der Bundeswehr
Universität München

Framework-Konzepte für Managementplattformen in föderierten softwarebasierten Netzen

Michael Steinke, M.Sc.

Vollständiger Abdruck der von der Fakultät für Informatik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften
(Dr. rer. nat.)

genehmigten Dissertation.

Gutachter:
Prof. Dr. Wolfgang Hommel
Prof. Dr. Helmut Reiser

Die Dissertation wurde am 16.05.2022 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Informatik am 08.07.2022 angenommen. Die mündliche Prüfung fand am 15.07.2022 statt.

Danksagung

Die vorliegende Arbeit entstand im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter der Professur für IT-Sicherheit von Software und Daten an der Universität der Bundeswehr München. Meinem Doktorvater und Inhaber der Professur, Prof. Dr. Wolfgang Hommel, möchte ich meinen besonderen Dank aussprechen. Seine uneingeschränkte Unterstützung, sein stets offenes Ohr und die vielen konstruktiven Anregungen haben wesentlich zum Gelingen dieses Vorhabens beigetragen. Ich bedanke mich auch sehr herzlich bei meinem Zweitgutachter, Herrn Prof. Dr. Helmut Reiser vom Leibniz-Rechenzentrum, dessen wertvolle Anmerkungen diese Arbeit abgerundet haben.

Mein Dank gilt auch allen Kolleginnen und Kollegen an der Professur und am Institut für Softwaretechnologie. Sie haben eine produktive und anregende Umgebung geschaffen, die nicht immer ganz ernst sein musste und mir viel Freude gebracht hat. Sie haben auch dazu beigetragen, dass ich den manchmal notwendigen Abstand zu meinem Promotionsvorhaben bekam.

Mein großer Dank gilt auch allen, die bei der Korrektur von Rechtschreibfehlern beigetragen haben: Meine beiden Gutachter, die mir nicht nur inhaltlich, sondern auch zur Rechtschreibung Rückmeldung gegeben haben, meine 93-jährige Oma Elisabeth und Katharina, die sich der Aufgabe mit Akribie widmeten.

Ich möchte mich auch bei meinen Eltern und meinem Bruder bedanken, die für mich alle Voraussetzungen bis hin zu diesem Schritt geschaffen und mich stets gefördert haben. Mein ausdrücklicher Dank gilt Katharina. Ihre Unterstützung, Geduld und ihr Verständnis erlaubten es mir, diesen Weg zu gehen.

München, Mai 2022

Kurzfassung

Heutige IT-Infrastrukturen unterscheiden sich stark von denen vor 20 Jahren. Hatte man in herkömmlichen Netzen zuvor noch einen sehr starken Fokus auf den Einsatz von IT-Ressourcen und Netzkomponenten als starre Verschmelzung aus Hardware und Funktion, so werden sie inzwischen in vielerlei Hinsicht als dynamisch, modular und vielschichtig verstanden: Paradigmen wie Netz- und Systemvirtualisierung sowie Network Functions Virtualization (NFV) erlauben einerseits eine feingranulare logische Kapselung verfügbarer IT-Ressourcen in verschachtelten virtuellen Netzen. Software-Defined Networking (SDN) beschreibt andererseits ein Managementparadigma für eine programmierbare sowie zentralisierte Verwaltung dieser Ressourcen. In dieser Arbeit werden derartige Netze als softwarebasierte Netze (SN) bezeichnet. Föderierte softwarebasierte Netze (FSN) beschreiben SNs, in denen IT-Ressourcen mehrerer Partner zusammengeschlossen, genutzt und auch gemeinsam koordiniert gemanagt werden. Das technische Werkzeug zur Unterstützung des Netzmanagements ist eine Managementplattform. Sie implementiert das Managementkonzept, die sogenannte Managementarchitektur.

FSNs können jedoch stark variierend ausgeprägt sein, beispielsweise in ihrem Zweck, der Zusammensetzung der IT-Ressourcen oder der Organisation des Managements. Eine einheitliche Managementarchitektur und eine diese umsetzende Managementplattform für FSNs kann es daher nicht geben, sondern vielmehr müssen darin fixe sowie an den jeweiligen Anwendungsfall anzupassende Bausteine berücksichtigt werden. Entsprechend erfüllen auch bestehende Managementplattformen nicht alle Anforderungen für einen geeigneten Einsatz in beliebigen FSNs und weisen höchstens in Teilmodellen eine gewisse Nutzbarkeit auf.

Das Ziel dieser Dissertation ist die Unterstützung der Spezifizierung und Implementierung von geeigneten Bausteinen von Managementarchitekturen in FSNs durch die Entwicklung und Bereitstellung entsprechender Framework-Konzepte. Ein besonderer Fokus dieser Arbeit liegt dabei in der geeigneten Realisierbarkeit dieser Architekturbausteine in einer Managementplattform, sowie der Definition von Schnittstellen der Bausteine untereinander im Sinne der Bereitstellung eines zusammenhängenden Gesamtkonzepts. Im Rahmen einer systematischen Vorgehensweise werden in dieser Dissertation dazu einerseits grundlegende Charakteristika des Betriebs von FSNs sowie andererseits von Föderationen in SNs neu erarbeitet. Auf dieser Basis werden begründet drei repräsentative Szenarien und 80 Anforderungen an Managementplattformen in FSNs abgeleitet und beschrieben. Den Kernbeitrag der Arbeit stellen die beschriebenen Framework-Konzepte dar. Sie können allein für sich stehend genutzt werden, um bestehende Managementplattformen in einzelnen Teilbereichen für das Management von FSNs zu komplementieren. Sie können jedoch auch als zusammenhängendes Gesamtkonzept genutzt werden, um anwendungsfallspezifisch geeignete Managementplattformen von Grund auf neu zu entwickeln. Die Frameworks wurden zu großen Teilen implementiert und ihre Eignung zur Beschreibung einer Managementarchitektur für FSNs im Kontext eines beispielhaften Szenarios angewendet. In einer zusätzlichen Evaluierung werden schließlich die Erfüllung von Performanzkriterien an die Frameworkimplementierung der Kernprozesse des Netzmanagements untersucht.

Abstract

Present IT infrastructures differ immensely from those of 20 years ago. Whereas in conventional networks, IT resources have been understood as an inflexible fusion of hardware and functionality, they are now comprehended as dynamic, modular and multi-layered: On the one hand, paradigms like network and system virtualization as well as Network Functions Virtualization (NFV) allow a fine-grained deployment of IT resources in nested virtual networks. Software Defined Networking (SDN), on the other hand, describes a management paradigm that allows a programmable and centralized management of those resources. In this work, these kinds of networks are referred to as Software Networks (SN). Federated Software Networks (FSN) describe SNs that consist of IT resources from multiple partners and that are used and managed by them collaboratively. The technical tool supporting network management is called a management platform. A management platform implements the management concept that is referred to as management architecture.

FSNs, however, may vary significantly, for instance in terms of their purpose, composition of IT resources or their organization. A unified management architecture and consequently a unified management platform that implements it cannot exist for FSNs, but rather components with fixed and adaptable pieces for different application cases. Consequently, existing management platforms do not meet all requirements for their use in arbitrary FSNs. They are rather usable in partial aspects.

The objective of this doctoral thesis is the specification and implementation of components of management architectures for FSNs and their implementation and provision via concepts of software frameworks. A special focus of this work is the practical usability of these architectural components in a management platform and the definition of interfaces between the components in order to provide a coherent holistic concept. In the course of a systematic approach, this thesis defines fundamental operational characteristics of FSNs and furthermore of characteristics of federations in SNs. On that basis, this thesis derives three well-founded representative scenarios along with 80 requirements for management platforms in FSNs. The major contribution of this work is the design of frameworks for management platforms. They can be used to complement existing management platforms in certain aspects in order to make them usable for the management of FSNs. They can also be used as a coherent overall concept in order to implement suitable management platforms for certain application cases from scratch. Major parts of the frameworks have been implemented and evaluated in an exemplary scenario. An additional evaluation assesses the ability of the frameworks' implementations to meet performance criteria of essential processes of network management.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation für föderierte softwarebasierte Netze . . . 2
- 1.2 Problemstellung und Zielsetzung . . . 3
- 1.3 Fragestellungen im Rahmen der Arbeit . . . 5
  - 1.3.1 Allgemeine Fragestellungen . . . 6
  - 1.3.2 Informationsmodell . . . 6
  - 1.3.3 Funktionsmodell . . . 6
  - 1.3.4 Organisationsmodell . . . 7
  - 1.3.5 Kommunikationsmodell . . . 7
- 1.4 Neue Beiträge und Kernbeiträge dieser Arbeit . . . 7
  - 1.4.1 Neue Beiträge als Grundlage für Kernbeiträge . . . 7
  - 1.4.2 Kernbeiträge der Arbeit . . . 8
- 1.5 Methodik und Aufbau der Arbeit . . . 8

2 Grundlegende Begriffe **11**
- 2.1 Softwarebasierte Netze . . . 12
  - 2.1.1 Netz- und Systemvirtualisierung . . . 12
  - 2.1.2 Software-Defined Networking (SDN) . . . 14
  - 2.1.3 Network Functions Virtualization (NFV) . . . 15
- 2.2 Föderationen im Kontext softwarebasierter Netze . . . 17
  - 2.2.1 Ausprägung von Föderationen in verwandten Bereichen . . . 17
  - 2.2.2 Charakteristika mit besonderer Manifestation in SNs . . . 20
- 2.3 Frameworks für Softwareanwendungen . . . 25
  - 2.3.1 Zweck und Arten von Softwareframeworks . . . 26
  - 2.3.2 Entwicklung und Dokumentation von Frameworks . . . 27
- 2.4 Netzmanagement und Managementplattformen . . . 28
  - 2.4.1 Managementarchitekturen . . . 29
  - 2.4.2 Managementplattformen . . . 31

3 Anforderungsanalyse **35**
- 3.1 Der Betrieb föderierter softwarebasierter Netze . . . 36
  - 3.1.1 Charakteristika des Betriebs . . . 36
  - 3.1.2 Anforderungen auf Basis der Charakteristika von FSNs . . . 38
- 3.2 Methodik zur Szenarienauswahl und Anforderungsableitung . . . 43
  - 3.2.1 Auswahl der Szenarien . . . 43
  - 3.2.2 Anforderungsableitung . . . 43
- 3.3 Szenario 1: Kooperation im Rahmen eines Projekts . . . 44
  - 3.3.1 IT-Infrastruktur . . . 44
  - 3.3.2 Organisation . . . 47
  - 3.3.3 Bedeutung für das Informationsmodell . . . 51
  - 3.3.4 Bedeutung für das Funktionsmodell . . . 53
  - 3.3.5 Bedeutung für das Kommunikationsmodell . . . 54

3.3.6 Einordnung des Szenarios . . . 55
3.4 Szenario 2: SNs als Wegbereiter der Digitalisierung in der Medizin . . . 55
3.4.1 Relevanz softwarebasierter Netze in der Telemedizin . . . 56
3.4.2 Föderationsbeziehungen und Aufgaben . . . 57
3.4.3 IT-Infrastruktur . . . 59
3.4.4 Organisation . . . 62
3.4.5 Bedeutung für das Informationsmodell . . . 66
3.4.6 Bedeutung für das Funktionsmodell . . . 67
3.4.7 Bedeutung für das Kommunikationsmodell . . . 70
3.4.8 Einordnung des Szenarios . . . 71
3.5 Szenario 3: Bedarfsabhängige Erweiterung von IT-Ressourcen . . . 71
3.5.1 Föderation und IT-Infrastruktur . . . 72
3.5.2 Organisation . . . 74
3.5.3 Bedeutung für das Informationsmodell . . . 76
3.5.4 Bedeutung für das Kommunikationsmodell . . . 76
3.5.5 Bedeutung für das Funktionsmodell . . . 76
3.5.6 Einordnung des Szenarios . . . 76
3.6 Anforderungen an Frameworks für Managementplattformen in FSNs . . . 77
3.6.1 Gewichtung und Ableitung der Anforderungen . . . 78
3.6.2 Hot-Spot Analyse . . . 78
3.6.3 Nicht-funktionale Anforderungen . . . 80
3.6.4 Funktionale Anforderungen . . . 81
3.6.5 Anforderungen an die Umsetzung des Informationsmodells . . . 84
3.6.6 Anforderungen an die Umsetzung des Kommunikationsmodells . . . 85
3.6.7 Anforderungen an die Umsetzung des Organisationsmodells . . . 90
3.6.8 Anforderungen an die frameworkspezifische Umsetzung . . . 92
3.6.9 Zusammenfassung der Kernanforderungen . . . 92

4 Gegenwärtiger Stand der Forschung und Technik 95
4.1 Kriterien zur Auswahl der untersuchten Vorarbeiten . . . 96
4.2 Standards zu Netzmanagementframeworks . . . 96
4.2.1 SNMP-Management-Framework . . . 96
4.2.2 NETCONF und YANG . . . 98
4.2.3 ETSI Zero Touch Network & Service Management (ZSM) . . . 100
4.2.4 Standards aus dem Cloud-Computing: CIMI und OCCI . . . 102
4.2.5 Standards für Netzereignisse und -Zustände . . . 104
4.3 Managementplattformen für Software-Defined Networking . . . 106
4.3.1 OpenDaylight (ODL) . . . 106
4.3.2 Open Network Operating System (ONOS) . . . 109
4.4 Managementplattformen für Network Functions Virtualization . . . 113
4.4.1 Übersicht über NFV-Plattformen . . . 113
4.4.2 Open Source MANO (OSM) . . . 113
4.5 Paradigmenübergreifende Managementplattformen . . . 118
4.5.1 XOS (CORD) . . . 118
4.5.2 OpenStack . . . 121
4.5.3 Open Network Automation Platform (ONAP) . . . 127
4.6 Zusammenfassung der Auswertung . . . 131

5 Framework-Konzepte für Managementplattformen in FSNs 135
5.1 Operative Managementprozesse . . . 136
5.1.1 Überwachung . . . 137
5.1.2 Steuerung . . . 138
5.2 Darstellung von Modellierungskonzepten . . . 139
5.3 Modellübergreifende Komponenten und Hilfselemente . . . 140
5.3.1 Hilfselemente . . . 140

5.3.2 Tagging von Elementen . . . 141
5.4 Informationsmodell . . . 141
5.4.1 Framework zur Modellierung von Managementobjektklassen . . . 142
5.4.2 Framework für Managementbeziehungen . . . 148
5.4.3 Framework für MOC-Funktionen . . . 153
5.4.4 Framework für Netzereignisse und -Zustände . . . 159
5.4.5 Framework für Alarme . . . 162
5.4.6 Unterscheidung von Ist- und Soll-Zustand . . . 164
5.5 Organisationsmodell . . . 164
5.5.1 Framework für Föderationsmodelle und -Beziehungen . . . 165
5.5.2 Framework für administrative Domänen . . . 167
5.5.3 Nutzer- und Rollen-Framework . . . 171
5.5.4 Framework für Managementaufgaben . . . 172
5.5.5 Mandantenfähiger Zugriff auf Informationselemente . . . 174
5.6 Kommunikationsmodell . . . 181
5.6.1 Southbound-Interface . . . 181
5.6.2 Northbound-Interface und Inter-Plattform-Kommunikation . . . 189
5.6.3 Datenbankanbindung . . . 202
5.6.4 Zentrale Komponenten des Kommunikationsmodells . . . 207
5.7 Funktionsmodell . . . 209
5.7.1 Funktionale Managementbereiche . . . 210
5.7.2 Erweiterung der Managementplattformfunktionalität . . . 211
5.7.3 Automatisierung der Kernprozesse . . . 213
5.7.4 Verwaltung von Managementanwendungen . . . 215
5.8 Zusammenfassung der Kernkonzepte für FSNs . . . 215

6 Prototypische Implementierung 219
6.1 Festlegung des Implementierungsrahmens . . . 220
6.2 Grundlegendes zur Implementierung . . . 221
6.2.1 Ausgangsbasis der Implementierung . . . 221
6.2.2 Allgemeine Umsetzung und Struktur . . . 221
6.3 Frameworks des Informationsmodells . . . 221
6.3.1 MOC-Framework . . . 222
6.3.2 Framework für Managementbeziehungen . . . 223
6.3.3 Framework für MOC-Funktionen . . . 224
6.3.4 Framework für Netzereignisse und -Zustände . . . 225
6.3.5 Framework für Alarme . . . 225
6.4 Frameworks des Organisationsmodells . . . 226
6.4.1 Framework für Föderationsmodelle und -Beziehungen . . . 226
6.4.2 Framework für administrative Domänen . . . 226
6.4.3 Nutzer- und Rollen-Framework . . . 227
6.4.4 Framework für Managementaufgaben . . . 227
6.4.5 Gültigkeitsbereiche und Zuständigkeiten . . . 228
6.4.6 Mandantenfähiger Zugriff auf Informationselemente . . . 228
6.5 Frameworks des Kommunikationsmodells . . . 230
6.5.1 Southbound-Interface . . . 230
6.5.2 Northbound-Interface . . . 233
6.6 Frameworks des Funktionsmodells . . . 236
6.6.1 Funktionserweiterung und Automatisierung der Kernprozesse . . . 236
6.6.2 Funktionale Managementbereiche . . . 237

7 Beispielhafte Anwendung der Frameworks 239
7.1 Beschreibung des Anwendungsszenarios . . . 240
7.1.1 Begründung der Ausprägungen des Basisszenarios . . . 240
7.1.2 Auswahl des Basisszenarios und Anpassungen . . . 241

7.2 Informationsmodell . . . 243
7.2.1 Managementobjektklassen . . . 243
7.2.2 Managementbeziehungen . . . 246
7.2.3 MOC-Funktionen . . . 249
7.2.4 Beispielhafte Netzinformationen . . . 252
7.2.5 Beispielhaftes Alarm-Schema . . . 255
7.3 Organisationsmodell . . . 257
7.3.1 Managementaufgaben . . . 258
7.3.2 Administrative Domänen und Überschneidungsbehandlung . . . 258
7.3.3 Nutzer und Rollen . . . 260
7.3.4 Gültigkeitsbereiche und Zuständigkeiten . . . 260
7.3.5 Föderationsmodell . . . 262
7.3.6 Zugriffsverwaltung . . . 263
7.4 Kommunikationsmodell . . . 268
7.4.1 Southbound-Interface . . . 268
7.4.2 Northbound-Interface . . . 272
7.5 Funktionsmodell . . . 275
7.5.1 Funktionale Managementbereiche . . . 275
7.5.2 Framework zur Erweiterung der Managementplattformfunktionalität . . . 276
7.6 Zusammenfassung des Kapitels . . . 278

8 Evaluierung der Frameworks 279
8.1 Evaluierungssystem . . . 279
8.2 Performanz von Kernprozessen . . . 280
8.2.1 Sichtbarkeits- und Zugriffsentscheidungen . . . 280
8.2.2 Benachrichtigungsprozess . . . 281
8.2.3 Closed-Loop-Automatisierung . . . 293
8.3 Abgleich der Anforderungen . . . 297
8.3.1 Anforderungen an das Funktionsmodell . . . 298
8.3.2 Anforderungen an das Informationsmodell . . . 300
8.3.3 Anforderungen an das Kommunikationsmodell . . . 300
8.3.4 Anforderungen an das Organisationsmodell . . . 301
8.3.5 Anforderungen an die Umsetzung . . . 302
8.3.6 Bewertung der Erfüllung der Anforderungen . . . 302

9 Fazit 305
9.1 Beantwortung der Forschungsfragen . . . 305
9.1.1 Allgemeine Fragestellungen . . . 305
9.1.2 Informationsmodell . . . 306
9.1.3 Funktionsmodell . . . 307
9.1.4 Organisationsmodell . . . 308
9.1.5 Kommunikationsmodell . . . 309
9.2 Zusammenfassung dieser Arbeit . . . 309
9.3 Ausblick auf weiterführende Forschungsfragen . . . 312

Literatur 319

Abbildungsverzeichnis 331

Tabellenverzeichnis 335

A Veröffentlichungen im Kontext dieser Arbeit 337

Kapitel 1

Einleitung

Inhalt

1.1 Motivation für föderierte softwarebasierte Netze 2
1.2 Problemstellung und Zielsetzung 3
1.3 Fragestellungen im Rahmen der Arbeit 5
1.3.1 Allgemeine Fragestellungen 6
1.3.2 Informationsmodell 6
1.3.3 Funktionsmodell 6
1.3.4 Organisationsmodell 7
1.3.5 Kommunikationsmodell 7
1.4 Neue Beiträge und Kernbeiträge dieser Arbeit 7
1.4.1 Neue Beiträge als Grundlage für Kernbeiträge 7
1.4.2 Kernbeiträge der Arbeit 8
1.5 Methodik und Aufbau der Arbeit 8

Rechnernetze sind bereits seit Jahrzehnten fest in den Alltag eines Jeden integriert und unterstützen die moderne Gesellschaft in der Erfüllung ihrer Aufgaben und Vorhaben. Die Kernelemente von Rechnernetzen werden üblicherweise als *Netzkomponenten* bezeichnet und stellen jeweils unterschiedliche Netzfunktionen bereit. Dazu zählen beispielsweise die Vernetzung von IT-Ressourcen mit Hilfe von *Switches* (z. B. Notebooks und Desktop-PCs), zur Vernetzung mehrerer Computernetze über *Router*, oder die Überwachung und Steuerung des Netzverkehrs durch z. B. *Network Intrusion Detection Systeme* und *Firewalls*. In herkömmlichen hardwarebasierten Netzarchitekturen (HN) und -konzepten erfolgt die Umsetzung der Netzfunktionen üblicherweise als aufeinander zugeschnittene Kombination aus jeweils spezieller physischer Komponente, welche IT-Ressourcen wie Rechenleistung oder Speicher bereitstellt, mit einer eigens angepassten Software zur Erfüllung der Netzfunktion. Derartige Systeme werden auch als „Hardwareappliances“ bezeichnet und stellen in der Regel ein in sich geschlossenes Gesamtsystem mit oft beschränkten Konfigurationsmöglichkeiten dar. Aufgrund einer für ihre jeweilige Funktion spezialisierte Hardware und darauf abgestimmte Software sind Hardwareappliances oft besonders performant in der Erfüllung ihrer Funktion. Auf der anderen Seite bieten sie jedoch einige Nachteile, die sich insbesondere durch auftretende Probleme und Schwierigkeiten im Management der Netze manifestieren:

- Die Installation von Hardwareappliances im Netz erfordert dedizierten räumlichen Platz und Strom [1].

- Ihre Entwicklung, Integration und ihr Betrieb sind mit großen Investitionen und erheblichem Aufwand verbunden und benötigen speziell ausgebildetes Personal [1].
- Mit kürzeren Innovationszyklen veralten Hardwareappliances sehr schnell [1].
- Sie haben eine fest integrierte Steuerlogik, d. h. sie sind nicht um neue Funktionen erweiterbar [2]. Klassische HN haben daher einen weitgehend statischen Charakter [3].
- HN und ihr Management skalieren nur sehr schlecht für eine sehr große Anzahl an Netzkomponenten und können nicht mehr manuell betrieben werden [3]. So erfordert die Konfiguration physischer Komponenten beispielsweise oft eine Behandlung vor Ort.

1.1 Motivation für föderierte softwarebasierte Netze

Softwarebasierte Netze (SN) sollen zur Lösung der Probleme von HN beitragen. In diesen Netzen sind Funktionen, die in HN ehemals hardwarebasiert bereitgestellt wurden, durch Softwarelösungen ersetzt worden [4]. Die wichtigsten Paradigmen hierbei sind die System- und Netzvirtualisierung, *Network Functions Virtualization* (NFV), sowie *Software-Defined Networking* (SDN) [5]. Eine detaillierte Erläuterung von SNs wird in Abschnitt 2.1 beschrieben. Im Kontext des NFV-Paradigmas zielen SNs darauf ab, Netzfunktionen generell in Computernetzen über leistungsfähige Server zu virtualisieren [1]. Die Steuerung des Netzes ist logisch zentralisiert und ermöglicht zudem seine Programmierbarkeit [3].

Durch die neuen Paradigmen vereinfachen SNs darüber hinaus den Betrieb mandantenfähiger und geographisch verteilter Netze [1]. Auf diese Weise bieten sie insbesondere für Föderationen eine ideale Basis. **Föderierte Softwarebasierte Netze** (FSN) beschreiben eine organisatorische Ausprägung von SNs, in denen zwei oder mehrere Organisationen in einer Kooperation IT-Ressourcen über SNs gemeinsam nutzen, um ein gemeinsames oder komplementäres Ziel zu erreichen. Der Hintergrund einer Föderation von IT-Netzen und -Ressourcen kann allgemein vielfältig sein. Beispielsweise kann eine Zusammenführung zweier ursprünglich getrennter Netze die Anbindung von Diensten erleichtern, wie es oft auch im wissenschaftlichen Umfeld im Rahmen von Forschungsnetzen wie dem Münchener Wissenschaftsnetz (MWN) oder dem darüberliegenden Deutschen Forschungsnetz (DFN) bereits Praxis ist. Auch kann durch FSNs beispielsweise eine nahtlose Integration von gemieteten IT-Ressourcen, z. B. aus Cloud-Computing-Diensten, mit vor Ort betriebenen IT-Ressourcen umgesetzt werden.

Eine beispielhafte, generell gehaltene Ausprägung von FSN, zeigt Abbildung 1.1. Physische IT-Ressourcen – handelsübliche Serversysteme – mehrerer Organisationen können jeweils über einen oder mehrere geographisch voneinander getrennte, aber miteinander verbundene Standorte verteilt sein. Die physischen IT-Ressourcen dienen ausschließlich als Hosts für Netz- und Systemvirtualisierung, welche potenziell logisch über mehrere Datenzentren hinweg miteinander verbunden sind. Virtualisierte IT-Ressourcen werden dabei über entsprechende Plattformen zentral verwaltet und bilden selbst ein auf die jeweilige Organisation beschränktes SN. Eine Föderation softwarebasierter Netze über mehrere Organisationen hinweg wird potenziell auf bereits bestehende Infrastrukturen der jeweiligen Organisation aufgebaut. Dabei stellt die gewollte Heterogenität der föderierten Netze eine besondere Herausforderung dar – Föderationspartner sollen die Möglichkeit haben, unterschiedliche Systeme in die Föderation einzubringen. Die Föderation muss über unterschiedliche physische Systeme, Technologien der Netz- und Systemvirtualisierung sowie optional zusätzliche heterogene Systeme zum *Management und zur Orchestrierung* (MANO, angelehnt an den gleichnamigen Begriff des ETSI-NFV-Architekturframeworks in [6]) hinweg realisiert werden.

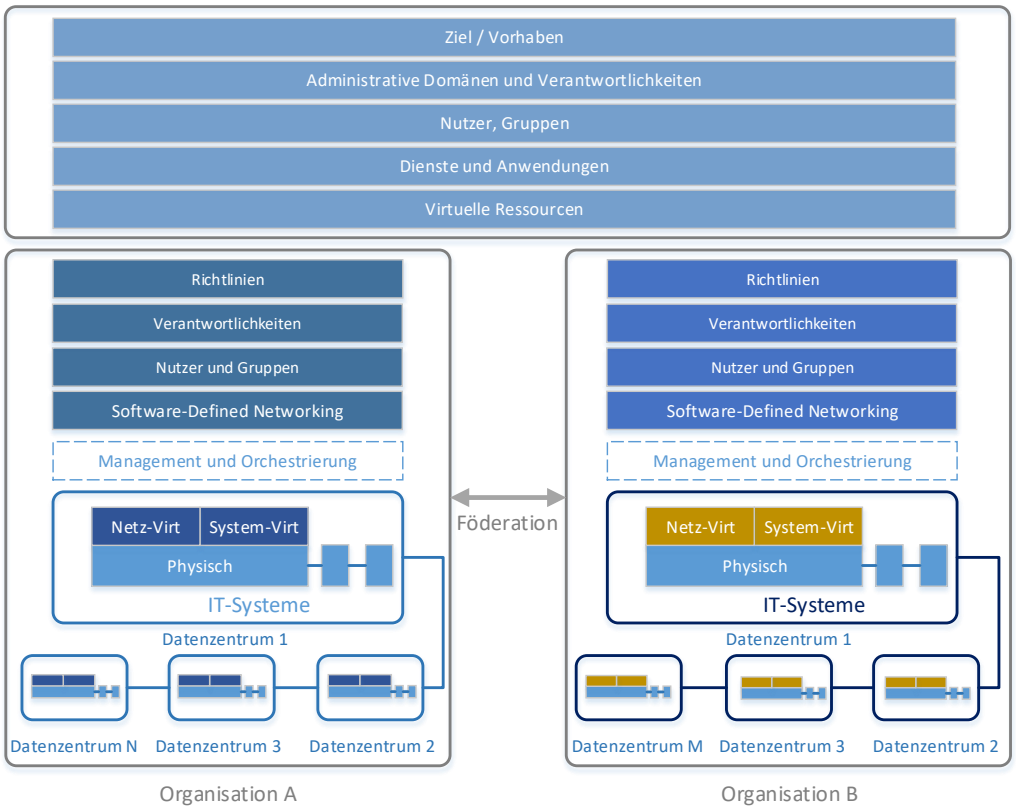


Abbildung 1.1: Überblick über die Architektur mit beispielhaften Komponenten in FSNs.

Auch die Organisation des Netzmanagements unterscheidet sich über die einzelnen Partner. Unterschiedliche Nutzer und Nutzergruppen erfüllen dabei in ihrer jeweils eigenen Organisation Verantwortlichkeiten, die über mehrere Organisationen hinweg heterogen festgelegt sind: Beispielsweise einerseits durch ein kleines IT-Team ohne weitere Untergruppierungen oder andererseits in einer anderen Organisation durch eine große und nach Aufgabengebiet untergliederte IT-Abteilung. Über unterschiedliche Organisationen werden zudem unterschiedliche Sätze an Richtlinien und Richtlinienausprägungen verfolgt, die in einer föderierten Umgebung berücksichtigt werden müssen.

Als Föderation stellen die Datenzentren der kooperierenden Partner schließlich virtuelle IT-Ressourcen bereit, die unter den Nutzern und im Kontext der Föderation definierten Nutzergruppen genutzt werden, um gemeinsame Anwendungen oder Dienste koordiniert betreiben zu können. Auch organisatorisch wird in der Föderation ein gültiges Modell mit administrativen Domänen sowie darin zu erfüllenden Verantwortlichkeiten verfolgt, um schließlich als Föderation ein gemeinsames Ziel zu erreichen.

1.2 Problemstellung und Zielsetzung

Eine wesentliche Grundvoraussetzung für den Nutzen und die Unterstützung von Anwendern bei der Erfüllung ihrer Aufgaben durch Computernetze ist ihre Zuverlässigkeit, beispielsweise hinsichtlich der Verfügbarkeit und Sicherheit von Diensten. **Netzmanagement** ist die Kernaufgabe zur Sicherstellung der Zuverlässigkeit eines Netzes. Es umfasst Aktivitäten, Methoden, Prozeduren und Softwarewerkzeuge zum Betrieb, der Administration, Aufrechterhaltung und der Bereitstellung von IT-Netzen und Diensten [7]. **Integriertes Management** (IM) befasst sich mit der Umsetzung eines disziplin- und funktionsübergreifenden Managements in hete-

rogenen Umgebungen unter Verwendung offener Programmier- und Nutzerschnittstellen. Systeme zum Management der Netze arbeiten auf Basis gemeinsam akzeptierter Kompatibilitätsstandards zusammen [8]. IM hat insbesondere auch durch den Anspruch, alle Netzressourcen in einer Umgebung zentral durch ein Softwarewerkzeug bzw. mehrere zusammenarbeitende Softwarewerkzeuge zu managen [9]. Insofern ermöglicht IM die Erfassung und Berücksichtigung des Gesamtzustands eines gemanagten Netzes durch die Zusammenführung der Funktionen und Informationen des Netzmanagements über ein zentrales System, das auf einer Managementplattform aufbaut. Beispielsweise können auftretende Störungen durch sicherheitsrelevante Ereignisse wie die Kompromittierung von Systemen und Diensten oder Schadsoftware ausgelöst werden.

Ein großer Teil vorhandener Managementplattformen und -konzepte wurde auf Basis etablierter HN entwickelt und auch nur unvollständig und schrittweise mit dem fließenden Übergang zu SNs in ihrem Funktionsumfang erweitert. Beispielsweise wurde die aktuellste Version des im Netzmanagement wichtigen *Simple Network Management Protocol* [10] im Jahr 2002 spezifiziert; SNs in ihrem heutigen Ausmaß waren zu diesem Zeitpunkt noch nicht vorhanden. Vor allem dem Konzept der durch SNs einfach nutzbaren föderierten Netze wird kaum Beachtung geschenkt. SNs beeinflussen das Netzmodell und Organisationsmodell. Möglichkeiten einer Netzföderation und der Kooperation zwischen mehreren Partnern, die durch bestehende Managementlösungen nur unzureichend abgedeckt werden, werden im Folgenden anhand zweier Beispiele verdeutlicht.

Administrative Domänen. In HN ist eine Netzföderation in der Regel auf die gemeinsame Nutzung vorab festgelegter Dienste beschränkt. Beispiele sind die Bereitstellung eines Netzzugangs durch den Netzbetreiber an Kunden und Föderationspartner oder der Zugriff zu zentralen Diensten wie es in Forschungs- und Wissenschaftsnetzen wie dem *Münchner Wissenschaftsnetz* üblich ist [11]. Administrative Domänen sind bei derartiger Umsetzung einer Netzföderation üblicherweise eindeutig getrennt, da der Betreiber des jeweils angebotenen und geteilten Dienstes in der Regel für die Zuverlässigkeit des Dienstes verantwortlich ist. Föderierte softwarebasierte Netze hingegen ermöglichen auf Basis darin geteilter physischer IT-Ressourcen der Föderationspartner die Konfiguration und Nutzung vieler voneinander isolierter virtueller Netze. Diese können durch beliebig viele Nutzer mandantenfähig und beliebig skalierbar erweitert und zentral konfiguriert werden. Der Betrieb von FSNs ist verglichen mit HN insofern grundlegend anders: Lediglich das Management der physischen Plattform von FSNs muss durch den jeweiligen Betreiber am Standort der Komponente vorgenommen werden. Die eigentlichen virtualisierten Netzfunktionen hingegen können von jedem Nutzer in der Netzföderation von überall gemanagt und unmittelbar umkonfiguriert werden. Managementplattformen müssen einen solchen Betrieb von FSNs unterstützen und vor allem technisch umsetzen können.

Heterogenität in FSNs. Eine weitere Herausforderung bei der Umsetzung von Managementplattformen ist die Heterogenität in FSNs. Beliebige softwarebasierte Netze sind dadurch nicht auf Basis einer fertigen Komplettlösung realisierbar, sondern eine Komposition vieler lose miteinander verbundener Systeme, wie von Moreno-Vozmediano im Kontext von Cloud-Föderationen beschrieben wird [12]. Cloud-Plattformen und Virtualisierungssysteme bieten jedoch keine standardisierten Managementschnittstellen. Die im IM geforderte Interoperabilität von für das Management relevanten Systemen ist daher in FSNs kaum möglich: Unterschiedliche Föderationspartner bauen softwarebasierte Netze üblicherweise auf jeweils unterschiedlichen Systemen und teils bereits vorhandener Infrastruktur auf. Managementplattformen in FSNs müssen daher mit verschiedensten Systemen der Virtualisierung, Steuerung und Orchestrierung interagieren und Funktionen sowie Informationen von diesen konsolidieren können. Die Schnittstellen von MANO-Systemen werden kontinuierlich vom jeweiligen Hersteller angepasst; die Wirksamkeit des IM ist direkt von der Fähigkeit der IM-Plattform abhängig, Funktionen über die entsprechende Schnittstelle in MANO-Systemen weiterzunutzen.

Klassische **Architekturen von Netzmanagementplattformen** sind daher aufgrund ihres gleichfalls eher statischen Charakters in FSNs weitgehend ungeeignet: Sie sind an die Charakteristika herkömmlicher HN, ihre beschränkte Konfigurierbarkeit sowie darin genutzter langlebiger standardisierter Protokolle des Netzmanagements (z. B. SNMP) angepasst. Deutlich flexiblere, dem schnellen Wandel der Umgebung, der Heterogenität und individuellen Anwendungsfällen in FSNs angepasste Architekturen in Managementplattformen sind notwendig, um die Zuverlässigkeit der Dienste, Komponenten und Informationen darin weiterhin über ein zentrales Softwarewerkzeug zu gewährleisten. Dazu müssen Managementplattformen selbst ausreichend Flexibilität und Konfigurierbarkeit bieten können.

Die **Zielsetzung** dieser Arbeit ist daher die *Generalisierung* der Softwarearchitektur für Managementplattformen in FSNs. Dazu werden einerseits fixe, andererseits notwendige flexible Komponenten der Softwarearchitektur anhand der Charakteristika von FSNs identifiziert und als Gesamtes in Form von zusammenhängenden Softwareframeworks konzipiert und ihre Kernkomponenten implementiert. Die resultierenden Softwareframeworks sollen Entwickler durch eine vorgegebene, jedoch *anpassbare Struktur* dabei unterstützen, Managementplattformen für verschiedene Anwendungsfälle von FSNs vergleichsweise einfach zu entwickeln oder bestehende Managementplattformen um anwendungsfallspezifisch fehlende Teile zu erweitern. Die Softwareframeworks sollen insbesondere dabei helfen, ein Management von FSNs basierend auf bereits bestehenden Infrastrukturen der Föderationspartner aufzubauen und diese nutzbar zu machen. Dabei wichtige Eigenschaften umfassen

- eine modulare Frameworkarchitektur zur einfachen Erweiterung der vier Teilmodelle einer Managementarchitektur (Organisationsmodell, Informationsmodell, Kommunikationsmodell und Funktionsmodell),
- die Modellierbarkeit und Berücksichtigung verschiedener Föderationsmodelle zwischen unterschiedlichen Organisationen und damit einhergehende Verantwortlichkeiten, administrative Domänen und anderer organisatorischer Aspekte,
- die Berücksichtigung besonderer Herausforderungen und Charakteristika in FSNs, z. B. Mandantenfähigkeit und Skalierbarkeit über mehrere Föderationspartner, welche auch durch Erweiterungen des Frameworks erhalten bleiben sollen,
- die Möglichkeit zur Einbindung beliebiger vorhandener Funktionen und Informationen aus insbesondere FSN-spezifischen Systemen, aber auch FSN-unspezifischen zu berücksichtigenden Systemen,
- geeignete systeminterne Prozesse und Datenmodelle, Datenstrukturen und Algorithmen zum automatischen Zusammenspiel der Modelle und unter Berücksichtigung der Erweiterbarkeit und Effizienz der Frameworks,
- eine breite Einsetzbarkeit in unterschiedlichen FSN-Anwendungsfällen.

1.3 Fragestellungen im Rahmen der Arbeit

Die in dieser Arbeit adressierten Problem- und Fragestellungen betreffen insbesondere geeignete ganzheitliche Umsetzungen des Netzmanagements in Managementplattformen für FSNs. Implementiert als Frameworks sollen sie generisch in beliebigen Ausprägungen von FSNs als Basis für Plattformen einsetzbar sein. Die in dieser Arbeit betrachteten Fragestellungen sind nach allgemeinen Aspekten und gemäß ihrer Zugehörigkeit zu einem der vier Teilmodelle einer Managementarchitektur gruppiert.

1.3.1 Allgemeine Fragestellungen

SNs – insbesondere im Kontext einer Föderation – unterscheiden sich grundlegend von herkömmlichen HN, nicht nur auf technischer Seite, sondern insbesondere durch eine unterschiedliche Betriebsweise. Als Grundlage zur Erstellung von Frameworks für Managementplattformen in FSNs müssen dabei grundlegende Fragen geklärt werden:

- **F1:** *Welche Eigenschaften im Betrieb unterscheiden (F)SNs von HN?*
- **F2:** *Wie wirken sich diese Eigenschaften auf Managementplattformen in FSNs aus?*
- **F3:** *Welche Funktionen und Eigenschaften müssen Managementplattformen in FSNs besitzen? Welche Aspekte davon müssen flexibel anwendungsfallspezifisch anpassbar, welche fix sein?*
- **F4:** *In welche Teilframeworks lassen sich die vier Teilmodelle einer Managementarchitektur in FSNs gruppieren und welche Schnittstellen gibt es zwischen ihnen?*

1.3.2 Informationsmodell

Das Informationsmodell beinhaltet Konzepte und Überlegungen bezüglich der Darstellung und Modellierung von Informationen, die für das Management von Bedeutung sind [8]. Aufgrund der Dynamik, Heterogenität und einer vielfältigen Nutzungsmöglichkeit von (F)SNs ist die Modellierung von Informationen eine der grundlegenden Herausforderungen. Zu klärende Fragestellungen umfassen die Folgenden:

- **F5:** *Wie wirkt sich die Dynamik in FSNs auf das Informationsmodell aus?*
- **F6:** *Welche Managementinformationen sind in FSNs zu berücksichtigen und wie hängen sie zusammen?*
- **F7:** *Wie lassen sich diese Informationen einfach modellieren und wie können neue, bisher unberücksichtigte Informationen in Verarbeitungsprozesse integriert werden?*
- **F8:** *Wie können Informationen in heterogenen FSNs normalisiert und vergleichbar gemacht werden?*

1.3.3 Funktionsmodell

Das Funktionsmodell bricht zum einen das gesamte Netzmanagement in Teilfunktionen herunter und adressiert für deren Implementierung notwendige feingranularere Funktionen und Dienste [8]. Für die Umsetzung des angestrebten Frameworks ergeben sich aufgrund der Umgebung folgende Fragestellungen:

- **F9:** *Wie kann Netzmanagement in a priori unbekannten Umgebungen unterstützt werden?*
- **F10:** *Welchen Einfluss hat der Übergang zu FSNs auf Managementfunktionen (z. B. FCAPS), Funktionsabhängigkeiten und Anforderungen an Managementplattformen selbst?*
- **F11:** *Wie kann der Funktionsumfang einer Managementplattform dynamisch erweitert werden?*

1.3.4 Organisationsmodell

Durch das Organisationsmodell einer Managementplattform werden im Netzmanagement involvierte Komponenten, deren Rollen und bereits grundlegende Kommunikationsmodelle beschrieben [8]. Im Rahmen der Umsetzung eines IM-Frameworks sind im Besonderen die folgenden Fragen zu klären:

- **F12:** *Wie ist die Föderations- und Organisationsstruktur in FSNs grundlegend aufgebaut und modellierbar?*
- **F13:** *Wie können Zugriffsberechtigungen auf Informationen in FSNs geeignet realisiert werden?*
- **F14:** *Wie können unterschiedliche Organisationsprinzipien der Föderationspartner in einer Plattform berücksichtigt werden?*

1.3.5 Kommunikationsmodell

Das Kommunikationsmodell beschreibt die Kommunikation zwischen den im Organisationsmodell identifizierten Komponenten im Management [8]. Aufgrund der in FSNs grundlegend veränderten Sichtweise (verglichen mit HN) des Managements – das Netzmanagement muss nicht mehr (nur) einzelne Komponenten, sondern heterogene Controller- / Managementsysteme selbst managen – sind andere Komponenten und Funktionsweisen zu berücksichtigen. Wesentliche Fragen umfassen dabei:

- **F15:** *Welche Netzkomponenten und Richtungen müssen bei der Kommunikation berücksichtigt werden?*
- **F16:** *Wie muss eine Managementplattformarchitektur gestaltet sein, damit sie ein zuverlässiges Management von FSNs erlaubt?*
- **F17:** *Wie können nicht-standardisierte Schnittstellen (insb. von Netzkomponenten) in einer Managementplattform handhabbar werden?*

1.4 Neue Beiträge und Kernbeiträge dieser Arbeit

In diesem Abschnitt werden wissenschaftliche Kernbeiträge dieser Arbeit für eine bessere Übersichtlichtkeit vorweggenommen. Dabei wird grundsätzlich zwischen einerseits im Rahmen dieser Dissertation herausgearbeiteten **neuen Beiträgen**, die so noch nicht in der bestehenden Literatur vorhanden waren, als auch andererseits resultierenden **Kernbeiträgen** der Arbeit unterschieden. Erstere sind notwendig, um eine fundierte Grundlage für die eigentlichen Kernbeiträge der Arbeit zu schaffen.

1.4.1 Neue Beiträge als Grundlage für Kernbeiträge

Die Erarbeitung einiger neuer wissenschaftlicher Beiträge umfasst die Klärung von *a)* grundlagennahen Forschungsaspekten, die das betrachtete Umfeld von Föderationen in SNs attribuierbar und für das Management greifbar machen sollen, andererseits *b)* szenarienbasierte Annahmen hinsichtlich der Anforderungen an Managementplattformen in FSNs.

Hinsichtlich *a)* bestehen neue Beiträge dieser Arbeit aus der Herausarbeitung bislang in der Literatur vernachlässigter grundlegender Eigenschaften von Föderationen in SNs in Form einer systematisch hergeleiteten **Morphologie** von Föderationscharakteristika, basierend auf einer begründeten Auswahl und Erweiterung ähnlicher Teilergebnisse aus verwandten Bereichen.

Diese Morphologie kann auch als Grundlage für Arbeiten mit ähnlichen Fragestellungen mit interorganisationalen Schwerpunkten herangezogen werden. Des Weiteren wird eine **allgemeine Charakterisierung des Betriebs** von SNs sowie FSNs vorgenommen. Diese ist besonders für die Herausarbeitung von Unterschieden des Managements von HN und (F)SNs von Bedeutung. Nicht nur für Managementlösungen, sondern auch aufseiten von Netz- und Sicherheitsfunktionen können diese Charakteristika beispielsweise für die Adaptierung geeigneter Funktionen darin verwendet werden. In Bezug auf *b)* werden insbesondere auch Szenarien für föderierte softwarebasierte Netze entwickelt, durch welche das Potenzial von FSNs aufgezeigt wird.

In diesen Beiträgen werden bereits einige wissenschaftliche Fragestellungen aus Abschnitt 1.3 geklärt, insbesondere F1, F2, F5 und teilweise F12. Andere wissenschaftliche Fragestellungen werden im Rahmen der eigentlichen Kernbeiträge geklärt.

1.4.2 Kernbeiträge der Arbeit

Die **Kernbeiträge** dieser Arbeit umfassen zunächst ein unter Berücksichtigung des aktuellen Stands der Forschung hergeleitetes und entworfenes Gesamtkonzept für ein in FSNs wiederverwendbares und erweiterbares Design von Managementplattformen in Form mehrerer zusammenhängender Frameworks.

Die Herleitung des Konzepts erfolgt auf Basis einer Vielzahl detailliert herausgearbeiteter szenarienspezifischer Anforderungen an eine jeweils geeignete Managementplattform und, unter anderem daraus abgeleitet, 80 Anforderungen an ein Framework – und folglich an eine szenarienübergreifende, adaptierbare Managementinfrastruktur. Das Design bestehender Managementplattformen und Managementkonzepte wird anhand der Anforderungen an ein im jeweiligen Bereich verwendbares Framework abgeglichen.

Der Schwerpunkt der in dieser Arbeit herausgearbeiteten **Konzepte für Frameworks** der Kernkomponenten für Managementplattformen in FSNs liegt im Besonderen einerseits auf der Nutzbarkeit des Frameworks als Ausgangsbasis und zur Ableitung einer szenarienspezifisch geeigneten Managementplattform für FSNs. Andererseits spielt vor allem die notwendige Funktionalität zur Unterstützung jeweils verwendeter Modelle, vor allem der gemanagten Umgebung, organisatorischer Abläufe und Abhängigkeiten in Föderationen eine wichtige Rolle. Kernelemente der Frameworks werden **implementiert** und im Rahmen eines beispielhaften **Evaluierungsszenarios** angewendet und hinsichtlich ihrer Nutzbarkeit und Performanzkriterien von Kernprozessen des Netzmanagements evaluiert.

1.5 Methodik und Aufbau der Arbeit

Die Methodik dieser Arbeit ist in Abbildung 1.2 zusammengefasst. Nachdem in diesem Kapitel der Hintergrund, die Motivation, Problemstellung und die daraus resultierenden und in dieser Dissertation näher betrachteten wissenschaftlichen Fragestellungen erläutert wurden, werden in Kapitel 2 grundlegende zusammenwirkende fachliche Bereiche und wichtige Begriffe erläutert. Zunächst werden darin SNs und ihre Paradigmen erläutert, gefolgt von einer Klärung des Föderationsaspekts in Netzen und ihre Ausprägungen, welche in einer von früheren Dissertationen mit verwandten Aufgabenstellungen abgeleiteten Morphologie von Föderations- und Organisationscharakteristika resultieren. Diese fließen in die in Kapitel 3 vorgestellte Anforderungsanalyse ein und dienen darin wesentlich zur begründeten Auswahl von drei repräsentativen Szenarien, sowie der Ableitung allgemeiner Charakteristika des Betriebs von FSNs. Beide bilden die Grundlage zur Aufstellung von Anforderungen an Managementplattformen in FSNs, die in einem Anforderungskatalog zusammengefasst werden, sowie einer Hot-Spot-Analyse, in der flexible und fixe Teile der in dieser Arbeit resultierenden Frameworks begründet festgelegt werden.

In Kapitel 4 wird schließlich der aktuelle Stand der Forschung und Technik mit dem aus der Anforderungsanalyse erstellten Anforderungskatalog abgeglichen. Dabei werden zum einen bestehende Standards und zum anderen bestehende Managementplattformen hinsichtlich ihrer Eignung evaluiert. Es werden Defizite und mögliche Lösungen aufgezeigt, die Teillösungen für diese Arbeit darstellen können.

In Kapitel 5 wird schließlich ein Gesamtkonzept für Frameworks für Managementplattformen in FSNs unter Berücksichtigung der Hot-Spot-Analyse und den Anforderungen entwickelt. Die Lösungen für die vier Teilmodelle von Managementarchitekturen werden durch eine Beschreibung der im Konzept unterstützten operativen Kernprozesse im Netzmanagement ergänzt. Die Frameworkkonzepte werden schließlich in Kapitel 6 implementiert und ihre Anwendbarkeit im darauffolgenden Kapitel 7 anhand eines begründeten Evaluierungsszenarios überprüft. In Kapitel 8 erfolgt schließlich die Evaluierung der Frameworks. Darin werden Kernprozesse des Managements auf Basis der zuvor beschriebenen prototypischen Implementierung hinsichtlich ihrer Performanz unter Berücksichtigung anwendungsspezifischer Ausprägungen evaluiert. Außerdem werden die erarbeiteten Lösungen im Kontext des Anforderungskatalogs bewertet.

Kapitel 9 beschließt diese Arbeit durch die Beantwortung der betrachteten Forschungsfragen, die Zusammenfassung der Kernergebnisse dieser Arbeit sowie einen Ausblick auf weiterführende Forschungsfragen, die an die Ergebnisse dieser Arbeit angeschlossen werden können.

Auf im Rahmen dieser Dissertation entstandene Veröffentlichungen wird jeweils im entsprechenden thematischen Abschnitt eingegangen. Eine Gesamtübersicht wird in Anhang A gezeigt.

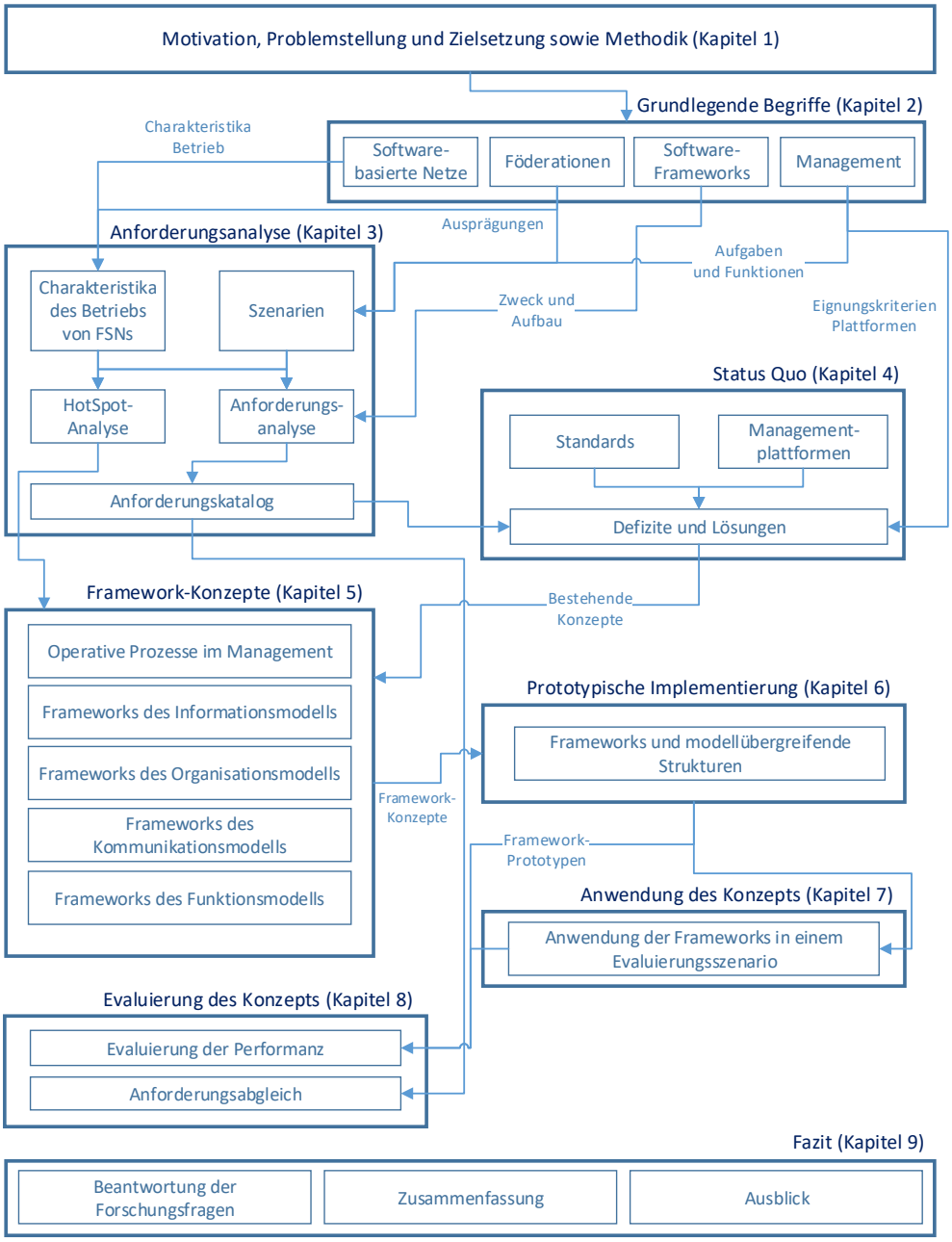


Abbildung 1.2: Vorgehensweise in dieser Arbeit.

Kapitel 2

Grundlegende Begriffe

Inhalt

2.1 Softwarebasierte Netze 12
2.1.1 Netz- und Systemvirtualisierung 12
2.1.2 Software-Defined Networking (SDN) 14
2.1.3 Network Functions Virtualization (NFV) 15
2.2 Föderationen im Kontext softwarebasierter Netze 17
2.2.1 Ausprägung von Föderationen in verwandten Bereichen 17
2.2.2 Charakteristika mit besonderer Manifestation in SNs 20
2.3 Frameworks für Softwareanwendungen 25
2.3.1 Zweck und Arten von Softwareframeworks 26
2.3.2 Entwicklung und Dokumentation von Frameworks 27
2.4 Netzmanagement und Managementplattformen 28
2.4.1 Managementarchitekturen 29
2.4.2 Managementplattformen 31

In diesem Kapitel werden grundlegende Begriffe aus verschiedenen, in dieser Arbeit zusammenwirkenden thematischen Bereichen geklärt. Im folgenden Abschnitt 2.1 wird der Begriff des **softwarebasierten Netzes** näher erläutert und eine erste Abgrenzung zu herkömmlichen Netzen formuliert. Abschnitt 2.2 geht auf die Besonderheiten von **Föderationen** von Infrastruktur und Diensten ein und umfasst eine ausführliche **Herleitung von Föderationscharakteristika** in FSNs basierend auf ähnlichen Bereichen. Abschnitt 2.3 erläutert den Zweck und Nutzen eines **Softwareframeworks** und zeigt Modelle zur Entwicklung auf. Gleichzeitig wird so eine Abgrenzung bezüglich der Entwicklung von Softwareanwendungen im herkömmlichen Sinne verdeutlicht. Abschnitt 2.4 klärt schließlich wichtige Begriffe und Elemente aus dem **Netzmanagement**. Schließlich werden ein typischer theoretischer Aufbau und wichtige Komponenten bisheriger Managementplattformen erläutert.

2.1 Softwarebasierte Netze

Softwarebasierte Netze setzen anstatt physischer Netzfunktionen auf in Software realisierte Netzlösungen [4]. Softwarebasierte Netze auszeichnende Kernparadigmata umfassen *Netz-* und *Systemvirtualisierung*, *Software-Defined Networking* (SDN) sowie *Network Functions Virtualization* (NFV) [5].

Ein softwarebasiertes Netz, wie es in dieser Arbeit betrachtet wird, erfüllt für das Netzmanagement zentrale Kerncharakteristika der drei Paradigmen:

- **Netz- und Systemvirtualisierung**: Eine hohe Flexibilität der IT-Infrastruktur durch Netz- und Netzkomponentenvirtualisierung, ursprünglich um die *Verknöcherung von Netzen* zu verhindern [13].
- **Software-Defined Networking**: Eine logisch zentralisierte Steuerung, bessere Automatisierbarkeit und Programmierbarkeit der Netze durch eine Trennung der Daten- und Kontrollebene [3].
- **Network Functions Virtualization**: Der grundsätzliche Einsatz virtualisierter Netzkomponenten auf sogenannter *Commercial-off-the-Shelf* -, d. h. handelsüblicher Hardware [6].

Der generelle Zweck der Weiterentwicklung von herkömmlichen Netzen zu softwarebasierten Netzen liegt insbesondere in einer Erleichterung ihres Managements. Die einzelnen Paradigmen werden in den folgenden Abschnitten näher erläutert.

2.1.1 Netz- und Systemvirtualisierung

Virtuelle IT-Infrastrukturen und -Netze, wie sie in dieser Arbeit im Vordergrund stehen, basieren grundlegend auf der Virtualisierung unterschiedlicher IT-Ressourcen: Zum einen der Virtualisierung von Systemen im Netz (sog. Systemvirtualisierung), zum anderen der Virtualisierung von Ressourcen zur Vernetzung selbst, d. h. Verbindungen zwischen Systemen (sog. Netzvirtualisierung).

2.1.1.1 Systemvirtualisierung

Systemvirtualisierung erlaubt eine effiziente Ausnutzung physischer IT-Ressourcen durch den parallelen Betrieb mehrerer virtueller Maschinen (VMs) auf demselben physischen Host [14]. Dienste können so in logisch getrennten Systemen ausgeführt werden. Nebeneffekte einer derartigen Architektur sind zum einen mehr Sicherheit durch die Systemtrennung verglichen mit einem Betrieb von Diensten auf demselben logischen System. Zum anderen erlauben VMs eine umfänglichere Managebarkeit im Vergleich zu physischen Systemen, da Funktionen wie das Einrichten, Starten, Ändern der Konfiguration und die Deinstallation von Systemen über APIs aus der Ferne durchführbar sind. Auch ist eine mehrschichtige Virtualisierung auf den VMs selbst gebräuchlich, beispielsweise um IT-Ressourcen weiter zu unterteilen.

Heutzutage vornehmlich eingesetzte Technologien der Systemvirtualisierung lassen sich gemäß [14] in zwei Kategorien einteilen: Zum einen Hypervisor-basierte Virtualisierung (auch Hardwarevirtualisierung genannt) und zum anderen Container-basierte Virtualisierung. Abbildung 2.1 zeigt die je nach Virtualisierungsart spezifische Architektur im Vergleich zu einem System, auf dem Anwendungen ohne Virtualisierung (System *(a)*) betrieben werden.

Hypervisor-basierte Virtualisierung (System *(b)*) basiert demnach auf einem sogenannten *Hypervisor*, über den virtuelle Systeme eingerichtet, Ressourcen wie CPU, Speicher und Netzadapter virtuell zugewiesen und ein isoliertes Gastbetriebssystem installiert werden, auf dem

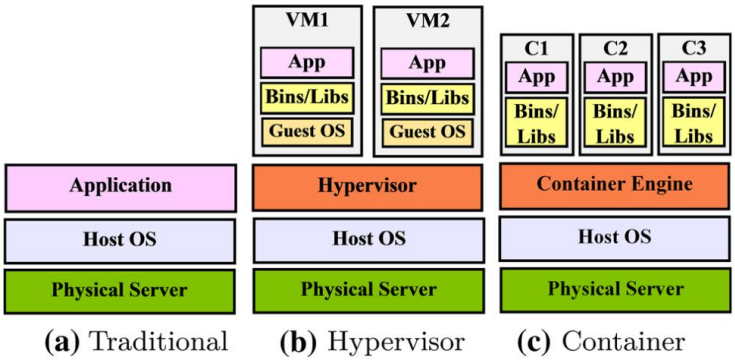


Abbildung 2.1: Arten der Systemvirtualisierung im Vergleich zu einem traditionellen System ohne Virtualisierung [14].

letztlich Anwendungen betrieben werden. Das Gastbetriebssystem kann sich dabei komplett vom Hostbetriebssystem bis hin zu Hardwarespezifika wie der Prozessorzielarchitektur unterscheiden. Diese Art der Virtualisierung bietet in der Regel mehr Sicherheit durch eine striktere Systemtrennung, weist aber Performanzeinbußen gegenüber Systemen von Typ (a) und (c) auf. Beispiele für Hypervisor sind *KVM* oder *XEN* [14].

Container-basierte Virtualisierung (System *(c)*) stellt demnach eine vergleichsweise leichtgewichtige Virtualisierungsart dar, in der die Trennung der Softwarecontainer – im Verlauf dieser Arbeit zur Vereinfachung auch als VMs bezeichnet – auf Basis des Hostbetriebssystems stattfindet. VMs und Hostbetriebssystem teilen sich entsprechend denselben Kernel. Daraus resultiert im Vergleich zur Hypervisor-basierten Virtualisierung in der Regel eine bessere Performanz bei weniger Sicherheit durch eine weniger strikte Systemtrennung. Beispiele für Containersysteme sind *Docker*, *LXC* oder *rkt* [14].

Beide Arten der Virtualisierung werden in dieser Arbeit als Infrastrukturkomponenten in FSNs berücksichtigt, insbesondere da sie durch ihre unterschiedlichen Charakteristika hinsichtlich Performanz und Sicherheit jeweils unterschiedliche Anforderungen an Anwendungen in FSNs bedienen können. Gleichzeitig stellen beide Arten von Systemen, Hypervisor sowie Containersysteme, wichtige Elemente der gemanagten IT-Infrastruktur in FSNs dar.

2.1.1.2 Netzvirtualisierung

Unter dem Begriff Netzvirtualisierung werden in dieser Arbeit insbesondere Protokolle und Technologien verstanden, über die physische Netzverbindungen in logische aufgeteilt werden können. Logische Netzverbindungen können selbst wiederum in mehrere logische Netzverbindungen unterteilt werden. Diese Funktionalität stellt in (föderierten) SNs eine wesentliche Basis für einige Kernaspekte der hier betrachteten gemanagten IT-Infrastrukturen dar. Dazu zählen insbesondere die Schaffung **mandantenfähiger Netze**, in denen mehrere Mandanten logisch getrennt voneinander IT-Ressourcen derselben Hardware-Basis nutzen können, oder auch **geographisch verteilte Netze**, indem mehrere Datenzentren virtuell zu einer gemeinsamen IT-Infrastruktur zusammengeschlossen werden können. Auch sind sie **fernwartbar**. Weitere Charakteristika von FSNs werden im Rahmen der Anforderungsanalyse in Abschnitt 3.1 beschrieben.

Gängige Verfahren zur Netzvirtualisierung umfassen beispielsweise die Verwendung von *Virtual Private Networks* (VPN), welche durch eine Vielzahl unterschiedlicher Protokolle realisiert werden können. Weitere gebräuchliche Lösungen sind beispielsweise das *Virtual eXtensible Local Area Network* (VXLAN), das besonders für sehr große Netze, wie sie in Föderationen durch mehrere kooperierende Partner wahrscheinlicher werden, geeignet sind und beispielsweise im

Vergleich zu klassischen VLANs das Anlegen von deutlich mehr (2^{24} statt 2^{12}) virtuellen Netzen erlauben [15]. Ein anderes gebräuchliches Protokoll mit analoger Funktionalität ist *Network Virtualization Using Generic Routing Encapsulation* (NVGRE) [16].

2.1.2 Software-Defined Networking (SDN)

HNs werden üblicherweise durch hardwarebasierte, monolithische Systeme realisiert, in denen ihre Funktion genauso wie Logik und Steuerung dezentralisiert im selben System integriert sind. Beispielsweise agieren herkömmliche Switches zur Weiterleitung des Netzverkehrs weitestgehend isoliert voneinander und mit statisch implementierter Logik.

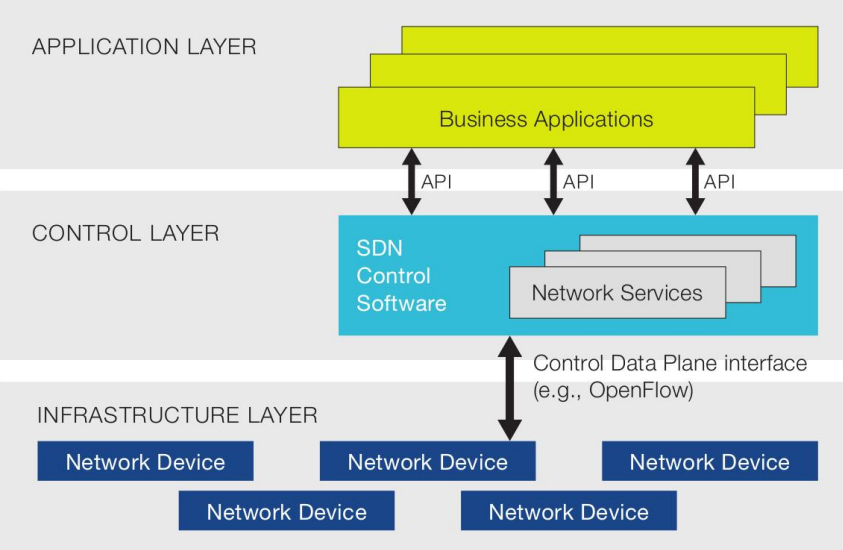


Abbildung 2.2: Dreischichtige SDN-Referenzarchitektur mit jeweiligen Komponenten [3].

Das Konzept hinter dem Begriff Software-Defined Networking beschreibt einen Ansatz zum Management von Netzkomponenten, in dem die Logik bzw. die Steuerung der Netzkomponenten aus einer zentralen Sicht implementierbar ist. Im Kern fordert SDN daher die Trennung der Funktion einer Komponente von ihrer Steuerung. Gemäß einem Whitepaper der *Open Network Foundation* [3] resultiert die in Abbildung 2.2 gezeigte Architektur von SDNs. Auf der *Infrastructure Layer*, welche oft auch *Data Plane* genannt wird, befinden sich die eigentlichen Netzkomponenten. Diese werden von einem sogenannten *SDN-Controller* auf der darüberliegenden Schicht, dem *Control Layer*, logisch zentralisiert gesteuert. SDN-Controller überwachen dabei üblicherweise einerseits den Zustand der Netzkomponenten und steuern diese andererseits. Die eigentliche Logik der Steuerung ist wiederum davon getrennt in, hier *Business Applications*, oft aber auch *Netz-* oder SDN-Anwendungen genannten Anwendungen, auf dem darüberliegenden *Application Layer* implementiert.

Die zwei Schnittstellen zwischen Komponenten der drei Schichten in der SDN-Architektur ist unterschiedlich realisiert. Für die Schnittstelle zwischen SDN-Controllern und SDN-Infrastruktur haben sich einige Protokolle für unterschiedliche Zwecke durchgesetzt. Für die Steuerung von SDN-Switches und -Routern wird beispielsweise häufig *OpenFlow* eingesetzt, das eines der im SDN-Kontext zuerst dazu entwickelten Protokolle darstellt [3]. OpenFlow ist nicht nur ein Protokoll, sondern beschreibt eine komplette Switch-Architektur und ein Kommunikationssystem [17]. Direkte Alternativen zu OpenFlow sind beispielsweise *Protocol Oblivious Forwarding* (POF), das Netzverkehr anders als OpenFlow unabhängig von Protokollfeldern steuerbar machen soll, aber dennoch nach einer analogen Architektur funktioniert [18]. Auch ist in diesem Kontext die Spezifikationssprache *Programming Protocol-independent Packet Processors* (P4) entstanden, die verwendet werden kann, um Netzverkehrweiterleitung auf Switches direkt zu

implementieren [19]. Weitere Spezifikationen wie das *Open vSwitch Database Management Protocol (OVSDB)* erfüllen dabei beispielsweise ergänzende Funktionen, wie die Konfiguration des *Open vSwitch*, einem rein softwarebasierten Switch [20]. Andere Protokolle, die im SDN-Kontext zum Zugriff auf Netzkomponenten eingesetzt werden, sind beispielsweise NETCONF (siehe Abschnitt 4.2.2), aber auch seit langem etablierte Ansätze wie das *Simple Network Management Protocol* (vgl. Abschnitt 4.2.1) können weiterhin unter dem Paradigma verwendet werden.

Die Schnittstelle zwischen SDN-Controllern auf dem Control Layer und Netzanwendungen auf dem Application Layer ist dagegen praktisch ohne etablierte Standardisierung, sondern variiert stark je nach SDN-Controllerimplementierung. Die Schnittstelle zwischen SDN-Controllern und Netzanwendungen stellt jedoch auch im Netzmanagement wichtige Anbindungspunkte einer Managementplattform an die gemanagte IT-Infrastruktur dar. Die fehlende Standardisierung an diesem Punkt stellt sich als eine der Herausforderungen im Management föderierter softwarebasierter Netze heraus.

2.1.3 Network Functions Virtualization (NFV)

Network Functions Virtualization beschreibt ein Paradigma, das durch das *European Telecommunications Standards Institute* (ETSI), einer Normungsorganisation mit speziellem Fokus auf den Telekommunikationsbereich, standardisiert wird. Wie in einem gemeinsamen Whitepaper [1] mehrerer Telekommunikationsunternehmen zusammengefasst ist, entstand die Idee hinter NFV aus der Forderung nach einfach handhabbaren und wiederverwendbaren Systemen, welche bis dato üblicherweise problematische hardwarebasierte und proprietäre Netzkomponenten ersetzen sollen. Demnach soll der Einsatz von virtualisierten Netzfunktionen und die Verwendung von *Commercial-off-the-Shelf* -Hardware Vorteile wie reduzierte Kosten, mandantenfähige Netzkomponenten, höhere Sicherheit und heterogene Netze bringen. Auf diese Weise ändert sich auch die Art der Bereitstellung von Netzdiensten insofern, dass *a)* Hardware und Software entkoppelt werden, *b)* Netzfunktionen flexibler und automatisierbar über unterschiedliche Datenzentren bereitgestellt werden können, und, dass *c)* eine hohe, einfache Skalierbarkeit resultiert [6].

2.1.3.1 Architektur

Um diese ursprüngliche Idee von NFV sind in den letzten Jahren vielerlei Teilkonzepte wie Use-Cases, Anforderungen und eine NFV-Referenzarchitektur definiert worden. Im Kontext dieser Arbeit ist insbesondere die NFV-Referenzarchitektur von Bedeutung, da sie Komponenten und Komponentenabhängigkeiten für das Netzmanagement definiert. Die NFV-Referenzarchitektur ist in Abbildung 2.3 gezeigt.

Die Grobstruktur der NFV-Referenzarchitektur ist in der ETSI Group Specification NFV 002 [6] beschrieben und setzt sich demnach aus drei Hauptblöcken zusammen: Der *NFV Infrastructure* (NFVI), den darauf betriebenen *Virtual Network Functions* (VNF), sowie dem *NFV Management and Orchestration*-(MANO)-Block. Darüber liegen Operations- und Business-Support-Systeme.

- **NFVI** enthält demnach alle physischen und daraus über eine Virtualisierungsschicht jeweils abgeleiteten virtuellen IT-Ressourcen (insbesondere Rechen-, Speicher- und Netzressourcen).

- **VNFs** ist die Menge der durch die NFVI realisierten virtuellen Netzfunktionen. *Element Management* (EM) steht für das Management der VNFs.

- **MANO** teilt sich in mehrere Komponenten zum Management und der Orchestrierung der jeweiligen zuvor genannten Blöcke und Schichten auf, die für das Netzmanagement von Bedeutung sind.

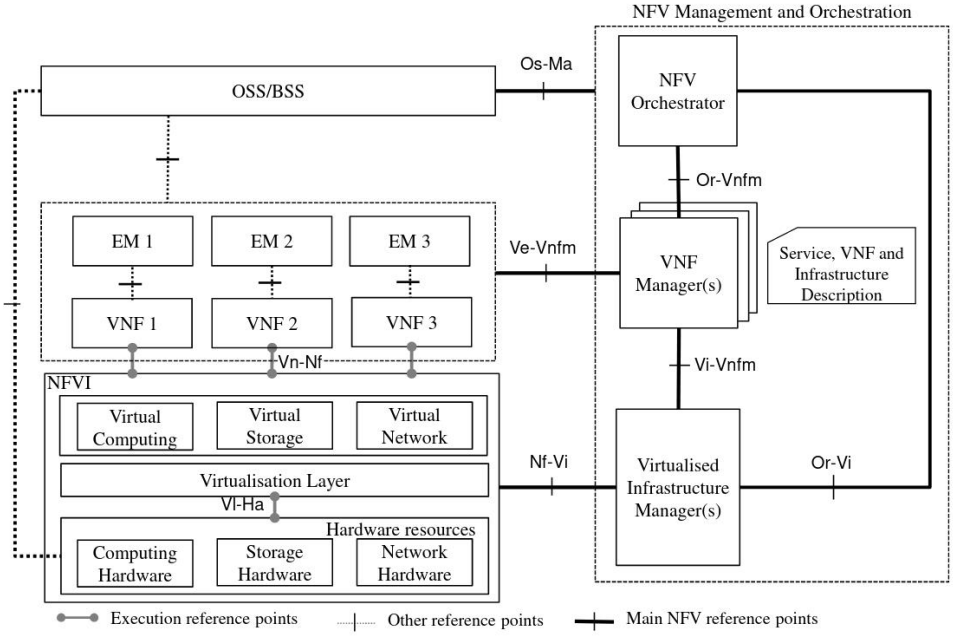


Abbildung 2.3: NFV-Referenzarchitektur [6].

Die in die Abbildung eingezeichnete Kanten beschreiben von im NFV-Framework benannte Schnittstellen zwischen den Komponenten. Beispielsweise beschreibt die Kante *Vi-Ha* im NFVI-Block die Schnittstelle zwischen Hardwareressourcen und der Virtualisierungsschicht. Die Schnittstellen sind auch im Netzmanagement von FSNs zu berücksichtigen. Sie werden insbesondere in den NFV-IFA-(Interfaces and Architecture)-Standards beschrieben. Darüber hinaus stellt ETSI ein umfangreiches Informationsmodell bereit, das ebenfalls auf Ressourcen- und Schnittstellenbeschreibungen eingeht.

2.1.3.2 Anschlusspunkte an das Netzmanagement

Anschlusspunkte der NFV-Referenzarchitektur an eine Netzmanagementplattform sind – neben Komponenten der Virtualisierungsschicht – vor allem die MANO-Komponenten *Virtualized Infrastructure Manager* (VIMs), *VNF Manager* und *NFV Orchestrator*.

- Ein **VIM** managt die IT-Ressourcen und Systeme der NFVI wie (oft jeweils mehrerer) Hypervisors und Dienste zur Verwaltung von Speicher und (virtuellen) Netzen. Auch erfüllt er Funktionen zur Verwaltung von VMs. Beispiele für Implementierungen von VIMs sind OpenStack oder OpenNebula. Gerade da diese Systeme SNs bereitstellen können, stellen sie selbst zentrale MOs aus Sicht des Netzmanagements dar.

- **VNF Manager** (VNFM) erfüllen Funktionen des Lifecycle-Managements für virtuelle Netzfunktionen. Als Teil des OpenStack-Projekts bietet der Dienst *Tacker*[I] beispielsweise derartige Funktionalität.

- **NFV Orchestrator** (NFVO) sind gemäß der Spezifikation für die Orchestrierung und das Management der NFV-Infrastruktur und -Software sowie Netzfunktionen zuständig. Ein Beispiel ist *Open Source MANO* (vgl. Abschnitt 4.4.2).

Gemäß der NFV-Architektur in Abbildung 2.3 wird im NFV-Standard in der Regel nur ein NFVO und VIM vorgesehen. In FSNs sind jedoch durchaus mehrere dieser Komponenten für Teilnetze

[I] `https://wiki.openstack.org/wiki/Tacker`

denkbar und werden entsprechend in dieser Arbeit berücksichtigt. Der in NFV zentrale Aspekt der Dienstorchestrierung (inkl. Modellierung, *Modell-Onboarding* und Lebenszyklusmanagement von Diensten) wird in dieser Arbeit **nicht** explizit berücksichtigt. Dazu existieren bereits mehrere Implementierungen wie Open Source MANO (vgl. Abschnitt 4.4.2) oder ONAP (vgl. Abschnitt 4.5.3), die später auch hinsichtlich ihrer Eignung für das Netzmanagement untersucht werden.

2.2 Föderationen im Kontext softwarebasierter Netze

> Eine **Föderation** bezeichnet eine Kooperation mindestens zweier voneinander *unabhängiger* Parteien (der Begriff *Partner* wird in dieser Arbeit synonym verwendet) zum Zweck der Erreichung eines *gemeinsamen* oder *komplementären* Ziels. Kernmerkmal der Föderation ist in dieser Arbeit die gemeinsame Verwendung von durch unterschiedliche Partner bereitgestellten IT-Ressourcen zum Zweck der Zielerreichung.

Im Mittelpunkt dieser Arbeit steht die Föderation von Infrastruktur und Diensten im Kontext von SNs. Als *Infrastruktur* werden das Netz sowie die zum Betrieb und Management notwendigen Komponenten angesehen, wie sie in den letzten Abschnitten zum jeweiligen Paradigma beschrieben wurden. Als *Dienst* wird in dieser Arbeit eine auf der Infrastruktur basierende definierte Funktion bezeichnet, auf die über definierte Schnittstellen zugegriffen werden kann. Beispielsweise ist ein *Virtual Private Network* (VPN) ein insbesondere in SNs elementarer Dienst, um mehrere jeweils untereinander lediglich über öffentliche Infrastruktur (z. B. die globale Internet-Infrastruktur) erreichbare Teilnetze zusammenzuschließen. Andere Dienste wie beispielsweise E-Mail, Netzwerkspeicher oder Versionierungssysteme werden als Anwendungen angesehen, deren Management kein Schwerpunkt dieser Arbeit bildet. Die Menge und Konfiguration der in einem Netz betriebenen Anwendungen ist üblicherweise stark vom Verwendungszweck und der Aufgaben abhängig, die durch das Netz unterstützt werden sollen.

Eine möglichst detaillierte Charakterisierung von Föderationen ist notwendig, um eine Basis für geeignete FSN-Szenarien zu legen. Ein Framework zur Entwicklung von Managementplattformen muss aus den unterschiedlichen Charakteristika resultierende Anforderungen berücksichtigen und in dieser Arbeit besonders einen Fokus auf in SNs ausgeprägte Charakteristika setzen. Entsprechend wird im folgenden Abschnitt eine Sammlung von Charakteristika von Föderationen aus verwandten Bereichen beschrieben, welche in Abschnitt 2.2.2 auf FSNs abgebildet wird.

2.2.1 Ausprägung von Föderationen in verwandten Bereichen

Eine umfangreiche Herleitung der Charakteristika von Föderationen mit Schwerpunkt bezüglich unterschiedlicher (dennoch eng verwandter) Hintergründe ist bereits in anderen Dissertationen bearbeitet worden. Michael Schiffers beschreibt in seiner Dissertation [21] Charakteristika von Föderationen im Sinne einer Virtuellen Organisation (VO) im Grid-Computing. Darauf aufbauend erweitert Daniela Pöhn in ihrer Dissertation [22] diese Klassifikation von Föderationen im Kontext des föderierten Identitätsmanagements. Eine ähnliche Klassifikation wird in der Dissertation von Patricia Marcu [23] bezüglich interorganisationaler Beziehungen zum organisationsübergreifenden Fehlermanagement vorgenommen.

2.2.1.1 Föderationen im Grid-Computing

Da besonders eine Föderation im Grid-Computing eng mit in dieser Arbeit betrachteten Föderationen zusammenhängt – der Kernaspekt der Föderation ist die Erreichung eines gemeinsamen Ziels durch zu diesem Zweck geteilte IT-Ressourcen – bilden die in [21] genannten Charakteristika eine geeignete Grundlage für diese Arbeit. Demnach wird eine VO („eine zeitlich begrenzte

Kooperation von Elementen [...], die Teile ihrer physischen oder logischen Ressourcen [...] derart zur Verfügung stellen, dass die gemeinsam vereinbarten Ziele unter Berücksichtigung lokaler und globaler Policies erreicht werden können [21]“) durch die folgenden Charakteristika beschrieben:

- Die **Bindungsintensität**, d. h. das Ausmaß der Aufgabe der Autonomie der einzelnen Partner in einer Kooperation.
- Der **Integrationsgrad**, der Grad der gemeinsamen Koordination der Rahmenbedingungen (*autonom*, *koordiniert* oder *integriert*).
- Die **Struktur** bezüglich der Koordination der Kooperation bzw. Föderation (*intra-* bzw. *interorganisational*).
- Die **Entscheidungsreichweite** eines Partners und Grad der Auswirkung auf die anderen Partner.
- Die **Richtung der Organisationsketten** in der Wertschöpfungskette (*horizontal*, *vertikal* oder *diagonal*).
- Die **räumliche Dimension**, beispielsweise von *lokal* bis hin zu *international*.
- Die **Dauer** der Kooperation, insbesondere bezüglich des Vorhandenseins einer allgemeinen zeitlichen Begrenzung.
- Der **Bereich** bzw. die Phase innerhalb der Wertschöpfungskette, welche durch die Kooperation abgedeckt ist (z. B. *Forschung und Entwicklung*, *Beschaffung*, *Fertigung*, *Montage*, *Vertrieb* sowie *Service*).
- Die **Kapazitäten** der eingebrachten Ressourcen (z. B. *komplementär* oder *redundant*).
- Die **Kompetenzen**, welche in die Kooperation eingebracht werden (analoge Ausprägung wie das Charakteristikum der *Kapazitäten*).
- Die **Anzahl der Partner**.
- Die **Koordination** der Kooperation; entweder *explizit* durch eine übergeordnete Stelle oder *implizit* (auf Basis lokaler Entscheidungen).
- Die **Gruppenstruktur** bzgl. ihrer Offenheit (*offen* bzw. *abgeschlossen*).
- Der **Gründungsprozess** der Kooperation; entweder *geplant* oder *spontan*.
- Die **Administration** der IT-Ressourcen bezüglich dem Grad der Automatisierung (*manuell*, *werkzeuggestützt*, *automatisiert*).
- Die **Betrachtungsebene** der kooperierenden Partner (d. h., eine Kooperation auf Basis einzelner Individuen, Organisationen oder eine Kooperation zwischen Föderationen).
- Die **Kooperationsstruktur** bzw. Koordination der Aufgaben einzelner Partner zur Zielerreichung (z. B. *sequenziell*, im *Master-Slave-Modell* oder *strukturlos*).

2.2.1.2 Föderiertes Identitätsmanagement

Die Abstimmung der Charakterisierung von Föderationen auf dieser Basis für den Kontext des föderierten Identitätsmanagements wurde in [22] durch eine Auswahl relevanter Aspekte, ihrer Anpassung und Verfeinerung vorgenommen. Zusätzlich werden identifizierte Charakteristika geeignet nach Zugehörigkeit in die übergeordneten Kategorien *a)* **Struktur und Dimension**, *b)* **Rahmenbedingungen** und **c) Aspekte des Vertrauens** gruppiert.

Zu *a)* gehören einerseits alle Charakteristika bezüglich der Kooperation der Föderation:

- Die **Struktur** der Kooperation an sich. Dieses Charakteristikum wurde auf Basis des gleichnamigen Punktes zur Charakterisierung von VOs genutzt, jedoch auf den Bereich FIM angepasst.
- Die **Anzahl der Teilnehmer**, ebenfalls analog zum Charakteristikum *Anzahl der Partner* für VOs mit angepasstem Satz an Ausprägungen.
- Die **Gruppenstruktur** mit gleichen Ausprägungen wie in VOs.

Andererseits umfasst die Gruppe ebenfalls Charakteristika bezüglich der Dimension der Föderation, insbesondere

- die **räumliche Dimension**, als auch
- die **organisatorische Dimension**, welche mehr dem Charakteristikum der *Betrachtungsebene* von VOs nach Schiffers [21] entspricht.

Zu *b)* den Rahmenbedingungen zählen die folgenden Aspekte:

- Die **Dauer der Föderation** analog zum gleichnamigen Charakteristikum von Föderationen im Sinne von VOs.
- die **Zusammenarbeit**, ein für den Bereich FIM zusätzlich eingeführtes Kriterium, welches den Kontext der Zusammenarbeit beschreibt, insbesondere im Sinne *nationaler Zusammenarbeit*, *gemeinsamer Projekte* und *Communitites*.
- Die **Koordination** der Föderation, ähnlich beschrieben wie im Kontext von VOs.
- Der **Gründungsprozess**, analog wie für VOs.

Die letzte Gruppierung *c)* bezüglich Vertrauensaspekten berücksichtigt weitere Charakteristika von Föderationen:

- Der **Circle of Trust** ist im FIM Kontext ebenfalls zusätzlich betrachtet worden und beschreibt die Veränderlichkeit der Gruppenstruktur innerhalb der Föderation – die Möglichkeit der Aufnahme neuer bzw. des Ausscheidens bestehender Parteien einer Föderation.
- Die **Bindungsintensität**, analog zum gleichnamigen Charakteristikum von VOs.
- Das **Vertrauen** zwischen den Organisationen untereinander, d. h. *direkt* oder *indirekt* (transitiv).

2.2.1.3 Interorganisationales Fehlermanagement

Eine dazu verwandte Charakterisierung von Beziehungen zwischen zwei oder mehr Organisationen wird ebenfalls in der Dissertation von Patricia Marcu [23] im Kontext des interorganisationalen Fehlermanagements vorgenommen. Ein besonderer Fokus ihrer Arbeit liegt auch auf dem Aspekt der Koordination einer gemeinsamen Diensterbringung. Die Koordination wird demnach (basierend auf dem Koordinationswürfel aus [24]) durch drei Merkmale charakterisiert:

- Die **Steuerung** im Sinne der Art der Koordination von Entscheidungen – entweder von *zentraler Stelle*, *gemeinsam* oder *lokal-autonom* pro Partner (vgl. *Koordination* in VOs und FIM).

- Die **Aufgabenzuordnung** an die jeweiligen Partner, wobei Aufgaben entweder *arbeitsteilig* aufgeteilt sind oder mehrere Partner dieselbe Aufgabe durchführen (*homogen*).

- Die **Kommunikation** der Informationsverteilung. Die Ausprägungen dieses Merkmals entsprechen den Kommunikationsstrukturen zum Informationsaustausch, hier konkret einer *Baum*-, *Stern*- oder *Peer-to-Peer*-Struktur.

Neben Charakteristika der Koordination werden weitere allgemeine Merkmale zur Beschreibung von Föderationen im interorganisationalen Fehlermanagement herangezogen:

- Die **Dienstbeziehung** im Allgemeinen (vgl. Charakteristikum *Richtung der Organisationsketten* nach Schiffers [21]), beschränkt auf die *vertikale* sowie die *horizontale* Dienstkomposition.

- Die **Dynamik** der Kooperation unter zeitlich-strukturellen Aspekten: Entweder mit *statischer* Ausprägung oder *dynamisch*, gekennzeichnet durch relativ häufige Umstrukturierungen innerhalb der Föderation.

- Die **Rollenvergabe** für Kooperationspartner innerhalb der Kooperation, mit ihrerseits entweder *statischem* oder *dynamischem* Charakter.

- Die **Werkzeugorientierung** (d. h. der Unterstützung durch Softwarewerkzeuge) bezüglich der Durchführung von Prozessen und der Organisation der Kooperation. Vergleichbar mit dem Charakteristikum *Administration* nach Schiffers [21].

- Der **Anzahl verantwortlicher Dienstleister** für den erbrachten Dienst als Ganzes und ausgeprägt durch entweder *einen* zentralen oder *mehrere* Verantwortliche.

Ein Überblick über die aus den verschiedenen Quellen und Kontexten bereits beschriebenen Charakterisierungen von Föderationen ist in Tabelle 2.1 in einer harmonisierten Form zusammengefasst.

2.2.2 Charakteristika mit besonderer Manifestation in SNs

Eine insbesondere für die Anforderungsanalyse dieser Arbeit notwendige geeignete Charakterisierungsmorphologie von Föderationen in FSNs ist aus Vorarbeiten in dem Bereich nicht bekannt. Entsprechend wird in diesem Abschnitt eine Herleitung, basierend auf der in den vorhergehenden Abschnitten durchgeführten Zusammenfassung aus anderen Bereichen vorgenommen. Auch werden neue, in Vorarbeiten nicht berücksichtigte Charakteristika, begründet eingeführt. Eine Zusammenfassung wird in Tabelle 2.2 gezeigt.

	Schiffers [21] Virtuelle Organisationen in Grids	**Pöhn** [22] Föderiertes Identitäts-management	**Marcu** [23] Interorg. Fehler-management
Bindungsintensität	■	■	
Integrationsgrad	■		
Struktur	■		
Entscheidungs-reichweite	■		
Richtung	■		■
Räumliche Dimension	■	■	
Dauer	■	■	
Bereich	■		
Kapazitäten	■		
Kompetenzen	■		
Anzahl Partner	■	■	
Koordination	■	■	■
Gruppenstruktur	■	■	
Gründungsprozess	■	■	
Administration	■		■
Betrachtungsebene	■	■	
Kooperationsstruktur	■	■	
Zusammenarbeit		■	
Circle of Trust		■	
Vertrauen		■	
Aufgabenzuordnung			■
Kommunikation			■
Dynamik			■
Rollenvergabe			■
Anzahl der Dienstver-antwortlichen			■

Tabelle 2.1: Harmonisierte Übersicht über Charakteristika von Föderationen aus der Literatur über thematisch verwandte Bereiche (Blau gefärbte Zellen entsprechen einer Berücksichtigung).

Charakteristikum	Ausprägung in FSNs		
Bindungsintensität	vertraglich		
Integrationsgrad	koordiniert		
Struktur	interorganisational		
Entscheidungsreichweite	mehrere Organisationen		
Richtung	irrelevant		
Räumliche Dimension	regional	überregional	gemischt
Dauer	kurzfristig	langfristig	
Bereich	irrelevant		
Kapazitäten	irrelevant		
Kompetenzen	irrelevant		
Anzahl Partner	Netzwerk		
Koordination	implizit	explizit	
Gruppenstruktur	flexibel	fest	
Gründungsprozess	irrelevant		
Administration	integriert und werkzeuggestützt		
Betrachtungsebene	Organisationen		
Kooperationsstruktur	irrelevant		
Zusammenarbeit	irrelevant		
Circle of Trust	vgl. *Dynamik*		
Vertrauen	indirekt	direkt	gemischt
Aufgabenzuordnung	statisch	dynamisch	
Kommunikation	irrelevant		
Dynamik	statisch	dynamisch	
Rollenvergabe	statisch	dynamisch	
Anzahl der Dienstverantwortlichen	mehrere		
Relation zw. administrativen Domänen	überlappend	disjunkt	
Zielsetzung	überlappend	komplementär	gemischt
Zusammensetzung der Infrastruktur	heterogen		
Domänenstruktur	global	hierarchisch	unabhängig
Art der Infrastrukturnutzung	symmetrisch	asymmetrisch	

Tabelle 2.2: Relevanz und Ausprägung identifizierter Charakteristika von Föderationen in SNs (Blau: verschiedene Ausprägungen; Grau: keine Relevanz; Weiß: statische Ausprägung).

2.2.2.1 Abbildung bestehender Föderationscharakteristika

Nicht alle Charakteristika wirken sich auf das Design von Managementplattformen aus. Beispielsweise können Charakteristika, die keinen erkennbaren Effekt auf das Design der Managementplattform haben, vernachlässigt werden. Dagegen müssen Ausprägungen relevanter Charakteristika derart gewählt werden, dass aus ihnen Anforderungen an das in dieser Arbeit entwickelte Framework zur Entwicklung von Managementplattformen in FSNs resultieren.

Das Charakteristikum der **Bindungsintensität**, wie es für VOs und das föderierte Identitätsmanagement beschrieben ist, stellt für das Management föderierter softwarebasierter Netze ebenfalls ein wichtiges Kriterium dar. Aufgrund der Föderation im Sinne einer Kooperation mehrerer unabhängiger Organisationen wird stets von einer vertraglich geregelten Ausprägung dieses Charakteristikums ausgegangen.

Mit der Bindungsintensität oft einhergehend ist daher auch das Merkmal des **Integrationsgrades**, welches bezogen auf das Management von FSNs, die Art der Installation und Verwal-

tung beschreibt. Da FSNs in der Regel zum Zweck einer gemeinsamen oder komplementären Problem- bzw. Interessenslösung initiiert werden, wird hier stets mindestens eine *koordinierte* Vorgehensweise angenommen. Eine tatsächliche organisatorische *Integration* in föderierten SNs ist hingegen in der Praxis kaum realistisch. Man kann eher davon ausgehen, dass die Organisation der Ressourcen in einer Föderation an der jeweiligen bereits bestehenden Organisation des jeweiligen Partners ausgerichtet ist.

Ebenfalls wird in FSNs davon ausgegangen, dass die **Struktur** der Koordinationsbefugnisse rein *interorganisational* verteilt ist, da die geteilten IT-Ressourcen selbst aus unterschiedlichen Organisationen stammen. Entsprechend wird ein Teil der unmittelbaren Kontrolle der jeweiligen Ressourcen stets bei der Organisation bleiben, in dessen Besitz sie sind.

Die **Richtung der Organisationsketten** ist, genau wie die **Kooperationsstruktur**, für das Management föderierter softwarebasierter Netze irrelevant, da aus ihr lediglich eine spezielle Sicht auf Abhängigkeiten von Ressourcen untereinander ausgedrückt werden kann. Insbesondere ist bei einer vertikalen Beziehung die Organisation von Ressourcen von einer oder mehrerer anderer Parteien abhängig. Auf einer Ebene von Diensten, Anwendungen und verteilten Systemen, die oft intransparent (i. S. v. ohne feste Bindung an spezielle physische Ressourcen) auf SNs aufbauen, ist jedoch eine feingranularere Sichtweite notwendig. Aufgrund generell geteilter IT-Ressourcen muss folglich stets mit geteilten Abhängigkeiten gerechnet werden. Aufgrund des Teilens der Infrastruktur in FSNs gilt auch, dass die **Entscheidungsreichweite** im Allgemeinen immer *mehrere* Organisationen direkt (z. B. erfordert die Wartung eines Virtualisierungshosts oft die Abschaltung darauf realisierter virtueller Maschinen) oder indirekt (z. B. durch den Umfang unmittelbar zur Verfügung stehender und nutzbarer Ressourcen) betrifft.

Der Aspekt der **räumlichen Dimension** ist für FSNs aufgrund unterschiedlicher technischer als auch organisatorischer Aspekte relevant. Auf Seiten technischer Aspekte bedeutet die räumliche Distanz für das Netz oft reale Einschränkungen wie beispielsweise eine höhere Latenz zwischen Infrastrukturknoten aus unterschiedlichen Organisationen sowie die Abhängigkeit von öffentlicher Infrastruktur wie der Internet-Infrastruktur und daraus resultierenden Abhängigkeiten. Aus organisatorischer Sicht bedeutet die räumliche Distanz Herausforderungen in beispielsweise der Koordination von Entscheidungen, sowie in der Konfigurierbarkeit von IT-Ressourcen, da diese möglicherweise nicht physisch zugreifbar sind. Entsprechend ist in FSNs zu berücksichtigen, dass die Ausprägungen das Ausmaß der räumlichen Dimension darstellt – entsprechend entweder *regional* (d. h. praktisch auch physisch noch relativ schnell erreichbar) oder *überregional*. Bei vielen Partnern oder mehreren stark verteilten Datenzentren ist zudem eine *gemischte* Form denkbar.

Die **Dauer** einer Kooperation im Rahmen von FSNs ist mindestens insofern relevant, da beispielsweise der Aufwand zur Einrichtung der Infrastruktur und Systeme zum Management der geteilten Ressourcen in angemessenem Verhältnis zum eigentlichen Nutzen der Plattform stehen soll. Entsprechend soll das Management *kurzfristig* (Wochen bis Monate) genauso wie *langfristig* (Monate bis Jahre) ausgeprägter FSNs möglich sein. Der **Gründungsprozess** spielt eine vergleichsweise untergeordnete Rolle, da der Fokus der Arbeit auf dem letztendlich resultierenden FSN liegt.

Der **Bereich** der Kooperation innerhalb der Wertschöpfungskette ist für das Management von FSNs nicht relevant. Der Fokus liegt auf der Realisierung der Ziele des Managements der Infrastruktur im Allgemeinen und nicht auf denen darüberliegender Anwendungsfälle.

Der Aspekt der **Kapazitäten** entfällt bei FSNs, da weder das Komplement noch die Redundanz in der Verteilung der eingebrachten Ressourcen für deren Management von besonderer Bedeutung sind und auch im Rahmen derselben Föderation mehrfach wechseln kann (z. B. bzgl. administrierter Dienste). Ähnliches gilt für den Aspekt der **Kompetenzen**: Diese beziehen sich in einer Föderation nicht notwendigerweise auf das Netzmanagement. Für das Netzmanagement

wird einheitlich angenommen, dass notwendige Komponenten für den Betrieb der föderierten IT-Infrastruktur von einem oder mehreren Partnern vorhanden sind.

Da die **Anzahl der Partner**, ob dyadisch, triadisch oder anders geartet, für das Management selbst nicht von Bedeutung ist, wird in dieser Arbeit allgemein von einem *Netzwerk* aus Partnern ausgegangen.

Die **Koordination** von FSNs wird zu einem gewissen Grad – aufgrund der verteilten Ressourcen aus unterschiedlichen Organisationen – *implizit* ausfallen, d. h. verteilt auf alle kooperierenden Partner (z. B. Patchmanagement der lokalen Infrastruktur). Eine davon abgesehen hauptsächlich *explizit* koordinierte Organisation der Föderation hingegen ist auf höheren Ebenen wie der Planung des Einsatzes gemeinsamer IT-Ressourcen in einer Föderation wünschenswert und sinnvoll. Der Fokus der **Administration** ist aufgrund der Komplexität von FSNs auf einem *integrierten*, werkzeuggestützten Ansatz.

Die **Gruppenstruktur** ist auch in Kooperationen im Rahmen von FSNs ein wichtiger Aspekt, der unter verschiedenen Ausprägungen berücksichtigt werden muss. Änderungen in der Zusammensetzung der Partner in einer Kooperation in FSNs bedeuten – aufgrund der Teilung von Ressourcen – folglich auch Änderungen in der Infrastruktur. Entsprechend wird in dieser Arbeit zwischen einer *flexiblen* sowie einer *festen* Gruppenstruktur unterschieden.

Die **Betrachtungsebene** ist vor allem für die Organisation im Management der Ressourcen relevant. In dieser Arbeit wird von Föderationen zwischen Organisationen ausgegangen, wobei aus Managementsicht die Kooperation einzelner Individuen darin abgedeckt ist. Eine Kooperation aus Föderationen könnte ebenfalls als Kooperation von Organisationen mit einer angepassten Umsetzung der Organisation der Infrastruktur, Gruppen und Verantwortlichkeiten betrachtet werden.

Die Art der **Zusammenarbeit** ist in diesem Kontext weniger relevant, das Hauptkriterium liegt auf der gemeinsamen Nutzung geteilter Ressourcen. Ein damit verbundenes Charakteristikum ist die Struktur der **Kommunikation**, die ebenfalls in dieser Arbeit nicht konkret betrachtet wird. Es wird eine funktionierende Kommunikation zwischen den Partnern angenommen. Die ebenfalls damit verknüpfte **Anzahl verantwortlicher Dienstleister** ist im Management von FSNs statisch: FSNs werden zwar nicht notwendigerweise zum Zweck der Diensterbringung genutzt, der verfolgte Zweck involviert jedoch szenarienübergreifend mehrere Partner.

Der Aspekt des **Circle of Trust** aus [22] wird in dieser Arbeit als Teilcharakteristikum des ebenfalls beschriebenen Charakteristikums der **Dynamik** aus [23] angesehen. Je nach Kooperationsart ist die Dynamik der Föderation (insbesondere bzgl. Kooperationspartner, IT-Ressourcen und organisatorischer Aspekte wie Zuständigkeitsbereiche und Richtlinien) ein Aspekt mit potenziell großen Auswirkungen auf die Organisation des Managements. Entsprechend werden in dieser Arbeit die Ausprägungen *statisch* und *dynamisch* berücksichtigt. Ein ebenfalls damit verknüpftes Charakteristikum ist das des **Vertrauens**, das – wie in [22] beschrieben – in Föderationen entweder *direkt* oder *indirekt* ausgeprägt sein kann. Bei einer höheren Anzahl an Föderationspartnern ist auch eine *gemischte* Form möglich. Auch die **Aufgabenzuordnung** sowie die **Rollenvergabe** liefern in diesem Kontext je nach Ausprägung Anforderungen an Managementplattformen. Beide werden einzeln berücksichtigt, basierend auf den Ausprägungen aus [23], als *statisch* und *dynamisch*.

2.2.2.2 Einführung neuer Föderationscharakteristika in FSNs

Neben den aus anderen Arbeiten und anderen Kontexten von Föderationen zwischen Organisationen relevanten Charakteristika, sind aus dem Bereich des Netzmanagements zudem Charakteristika betreffend die Organisation sowie technischer Aspekte zu berücksichtigen. Folgende Charakteristika sind daher in diesem Kontext neu und in dieser Arbeit hinzugefügt worden.

Ein die Umsetzung des Managements beeinflussender Aspekt ist die **Zielsetzung** der jeweiligen Partner in einer Föderation. Organisationen gehen eine Föderation üblicherweise aus einem der zwei Gründe ein: Entweder sie haben eine gemeinsame Zielsetzung und bündeln ihre Ressourcen zur Zielerreichung oder sie verfolgen sich ergänzende Ziele, beispielsweise wie es in einer Beziehung zwischen einem IT-Ressourcenprovider und seinen Kunden der Fall ist. Entsprechen kann die Zielsetzung zwischen Föderationspartnern *überlappend* oder *komplementär* sein. Bei mehr als zwei Partnern ist ebenfalls eine *Mischform* aus beiden Ausprägungen möglich.

So ist auch die **Relation zwischen administrativen Domänen** untereinander ein wichtiges Kriterium, das insbesondere durch eine Managementplattform unterstützt werden muss. Eine administrative Domäne beschreibt einen Zuständigkeitsbereich im Management von IT-Ressourcen. In einer Föderation gibt es klassischerweise mindestens *zwei*, je nach Ausprägung auch deutlich mehr administrative Domänen. Da jeder Föderationspartner in der Regel eigene IT-Ressourcen in eine Föderation einbringt, obliegt mindestens die Verantwortlichkeit für die physischen IT-Ressourcen dem jeweiligen Eigentümer. Weitere administrative Domänen können beispielsweise durch mehrere Partner oder durch die Unterteilung der vorhandenen IT-Ressourcen in weitere, künstliche administrative Domänen (beispielsweise nach Funktion und Verwendung der Ressourcen) entstehen. Administrative Domänen können *überlappend* sein, falls mehrere Partner für dieselbe IT-Ressource denselben Zuständigkeitsbereich übernehmen, oder sind andernfalls *disjunkt*.

Des Weiteren ist die **Domänenstruktur** in der föderierten Infrastruktur von Bedeutung. Sie ist in der Regel abhängig von anderen bereits oben genannten Charakteristika wie dem Integrationsgrad, der Entscheidungsreichweite, der Koordination und auch der Relation zwischen Domänen innerhalb einer Föderation. Die Domänenstruktur kann einerseits *hierarchisch* sein, beispielsweise bzgl. der gesamten föderierten Infrastruktur mit bis hin zu beliebig feingranular unterteilten Submengen der Infrastruktur. Andererseits kann sie einheitlich *global* sein, d. h. die IT-Ressourcen der jeweiligen Partner werden dediziert der föderierten Infrastruktur zur Verfügung gestellt oder sie kann individuell *unabhängig* für die jeweilige Unterteilung der Infrastruktur sein.

Ein zusätzlicher wichtiger Aspekt, besonders für eine Managementplattform, ist die **Zusammensetzung der gemanagten Infrastruktur** im Sinne vorhandener Architekturen (z. B. Prozessorarchitekturen, Netzarchitekturen), Art gemanagter Systeme und Tools (z. B. bzgl. Monitoring und Management). Vor allem in Föderationen ist eine stark *heterogen* ausgeprägte, gemeinsam genutzte Infrastruktur aufgrund einer möglichen Absenz einer zentralen Koordinierungsinstanz oft der Fall. Eine andere mögliche Ausprägung dieses Charakteristikums ist eine *homogene* Zusammensetzung der Infrastruktur. In dieser Arbeit wird jedoch stets von einer heterogenen Zusammensetzung ausgegangen.

Ebenfalls ist die **Art der Infrastrukturnutzung** für das Management und folglich Managementplattformen von Bedeutung. Die föderierte Infrastruktur kann einerseits *symmetrisch* (gleichmäßig) oder *asymmetrisch* unter den Föderationspartnern genutzt werden.

2.3 Frameworks für Softwareanwendungen

Frameworks per se im Kontext der Softwareentwicklung zählen inzwischen kaum noch zu aktuellen Forschungsschwerpunkten in der heutigen Informatik. Ihr Höhepunkt liegt bereits ein Vierteljahrhundert zurück, als parallel mit der Entwicklung des damals aufkommenden und bis heute in der Softwareentwicklung stark verbreiteten Programmierparadigmas der *objektorientierten Programmierung* (OOP) Konzepte zur Verbesserung der Qualität von Software von besonderem Interesse der weltweiten Forschergemeinde aus diesem Bereich waren. Zur Qualitätsverbesserung von Software tragen Frameworks und OOP gemeinsam mit verwandten Konstrukten wie *Design Patterns* in ähnlicher Art und Weise bei, insbesondere durch eine resultie-

rende Wiederverwendbarkeit [25]. So erlaubt OOP die Wiederverwendung von Software (z. B. Klassen, Schnittstellen, usw.) allgemein, hingegen Frameworks die Wiederverwendung des **Designs** von Softwareanwendungen [25]. Entsprechend wird das Konzept von Frameworks in der Literatur ebenfalls fast ausschließlich im Kontext der objektorientierten Programmierung verwendet.

2.3.1 Zweck und Arten von Softwareframeworks

Die Verwendung von Frameworks ist hingegen auch heute hochaktuell – inzwischen existieren praktisch unzählige Frameworks für verschiedenste Anwendungszwecke, beispielsweise im Bereich der Webentwicklung, für Nutzerschnittstellen, Testframeworks oder spezieller Bereiche wie Machine Learning (z. B. TensorFlow). Sie versprechen in der Regel einen schnellen Einstieg in die Praxis mit einer Vielzahl domänenspezifischer Anwendungsfälle.

2.3.1.1 Zweck von Softwareframeworks

Eine zentrale, einheitlich akzeptierte Definition von Frameworks wird von den unterschiedlichen Autoren der vielen dazu bestehenden wissenschaftlichen Arbeiten nicht verwendet [26]. Dennoch finden sich einige wenige Kernaspekte, die Frameworks auszeichnen:

1. **Wiederverwendbarkeit** von Softwaredesign und Komponenten [25][26].
2. Anwendung zu einem **domänenspezifischen Zweck** [25][27][28].
3. **Unvollständigkeit** im Sinne einer lauffähigen Softwareanwendung [25][26].

Frameworks werden üblicherweise von Personen zur Lösung eines Problems innerhalb eines Fachgebiets entwickelt und implementiert, in der sie selbst sachkundig sind. Sie sind üblicherweise ausreichend generisch als auch konkret, damit sie insbesondere von weniger sachkundigen Entwicklern genutzt werden können, um vom Framework berücksichtigte und nicht berücksichtigte Aufgaben in einem speziellen Anwendungsfall zu lösen [28]. Als allgemeinste und im Kontext dieser Arbeit verwendete Definition eines Frameworks für Softwareanwendungen wird die Folgende genutzt:

> Ein **Framework für Softwareanwendungen** stellt ein generisches Design sowie eine wiederverwendbare Implementierung des Designs innerhalb einer gegebenen Domäne bereit [27].

2.3.1.2 Arten von Softwareframeworks

Gemäß [29] unterscheidet man grundlegend drei verschiedene Kategorien von objektorientierten Frameworks:

1. **Application Frameworks**, welche breite Funktionalität innerhalb einer Softwareanwendung bereitstellen sollen. Darunter fallen beispielsweise Frameworks für graphische Benutzeroberflächen (engl. „graphical user interfaces“, GUIs) oder Web-Anwendungen wie *Django*[II] oder *Spring*[III].
2. **Domain Frameworks**, welche Softwareentwickler bei der Entwicklung von Anwendungen für oder innerhalb einer bestimmten Domäne unterstützen sollen und oft umge-

[II] `https://www.djangoproject.com/`
[III] `https://spring.io/projects/spring-framework`

bungsabhängig (z. B. bzgl. der Organisation, in der die Anwendung Einsatz findet) implementiert werden müssen (z. B. Finanz- und Buchhaltungsanwendungen).

3. **Support Frameworks**, stellen eine Untergruppe von Domain Frameworks dar mit Hauptfokus der Unterstützung von eigentlichen Anwendungen auf Ebene von darunterliegenden computerspezifischen Systemen (z. B. Dateisysteme oder Speichermanagement).

Die im Rahmen dieser Arbeit entwickelten Frameworks sind entsprechend *Domain Frameworks*, die die Entwicklung von geeigneten Managementplattformen für föderierte softwarebasierte Netze unterstützen sollen. Eine Framework-Instanz (d. h. eine Managementplattform, die auf Basis der Frameworks entwickelt wurde) wird üblicherweise für Anwendungsfälle entwickelt, deren Anforderungen herkömmliche, bereits bestehende Managementplattformen nicht komplett erfüllen können. Durch das Framework soll der Aufwand zur Entwicklung einer geeigneten Plattform drastisch reduziert werden, wobei gleichzeitig das Auftreten von Designfehlern in der resultierenden Anwendung minimiert werden soll.

Nach [30] wird in der Umsetzung von Softwareframeworks üblicherweise zwischen Whitebox- und Blackbox-Frameworks unterschieden:

- Bei **Whitebox**-Frameworks erfolgt die eigentliche Anwendungsentwicklung durch einen vererbungsbasierten Ansatz. Das Framework stellt entsprechend Klassen bereit, die bei Anwendung spezialisiert werden müssen.

- **Blackbox**-Frameworks werden hingegen durch Parametrisierung gegebener Klassen angewendet. Blackbox-Frameworks bieten in der Regel weniger Flexibilität, sind jedoch einfacher anzuwenden.

Eine harte Klassifizierung von Frameworks ist in der Praxis jedoch kaum möglich und meist setzt sich ein Framework aus Whitebox- und Blackbox-Komponenten zusammen [30].

2.3.2 Entwicklung und Dokumentation von Frameworks

Die Entwicklung von Frameworks unterscheidet sich nach [30] wesentlich von der Entwicklung von Softwareanwendungen: Angefangen bei der Konzeption und Anforderungsanalyse über die Art der Unterstützung (vgl. vorheriger Abschnitt), die Dokumentation bis zum Testen. Gemeinsam haben sie, dass meist eine Anforderungsanalyse in der Entwicklung des Kerndesigns und danach oft um einzelne inkrementelle Erweiterungen mündet.

2.3.2.1 Entwicklung und Testen von Frameworks

Gemäß [30] beginnt die Entwicklung eines Softwareframeworks ähnlich wie die einer Softwareanwendung mit einer domänenspezifischen Anforderungsanalyse. Anders als bei Anwendungen muss jedoch darauf geachtet werden, dass ein Framework nicht nur für einen spezifischen Anwendungsfall, sondern für alle Anwendungsfälle geeignet konzipiert ist und entsprechend geeignete Anforderungen definiert sind. Bei der Konzepterstellung für ein Framework muss darüber hinaus darauf geachtet werden, dass die Granularität geeignet unterstützt wird. So ist eine Aufteilung eines Frameworks in mehrere Teilframeworks, die unterschiedliche Aspekte unterstützen, denkbar.

Generell ist ein Framework in **Hot-Spots** und **Frozen-Spots** eingeteilt. Frozen-Spots beschreiben demnach Teile eines Frameworks, die bereits fix implementiert sind, wohingegen Hot-Spots durch den Anwender angepasst werden können oder müssen, um auf den jeweils verfolgten Anwendungsfall zu passen [31]. In dieser Arbeit werden die zwei Begriffe weitergefasst verwendet und sie beziehen sich auch auf Frameworkfunktionalität an sich.

Das Testen eines Frameworks unterscheidet sich im Vergleich zu Anwendungen nach [30] ebenfalls wesentlich. Da ein Framework keine lauffähige Anwendung ist, kann es nicht zum Zweck der Fehlererkennung ausgeführt werden. Vielmehr ist es notwendig, dass ein Framework anhand einer oder mehrerer konkreter Anwendungsfälle getestet wird.

2.3.2.2 Dokumentation eines Frameworks

Nach [30] betrifft die Dokumentation von Softwareframeworks zwei Zwecke. Einerseits muss eine Dokumentation die Entwicklung des Frameworks, andererseits seine eigentliche Nutzung zur Entwicklung einer Softwareanwendung unterstützen. Auch in dieser Arbeit sind potenziell beide Aspekte wichtig, da die hier entstehenden Frameworks einerseits offen für Weiterentwicklungen, andererseits für ihre Nutzung sein sollen. Eine Übersicht über aus anderen wissenschaftlichen Arbeiten entwickelten Ansätzen zur geeigneten Dokumentation von Frameworks wird in [30] beschrieben. Diese umfassen einen beispielbasierten Kochbuch-Ansatz, eine Beschreibung durch eine *Pattern Language* (nach [28] eine Art strukturierter Essay mit Fokus auf der Anwendung eines Frameworks, nicht seiner Funktionsweise) oder durch eine *Framework Description Language*. Nach [30] haben die meisten Ansätze gemein, dass sie

- den Zweck,
- eine Anwendungsbeschreibung,
- und den Zweck von Beispielanwendungen

beschreiben. Eine kombinierte Beschreibung mit unterschiedlichen Verfahren, unter anderem auch Beispiele oder Architekturdarstellungen, ist demnach am geeignetsten. Nach [28] spielen insbesondere Beispiele eine Schlüsselrolle bei der Dokumentation von Frameworks.

2.4 Netzmanagement und Managementplattformen

> Gemäß Hegering, Abeck und Neumair [8] umfasst **Management** alle Maßnahmen zur Sicherstellung eines effektiven und effizienten Betriebs von Systemen und Ressourcen gemäß den Zielen des Unternehmens. **Netzmanagement** beschreibt demnach einen Teilbereich darin, der sich auf das Kommunikationsnetz und darin befindliche Komponenten beschränkt.

Klassische Objekte des Netzmanagements sind demnach beispielsweise Leitungen oder Netzfunktionen zur Übertragung und Vermittlung des Netzverkehrs. Objekte, die hingegen im Netzmanagement von SN in dieser Arbeit betrachtet werden, erweitern die klassischen Objektegruppen. Bereits im Zuge der Beschreibung der Paradigmen in SNs in Abschnitt 2.1 sind einige neue Netzkomponenten, die sich nicht in diese klassische Definition einordnen lassen, herausgestellt worden, beispielsweise SDN-Controller, Hypervisor oder MANO-Systeme. Da Netzfunktionen in SNs de facto auf VMs realisiert werden, verschiebt sich jedoch auch der Fokus und deckt in dieser Arbeit Teile des Managements ab, die klassischerweise dem Systemmanagement zugeordnet werden. Gemäß [32] bestehen die Kernaufgaben eines Netzmanagementsystems in der Bereitstellung, Konfiguration und der Überwachung der IT-Ressourcen im Netz. Die Bereitstellung und Konfiguration werden in dieser Arbeit als Teilbereiche der Steuerung betrachtet, die gemeinsam mit der Überwachung als die zwei Kernprozesse des Netzmanagements angesehen werden.

2.4.1 Managementarchitekturen

Eine wesentliche Herausforderung im Netzmanagement liegt basierend auf [8] in der Heterogenität des jeweils betrachteten Netzes, beispielsweise hinsichtlich der darin kommunizierenden Komponenten und Systeme, zu berücksichtigenden Informationen, organisatorischer Aspekte, Kommunikationssysteme und Protokollen, Funktionen und Konfigurationsmöglichkeiten. Demnach wird eine Spezifizierung der im jeweiligen Fall notwendigen Ausprägungen davon als Managementarchitektur bezeichnet. Sie dient als Grundlage zur Entwicklung interoperabler Managementlösungen durch unterschiedliche Hersteller.

2.4.1.1 Teilmodelle

Das Gesamtkonstrukt der Managementarchitektur kann nach [8] in vier Teilmodelle heruntergebrochen werden:

- **Informationsmodell:** Da, wie in [8] beschrieben wird, es in heterogenen Netzen kein einheitliches Verständnis von Informationen gibt, ist es die Aufgabe des Informationsmodells, dieses herzustellen. Darin werden für das Management relevante Informationen festgelegt und für das gemanagte Netz im notwendigen Maß modelliert. Eine Repräsentation einer gemanagten Ressource wird als **Managementobjekt** (MO) bezeichnet [33]. Das Informationsmodell gibt entsprechend die Basis zur Spezifizierung von Managementobjekten vor. Demnach relevante Aspekte sind beispielsweise die Identifikation von MOs, ihre Zusammensetzung, ihr Steuerungsverhalten und Beziehungen zu anderen MOs. Beziehungen zwischen zwei oder mehreren MOs (nach [34] bspw. Verbindungs- oder Existenzbeziehungen) werden als **Managementbeziehungen**, oder hier auch synonym als **MO-Abhängigkeiten** bezeichnet. Eine Managementplattform muss entsprechend benutzerdefinierte Informationselemente und ihre Modellierung unterstützen. Als MO wird in dieser Arbeit in erster Linie eine gemanagte Netzkomponente selbst bezeichnet.

- **Organisationsmodell:** Im Organisationsmodell werden organisatorische Aspekte des Managements wie Kooperationsmodelle, Rollen, Gruppen oder Domänen festgelegt.

> Eine **administrative Domäne** (teilweise auch kurz **Domäne**) beschreibt einen administrativen Zuständigkeits- oder Verantwortungsbereich innerhalb des Netzmanagements. Basierend auf RFC 3198 [35] umfasst sie eine Menge an Elementen und Diensten, die in einer abgestimmten Art und Weise administriert werden. Entsprechend wird in dieser Arbeit insbesondere auch eine Menge an berechtigten Nutzern in bestimmten Rollen und mit definierten Verantwortlichkeiten und damit verbundenen Aufgaben als Teil administrativer Domänen angesehen.

Gerade in FSNs betrifft das einige Aspekte, die im Vergleich zu herkömmlichen Netzen deutlich an Bedeutung gewinnen und im Management berücksichtigt werden müssen. Beispielsweise gibt es, wie in Abschnitt 2.2.2 ausgeführt wurde, unterschiedliche Arten und Ausprägungen von Föderationen zwischen Partnern, deren Modellierung Teil des Organisationsmodells ist. Wie in Abschnitt 2.1 beschrieben wurde, legen des Weiteren die Paradigmen in SNs die technische Basis für den Betrieb von mandantenfähigen Netzen und Netzkomponenten, sodass sich überschneidende administrative Domänen sehr viel häufiger vorkommen als in herkömmlichen Netzen. Administrative Domänen können sich überschneiden, wenn MOs oder Nutzer mehreren administrativen Domänen zugeordnet sind. Das führt zwangsläufig auch zu Überschneidungen hinsichtlich Verantwortlichkeiten bezüglich der Umsetzung von Managementaufgaben. Weitere wesentliche Aspekte für die Organisation werden neben technischen Aspekten im Rahmen einer

Ableitung der Betriebscharakteristika von FSNs in Abschnitt 3.1 abgeleitet. Eine Managementplattform muss die Modellierung organisatorischer Aspekte und ihren Einfluss im Management unterstützen.

- **Kommunikationsmodell:** Im Kommunikationsmodell werden nach [8] Konzepte zum Austausch von Managementinformationen zwischen den Akteuren festgelegt. Das beinhaltet klassischerweise Informationen der Steuerung, der Überwachung (bzw. Monitoring) oder auch asynchrone Ereignismeldungen, die an die Managementplattform von einem MO geschickt werden. Demnach werden im Kommunikationsmodell die kommunizierenden Partner, Kommunikationsmechanismen sowie Syntax und Semantik der ausgetauschten Informationen festgelegt. Für eine technische Managementplattform bedeutet das besonders die Unterstützung unterschiedlicher Kommunikationsmechanismen und die Bereitstellung entsprechender Schnittstellen. In herkömmlichen Netzen kommen dazu beispielsweise häufig auch **Agentensysteme** auf MOs zum Einsatz, wie beispielsweise im SNMP-Managementframework (siehe Abschnitt 4.2.1).

- **Funktionsmodell:** Nach [8] wird im Funktionsmodell die Gesamtaufgabe des Managements in einzelne Funktionsbereiche aufgeteilt. Für einzelne Funktionsbereiche werden selbst wiederum feingranularere Funktionalitäten mit ihren jeweiligen Aufrufkonventionen definiert. Darüber hinaus werden im Funktionsmodell Verwaltungszustände definiert. Die generellen Funktionsbereiche des Managements sind nicht fest definiert, jedoch haben sich unterschiedliche Ansätze durchgesetzt und wurden als Standards festgelegt. Die wohl bekannteste Ausprägung stellen die fünf von der ISO [33] beschriebenen und als *FCAPS* abgekürzten Funktionsbereiche dar:

  - *Fault Management* umfasst die Detektierung, Isolierung und Behebung von Störfällen im Netz, durch Führung und Auswertung von Logdaten, der Reaktion auf Störmeldungen, anschließender Rückverfolgung und Identifizierung sowie Diagnose und Behebung von Störungen.

  - *Configuration Management* definiert, steuert, überwacht und stellt Informationen bereit, die zum Betrieb untereinander verbundener Dienste notwendig sind. Kernfunktionen dazu umfassen u. a. die Konfiguration von Systemen, die Initialisierung und das Entfernen von MOs und die bedarfsabhängige Bereitstellung von Zustandsinformationen eines Systems.

  - *Accounting Management* behandelt die Erhebung, Identifizierung sowie Mitteilung der Nutzung und anfallender Kosten von Ressourcen sowie der Möglichkeit zur Nutzungsbeschränkung auf Basis einer Kostenobergrenze.

  - *Performance Management* befasst sich mit der Sicherstellung der Effektivität der Systeme und ihrer Kommunikation untereinander über die Sammlung Informationen des Netzbetriebs.

  - *Security Management* unterstützt die Anwendung von Sicherheitsrichtlinien durch Funktionen zur Einrichtung, Steuerung und dem Entfernen von Sicherheitsdiensten und -Mechanismen; darüber hinaus durch die Verteilung von sicherheitsrelevanten Informationen und der Berichterstattung von Sicherheitsereignissen.

  Eine Managementplattform muss entsprechend Möglichkeiten zur Festlegung von Funktionsbereichen und Funktionen bereitstellen, sowie mit Elementen aus den anderen Teilmodellen verknüpfen können. Beispielsweise ist eine Verantwortlichkeit und Aufgabe meist mit dem Privileg der Benutzung von Funktionen der Managementplattform verbunden oder mit Informationen (z. B. Sicherheitsereignisse mit dem Sicherheitsmanagement).

2.4.1.2 Werkzeugunterstützung

Die Werkzeugunterstützung des Netzmanagements kann gemäß [8] in drei Formen umgesetzt werden:

- **Isolierter Ansatz:** In dieser Umsetzung gibt es demnach für jedes Problem ein eigenes isoliertes Werkzeug. Isoliert insofern, dass die Werkzeuge untereinander nicht kompatibel sind und daher unabhängig voneinander arbeiten.

- **Koordinierter Ansatz:** Dieser Ansatz erweitert den isolierten Ansatz insofern, dass einzelne Werkzeuge teilweise kommunizieren und Ergebnisse anderer Werkzeuge als Eingabe verwenden. Auch wird in diesem Ansatz oft eine Oberflächenintegration durchgeführt, d. h., eine gemeinsame Nutzeroberfläche für die Werkzeuge eingesetzt.

- **Integrierter Ansatz:** Ein integrierter Ansatz achtet auf die herstellerübergreifende Interpretierbarkeit von Informationen und ihren Austausch über definierte Schnittstellen sowie Protokolle. Darüber hinaus wird eine Zusammenführung von Netz-, System- und Anwendungsmanagement vorgesehen.

Die Frameworkkonzepte, die in dieser Arbeit erstellt werden, zielen primär auf die Realisierung eines integrierten Ansatzes ab, in dem Informationen der vier zuvor beschriebenen Teilmodelle einer Managementarchitektur einerseits als herstellerübergreifende Schnittstelleninformation eingesetzt werden können und andererseits als Frameworkkomponenten aufeinander abgestimmte Schnittstellen bereits vorsehen. Auch sind die Frameworks als Basis für Teillösungen in einem koordinierten Ansatz denkbar. Ein isolierter Ansatz ist in größeren IT-Umgebungen nicht praktikabel und nicht im Sinne der in dieser Arbeit entwickelten Frameworks.

2.4.2 Managementplattformen

> Gemäß [8] sind **Managementplattformen** „Trägersysteme für Managementanwendungen“. Managementplattformen implementieren demnach Managementarchitekturen.

Abbildung 2.4 illustriert allgemeine Zusammenhänge von Komponenten im Netzmanagement, basierend auf [8]. Die Managementplattform – auch Managementinfrastruktur genannt – steht zwischen der gemanagten IT-Infrastruktur und der zum Management genutzten **Managementanwendungen** (synonym auch als Managementapplikation bezeichnet). Die Schnittstelle zur gemanagten IT-Infrastruktur wird als *Southbound-Interface* (SBI), die Schnittstelle zu den Managementanwendungen als *Northbound-Interface* (NBI) oder auch *Application Programming Interface* (API) der Managementplattform bezeichnet. Die Managementplattform bildet zusammen mit den Managementanwendungen schließlich das Managementsystem. In dieser Arbeit wird zudem zugunsten einer eindeutigen Trennung der Begriff **Managementplattforminstanz** als konkrete Implementierung einer Managementplattform verwendet.

2.4.2.1 Gemanagte Infrastruktur und Managementobjekte

Als gemanagte IT-Infrastruktur wird in dieser Arbeit die Menge an Komponenten in einem FSN bezeichnet, die im Netzmanagement berücksichtigt wird. Eine Netzmanagementplattform, wie sie in dieser Arbeit betrachtet wird, beschreibt ein anwendungsspezifisch dienliches Modell der gemanagten IT-Infrastruktur und ihrer IT-Ressourcen. MOs beziehen sich in dieser Arbeit ausschließlich auf Ressourcen des Netzmanagements von FSNs; sie können grundlegend unterschiedlich hinsichtlich ihrer Komponentengruppe, Netzfunktion, Implementierung und Version ausgeprägt sein. In Abbildung 2.4 wird dieser Aspekt durch unterschiedliche Farben der MOs als Teil einer modellhaften gemanagten Infrastruktur verdeutlicht. MOs mit derselben Farbe stellen folglich Elemente der gemanagten Infrastruktur mit gleichartiger Ausprägung dar,

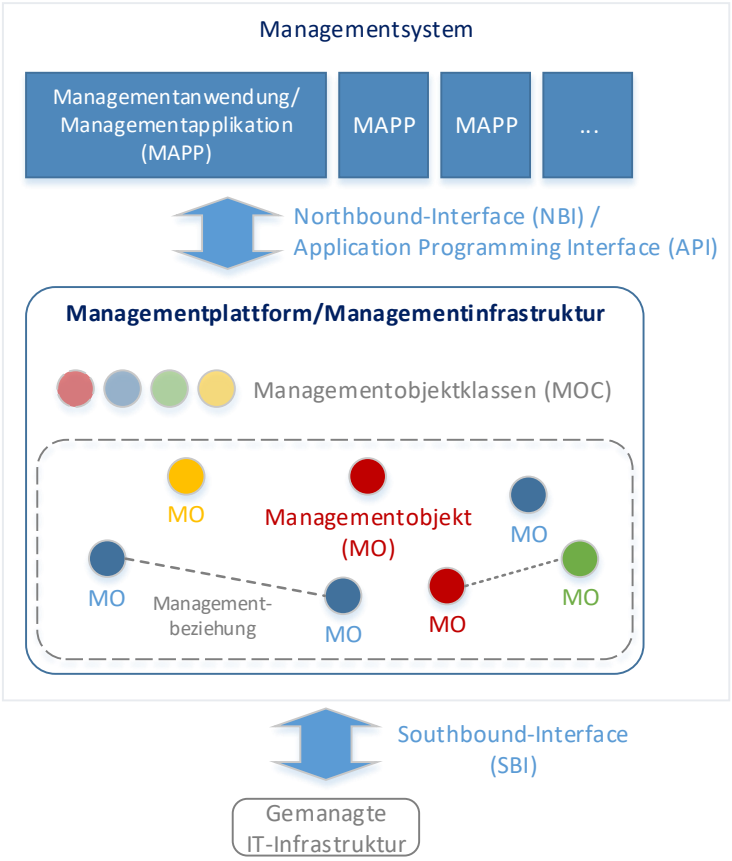


Abbildung 2.4: Generelle Zusammenhänge von Komponenten im Netzmanagement basierend auf [8].

beispielsweise dasselbe Modell eines SDN-Switches mit gleichem Softwareversionsstand. Wie bereits in Abschnitt 2.4.1.1 beschrieben wurde, wird die Beschreibung von MOs als Teil des Informationsmodells einer Managementarchitektur berücksichtigt. Gemäß [8] basiert die Beschreibung eines MO auf sogenannten **Managementobjektklassen** (MOC). MOs sind entsprechend Instanzen von MOCs.

Darüber hinaus bestehen zwischen einzelnen oder mehreren MOs **Managementbeziehungen** (hier auch synonym als **MO-Abhängigkeiten** bezeichnet) [34]. In FSNs sind Managementbeziehungen anders ausgeprägt als in herkömmlichen Netzen. Neu zu berücksichtigende umfassen beispielsweise Steuerbeziehungen zwischen FSN-spezifischen MOs wie SDN-Controller und SDN-fähiger Infrastruktur, oder auch Existenzbeziehungen zwischen VMs und Hypervisors. In Abbildung 2.4 werden Managementbeziehungen beispielhaft über Kanten zwischen zwei MOs repräsentiert.

2.4.2.2 Kernbausteine von Managementplattformen

Nach [8] besteht eine Managementplattform im Wesentlichen aus der eigentlichen sog. Infrastruktur, den Managementanwendungen, Schnittstellen und Entwicklerwerkzeugen. Die Infrastruktur stellt die eigentliche Managementplattform bereit und implementiert ihr Kernsystem, d. h., die eigentliche Logik [23]. Die zwei anderen Hauptbestandteile der Infrastruktur – und mit dem Kernsystem verbunden – sind einerseits der Kommunikationsbaustein zur Kommunikation mit der gemanagten IT-Infrastruktur (hier als SBI bezeichnet), sowie andererseits die Informationsverwaltung. Letztere greift auf eine Datenbank mit MOs zu. Die Infrastruktur ist

über eine API (hier als NBI bezeichnet) mit Managementanwendungen verbunden. Diese können sog. Basisanwendungen sein, die grundlegende Funktionalität wie einen Ereignismanager oder Zustandsmonitor implementieren, d. h., den rudimentären Zugriff auf MOs erlauben, und darauf aufsetzende Managementanwendungen mit den so bereitgestellten Funktionen bedienen können. In dieser Arbeit wird jedoch nicht explizit zwischen Basisanwendungen und Managementanwendungen unterschieden – insbesondere auch, da eine strikte Trennung nicht szenarienübergreifend haltbar ist. Dennoch ist die Beachtung des Konzepts der Nutzung von durch Managementanwendungen aufbereiteten Informationen durch andere Managementanwendungen wichtig und notwendig im Netzmanagement und wird auch in dieser Arbeit berücksichtigt.

Kapitel 3

Anforderungsanalyse

Inhalt

3.1 Der Betrieb föderierter softwarebasierter Netze 36
3.1.1 Charakteristika des Betriebs 36
3.1.2 Anforderungen auf Basis der Charakteristika von FSNs 38
3.2 Methodik zur Szenarienauswahl und Anforderungsableitung 43
3.2.1 Auswahl der Szenarien 43
3.2.2 Anforderungsableitung 43
3.3 Szenario 1: Kooperation im Rahmen eines Projekts 44
3.3.1 IT-Infrastruktur 44
3.3.2 Organisation 47
3.3.3 Bedeutung für das Informationsmodell 51
3.3.4 Bedeutung für das Funktionsmodell 53
3.3.5 Bedeutung für das Kommunikationsmodell 54
3.3.6 Einordnung des Szenarios 55
3.4 Szenario 2: SNs als Wegbereiter der Digitalisierung in der Medizin 55
3.4.1 Relevanz softwarebasierter Netze in der Telemedizin 56
3.4.2 Föderationsbeziehungen und Aufgaben 57
3.4.3 IT-Infrastruktur 59
3.4.4 Organisation 62
3.4.5 Bedeutung für das Informationsmodell 66
3.4.6 Bedeutung für das Funktionsmodell 67
3.4.7 Bedeutung für das Kommunikationsmodell 70
3.4.8 Einordnung des Szenarios 71
3.5 Szenario 3: Bedarfsabhängige Erweiterung von IT-Ressourcen 71
3.5.1 Föderation und IT-Infrastruktur 72
3.5.2 Organisation 74
3.5.3 Bedeutung für das Informationsmodell 76
3.5.4 Bedeutung für das Kommunikationsmodell 76
3.5.5 Bedeutung für das Funktionsmodell 76
3.5.6 Einordnung des Szenarios 76
3.6 Anforderungen an Frameworks für Managementplattformen in FSNs . . 77
3.6.1 Gewichtung und Ableitung der Anforderungen 78
3.6.2 Hot-Spot Analyse 78
3.6.3 Nicht-funktionale Anforderungen 80

3.6.4 Funktionale Anforderungen 81
3.6.5 Anforderungen an die Umsetzung des Informationsmodells 84
3.6.6 Anforderungen an die Umsetzung des Kommunikationsmodells . . . 85
3.6.7 Anforderungen an die Umsetzung des Organisationsmodells 90
3.6.8 Anforderungen an die frameworkspezifische Umsetzung 92
3.6.9 Zusammenfassung der Kernanforderungen 92

Dieses Kapitel dient der Ermittlung von Anforderungen an Frameworks zur Entwicklung von Managementplattformen für FSNs. Die Ableitung relevanter Anforderungen erfolgt zum einen auf Basis einer Herleitung relevanter Charakteristika des Betriebs und Managements von FSNs in Abschnitt 3.1. Zum anderen erfolgt sie auf Basis der in den Abschnitten 3.3 bis 3.5 beschriebenen Szenarien mit variierenden technischen sowie organisatorischen Ausprägungen von FSNs. Die Vorgehensweise zur Herleitung der in diesem Kapitel betrachteten Szenarien wird davor in Abschnitt 3.2 beschrieben. Eine szenarienübergreifende Ableitung von Anforderungen an ein Design für Frameworks für Managementplattformen in FSNs wird schließlich in Abschnitt 3.6 vorgenommen. Eine Grobunterteilung der Anforderungen erfolgt hinsichtlich ihres Einflusses auf das **Informationsmodell**, das **Organisationsmodell**, das **Kommunikationsmodell** sowie das **Funktionsmodell** von Managementplattformen. Zusätzlich werden **nicht-funktionale** Anforderungen aufgestellt. Da sich die in dieser Arbeit definierten Anforderungen nicht an eine klassische Softwareanwendung richten, sondern an Softwareframeworks (vgl. Abschnitt 2.3.2), wird dieser Aspekt durch eine **Hot-Spot-Analyse** als Teil des Abschnitts 3.6 durchgeführt.

3.1 Der Betrieb föderierter softwarebasierter Netze

Netzmanagement ist abhängig von einigen Faktoren mit einem enormen Aufwand verbunden. Beispielsweise anhand der Größe des Netzes (geographisch aber auch anhand der Anzahl der in das Netz integrierten Geräte) sowie der Heterogenität darin vernetzter Komponenten und Anwendungen, Nutzer- und Anwendergruppen. Paradigmen in SNs (vgl. Abschnitt 2.1) verfolgen in erster Linie den Zweck, IT-Netze einerseits effizienter zu nutzen als auch andererseits einfacher betreiben zu können. In SNs hat sich der Betrieb verglichen mit herkömmlichen Netzen stark verändert. Durch den Aspekt der Föderation kommen darüber hinaus weitere Charakteristika des Betriebs hinzu, welche inhärent gegeben und von Managementplattformen zu berücksichtigen sind. Die Effektivität und Qualität des Netzmanagements sind wesentlich von der Eignung der Managementplattform abhängig. Die in dieser Arbeit entwickelten Frameworks für Managementplattformen sollen an diese Charakteristika angepasst sein.

3.1.1 Charakteristika des Betriebs

Einige zentrale Charakteristika des Betriebs von föderierten softwarebasierten Netzen wurden in Vorarbeiten zu dieser Dissertation analysiert und größtenteils in [36] veröffentlicht. Dazu gehören sowohl auf den Charakteristika aufbauend identifizierte Diskrepanzen zu funktionalen sowie nicht-funktionalen Anforderungen an Managementplattformen in FSNs, als auch mögliche Maßnahmen zu ihrer Überwindung.

3.1.1.1 Charakteristika in SNs

SNs für sich, und damit zunächst unabhängig von Föderationsaspekten, haben demnach die folgenden Charakteristika:

- **Ressourcenschichtung.** SNs bestehen auf Basis darunterliegender physischer IT-Ressourcen bzw. auch verschachtelt auf Basis virtueller (als auch manchmal selbst softwarebasierter) Netze. Ihr Aufbau (z. B. bzgl. Topologie und darin vernetzter Komponenten) ist praktisch beinahe komplett unabhängig von darunterliegenden IT-Ressourcen.

- **Remote Konfigurierbarkeit.** Paradigmen von SNs ermöglichen eine remote, logisch zentralisierte Konfiguration des Netzes. Anders als bei Hardware-Appliances, welche oft vor Ort und manuell konfiguriert werden müssen (bspw. oft das Aufspielen neuer Firmware, die Vernetzung mit anderen Komponenten mit entsprechenden Kabeln oder auch der komplette Austausch der Appliance), können praktisch alle Aufgaben in Software gelöst werden. Beispielsweise werden virtuelle Komponenten durch Mechanismen der Netzvirtualisierung vernetzt; der Austausch einer virtuellen Appliance kann durch den Austausch einer VM-Instanz realisiert werden. Jedoch unterstützen selbst physische Appliances im SDN-Paradigma ebenfalls Funktionen zur Steuerung aus der Ferne, beispielsweise über OpenFlow.

- **Geographische Verteilung.** SNs unterstützen reibungslos eine geographische Verteilung der IT-Ressourcen. Die darunterliegenden Komponenten werden beispielsweise über ein virtuelles privates Netz verbunden; die Verteilung wird auf darauf aufgebauten SNs realisiert und beeinflusst deren Konfiguration nur indirekt. So werden durch SNs beispielsweise auch föderierte Infrastrukturen besser unterstützt. Die Verbindung nutzt oft öffentliche Netzinfrastruktur wie die des Internets.

- **Unmittelbare Steuerung.** Da die Konfiguration von SNs und seinen Komponenten hauptsächlich durch Anpassungen von Softwarezuständen geändert werden kann, erfolgen sie zeitlich praktisch unmittelbar (d. h. üblicherweise innerhalb weniger Sekunden bis Minuten). Bei herkömmlichen hardwarebasierten Netzen hingegen können Konfigurationsänderungen (z. B. Änderung in der Netztopologie und Vernetzung, Aufsetzen neuer oder auch die Außerbetriebnahme bestehender Systeme) durchaus einige Stunden bis Tage dauern.

- **Ressourcen-Pooling.** Da SNs oft Teile der Infrastruktur von *Cloud-Computing*-Lösungen bereitstellen (z. B. Virtualisierung und zentralisierte Steuerung), ist an dieser Stelle das Charakteristikum des Ressourcen-Poolings davon abgeleitet (vgl. [37]). SNs ermöglichen durch Virtualisierung ein *Pooling* (engl. *Bündelung*) von IT-Ressourcen, welche gemeinsam auf Basis einer zentralisierten Sicht genutzt werden können. Dies ist ein weiterer Aspekt, um föderierte Infrastrukturen besser zu unterstützen: Partner der Föderationen können entsprechend dezentrale Ressourcen zusammenschließen, welche gemeinsam komplett oder wiederum unterteilt in weitere Pools genutzt werden können.

- **Skalierbarkeit.** Softwarebasierte Netze sind üblicherweise gut skalierbar. Die physische und damit auch virtuelle Infrastruktur kann praktisch immer erweitert werden, ohne dass Konfigurationsänderungen auf darüberliegenden SNs notwendig sind. Die Ressourcen können einfach integriert und genutzt werden.

- **Ausmaß des Netzes.** SNs sind gemessen an Komponenten und Systemen im Netz, jedoch auch anhand ihrer geographischen Ausbreitung oft deutlich größer als herkömmliche Netze. Sie können auf der bestehenden physischen Plattform (abhängig von ihrer zur Verfügung stehenden Leistung) potenziell weitere hunderte bis mehrere tausende isolierte Systeme umfassen, welche in diesem Sinne vollwertige Systeme darstellen können.

- **Heterogenität.** Eine Heterogenität an Komponenten, Protokollen, Schnittstellen und Systemen, wie sie bereits in herkömmlichen Netzen stark verbreitet ist, verstärkt sich weiterhin in SNs. Einflussfaktoren sind die verschiedenen Paradigmen und jeweils unterschiedliche Implementierungen paradigmenspezifischer Komponenten wie VIMs mit uneinheitlichen Schnittstellen und Funktionen (vgl. [38]). Besonders in föderierten Netzen kann dieser Aspekt noch stärker ausgeprägt sein, da unterschiedliche Partner nicht selten auch unterschiedliche Systeme benutzen und in die Föderation einbringen.

- **Weitgehend agentenlose Ansätze.** Da in FSNs zentralisierte Komponenten wie SDN-Controller, VIMs oder Virtualisierungssysteme die Funktionalität zur Überwachung und Steuerung der gemanagten Infrastruktur mitbringen, ist meist kein flächendeckender Einsatz (z. B. pro MO) von Agentensystemen notwendig.

3.1.1.2 Charakteristika in FSNs

Die im vorherigen Abschnitt beschriebenen Charakteristika treffen unterschiedlich ausgeprägt auf softwarebasierte Netze zu. Eine Föderation auf Basis von softwarebasierten Netzen bringt weitere Betriebscharakteristika mit sich. Darunter mindestens die Folgenden:

- **Vertrauen.** Wie bereits in Abschnitt 2.2.2 beschrieben ist, ist Vertrauen ein wichtiger Aspekt im Betrieb von FSNs. Das Vertrauen zwischen den Kooperationspartnern kann stark als auch kaum vorhanden und von Partner zu Partner unterschiedlich sein, wodurch das Netzmanagement beeinflusst wird. Beispielsweise kann es sein, dass einige Partner ihre Systeme und Dienste nicht auf IT-Ressourcen von Partnern betreiben wollen, denen sie nicht vertrauen.

- **Domänenbildung.** Administrative Domänen sind wesentlicher Bestandteil von FSNs, da mehrere Parteien IT-Ressourcen darin betreiben. Eine sich verstärkende Ausprägung ist die Überschneidung von administrativen Domänen. Da SNs auf geteilten physischen IT-Ressourcen aufbauen, werden sowohl virtuelle Systeme und Netze als auch Dienste von Partnern auf Infrastruktur mehrerer Parteien betrieben. Entsprechend ist nicht nur der eigentliche Besitzer der physischen Systeme für deren Management zuständig, sondern durch implizierte Abhängigkeiten auch die Verantwortlichen für die darauf laufenden virtuellen Systeme und Dienste. Genauso andersherum, da der Betrieb der physischen Systeme darüberliegende Systeme direkt beeinflusst.

- **Betriebs- und Managementkonzepte.** Die unterschiedlichen Partner in einer Föderation bringen üblicherweise jeweils ein eigenes Verständnis von Management sowie eigene Betriebs- und Managementkonzepte in die Föderation ein. Dazu zählen beispielsweise eigene Systeme und Werkzeuge mit ihren jeweiligen unterschiedlichen Schnittstellen, aber auch organisatorische Aspekte wie festgelegte Gruppen und Verantwortlichkeiten.

- **Entscheidungsauswirkungen.** Lokale Entscheidungen von Föderationspartnern haben möglicherweise Auswirkungen auf alle anderen Parteien. Beispielsweise hat der Ausstieg eines Partners aus der Föderation einen großen Einfluss auf alle anderen Partner. Komponenten und Teilnetze, die auf seinen Ressourcen zur Verfügung gestellt werden, müssen nach Ausscheiden des Partners auf anderen noch verfügbaren IT-Ressourcen umgezogen und betrieben werden.

- **Aufteilung von Ressourcen.** Physische und virtuelle Ressourcen in FSNs können in viele Segmente unterteilt werden. Die Aufteilung kann beispielsweise auf die unterschiedlichen Mandanten im FSN aufgeteilt werden.

3.1.2 Anforderungen auf Basis der Charakteristika von FSNs

Im Rahmen einer ersten allgemeinen Anforderungsanalyse auf Basis der zuvor aufgestellten Betriebscharakteristika werden Akteure und Anforderungen an Managementplattformen in FSNs aufgestellt. Da FSNs eine spezielle Ausprägung klassischer Netze darstellen, bleibt eine Menge herkömmlicher Akteure weiterhin bestehen. Weitere FSN-spezifische Akteure ergeben sich dagegen aus der Paradigmendefinitionen von SNs, die in Abschnitt 2.1 beschrieben wurde.

3.1.2.1 Allgemeine Akteure in FSNs

> Ein **Akteur** beschreibt in dieser Arbeit eine Person oder ein System, das im Kontext des Netzmanagements mit der Managementplattform interagiert.

Eine Übersicht über alle Akteure ist in Abbildung 3.1 als Baumstruktur mit generellen Gruppierungsknoten (hellblau) und tatsächlichen Akteuren (dunkelblau) illustriert. Kindknoten repräsentieren darin Spezialisierungen ihrer jeweiligen Elternknoten. Wie bereits im Kontext von SNMP in RFC 3414 [39] beschrieben wird, können Nutzer bzw. Akteure im Netzmanagement sowohl Personen als auch Anwendungen sein. Diese Unterteilung wird auch in diesem Ansatz als Ausgangspunkt zur Klassifikation von Akteuren genutzt. Entsprechend wird grob zwischen **Nutzern** sowie **Managementobjekten** als Akteuren mit der Managementplattform unterschieden.

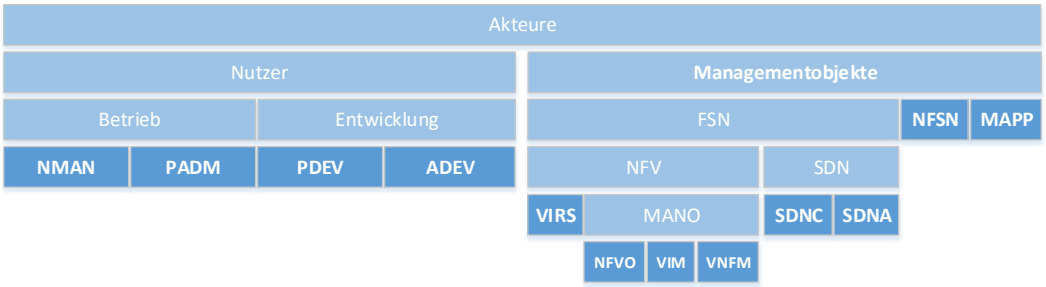


Abbildung 3.1: Übersicht und Gruppierung der Akteure im Netzmanagement in FSN (Hellblau: Gruppierungsknoten; Dunkelblau: Akteure).

Einen wichtigen Block der Akteure einer Managementplattform bilden die menschlichen Nutzer. Diese können generell in Akteure des Betriebs und der Entwicklung im Netzmanagement eingeteilt werden. Der zentrale Benutzer im Betrieb ist der **Netzmanager** (NMAN), welcher im Netzmanagementbetrieb des FSN einen bestimmten Verantwortungsbereich hat. Da Verantwortungsbereiche organisationsübergreifend üblicherweise nicht einheitlich definiert sind, ist dieser Akteur entsprechend generisch beschrieben. Ein Netzmanager gehört zu einer bestimmten Partnerorganisation in der Föderation. Beispielsweise erfüllt ein NMAN im Kontext einer administrativen Domäne eine bestimmte Aufgabe und Verantwortlichkeit. Eine ähnliche, aber spezifischere Rolle im Betrieb des Netzmanagements ist der **Managementplattformadministrator** (PADM). Ein PADM erfüllt administrative Tätigkeiten betreffend die Plattform selbst. Neben den Rollen des Betriebs existieren Rollen der Entwicklung eines Managementsystems. Ein darin zentraler Akteur ist der **Managementplattformentwickler** (PDEV). Ein PDEV ist dafür zuständig, dass die genutzte Managementplattform für den jeweiligen Anwendungsfall bzw. die jeweilige Umgebung einsetzbar ist. Beispielsweise implementiert ein PDEV passende Schnittstellen, Funktionen und Konnektoren für die Managementplattform. Das ergänzende Gegenstück dazu ist der **Managementanwendungsentwickler** (ADEV). Dieser entwickelt Managementanwendungen passend zur Managementplattform. Beispielsweise entwickelt ein ADEV eine graphische Nutzeroberfläche zur Visualisierung des Status von Netzkomponenten.

Neben den unterschiedlichen Nutzerklassen müssen ebenfalls Systeme und Anwendungen als Akteure berücksichtigt werden. Diese unterteilen sich in FSN-spezifische und unspezifische Akteure. In der Kategorie der FSN-spezifischen Akteure kann zwischen **NFV**- sowie **SDN**-spezifischen Komponenten unterschieden werden. Akteure aus dem NFV-Paradigma sind insbesondere Komponenten der Virtualisierungsschicht, die durch die generische Komponente des **Virtualisierungssystems** (VIRS) beschrieben werden. Darüber hinaus sind die Komponenten des **MANO**-Blocks, der **NFV Orchestrator** (NFVO), **Virtualized Infrastructure**

Manager (VIM) und **VNF Manager** (VNFM) zu berücksichtigen (vgl. Abschnitt 2.1.3). Anders als in der NFV-Referenzarchitektur wird in dieser Arbeit die Möglichkeit mehrerer NFVOs und VIMs in einer Föderation explizit berücksichtigt. Vor allem aus dem SDN-Bereich ist der **SDN-Controller** (SDNC) die zentrale Komponente zur Steuerung des Netzes. Hingegen zählen SDN-Switches in SDNs nicht zu direkten Akteuren mit einer Managementplattform, da die Steuerung konzeptionell ausschließlich auf den SDNC zentralisiert wurde. Eine im Vergleich zu herkömmlichen Netzen außerdem neue Komponente ist die auf dem SDNC aufgesetzte **SDN-oder Netzanwendung** (SDNA). Diese nutzt in erster Linie Funktionen eines SDNC, um das Netz zu konfigurieren. Andererseits sind auch Schnittstellen zu Managementanwendungen der Managementplattform denkbar, um beispielsweise weitere Informationen aus anderen Bereichen mit in die Netzkonfiguration einzubeziehen.

Nicht-FSN-spezifische Systeme (NFSN), die als Akteure mit einer Managementplattform interagieren, werden in dieser Arbeit generisch als solche bezeichnet. NFSNs können beispielsweise Kommunikationsdienste wie E-Mail sein, oder auch Netzdienste wie ein Intrusion-Detection-System (IDS) oder eine Firewall, die direkt Überwachungsdaten an die Managementplattform schicken. Eine stets vorhandene Spezialisierung nicht-FSN-spezifischer Systeme stellt der Akteur **Managementanwendung** (MAPP) dar. MAPPs nutzen die Managementplattform als Infrastruktur, um Managementaufgaben zu erfüllen und NMAN in ihren Aufgaben zu unterstützen. Auch ist die Überwachung von MAPPs selbst notwendig, um die Zuverlässigkeit des Netzes sicherzustellen.

3.1.2.2 Allgemeine Anforderungen

Auf Basis der beschriebenen Charakteristika und Herausforderungen im Betrieb von FSNs lassen sich bereits szenarienübergreifend geltende Anforderungen an Managementplattformen in FSNs ableiten. Die Anforderungen werden gemäß dem in Abschnitt 3.6 definierten Gewichtungsschema als *essenziell*, *wichtig* oder *empfehlenswert* priorisiert.

3.1.2.2.1 Automatisierung der Kernprozesse. FSNs sind hochdynamisch und bieten umfängliche Möglichkeiten zur Automatisierung des Managements. Eine Managementplattform muss daher die Automatisierung der Managementkernprozesse Überwachung und Steuerung erlauben. Ein Framework muss entsprechend eine Komponentenstruktur vorgeben, durch die eine Automatisierung implementierbar ist. Diese Anforderung ist *essenziell*, da sie die Hauptfunktionalität von Managementplattformen beschreibt.

3.1.2.2.2 Quasi-Echtzeitfähigkeit. FSNs sind potenziell durch durchgehende Virtualisierung und aufeinander aufgesetzte Schichten, weiterhin verstärkt durch eine hohe Anzahl an Partnern und Mandanten, oft deutlich größer als herkömmliche Netze (gemessen an der Anzahl der Systeme im Netz). Informationen der entsprechend vielen Datenquellen (vgl. Anforderung *Skalierbarkeit*) müssen zuverlässig und möglichst in Echtzeit bearbeitet werden können. Dazu muss die Plattform selbst ausreichend performant (beispielsweise über geeignete Parallelisierung) sein. Diese Anforderung ist *essenziell*, da ansonsten veraltete Daten resultieren können.

3.1.2.2.3 Unterbrechungsfreier Betrieb. Die Managementplattform stellt die zentrale technische Schnittstelle zu Managementfunktionen dar. Der Ausbau der auf dem Framework entwickelten Plattformen darf den Betrieb nicht unterbrechen und die Möglichkeit zum Management nicht beeinträchtigen. Da einerseits kurze geplante Ausfälle dennoch akzeptabel sein können und andererseits einfache Maßnahmen zur Umgehung dieser Anforderung existieren (z. B. Aufsetzen eines Zweitsystems), ist diese Maßnahme als *empfehlenswert* eingestuft.

3.1.2.2.4 Nachvollziehbarkeit von Aktionen. Durch die voneinander abhängige Infrastruktur und mehrerer auf der Infrastruktur agierender Partner auf unterschiedlichen Ebenen

ist es erforderlich, dass Aktionen nachvollziehbar sind, um beispielsweise auftretende Probleme und ihre Ursachen zu finden. Insbesondere automatisiert durchgeführte Aktionen müssen auch nachträglich nachvollziehbar sein. Diese Anforderung ist *empfehlenswert*, da sie vor allem Nachvollziehbarkeit an sich unterstützt, jedoch auch ohne sie effektives Management möglich ist.

3.1.2.2.5 Logische Zentralisierung. Automatisiertes Netzmanagement ist praktisch nur mit einer zentralen Datenbasis und somit gesamtheitlichen Perspektive über die gemanagte Umgebung möglich. Auch wenn sich ein FSN selbst sowohl geographisch als auch administrativ über mehrere Bereiche erstreckt, so ist das Netz aufgrund von Abhängigkeiten untereinander im Management als Gesamtes zu erfassen. Beispielsweise müssen Auswirkungen lokaler Entscheidungen einzelner Partner für das ganze Netz berücksichtigt werden. Aus den genannten Gründen ist diese Maßnahme als *essenziell* eingestuft.

3.1.2.2.6 Agentenloses Management. MOs in FSNs bieten umfängliche Überwachungs- und Steuerungsfunktionen selbst an. Eine Managementplattform muss in der Lage sein, FSNs ohne eine unbedingt notwendige Installation von Agentensystemen in der gemanagten Infrastruktur zu überwachen und zu steuern. Die Anforderung ist *essenziell*, da die Möglichkeit zum Verzicht auf eine Installation und Konfiguration von Agentensystemen ein Grundcharakteristikum in FSNs ist.

3.1.2.2.7 Resilienz und Fehlertoleranz. Die Skalierbarkeit von Software-Systemen wird nicht selten über einen verteilten modularen Ansatz (z. B. mit sogenannten *Microservices* und entsprechender Netzkommunikation) gelöst. Dennoch muss die Plattform auch in Ausnahmefällen, beispielsweise Verbindungsabbrüchen und fehlender Konnektivität funktionieren. Da sich FSNs oft über öffentliche IT-Infrastrukturen wie das Internet erstrecken, ist dies ein besonders relevanter Aspekt und daher *essenziell*.

3.1.2.2.8 Sichere Kommunikation. Dadurch, dass sich FSNs oft über zum Teil öffentliche IT-Infrastrukturen erstrecken, muss auch die Kommunikation zwischen Nutzern zu Managementapplikationen sowie von Managementanwendungen zur Managementplattform abgesichert sein. Des Weiteren muss jegliche designabhängig notwendige interne Kommunikation (z. B. zwischen Datenbank und Plattform) abgesichert werden. Diese Anforderung ist (vor allem in über öffentliche Infrastruktur verbundene FSNs) *essenziell*.

3.1.2.2.9 Skalierbarkeit. Da FSNs und die darunterliegenden Technologien in der Regel eine gute Skalierbarkeit bieten, muss eine Managementplattform diese in gleicher Weise erfüllen. Eine Skalierbarkeit muss nicht nur in Hinblick auf die Größe der gemanagten Umgebung, sondern auch auf die Anzahl der Föderationspartner, Nutzer, Mandanten, administrative Domänen, Managementanwendungen und besonders auch Informationen berücksichtigt werden. Die Anforderung kann höchstens in kleinen und starren FSNs vernachlässigt werden und ist daher *wichtig*.

3.1.2.2.10 Berücksichtigung von Ressourcenabhängigkeiten. Durch die Möglichkeit zur beliebig tiefen Schachtelung von SNs und virtueller Komponenten müssen daraus resultierende Abhängigkeiten abgebildet und beispielsweise unter Aspekten der MOC-Modellierung, Kommunikation und Steuerung von MOs sowie Verfügbarkeit von Teil-Netzen berücksichtigt werden. Daher wird diese Anforderung als *essenziell* eingestuft.

3.1.2.2.11 Adressierbarkeit von MOs. MOs im gemanagten Netz müssen über mehrere sowohl über administrative Domänen als auch Virtualisierungsebenen eindeutig identifiziert und adressiert werden können. Andernfalls ist kein effektives Management der Umgebung möglich, wodurch diese Anforderung als *essenziell* eingestuft ist.

3.1.2.2.12 Geographische Ressourcenverteilung. Die geographische Verteilung von Ressourcen kann den Betrieb des Netzes enorm beeinflussen. Beispielsweise sollte ein kritisches verteiltes System unabhängig von öffentlicher Netzinfrastruktur installiert sein. Die geographische Verteilung von Ressourcen muss entsprechend im Informationsmodell abbildbar sein und auch im Funktionsmodell berücksichtigt werden können. Die Anforderung kann in Szenarien ohne geographische Verteilung nicht berücksichtigt werden und ist daher als *wichtig* eingestuft.

3.1.2.2.13 Datenaktualität. Wie beschrieben, zeichnen sich SNs auch durch unmittelbare Konfigurierbarkeit innerhalb weniger Minuten aus. Die gemanagte Umgebung kann sich entsprechend schnell ändern (vor allem mit vielen Parteien). Beispielsweise sind so auch Aktivitäten wie Ressourcendiscoveries (d. h. Scannen des Netzes nach Systemen), Schwachstellen-, Dienst- und Port-Scans betroffen und müssen aktuelle Daten liefern. Diese Anforderung ist *essenziell*.

3.1.2.2.14 Heterogene Managementkonzepte. Gerade in dezentral organisierten Föderationen müssen heterogene Management- und Betriebskonzepte, beispielsweise hinsichtlich Aufgaben und Verantwortlichkeiten, berücksichtigt werden können. Diese Anforderung gilt nicht für alle Szenarien und ist daher *wichtig*.

3.1.2.2.15 Administrative Domänen. Durch die in FSNs praktisch immer vorkommenden administrativen Domänen muss im Management sichergestellt sein, dass sowohl physische als auch virtuelle IT-Ressourcen zunächst ihren jeweiligen Domänen zugewiesen werden können. Diese Anforderung ist in Föderationen nur selten vernachlässigbar und daher *wichtig*.

3.1.2.2.16 Domänenüberschneidung. Wie beschrieben treten in FSNs oft Überschneidungen administrativer Domänen auf. Verantwortlichkeiten und Zuständigkeiten müssen im Netzmanagement jedoch stets eindeutig sein, daher ist die Anforderung als *essenziell* eingestuft.

3.1.2.2.17 Mandantenfähigkeit. Wie in den Charakteristika des Betriebs von FSNs beschrieben, bieten sie eine einfache Aufteilung von Ressourcen auf potenziell viele Mandanten. Eine Managementplattform muss entsprechend selbst als zentrale Managementinstanz in einem FSN den Zugriff durch und die Trennung von vielen Mandanten unterstützen. Diese Anforderung ist in jedem Fall *essenziell*.

3.1.2.2.18 Föderationsbeziehungen. Anders als in herkömmlichen Netzen und SNs müssen in FSNs besonders auch Beziehungen zwischen Parteien in einer Föderation berücksichtigt werden. Dazu zählen beispielsweise Vertrauensbeziehungen. Entsprechende Funktionen und Modelle müssen dies unterstützen. Diese Anforderung ist in manchen Szenarien notwendig und daher für ein generisches Design *wichtig*.

3.1.2.2.19 Dokumentation. Da alle Funktionen eines Frameworks von Entwicklern nutzbar sein sollen, muss dessen Nutzung ausführlich und insbesondere geeignet dokumentiert sein. Diese Anforderung ist *empfehlenswert*, da sie die Nutzbarkeit des Designs erhöht, das Plattformdesign jedoch nicht einschränkt.

3.1.2.2.20 Unterstützung der Implementierung. Die Implementierung einer geeigneten Managementplattform muss unterstützt werden, beispielsweise durch die Möglichkeit zum Einsatz unterschiedlicher geeigneter Programmiersprachen. Entsprechend sollen die Nutzbarkeit und die Motivation zur Nutzung erhöht werden und ist daher *empfehlenswert*.

3.1.2.2.21 Langlebigkeit des Frameworks. Ein Framework muss möglichst langlebig gestaltet sein. Die Entwicklung von FSNs kann nicht als abgeschlossen betrachtet werden und neue Systeme, Konzepte und Paradigmen in dieser Richtung sind weiterhin in Entwicklung. Entsprechend ist diese Anforderung als mindestens *wichtig* angesehen, da konkrete aktuelle Umsetzungen von FSNs nicht betroffen sind.

3.1.2.2.22 Erweiterbarkeit des Frameworks. Das Framework muss entsprechend (vgl. Anforderung *Langlebigkeit des Frameworks*) erweiterbar und an neue Konzepte im Bereich softwarebasierter Netze anpassbar sein. Die Anforderung ist *wichtig*.

3.1.2.2.23 Grad der Designfreiheit. Ein Aspekt der insbesondere für Frameworks relevant ist, auch bezüglich der vorhergehend beschriebenen Anforderungen, ist ein geeigneter Kompromiss zwischen Nutzbarkeit und Erweiterbarkeit. Die Nutzung muss einerseits einfach und möglichst unmittelbar sein, andererseits soll die Funktion und Erweiterbarkeit nicht eingeschränkt sein. Diese Anforderung ist *empfehlenswert* und erhöht generell die Nutzbarkeit eines Frameworks.

3.1.2.2.24 Nutzerfreundlichkeit. Eine Anforderung, die sich implizit aus den Charakteristika ableitet, ist die einfache Nutzbarkeit einer Managementplattform. Die Charakteristika beschreiben, trotz einiger Vorteile durch neue Paradigmen, auch komplexere Umgebungen. Es ist *wichtig*, dass diese Komplexität jedoch nicht über eine Managementplattform an die Nutzer weitergetragen wird; sie muss vielmehr die Umgebung vereinfachen.

3.2 Methodik zur Szenarienauswahl und Anforderungsableitung

Die folgenden Szenarien komplementieren in der Anforderungsanalyse die bereits beschriebenen generellen Anforderungen an Managementplattformen in FSNs. Sie zeigen konkrete technische und organisatorische Ausprägungen von FSNs auf.

3.2.1 Auswahl der Szenarien

Die Auswahl der Szenarien basiert auf der in Abschnitt 2.2 hergeleiteten Morphologie, um unterschiedliche Ausprägungen von FSNs berücksichtigen zu können. Tabelle 3.1 umfasst die relevanten und dynamischen Charakteristika der Morphologie, anhand der die Szenarien klassifiziert werden. Für die Anforderungsanalyse wird in dieser Arbeit davon ausgegangen, dass Ausprägungen der einzelnen Charakteristika für sich stehen und Anforderungen unabhängig von Kombinationen gelten. Entsprechend ist es für die Anforderungsanalyse vor allem von Bedeutung, dass jede Charakteristikumsausprägung in mindestens einem der Szenarien berücksichtigt wird.

3.2.2 Anforderungsableitung

Das generelle Ziel der Frameworks für Managementplattformen ist ihre Anpassbarkeit an und folglich Nutzbarkeit in vielerlei FSN-Umgebungen. Jedes der folgenden Szenarien beschreibt daher eine konkrete prototypische FSN-Umgebung mit unterschiedlichen Ausprägungen hinsichtlich Föderation und Management.

Im jeweiligen Szenario werden notwendige Eigenschaften und Funktionen einer darin geeigneten Managementplattform adressiert und jeweils über Randnotizen in den Szenarienbeschreibungen hervorgehoben. Sie dienen als Grundlage für eine szenarienübergreifende Anforderungsanalyse in Abschnitt 3.6 in Form einer *a)* Hot-Spot-Analyse sowie *b)* der Ableitung von Anforderungen an die resultierenden Frameworks.

Charakteristikum	Ausprägung in FSNs		
Räumliche Dimension	regional	überregional	gemischt
Dauer	kurzfristig		langfristig
Koordination	implizit		explizit
Gruppenstruktur	flexibel		fest
Vertrauen	indirekt	direkt	gemischt
Aufgabenzuordnung	statisch		dynamisch
Dynamik	statisch		dynamisch
Rollenvergabe	statisch		dynamisch
Relation zw. administrativen Domänen	überlappend		disjunkt
Zielsetzung	überlappend	komplementär	gemischt
Domänenstruktur	global	hierarchisch	unabhängig
Art der Infrastrukturnutzung	symmetrisch		asymmetrisch

Tabelle 3.1: Vorlage einer Morphologie von Föderationen in SNs zur Einordnung der Szenarien.

Bezüglich *a)* werden dazu szenarienübergreifend notwendige Eigenschaften und Funktionen einer Managementplattform verglichen. Sie treffen entweder immer zu und sind folglich Frozen-Spots, oder sie unterscheiden sich in den unterschiedlichen Szenarien und stellen Hot-Spots dar. Bei *b)* der Anforderungsanalyse an die Frameworks werden letztlich szenarienübergreifende Anforderungen an das Framework für Managementplattformen beschrieben.

3.3 Szenario 1: Kooperation im Rahmen eines Projekts

Die Zusammenarbeit mehrerer voneinander autonomer Unternehmen ist üblicherweise dann notwendig, wenn die Erreichung eines Ziels die mittel- oder langfristige Zusammenführung von Ressourcen wie Personal oder Material erfordert. Ein Beispiel hierfür ist die Kooperation von Einrichtungen aus Forschung und Wirtschaft im Rahmen eines gemeinsamen Projekts. Derartige Projekte werden vielfach pro Jahr initiiert und haben oft jeweils wiederum eine Laufzeit von mehreren Jahren. Beispielsweise wurde das unter dem EUREKA/CELTIC-NEXT-Dachverband initiierte Projekt *SENDATE-PLANETS*, wie der CELTIC-NEXT-Website [40] entnommen werden kann, im Jahr 2016 mit einer Laufzeit von drei Jahren und mehr als zwei Dutzend Projektpartnern aus Deutschland und Finnland gestartet. Das übergeordnete Ziel von SENDATE-PLANETS war demnach die Entwicklung einer sicheren und flexiblen IT-Infrastruktur aus geographisch verteilten Datenzentren. Die Ausgangslage der Beziehung der Partner ist verschiedenartig: Einige Partner haben bereits in jeweils einem anderen Kontext miteinander kooperiert und sind untereinander bekannt, andere hatten bis dato keinen Kontakt zueinander. Die folgenden Abschnitte zu IT-Infrastruktur und Organisation beziehen sich auf die in Abbildung 3.2 gezeigte beispielhafte Umsetzung eines FSN im Kontext des Szenarios.

Aspekte des Vertrauens

3.3.1 IT-Infrastruktur

Zur effektiven Erfüllung der Aufgaben und einer effizienten Zusammenarbeit der Projektpartner ist eine gemeinsame, unter den Parteien gleichberechtigt nutzbare föderierte Infrastruktur zur Koordination des Projekts und als Plattform für eine gemeinsame Entwicklungs- und Testumgebung notwendig. Unter Ersteres fallen Kommunikationsdienste wie E-Mail, Software für Videokonferenzen und Messaging-Dienste, sowie Systeme zur Koordination und Dokumentation, z. B. gemeinsame Kalender, Dateiablagesysteme oder ein gemeinsamer Webauftritt. Die Infrastruktur für gemeinsame Entwicklungsumgebungen umfasst weitere Komponenten und Dienste, wie virtuelle private Netze (VPN), Versionierungssysteme, Software zum Projektmanagement, Wiki-Software und Virtualisierungssysteme zum Testen von Anwendungen. Zur ef-

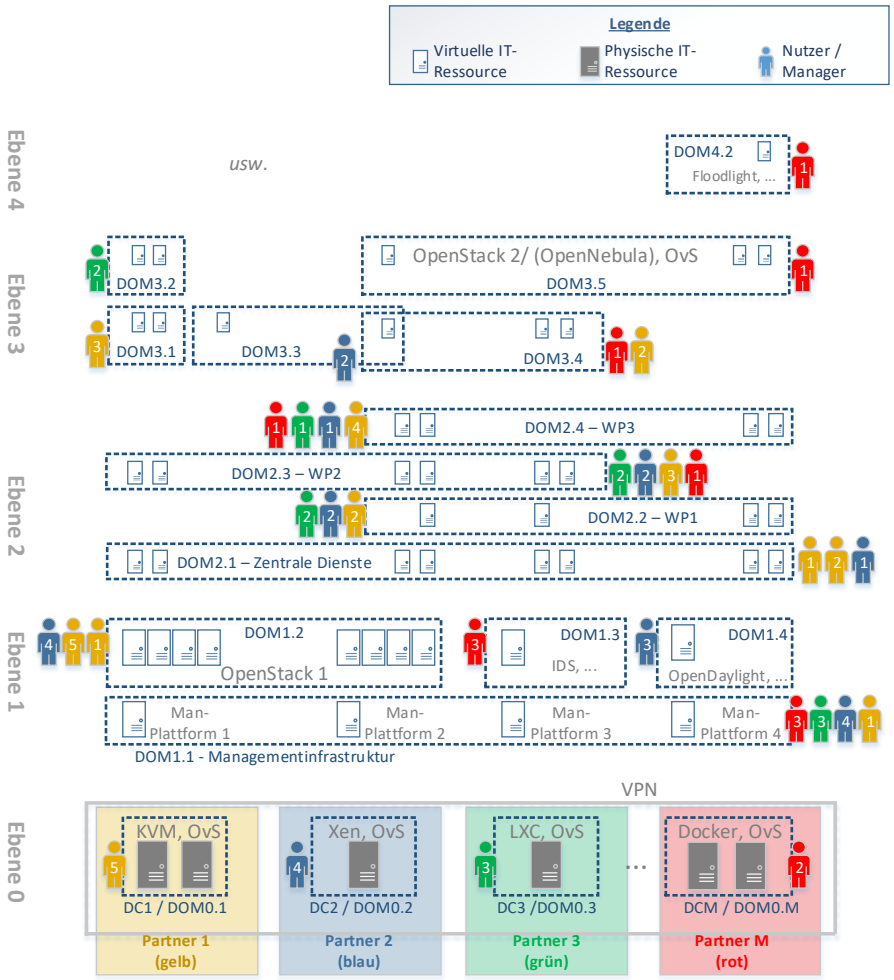


Abbildung 3.2: Exemplarische mehrebnige IT-Infrastruktur für Szenario 1. Die Ebenen beschreiben aufeinander aufbauende Virtualisierungsschichten.

fizienten Organisation der Dienst-Infrastruktur ist es zudem sinnvoll, weitere übergreifende Dienste wie Verzeichnisdienste oder Zugangsgateways zentralisiert anzubieten.

3.3.1.1 Aufbau der föderierten Infrastruktur

Heterogenität der Infrastruktur

Grundfunktionen zur Kommunikation und Organisation (z. B. E-Mail) werden zu Beginn des Projekts bei dynamischer bedarfsspezifischer Erweiterung der Dienste bereitgestellt. Die Erweiterung erfolgt dann durch einen zuständigen Partner oder auf freiwilliger Basis durch einen Partner. Die föderierte IT-Infrastruktur ist im Szenario in mehrere Datenzentren unterteilt. Jeder Partner stellt genau ein Datenzentrum (*DC1* bis *DCM* auf *Ebene 0*) bereit, das die vom jeweiligen Partner in die Föderation eingebrachten IT-Ressourcen kapselt. Einige Projektpartner betreiben bereits selbst eigene IT-Infrastrukturen, basierend auf Teilaspekten von SNs, mit denen sie Erfahrungen haben oder für ihr Vorhaben benötigen und folglich in die Föderation einbringen: Unterschiedliche VIRS oder VIMs, SDN-Infrastrukturkomponenten und -Controller bilden folglich ein stark heterogenes FSN. Die Vernetzung wird über entsprechend geeignete Netzvirtualisierungslösungen realisiert (vgl. Abschnitt 2.1.1.2) und kann flexibel gemäß unterschiedlicher Kriterien beispielsweise für Untervorhaben, nach Vertrauen, Sicherheitskriterien für Daten und Systeme oder zur Gruppierung ähnlicher Systeme in Subnetze erfolgen.

Durch geschachtelte Virtualisierung werden die Ressourcen in unterschiedliche Ebenen eingeteilt. Die unterste Ebene, Ebene 0, umfasst die von den Partnern unmittelbar zur Verfügung gestellten Ressourcen in den einzelnen Datenzentren. Die IT-Ressourcen auf Ebene 0 sind über ein VPN verbunden, aufgesetzt auf öffentlicher Infrastruktur wie dem Internet. Ressourcen auf darauf aufbauenden Ebenen (d. h. Ebene 1 bis Ebene N) werden durch Ressourcen darunterliegender Ebenen realisiert. Beispielsweise werden Ressourcen der Ebene 2 durch Ressourcen der Ebene 1 und der Ebene 0 durch gängige VIRS wie KVM, Xen, LXC und Docker realisiert. Durch VIMs wie *OpenStack 1* wird ein Teil von Ebene 1 durch Partner 1 und Partner 2 aufgespannt, in der virtuelle Ressourcen der NFVI (z. B. Compute-Node, Storage-Node, Controller-Dienste, usw.) virtualisiert auf Ressourcen der Ebene 0 aufgesetzt werden. Die horizontale Ausrichtung der Ressourcen in Abbildung 3.2 ist dabei wesentlich: Sie beschreibt die Realisierungsabhängigkeit zu einem System auf den darunterliegenden Ebenen. *OpenStack 1* wird im beschriebenen Szenario auf Basis von IT-Ressourcen von Partner 1 sowie Partner 2 realisiert und bildet eine eigene administrative Domäne auf Ebene 1. Zusätzlich werden auf Ebene 1 beispielsweise notwendige Dienste zur Steuerung der SDN-Komponenten auf Ebene 0 wie einem SDN-Controller sowie beispielsweise Sicherheitsfunktionen wie ein NIDS betrieben, welche ebenfalls in getrennten Domänen organisiert werden. Das SDN der Ebene 0 wird über OpenFlow-fähige virtuelle und physische Netzkomponenten wie dem Open vSwitch (OvS) in der Datenebene und OpenDaylight in der Kontrollebene initial für diese Ebene realisiert.

Vielschichtige Virtualisierung

Des Weiteren tauchen in FSNs und dem beschriebenen Szenario in praktisch beliebigen Schichten FSN-spezifische Komponenten aus den Bereichen SDN oder NFV wieder auf. Beispielsweise befindet sich auf Ebene 3 und, basierend auf Ressourcen aus Ebene 2 aufgesetzt, ein weiterer VIM. Dieser bildet ebenfalls ein eigenes SDN-fähiges virtuelles Netz, welches über einen dedizierten SDN-Controller gesteuert wird. Weitere gleichartige, jedoch heterogene Konstellationen sind auf Ebene 3 und darüber denkbar. Die auf mehreren Ebenen neu entstandenen Ressourcen werden von dem jeweiligen verantwortlichen NMAN dezentral, aber durch die logisch zentralisierte Managementplattform gesteuert, in die neuen Domänen eingeteilt.

Der Zweck von Ebene 1 ist die teilweise Bündelung von Ressourcen der Ebene 0 und der Betrieb der Managementinfrastruktur. Ebene 2 stellt IT-Ressourcen für die einzelnen Arbeitspakete im Projekt bereit. Darüber hinaus dient sie zur Realisierung zentraler Dienste für das Projekt. Ebene 3 und darüber liegende Ebenen stellen einzelne und gemeinsame IT-Infrastrukturen für die Entwicklung und Evaluierung auf Basis von Arbeitspaket-(WP)-Ressourcen bereit.

3.3.1.2 Die Managementinfrastruktur

Verteilte Installation

Im Szenario nutzen die Parteien eine gemeinsame homogene, aber verteilte Managementplattform, die aus mehreren Managementplattforminstanzen besteht. In jedem Datenzentrum wird mindestens eine unabhängige Instanz aufgesetzt, wodurch einerseits eine höhere Ausfallsicherheit geschaffen wird und andererseits geringere Antwortzeiten von Managementplattforminstanzen zur gemanagten Infrastruktur in einem Datenzentrum erreicht werden. Diese Managementplattformen werden jedoch nicht direkt auf Ebene 0, sondern in einer Managementdomäne auf Ebene 1 betrieben, die alle Ressourcen der Managementinfrastruktur umfasst. Rollen und Verantwortlichkeiten diesbezüglich umfassen entsprechend Tätigkeiten zum Management der Managementplattform selbst. Die Einrichtung und Konfiguration der gemeinsamen Managementplattformen muss zentral erfolgen, um mehrfachen gleichartigen Aufwand und damit verbundene Fehleranfälligkeit und Inkompatibilität in der Konfiguration zu vermeiden. Durch einen Zugang zu einer zentralisierten Managementschnittstelle, wie einer breit-erreichbaren Weboberfläche, können die Föderationspartner vorhandene Dienste einsehen und gemäß ihrer Rolle im System alle Dienste verwalten. Ein Parallelbetrieb von Diensten mit inkonsistenten Datenbeständen wird auf diese Weise vermieden.

Selbstkonfiguration

MO-Register

3.3.2 Organisation

In einem Forschungsprojekt mit vielen Partnern haben sich einerseits verschiedene organisatorische Maßnahmen und Strukturen bewährt, andererseits weisen sie bestimmte allgemeine Herausforderungen in der Bereitstellung und Aufrechterhaltung des Dienstbetriebs auf. Im Folgenden werden zum einen Herausforderungen im Netzmanagement, angelehnt an das Projekt SENDATE-PLANETS, sowie eine beispielhafte organisatorische Umsetzung für FSNs erläutert.

3.3.2.1 Projektorganisation und -Struktur

Projekte dieser Größe sind oft hierarchisch in weitere Subprojekte bzw. Teilvorhaben unterteilt, welche sich durch die Zergliederung des Gesamtziels in einzelne komplementäre Teilziele ergeben und denen die kooperierenden Partner, je nach ihren Einzelbeiträgen zur Erreichung der Ziele, ein- oder mehrfach zugeordnet sind. So ist beispielsweise SENDATE-PLANETS in drei Arbeitspakete (engl. *Work Packages* bzw. WPs) mit Fokus auf IT-Sicherheit (WP1), Architektur (WP2) und Vernetzung (WP3) unterteilt [41]. Die Arbeitspakete haben jedoch keine operative Relevanz im Netzmanagement, sondern stellen Arbeitsgruppen mit gemeinsamen Forschungsschwerpunkten dar. Die Zuordnung der Partner zu aus der Aufteilung resultierenden Arbeitsgruppen erfolgt dabei nicht ausschließlich zum Start des Projekts, sondern unter Umständen auch während der Laufzeit, zum Beispiel durch nachträgliche Umstrukturierung der Partner (Hinzufügen, Anpassung oder Entfernen eines Partners und dessen Aufgabenbereichs während der Projektlaufzeit), von Standorten und Datenzentren.

Skalierbarkeit

Genau wie das Gesamtprojekt werden die einzelnen Subprojekte dabei von einem der je zum Arbeitspaket zugehörigen Partner koordiniert. Beispielsweise durch das Ansetzen regelmäßiger sowie nicht planmäßiger Treffen und Besprechungen, der Einforderung, Aufarbeitung und Dokumentation von Statusberichten zur Fortschritts- und Problembeschreibung, der Präsentation gemeinsamer Ergebnisse an die Projektpartner der anderen Subprojekte sowie dem Koordinator des Gesamtprojekts. Auch fällt die Organisation intensivierter multilateraler Zusammenarbeit der Partner (z. B. gemeinsame Testumgebungen und -Fälle und Implementierungen), oder der Mitteilung aktueller Ereignisse wie geplante Konferenzen und Workshops darunter. Die Kommunikation, Absprachen und eine enge Kooperation finden unabhängig von der Koordination durch die (Sub-)Projektleitung statt, sowohl innerhalb von Arbeitsgruppen, als auch arbeitsgruppenübergreifend.

3.3.2.2 Herausforderungen der Organisation

Der Betrieb von Diensten auf freiwilliger Basis bietet insbesondere mehr Flexibilität und ein isoliertes Management der Komponenten voneinander, aufgeteilt auf den jeweils verantwortlichen Partner. Oft findet dieser Ansatz auch im Hintergrund Anwendung im Rahmen einer engen Zusammenarbeit zweier oder mehr Partner innerhalb des Projekts. So stehen individuell betriebene Dienste meist nicht allen Partnern in gleicher Weise zur Verfügung und zentrale Dienste (z. B. Verzeichnisdienste) fehlen, sodass Nutzer für jeden Dienst einzeln registriert und verwaltet werden müssten. Mögliche Umstrukturierungen während der Laufzeit des Projekts, wie personelle Änderungen, münden nicht nur in deutlichem Mehraufwand, sondern stellen auch potenzielle Fehlerquellen dar. Oft werden Dienste aufgrund von fehlender Kenntnis bereits vorhandener Dienste innerhalb eines Projekts ebenfalls parallel von mehreren Partnern realisiert – zum Beispiel mehrere individuell innerhalb der einzelnen Subprojekte betriebene Dateiablagesysteme. Um diesen *Schattenbetrieb* von Diensten im Projekt zu vermeiden, ist eine flexible Organisation hinsichtlich Verantwortlichkeiten und Zugriffsmöglichkeiten notwendig. Nutzer sollen ebenfalls die Möglichkeit bekommen, einzelne IT-Ressourcen oder Teilnetze verwalten zu können.

Flexible Organisation

Herausforderungen treten ebenfalls bei Umstrukturierungen des Projekts auf, beispielsweise bei nachträglicher Zuordnung von bestehenden Partnern zu anderen Subprojekten oder gar

bei Ausscheiden eines Partners aus dem Projekt. Zum einen müssten Zugangsdaten des Partners angepasst werden, zum anderen müssen Dienste, die durch den ausscheidenden Partner selbst betrieben wurden, durch einen anderen Partner ersetzt werden. In FSNs ist das Ausscheiden eines Projektpartners während der Laufzeit des Projekts weniger kritisch; Dienste, die auf Basis virtueller Maschinen oder Container betrieben werden, können problemlos unter Beibehaltung ihres aktuellen Zustands auf einen Host eines anderen Partners migriert werden. Dennoch ist der Betrieb zentraler Dienste in FSNs weiterhin notwendig.

Die konkrete Umsetzung des Managements der föderierten Infrastruktur kann je nach Projekt stark variieren. Üblicherweise jedoch obliegen der Betrieb und das Management der vom jeweiligen Partner selbst direkt eingebrachten IT-Ressourcen. Bei dieser naiven, isolierten Art des Managements entstehen weitere Probleme, beispielsweise kann im Falle eines Ausfalls der Infrastruktur eine Abwesenheit (z. B. durch Urlaub) zuständiger Personen zu Einschränkungen vieler Partner führen, welche auf die Infrastruktur angewiesen sind. Bei der Übertragung dieses isolierten Ansatzes auf eine föderierte IT-Infrastruktur wird die Problematik eines von Partner zu Partner heterogenen Managementansatzes deutlich, insbesondere hinsichtlich der Organisation. Aufgaben, Verantwortlichkeiten, Arbeitsgruppenstrukturen, jedoch auch Managementwerkzeuge sind bei jedem Partner unterschiedlich festgelegt (vgl. Herausforderung *Betriebs- und Managementkonzepte* in Abschnitt 3.1.1.2). Gerade durch die Trennung der Infrastruktur in mehrere Arbeitspakete ist hier arbeitspaketübergreifend Flexibilität in der Umsetzung von Managementkonzepten notwendig, da diese potenziell nicht einheitlich umgesetzt werden kann und beispielsweise von der Art der Nutzung oder personeller Ressourcen abhängig ist. Das Management von FSNs kann im Projekt durch Personen aller Parteien durchgeführt werden.

Heterogene Managementkonzepte

3.3.2.3 Generelle Verantwortlichkeiten und Rollen

Jedes Unternehmen bringt jedoch nicht nur IT-Ressourcen in die Föderation ein, sondern ebenfalls menschliche Ressourcen. Die unterschiedlichen Mitarbeiter jedes Föderationspartners sind in Abbildung 3.2 farblich entsprechend ihrer Zugehörigkeit zu einem bestimmten Partner markiert. Die einzelnen Individuen sind durch ihre Nummer in Kombination mit ihrer Farbe eindeutig kodiert: Beispielsweise beschreiben alle gelbe Figuren mit Beschriftung „2“ denselben Nutzer. Im Folgenden werden einzelne Individuen durch das Abkürzungsschema *Mitarbeiter-Farbe-Zahl* identifiziert.

Die Zugehörigkeit eines Mitarbeiters erstreckt sich üblicherweise auf mehr als eine administrative Domäne; Mitarbeiter sind oft entweder in unterschiedlichen Arbeitsgruppen des Projekts tätig, oder ihr Zuständigkeitsbereich erstreckt sich zusätzlich auf die Administration von IT-Ressourcen oder das Vorhaben unterstützende zentrale Dienste. Die Zugehörigkeit zu administrativen Domänen wird im folgenden Absatz beispielhaft für Partner 1 erläutert.

Im Szenario bringt Partner 1 (gelb) beispielsweise verhältnismäßig viele IT-Ressourcen in das Projekt ein (in Domäne DOM0.1), welche durch Mitarbeiter-Gelb-5 verwaltet werden. Wie beschrieben, haben sich Partner 1 und Partner 2 darauf geeinigt, ihre Ressourcen über ein gemeinsam betriebenes VIM (OpenStack 1) zu verwalten. Domäne DOM1.2 setzt entsprechend auf Ressourcen der Domänen DOM0.1 und DOM0.2 auf – das Management von DOM1.2 wird durch Mitarbeiter von Partner 1 (wiederum Mitarbeiter-Gelb-{1,5}) zusammen mit einem Mitarbeiter von Partner 2 (Mitarbeiter-Blau-4) umgesetzt. Mitarbeiter-Gelb-5 hat entsprechend Aufgaben in zwei Domänen. Partner 1 hat zusätzlich Mitarbeiter, die in unterschiedlichen Arbeitsgruppen des Projekts aktiv sind. Mitarbeiter-Gelb-2 ist beispielsweise in WP1 und verwaltet zudem (genauso wie Mitarbeiter-Gelb-1 zusätzlich) zentrale Dienste des Projekts (DOM2.1). Mitarbeiter-Gelb-3 ist in WP2 tätig (DOM 2.3) und hat aufbauend auf genau diesen Ressourcen eine Menge an persönlichen virtuellen Ressourcen in Domäne DOM3.1 aufgesetzt, deren Management in seine Zuständigkeit fällt. Mitarbeiter-Gelb-3 verarbeitet auf Ressourcen in Domäne DOM3.1 vertrauliche Daten und zielt entsprechend auf möglichst hohe Sicherheit ab. Ressourcen in der Domäne setzen daher (mittelbar über die Domäne DOM2.3, welche wiederum

Mehrfachzuweisung Nutzer zu Domänen

Transparente Ressourcenzugehörigkeit

auf Ressourcen in Domäne DOM1.2 basieren) auf Ressourcen auf, welche von seinem eigenen Unternehmen (Partner 1) gestellt werden. Neben den Tätigkeiten in den einzelnen Arbeitsgruppen kommt es darüber hinaus zu individuellen Kooperationen von Partnern aus unterschiedlichen Arbeitsgruppen. Auch dazu werden neue virtuelle IT-Ressourcen dediziert eingesetzt und auf den Ressourcen des WP realisiert. So ist es beispielsweise in Domäne DOM3.4 der Fall, in der Mitarbeiter-Rot-1 im Rahmen von Tätigkeiten innerhalb des WP3 mit Mitarbeiter-Gelb-2 aus WP1 kooperieren, bspw. bezüglich einer gemeinsamen Systemintegration.

Aspekte des Vertrauens

Für die initiale Einrichtung der Managementplattformen sind mehrere PADM verantwortlich, die zuvor bestimmt werden müssen. Da die Managementdomäne die Ressourcen aller Partner (d. h. Partner 1 bis Partner M) umfasst, ist es sinnvoll, dass jeder Partner einen PADM stellt, der sich jeweils lokal als auch parteienübergreifend um die Plattform kümmert. Die Erstellung eines initialen Satzes an administrativen Domänen, wird hingegen durch NMANs vorgenommen. Sie können gemäß ihres Zuständigkeitsbereichs IT-Ressourcen erstellen und diese (oder auch bereits erstellte Ressourcen) in Domänen untergliedern. Alle NMANs benötigen eine Übersicht über bereits bestehende Domänen und deren eindeutigen Zweck sowie Abhängigkeiten; andernfalls könnten Domänen mehrfach erstellt oder falsch konfiguriert werden. Um die Konsistenz von Zuständigkeiten zu sichern, können NMANs neue Domänen ausschließlich auf bestehenden Domänen aufsetzen, in denen sie die entsprechenden Berechtigungen haben.

Domänenübersicht

Dezentrale Domänenverwaltung

3.3.2.4 Domänenstruktur

Der initiale Satz der administrativen Domänen orientiert sich im Szenario an der in Abschnitt 3.3.2.1 beschriebenen Struktur des Projekts und insbesondere den verschiedenen Arbeitsgruppen. Jede Arbeitsgruppe bekommt initial eine Menge an virtuellen IT-Ressourcen in jeweils einer administrativen Domäne (Domänen DOM2.2, DOM2.3, DOM2.4) zur Verfügung gestellt. Die jeweils verantwortlichen NMANs sind in den Domänen DOM0.1 bis DOM0.M für die Einteilung direkt darauf aufbauender Domänen verantwortlich. Die Koordinierung der Erstellung von Domänen, welche auf mehreren darunterliegenden Domänen basieren, muss entsprechend durch die Plattform koordiniert werden. Generalisiert betrachtet müssen domänenübergreifende Aktionen und Aktivitäten allgemein unterstützt und koordiniert werden. Die den Arbeitsgruppen bereitgestellten virtuellen IT-Ressourcen spannen beabsichtigt über mehrere verteilte und von unterschiedlichen Partnern bereitgestellte IT-Ressourcen. So wird die im Forschungsprojekt notwendige Heterogenität der Systeme und geographische Verteilung erfüllt und eine Unabhängigkeit von einem bestimmten Datenzentrum gewährleistet. Auch werden auf Ebene 2 in einer administrativen Domäne DOM2.1 IT-Ressourcen für die in Abschnitt 3.3.1 beschriebenen zentralen Dienste zusammengefasst.

Koordinierung domänenübergreifender Aktivitäten

Eine Besonderheit in der Domänenstruktur ist in der Überschneidung der Domänen DOM3.3 sowie DOM3.4 gezeigt. Diese betrifft eine virtuelle Ressource (z. B. einen Dienst), an dem WP-übergreifend gearbeitet wird, und die folglich im Zuständigkeitsbereich mehrerer Personen und potenziell unterschiedlicher Aufgabendefinition liegt.

Domänenüberschneidung

3.3.2.5 Managementfunktionen und -Anwendungen

Trotz der Unterteilung des Netzes in mehrere administrative Domänen sind für das Netz als Gesamtes dennoch Managementaufgaben definiert, damit es zweckdienlich ist. Die Föderationspartner einigen sich auf solche Vorgaben und netzweite Strukturen, die von NMANs in den einzelnen administrativen Domänen umgesetzt werden. Funktionale Managementbereiche umfassen entsprechende Aufgaben, durch welche Verantwortlichkeiten definiert werden. Die einzelnen Verantwortlichkeiten werden technisch vor allem durch Managementwerkzeuge unterstützt, die in einem Netzmanagementsystem durch Managementanwendungen gegeben sind. Zuständigkeiten erlauben hier daher den Zugriff von NMANs auf zweckdienliche Managementanwendungen. Auf diese Weise ist ein funktionaler Managementbereich klar definiert. Im Szenario wird daher netzweit die Erfüllung von zwei, aus dem ISO FCAPS-Modell [33] abgelei-

Netzweite Vorgaben

Aufgabenzuweisung

Aufgabenübersicht und Zugriff auf Managementanwendungen

teten Managementaufgaben vorgegeben, die in den einzelnen administrativen Domänen umgesetzt werden müssen:

FB1 Das **Instanzen- und Domänen-Management**, insbesondere die Erstellung, Einrichtung (Konfiguration) und Auflösung von VMs, Netzfunktionen und Domänen, ausgehend von einer konkreten, darunterliegenden Domäne.

FB2 Das **Fehler-, Performanz- und Sicherheitsmanagement**, welche abgeleitet von den gleichnamigen Funktionsbereichen der ISO aus [33] übernommen und aufgrund von Aufgabenüberschneidungen zusammengefasst wurden. Der funktionale Zweck von FB2 besteht in der Sicherstellung eines sicheren und zuverlässigen Netzbetriebs.

Innerhalb des Projekts ist ein Abrechnungsmanagement wie von der ISO vorgeschlagen nicht notwendig. Beispielhafte Managementanwendungen zur Unterstützung des funktionalen Bereichs FB1 sind

- eine netzweite Quasi-Echtzeit Resource-Discovery-Anwendung zum automatischen Erfassen aller MOs im Netz und Dokumentation von Veränderungen,
- eine Domänen-parametrisierbare, VIM- und VIRS-agnostische zentrale Managementanwendung zur Einsicht, Erstellung, Konfiguration und Auflösung von VMs und Containern mit einheitlicher graphischer Nutzerschnittstelle,
- eine Domänen-parametrisierbare, SDN-Controller-agnostische zentrale Managementanwendung zur Steuerung des Netzverkehrs,
- eine Domänen-parametrisierbare Managementanwendung zur Einsicht, Erstellung, Konfiguration und Auflösung von administrativen Domänen,
- und eine Verbindung zu einem Ticketsystem zur Bearbeitung von Nutzeranfragen (z. B. das Einrichten einer neuen VM).

Notwendige Aufgaben im funktionalen Bereich FB2 sind hingegen die Auswertung von Logdaten sowie die Bearbeitung von Störfällen und Sicherheitsfunktionen. Verantwortliche für den Bereich FB2 benötigen für ihre jeweiligen Domänen Zugriff auf diese Daten, genauso wie eine Möglichkeit zur Konfiguration von derartigen Netz-/Sicherheitsfunktionen (z. B. Einrichten einer Firewall-Regel, realisiert über eine SDN-Applikation). Beispielhafte Anwendungen für Verantwortliche des Funktionsbereichs FB2 sind jeweils eine Domänen-parametrisierbare Managementanwendung

- zur Visualisierung von Informationen aus verfügbaren Netz- und Sicherheitsfunktionen und deren jeweilige Konfiguration,
- zur Einrichtung neuer und Entfernung bestehender Sicherheitsfunktionen im Netz,
- zur zentralen Einsicht und Auswertung von Log-Einträgen aus VMs und Containern,
- sowie ein Ticketsystem zur Bearbeitung von Störfällen und Sicherheitsvorfällen.

3.3.2.6 Organisation innerhalb Domänen

In einer Domäne sind oft mehrere NMANs von unterschiedlichen Partnern. Die Aufgabenverteilung innerhalb einer administrativen Domäne muss geeignet gestaltet sein, damit alle funktionalen Managementbereiche und alle damit verbundenen Aufgaben abgedeckt sind, welche netzweit bzw. domänenübergreifend vorgegeben sind. Die im vorherigen Abschnitt beschrie-

Breiter Zugriff auf Managementanwendungen

benen netzweiten Managementanwendungen dienen den Verantwortlichen dabei als wichtiges technisches Mittel bei der Erfüllung ihrer Aufgaben. Folglich müssen netzweite Managementanwendungen von allen Domänen aus erreichbar und nutzbar sein.

Anwendungsbereich von MAPPs

Zugriffsrechte Managementanwendungen

Sichere Anbindung von Managementanwendungen

Jedoch gibt es ebenfalls eine zweite Perspektive auf Managementanwendungen in Multi-Domänen-Umgebungen. Im konkreten Szenario haben sich beispielsweise Mitarbeiter-Rot-1, aus dem WP-Architektur und Mitarbeiter-Gelb-2 aus WP1 zusammengeschlossen, um eine Integration ihrer jeweils im Projekt entwickelten Anwendungen zu testen: Einerseits ein IDS für SDNs und andererseits eine Anwendung zur Korrelation von Informationen aus mehreren Sensoren im Netz, die im Folgenden „SDN-Monitor“ genannt wird. Um weitere Testdaten zu bekommen, ist der SDN-Monitor als Managementanwendung zusätzlich an die Managementplattform angebunden und wertet fingierte Informationen aus zu Testzwecken installierten Sicherheitsfunktionen aus der Domäne DOM3.4 aus. Der SDN-Monitor soll jedoch in erster Linie innerhalb der Domäne DOM3.4 zugreifbar und nutzbar sein und entsprechend keine öffentliche Managementanwendung sein. Um jedoch auch anderen interessierten Projektpartnern die Anwendung des SDN-Monitors zugänglich zu machen, sollen auch Mitarbeiter-Blau-2 und Mitarbeiter-Grün-2 aus WP1 Zugang dazu erhalten. Da so wie Mitarbeiter-Gelb-2 potenziell viele weitere NMANs in einem derartigen Szenario Managementanwendungen entwickeln und mit der Managementplattform koppeln, müssen insbesondere an dieser Stelle geeignete Sicherheitsmaßnahmen ergriffen werden, damit Managementanwendungen nur Funktionen und Informationen verarbeiten können, welche den Rechten und Verantwortlichkeiten des jeweiligen Nutzers der Anwendung entsprechen. Des Weiteren muss die Kommunikation von Managementanwendungen mit der Managementplattform abgesichert werden, da Zugriffe, wie beschrieben, oft über mehrere Domänen und auch über mehrere nicht inhärent abgesicherte Netze verlaufen.

Koordinierte Verwaltung von Domänen

Organisatorische Vorgaben in Domänen

Eine Managementplattform muss jedoch auch Sicherheitsaspekte bei der Verwaltung von Ressourcen einer Domäne berücksichtigen. Im Szenario erstellt beispielsweise Mitarbeiter-Gelb-3 eine neue Domäne DOM3.1 auf Basis von Ressourcen der Domäne DOM2.3. Ein solcher Fall muss durch eine Managementplattform geeignet koordiniert werden. Es muss beispielsweise verhindert werden, dass ein NMAN zu viele Ressourcen unkontrolliert reserviert. Auch muss beispielsweise bestimmt werden, ob Konfigurationen (z. B. das Setzen einer Routing-Regel im SDN) durch dritte NMANs verändert werden dürfen. Zur Koordination der Zuständigkeiten innerhalb einer Domäne gehören beispielsweise auch die Bildung von Gruppen, die Beschreibung von Aufgaben und Funktionen (im Sinne von funktionalen Bereichen des Managements wie z. B. FCAPS) oder auch unterschiedliche Sichten und abrufbare Informationen eines NMAN. Auch ist es sinnvoll, je nach Größe einer Domäne, Rollen zu untergliedern, beispielsweise in einen Domänenmanager und einen Komponentenmanager. Die Unterteilung muss jedoch entsprechend flexibel und je nach Szenario möglich sein.

3.3.3 Bedeutung für das Informationsmodell

Im Szenario sind wesentliche Aspekte zur Modellierung des Netzes, der MOs darin und von Informationen des Netzmanagements, die keinen Zustand eines MOs darstellen, herausgearbeitet worden. Sie werden im Folgenden explizit erläutert.

3.3.3.1 Modellierung von MOs

Erweiterbarkeit Informationsmodell

Berücksichtigung FSN-spezifische MOCs

Die hohe Heterogenität von Netzkomponenten im Szenario (vgl. Abschnitt 3.3.1.1) erfordert die flexible Erweiterbarkeit des Informationsmodells in Hinsicht von Managementobjektklassen (MOCs), Abhängigkeiten und Informationstypen. Von einer MOC sind zudem oft unterschiedliche Produkte im Einsatz. Im Szenario existieren beispielsweise zwei unterschiedliche SDN-Controller, OpenDaylight sowie Floodlight, mit ähnlichem Funktionsumfang, jedoch unterschiedlichen NBI-Ausprägungen. Das NBI von FSN-spezifischen MOs stellt jedoch die wichtigste Schnittstelle zur Managementplattform dar.

- Im Informationsmodell muss daher der abstrakte Typ von MOCs beschreibbar sein (z. B. SDN-Controller).
- MOCs eines Typs müssen alle durch ihn beschriebenen Aktionen bereitstellen (z. B. Installation einer OpenFlow-Regel im Netz),
- können weitere produktspezifische Aktionen definieren und
- müssen parametrisierbare Schnittstellenbeschreibungen (z. B. Verbindungsinformationen wie URLs, SOAP-Template) und Funktionsparameter bieten, genauso wie Rückgabewerte unterstützen.

Typen, Produkte, Aktionen und Schnittstellen von MOs

Die Einführung von MOC-Typen würde praktisch einer Normalisierung von MO-Funktionen entsprechen. Diese ist zwingend notwendig, um eine einheitliche technische Basis des Managements zu schaffen. Ein NMAN ruft aus Managementsicht folglich nicht eine Schnittstelle auf einem MO auf, sondern er führt eine für einen MOC-Typen einheitliche Aktion auf einem MO desselben Typs aus. Wenn eine Zuweisung von MO-Schnittstellen zu MOC-Typ-Aktionen möglich ist, kann beispielsweise auch die in Abschnitt 3.1.2.2 Automatisierbarkeit durch Vereinheitlichung erfolgen.

Normalisierung von MO-Schnittstellen durch MOC-Aktionen

3.3.3.2 Modellierung des Netzes und von Managementbeziehungen

Ein im Szenario wichtiger Aspekt ist die Notwendigkeit zur korrekten Beschreibung von Netzen in FSNs. Das Gesamtnetz besteht praktisch aus dem auf Ebene 0 gezeigten VPN-Netz, welches mehrere geographisch verteilte Datenzentren verbindet. Die geographische Verteilung muss im Informationsmodell berücksichtigt werden, um beispielsweise davon abhängige Performanz- (z. B. durch eine geringe Latenz) und Sicherheitsaspekte – Daten und Dienste bleiben beispielsweise im lokalen Datenzentrum eines Partners – nutzen zu können.

Berücksichtigung geographische Verteilung

Ein weiterer Aspekt, der im Szenario besonders deutlich wird, ist die Berücksichtigung von Abhängigkeiten zwischen MOs in FSNs. Beispielsweise werden einige MOs in den Domänen auf Ebene 2 von einem VIRS in Domäne DOM1.2 realisiert, orchestriert werden sie jedoch von der OpenStack 1-Instanz in derselben Domäne. Folglich muss aber das Management derartigen MOs durch die OpenStack 1-Instanz erfolgen, und nicht durch das darunterliegende VIRS. Das VIRS in Domäne DOM1.2 wiederum ist realisiert auf einem VIRS entweder in Domäne DOM0.1 oder DOM0.2, welche nicht durch ein VIM verwaltet werden. Ähnliches gilt für die SDN-Komponenten im gesamten Netz und allen Domänen. Ihre VM-Instanzen basieren jeweils auf einem VIRS, welche selbst wiederum auf einem anderen VIRS basieren. Zudem kommt eine weitere Abhängigkeit dazu, da SDN-Komponenten jeweils von einem konkreten SDNC gesteuert werden. Derartige Abhängigkeiten müssen im Management in der Konfiguration des Netzes berücksichtigt werden. Ähnliches gilt für Realisierungsabhängigkeiten von MOs (z. B. MO-1 läuft-auf MO-2 läuft-auf MO-3), die durch VIRS und ihre VMs, auch geschachtelt, generiert werden.

Berücksichtigung von Managementbeziehungen untereinander

3.3.3.3 Ereignisse und Zustände

Ein wichtiger Bestandteil des Informationsmodells sind zudem Ereignisse aus Netz- und Sicherheitsfunktionen. Die Informationen können dabei beliebiger Art sein und dürfen nicht durch das Informationsmodell der Managementplattform eingeschränkt werden, sollen aber möglichst wiederverwendet werden können. Eine gewisse Standardisierung ist jedoch Grundlage für andere Anforderungen und insbesondere eine Automatisierbarkeit des Managementprozesses und muss daher nachgelagert stattfinden. Die Normalisierung muss praktisch Prototypen vorsehen, in die ähnliche Informationen aus unterschiedlichen Quellen zusammengefasst werden. Bei der Auswertung von Ereignissen von Netz- und Sicherheitsfunktionen ist die chronologische Auswertung besonders wichtig. Alle Informationen müssen daher mit einem

Normalisierung von Ereignissen und Informationen

Zeitstempel

Zuordnung Ereignisse zu MOs

Zeitstempel versehen sein. Für das Kriterium der Automatisierbarkeit des Managements müssen zudem Abhängigkeiten von Ereignissen oder Zuständen zu MOs berücksichtigt werden. Beispielsweise meldet ein IDS in Domäne DOM1.3 von einem augenscheinlich kompromittierten System mit der IP-Adresse `192.168.55.23`. Aus Sicht des Netzmanagements muss dazu klar sein, zu welchem virtuellen Netz (bei unter Umständen mehrfach vorkommenden Adressbereichen, jedoch voneinander unabhängigen Netzen) und welches konkrete MO darin mit dieser IP-Adresse betroffen ist. Neben der Abhängigkeit der Betroffenheit, ist für die Auswahl geeigneter Maßnahmen unter Umständen die Abhängigkeit der Quelle der Information (z. B. kann die Quelle selbst zur Implementierung von Maßnahmen direkt genutzt werden wie ein SDN-Controller, oder ist ausschließlich passiv wie ein klassisches IDS).

3.3.4 Bedeutung für das Funktionsmodell

In diesem Abschnitt werden entsprechend weniger notwendige Funktionen beschrieben, die zum Management von Szenario 1 erforderlich sind. Der Fokus liegt mehr auf grundlegenden Funktionen, die eine Managementplattform bereitstellen muss, um notwendige Managementanwendungen im Szenario realisieren zu können.

3.3.4.1 Basisfunktionen

Mandantenfähigkeit

Generell wird das Netzmanagement durch viele unterschiedliche Partner und Mandanten ausgeführt. Der Zugriff auf Informationen und Funktionen zum Management der Umgebung muss daher grundlegend mandantenfähig ausgelegt sein. Folglich dürfen über Managementanwendungen nur auf Inhalte der Managementplattform zugegriffen werden, auf die der entsprechende Nutzer auch Berechtigungen hat.

Zustandsoperationen durch MAPPs

Neben einem lesenden Zugriff auf Informationen müssen Managementanwendungen auch in der Lage sein, Informationen auf der Managementplattform zu schreiben. Die Managementplattform an sich bekommt von Seiten der MOs üblicherweise nur einen einfachen Ist-Zustand der Umgebung. Wie bereits in Abschnitt 3.3.2.6 (am Beispiel der fiktiven Software SDN-Monitor) beschrieben wurde, ist jedoch auch die Auswertung und Korrelation von Informationen ein wichtiger Aspekt des Managements. Diese und ähnliche Aufgaben können flexibel durch Managementanwendungen ausgeführt werden, die Plattform muss jedoch auch die Weiterverwendung derartiger Informationen für andere Managementanwendungen ermöglichen.

Weitere Anforderungen des Funktionsmodells

Einige Inhalte, welche prinzipiell dem Funktionsmodell anrechenbar sind, wurden bereits in vorherigen Abschnitten auf Basis der Herleitung aus anderen Bereichen (insb. der Inter- und Intra-Domänenorganisation) geklärt. Dazu zählen die *Selbstkonfigurierbarkeit der Plattform*, die *Übersicht über Domänen*, *netzweite Managementvorgaben*, die *Zuweisung von Aufgaben zu funktionalen Managementbereichen*, eine *aufgabenbasierte Übersicht und Zugriffsbeschränkung auf Managementanwendungen*, die *Anwendungsbereich von MAPPs*, der *autorisierte Zugriff auf Informationen und Funktionen* sowie *domänenspezifische Vorgaben*.

3.3.4.2 Funktionszugriff

Bereitstellung Programmierschnittstelle

Unter dem Aspekt der Nutzbarkeit von Funktionen und Informationen der Managementplattform besteht ihre grundlegendste Funktion in der Bereitstellung einer Programmierschnittstelle (engl. *Application Programming Interface* bzw. API). Im Szenario kommen Nutzer und Entwickler von Managementanwendungen in der Regel aus unterschiedlichen Organisationen ohne gemeinsame Vorgaben der Softwareentwicklung (bzgl. verwendeter Programmiersprachen und -Modelle, Entwicklungsumgebungen, usw.). Daher muss die API eine möglichst breite Nutzung unterstützen. Die API muss zudem innerhalb des FSN über das Netz nutzbar und nicht nur auf lokale Schnittstellen beschränkt sein.

Eine Gemeinsamkeit fast aller Managementanwendungen ist ihr Bezug zu einer administrativen Domäne. Ihre Funktionalität bezieht sich daher oft auf Ressourcen in einer bestimmten Domäne. Eine Managementplattform muss Ressourcen ihren zugehörigen Domänen zuweisen und Mechanismen zur Verfügung stellen, damit berechtigte Nutzer über eine Managementplattform ihre Domänen managen können. Eine weitere Auffälligkeit ist der transparente Zugriff von Managementanwendungen aus auf zentrale Systeme von FSNs (z. B. SDN-Controller, VIRS, usw.). Wie bereits in Abschnitt 3.3.3 beschrieben, soll sich ein Nutzer von Managementanwendungen nicht um die Art der eigentlich ausführenden Komponenten kümmern müssen.

Parametrisierbarkeit nach Domänen

MO-typagnostischer Zugriff auf Funktionen

3.3.5 Bedeutung für das Kommunikationsmodell

Durch das erste Szenario sind wesentliche Anforderungen an die Schnittstellen einer Managementplattform sowie die eigentliche Systemarchitektur erläutert worden. Im Folgenden wird auf beides explizit eingegangen.

3.3.5.1 Schnittstellenanforderungen

Eine Managementplattform hat, wie in Abschnitt 2.4.2 beschrieben wurde, mit dem NBI und dem SBI grundlegend zwei Kommunikationsrichtungen. Beide Richtungen haben eine unterschiedliche Zielsetzung und daher unterschiedliche Anforderungen:

Das **SBI** soll insbesondere hinsichtlich der Anbindbarkeit von Komponenten möglichst flexibel sein, da MOs in FSNs keine einheitliche Schnittstelle bereitstellen. Folglich muss eine Managementplattform für stark heterogene FSNs die Möglichkeit zur Erweiterbarkeit der Schnittstelle zwischen Plattform und MOs ermöglichen. Dazu zählen beispielsweise Schnittstellen der Plattform für Push-basierte Kommunikation ausgehend von MOs, jedoch auch Konnektoren für Pull-basierte Kommunikation, initiiert durch die Plattform selbst. Beispielsweise sind VIMs, SDN- und Virtualisierungssysteme im Szenario meist Pull-basiert, müssen also von der Managementplattform aktiv angefragt werden. Andere Funktionen, wie beispielsweise über SDN-Anwendungen implementierte NIDS oder Firewall, haben hingegen auch einen Push-basierten Charakter, da sie Ereignisse oder Informationen aktiv an die Managementplattform senden. Hier ist insbesondere für Letzteres zusätzlich eine weitere Lösung zur Authentifizierung notwendig, damit nur berechtigte MOs erreichbare Schnittstellen der Managementplattform nutzen und Informationen eintragen können. Andernfalls hätten Angreifer die Möglichkeit zur Manipulation des Netzzustands durch Erstellung gefälschter Informationen sowie den unberechtigten Gebrauch von Speicherressourcen. Viele MOs aus dem Szenario bieten eine HTTP-basierte REST-Schnittstelle (z. B., die SDN-Controller OpenDaylight und Floodlight), welche teilweise über Protokolle zur sicheren Kommunikation wie TLS übertragen werden. Diese sicheren Kommunikationskanäle müssen nutzbar sein.

Flexibilität SBI

Erweiterbarkeit SBI

Push- & Pull-Mechanismen SBI

Authentifizierung am SBI

Verschlüsselung, Integritätsschutz SBI

Das **NBI**, defacto realisiert durch die **API** zwischen der Managementplattform und Managementanwendungen, soll dagegen möglichst breit nutzbar und effizient (d. h. sparsam und performant) sowie sicher gestaltet sein. Die API muss in erster Linie die Funktionen der Managementplattform abbilden können, auch wenn diese bei Bedarf angepasst oder erweitert werden. Entsprechend muss die API erweiterbar sein. Beispielsweise können im Szenario so auch Funktionen (und Informationen) unterschiedlicher NIDS, Firewalls oder anderer Netzkomponenten genutzt werden. Für die Ausführung von Funktionen der Managementplattform selbst wird ein Pull-Mechanismus benötigt. Managementanwendungen greifen entsprechend aktiv auf Informationen und Funktionen der Managementplattform zu. Für eine sparsame Kommunikation ist ein Push-Mechanismus notwendig. Er realisiert einen Benachrichtigungsdienst von Managementplattform zu Managementanwendungen. Eine weitere generelle Maßnahme, die mit der Anforderung der Sparsamkeit der Kommunikation verbunden ist, ist die Verwendung zustandsbehafteter Kommunikation. Die Maßnahme wird am Beispiel einer Managementanwendung zur Korrelation von Sicherheitsereignissen deutlich: Sicherheitsereignisse werden im Szenario von mehreren Sicherheitsfunktionen (z. B. IDS, usw.) sukzessiv gesammelt und

Sparsamkeit der API-Kommunikation

Nutzbarkeit, Effizienz API

Erweiterbarkeit der API

Pull- & Pull-Mechanismus API

müssten entsprechend sukzessiv und gebündelt von der Plattform zur jeweiligen Managementanwendung gesendet werden. Informationen, die im Korrelationsprozess bereits berücksichtigt wurden, müssen nicht erneut ausgetauscht werden, wodurch Netzverkehr eingespart wird.

Zustandsbehaftete und Bulk-Kommunikation

Neben der Art, wie Informationen übertragen werden, spielt auch die Art der Nutzung der API im Szenario eine wichtige Rolle. Entwickler von Managementanwendungen kommen im Szenario potenziell von allen Partnern mit unterschiedlichem Hintergrund. Entsprechend breit muss die API zugreifbar sein, beispielsweise unabhängig von der Benutzung einer bestimmten Programmiersprache. Damit dennoch nur berechtigte Nutzer auf die Managementplattform-API zugreifen dürfen, muss die API Authentifizierungs- und Autorisierungsmechanismen bereitstellen. Auch eine sichere Kommunikation im Sinne der Vertraulichkeit und Integrität ist notwendig. Anders als beim SBI, welche auch stark von den Funktionen der MOs abhängig ist (manche MOs bieten beispielsweise keine TLS-abgesicherte Kommunikation an), ist bei der API die Managementplattform in der Pflicht, die Verwendung entsprechender Protokolle zur Absicherung bei der API durchzusetzen.

Breite Nutzung der API

AuthN, AuthZ am API

Sichere Anbindung von MAPPs

3.3.5.2 Systemarchitektur

Ein anderer wichtiger Aspekt des Kommunikationsmodells ist die Eignung der Systemarchitektur der Managementplattform selbst. Aufgrund der verteilten Installation der Managementplattform im Szenario ist eine Managementplattform als verteiltes System notwendig, das in jedem Datenzentrum als eigene Instanz installiert werden kann, im Verbund aber als System arbeitet. Die jeweils zweckdienliche Architektur der Managementplattform muss für jedes Szenario fallspezifisch umsetzbar sein. Unterschiedliche Architekturen der Managementplattform wirken sich zudem auf die Art der Anbindung der Datenbank aus. Diese kann szenarienspezifisch beispielsweise zentral als eigener Dienst und über das Netz an die Managementplattforminstanzen angebunden sein oder dezentral umgesetzt werden, d. h. lokal für jede Managementplattforminstanz.

Verteilte Installation

Flexible Datenbankanbindung

3.3.6 Einordnung des Szenarios

Die Einordnung des Szenarios ist in Tabelle 3.2 zusammengefasst: Aufgrund der europäisch international verteilten Partner ist die räumliche Dimension überregional und die Dauer mit mehreren Jahren als langfristig einzustufen. Die Koordination ist implizit ohne dedizierte zentrale Instanz; die Gruppenstruktur aufgrund des Eintritts neuer und dem Ausscheiden bestehender Partner in der Praxis flexibel. Das Vertrauensverhältnis zwischen den Partnern untereinander ist gemischt – einige haben bereits miteinander gearbeitet, die meisten hatten zuvor jedoch keine Beziehung zueinander. Die Aufgabenzuordnung, Dynamik der Infrastruktur, Dienste und Organisation, involvierte Personen sowie die Rollenvergabe haben einen dynamischen Charakter mit gleichmäßig untereinander genutzten IT-Ressourcen und Aufgaben und somit überlappenden administrativen Domänen. Die Zielsetzung ist in diesem Fall ebenfalls überlappend – die Föderation wird zum Zweck einer gemeinsamen Zielerreichung genutzt. Da unterschiedliche Partner jeweils verschiedene Systeme in die Föderation einbringen, ist die Zusammensetzung der Infrastruktur stark heterogen. Im Falle des Projekts gibt es keine zentrale Koordinierungsinstanz und entsprechend keine globalen oder hierarchischen, sondern viele unabhängige Domänen. Die von den Partnern eingebrachte Infrastruktur wird symmetrisch von allen Partnern genutzt.

3.4 Szenario 2: SNs als Wegbereiter der Digitalisierung in der Medizin

SNs gelten bereits seit ihrem Aufkommen als Wegbereiter-Technologien und als wichtige Teillösung für eine Digitalisierung in unterschiedlichen Geschäftsfeldern und Anwendungsfällen. Dazu zählen beispielsweise autonomes Fahren, das *Internet of Things* (IoT) und die Vernetzung

Charakteristikum	Ausprägung in FSNs		
Räumliche Dimension	regional	überregional	gemischt
Dauer	kurzfristig		langfristig
Koordination	implizit		explizit
Gruppenstruktur	flexibel		fest
Vertrauen	indirekt	direkt	gemischt
Aufgabenzuordnung	statisch		dynamisch
Dynamik	statisch		dynamisch
Rollenvergabe	statisch		dynamisch
Relation zw. administrativen Domänen	überlappend		disjunkt
Zielsetzung	überlappend	komplementär	gemischt
Domänenstruktur	global	hierarchisch	unabhängig
Art der Infrastrukturnutzung	symmetrisch		asymmetrisch

Tabelle 3.2: Einordnung von Szenario 1.

von Industrieanlagen, jedoch auch die zunehmende Digitalisierung in der Medizin [42]. In diesem Abschnitt wird beispielhaft der Einsatz von SNs zur Digitalisierung und Automatisierung eines Teilbereichs der *Telemedizin* betrachtet (auch als *E-Health* bezeichnet, vgl. auch im Glossar des Bundesministeriums für Gesundheit [43]). Sie adressiert eine remote Patientenüberwachung und -Versorgung. Telemedizin beschreibt laut der Definition der Bundesärztekammer die „Gesundheitsversorgung der Bevölkerung in den Bereichen Diagnostik, Therapie und Rehabilitation sowie bei der ärztlichen Entscheidungsberatung über räumliche Entfernungen (oder zeitlichen Versatz) hinweg“ [44]. Fortschritte in der Vernetzung medizinischer Geräte und einer automatisierten Auswertung von Gesundheitsdaten können beispielsweise auch in kritischen Situationen und schwierigen Umgebungen hilfreich sein: Zum Beispiel in ländlicheren Gegenden mit karger medizinischer Versorgung (im Sinne von aus Patientensicht schnell erreichbaren medizinischen Einrichtungen) oder auch in medizinischen Notfallsituationen, in denen eine möglichst zeitnahe Handlungsreaktion von Bedeutung ist [45].

3.4.1 Relevanz softwarebasierter Netze in der Telemedizin

Aspekte softwarebasierter Netze sind bereits jetzt schon Teil der Infrastruktur und von Konzepten, um medizinische Anwendungen zu realisieren. In [45] gehen die Autoren beispielsweise generell auf Anwendungsfälle von Cloud-Computing in der Telemedizin ein. So kann Cloud-Computing durch eine geeignete Infrastruktur beispielsweise für die Archivierung, Auswertung und den Zugriff auf Bilddaten (resultierend aus bildgebenden Verfahren wie z. B. der Magnetresonanztomographie und Sonographie), oder als Basis zur Teleberatung (z. B. via Telekonferenzen zwischen Patient und Arzt) genutzt werden. Auch wird remote Patientenüberwachung beschrieben, in der basierend auf Cloud-Computing-Lösungen Vital- und Physiologiedaten ausgewertet werden.

Eine konkretere Cloud-gestützte Architektur zur Sammlung und Verarbeitung von Informationen aus dem Patientenmonitoring wird in [46] beschrieben. Demnach sind für den Anwendungsfall der Datenverarbeitung in der Telemedizin ebenfalls Herausforderungen anzutreffen, wie sie auch in FSNs vorhanden sind – starke Heterogenität von Systemen, Protokollen und APIs, der Bedarf an einer möglichst einfachen Konfigurierbarkeit der Komponenten, einer flexiblen Nutzung und hohen Skalierbarkeit, sowie das Management der Komponenten. Eine Cloud-Infrastruktur bietet über Virtualisierung mehr Flexibilität und ermöglicht die automatisierte Verarbeitung aggregierter Informationen; außerdem bietet sie eine breit verfügbare (in der Regel webbasierte) Einsichtsmöglichkeit für medizinisches Personal.

Ebenfalls gibt es Ansätze für Patientenmonitoring und -Überwachung, die über Cloud-Computing hinausgehen. So beschreiben die Autoren in [47] auch Anwendungen für sogenanntes *Fog*- und *Edge*-Computing. Darin werden die Daten nicht ausnahmslos zentral in einem üblicherweise weit entfernen Rechenzentrum, sondern teilweise nah oder direkt an der Datenquelle ausgewertet. Sich in der Praxis dafür eignende Geräte sind beispielsweise der Sensor (z. B. *smarte* Systeme zur Blutglukoseüberwachung für Diabetes-Patienten), mobile private Geräte des Patienten (insb. Smartphones und Tablet-PCs), oder andere private IT-Systeme im lokalen Netz. Eine Auswertung von Patientenüberwachungsdaten nah am Patienten hat den Vorteil, dass eine Übertragung der Daten vermieden werden kann und entsprechende Geräte so auch für Patienten in Gegenden mit schlechter Netzanbindung einsetzbar sind. Entsprechend wird das System zuverlässiger, da seine Funktionsfähigkeit nicht von vielen Komponenten (z. B. Netzkomponenten und Cloud-Dienste) abhängig ist. Daher ist es für kritische Systeme geeignet, unter anderem für sogenannte *Closed-Loop-Systeme*, in denen eine geeignete automatisierte Reaktion auf Basis der zuvor erhobenen Messdaten folgt. Ein Beispiel dafür ist eine Closed-Loop-Insulinpumpe, welche dem Patienten automatisch eine geeignete Menge Insulin aufgrund unmittelbar zuvor gemessener Gesundheitsdaten (z. B. Blutglukosespiegel) verabreicht.

Da SNs und seine Paradigmen bereits einen wichtigen Teil der Infrastruktur von Cloud-, Fog- und Edge-Computing ausmachen, ist es üblicherweise mit gewissen Ausprägungen bereits in die genannten Anwendungen integriert. Das folgende Szenario orientiert sich an den beschriebenen Beispielen, legt darin jedoch einen verstärkten Fokus auf Ausprägungen und Chancen von SNs. Des Weiteren werden Föderationsaspekte der unterschiedlichen, im Szenario kooperierenden Parteien, sowie damit verbundene Besonderheiten im Management der Infrastruktur berücksichtigt.

3.4.2 Föderationsbeziehungen und Aufgaben

Der Fokus dieses konkreten Anwendungsfalls liegt auf der Beschreibung einer sich auf FSNs stützenden Infrastruktur zur flexiblen, zuverlässigen und sicheren Patientenüberwachung und Patientenversorgung aus der Ferne. Sie erlaubt die Protokollierung und Auswertung von Vitaldaten und die aktive Beeinflussung der Gesundheit und physiologischer Funktionen von Patienten mittels z. B. Herzschrittmachern, Hörgeräten oder auch Insulinpumpen. In die Bereitstellung und Nutzung der Infrastruktur sind üblicherweise unterschiedliche Parteien involviert, mindestens Patienten sowie eine medizinische Einrichtung zur Analyse der Vitaldaten und entsprechender Therapierung der Patienten. Jedoch sind auch komplexere Kooperationsstrukturen möglich und oft sinnvoll. So ist es durchaus realistisch, dass eine medizinische Einrichtung zur Analyse der Vitaldaten von Patienten nicht nur auf eigene IT-Infrastruktur setzt, sondern für diese Aufgabe ebenfalls wie im Szenario in Abschnitt 3.5 zusätzlich bedarfsabhängig weitere Rechenkapazitäten durch einen Hostinganbieter in Anspruch nimmt. Ebenfalls ist die Bereitstellung und Wartung für spezialisierte medizinische Geräte am Patienten und darauf installierten Anwendungen ein wichtiger Aspekt. Beispielsweise sind temporär eingesetzte medizinische Geräte für Patienten aufgrund hoher Kosten nicht geeignet finanzierbar. Für diese Aspekte gibt es teilweise Leasing-Verträge für derartige Geräte, welche insbesondere die Finanzierung ermöglichen (vgl. z. B. [48]).

Eine Zusammenfassung der Föderationspartner für dieses Szenario sowie deren Abhängigkeiten ist in Abbildung 3.3 illustriert. Da neue und sich technologisch auf dem neuesten Stand befindliche Ansätze und Maßnahmen insbesondere durch ein **Universitätsklinikum** eingesetzt werden, bildet es in diesem Szenario die zentrale medizinische Einrichtung.

Die Rolle des **Patienten** bleibt hingegen entsprechend generisch. Für das Szenario wird jedoch angenommen, dass sein Gesundheitszustand durch eine Erkrankung (z. B. Diabetes Typ 1) eingeschränkt ist und einer ständigen Überwachung sowie Behandlung bedarf. Die Behandlung und potenziell notwendige Therapie (z. B. mit Insulin) wird in Form eines medizinischen Gerätes (**MedG**) am Patienten kurzfristig und dynamisch an die fortlaufend gemessenen Vitaldaten

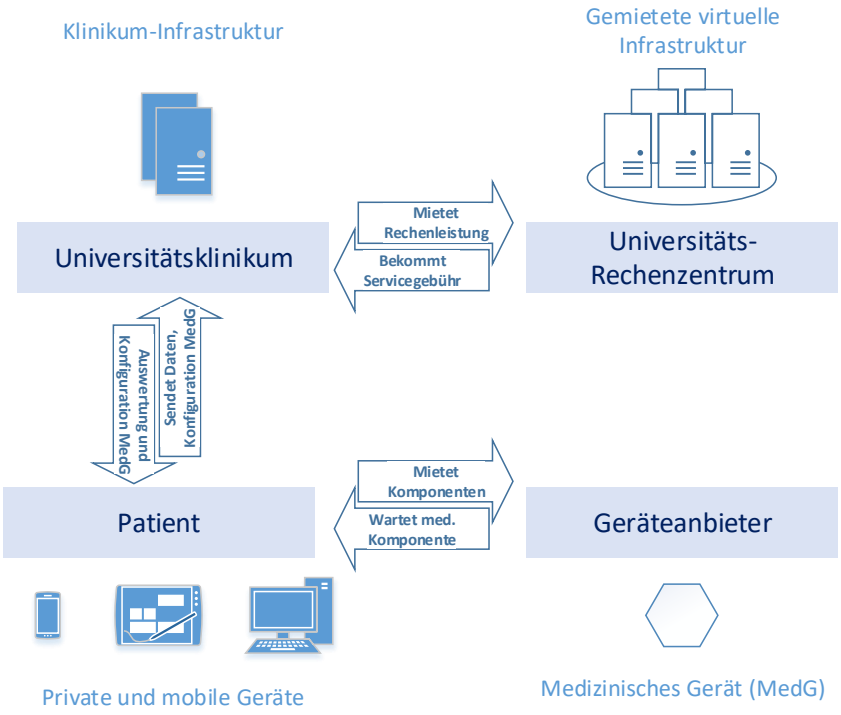


Abbildung 3.3: Beziehungen und generelle Infrastruktur der Föderationspartner bei SNs in medizinischen Anwendungen.

(z. B. Blutglukosegehalt) angepasst. Unterschiedliche Patienten benötigen folglich unterschiedliche MedGs zur Überwachung und Behandlung.

Die Beziehung zwischen Universitätsklinikum und Patient besteht hinsichtlich der eingesetzten Infrastruktur insofern einerseits aus der Bereitstellung von Vitaldaten des Patienten (gesammelt über das MedG) an das Universitätsklinikum, und andererseits aus der Aufgabe des Universitätsklinikums, die übermittelten Daten fortlaufend zu analysieren und gegebenenfalls die Konfiguration des MedG entsprechend einer zustandsabhängig geeignetsten Therapie anzupassen. Darüber hinaus kann ein Patient selbst seine Daten über einen Dienst des Universitätsklinikums einsehen und das MedG in gewissem Rahmen konfigurieren.

Die Aspekte der Bereitstellung und Wartung des MedG werden in diesem Szenario durch die jeweiligen **Geräteanbieter** übernommen. Der Patient bekommt sein MedG über einen Leasingvertrag. Gleichzeitig werden wichtige Softwareupdates sowie technische Störfälle des Gerätes durch den Geräteanbieter behandelt – entsprechend garantiert er für die Zuverlässigkeit und Sicherheit des MedG.

Aspekte des Vertrauens

Die Anmietung des Universitätsklinikums von weiteren Rechenkapazitäten hat einige Vorteile, wie zum Beispiel die Auslagerung gewisser Netz- und System-Managementaufgaben und eine bedarfsabhängige Kostenoptimierung. In diesem Szenario wird davon ausgegangen, dass die IT-Infrastruktur des Universitätsklinikums und der entsprechenden Universität teilweise von einem **Universitätsrechenzentrum** bereitgestellt und gewartet werden. Entsprechend kann das Universitätsklinikum auf der vorhandenen Vertrauensbasis aufbauend weitere Dienste kostenpflichtig buchen. Das Universitätsrechenzentrum dient in diesem Fall als regional ansässiger Hosting-Provider für das Universitätsklinikum. Das Klinikum mietet je nach Bedarf die notwendige physische oder virtuelle Infrastruktur gemäß einem Service-Level-Agreement und darin zugesicherter Dienstgüte, beispielsweise bezogen auf die für den Anwendungsfall

besonders wesentliche Verfügbarkeit der Infrastruktur. Das Universitätsrechenzentrum stellt die Ressourcen gegen eine abgestimmte Gebühr zur Verfügung. Wie am Beispiel des Leibniz-Rechenzentrums, dem Universitätsrechenzentrum der Münchener Universitäten, zu sehen, sind auch im Hochschul- und wissenschaftlichen Bereich derartige Dienste verfügbar [49].

Aufgrund dessen, dass sich der Kern der Föderation, das Universitätsklinikum und das Universitätsrechenzentrum, üblicherweise in direkter Umgebung (in der Regel derselben Stadt bzw. desselben Landkreises) befinden, sowie dass in der Praxis die medizinischen Komponenten am Patienten, das MedG, immer durch den Geräteanbieter oder einen geeigneten Vertreter auch physisch zugreifbar sein muss (z. B. im Falle eines Austauschs der Komponente), wird das Szenario als regional betrachtet.

3.4.3 IT-Infrastruktur

Die im Szenario kooperierenden Partner stellen jeweils eigene IT-Infrastrukturen bereit, die zu einer zusammenwirkenden föderierten Infrastruktur zusammengeschlossen wird. Die Geräte der Partner, Aspekte von SNs sowie betriebene Dienste bilden insofern praktisch den Kern des Dienstes der Patientenüberwachung und -Versorgung.

3.4.3.1 Überblick über die Infrastruktur der Föderation

Die grundlegend von jeder Partei innerhalb der Föderation bereitgestellte Infrastruktur ist in Abbildung 3.3 illustriert. Das Universitätsklinikum betreibt ein eigenes Klinikums-Rechenzentrum, um einerseits Dienste zur Unterstützung des alltäglichen medizinischen Betriebs bereitzustellen, und andererseits um einen gewissen Anteil der aggregierten Daten aus der Patientenüberwachung selbst und unabhängig von entfernten Ressourcen analysieren zu können. Das bietet einerseits einen gewissen Grad an Sicherheit, eine höhere Dienstverfügbarkeit gegenüber Netzausfällen zum Universitätsrechenzentrum, andererseits ist dennoch über die externen bedarfsorientiert beanspruchten Ressourcen eine Skalierbarkeit in Hinblick auf zusätzliche Patienten und Daten möglich.

Neben den bereits erwähnten Komponenten – das Universitätsrechenzentrum stellt physische oder virtuelle Ressourcen für die erforderliche Rechen- und Speicherleistung bereit, der Geräteanbieter liefert die medizinischen Komponenten für Patienten – bringt auch ein Patient an sich weitere Komponenten in die föderierte softwarebasierte Infrastruktur ein. Für medizinische Anwendungen ist die von Patienten eingebrachte Infrastruktur vor allem für die Konfiguration des MedG sowie zur Einsicht der vom MedG bereitgestellten Informationen von Bedeutung. Üblicherweise haben die meisten privaten Geräte Internetzugang, beispielsweise über lokale Netze oder das Mobilkommunikationsnetz. Dafür besonders von Relevanz sind beispielsweise private Smartphones und Tablet-PCs, Notebooks oder Desktop-PCs. Patienten-Anpassungen an der Konfiguration des MedG werden über eine entsprechende Anwendung des jeweiligen Patienten auf seinen privaten Geräten durchgeführt, welche auf den dafür vorgesehenen Dienst am Klinikums-Rechenzentrum zugreift. Zudem hat ein Patient die Möglichkeit, seine eigenen über das MedG gesammelten Gesundheitsdaten sowie Systeminformationen (beispielsweise Batteriestatus, Verbindungsstatus und Verbindungsqualität, Fehler und Störungen, usw.) einzusehen. Dazu bekommt er eine für diese Zwecke mit entsprechenden Diensten ausgestattete direkt nutzbare virtuelle Maschine mit eigenen spezialisierten Schnittstellen, den *Health-Monitor*, welchen er auf seinem Heimrechner installieren kann. Über entsprechende Applikationen für mobile Geräte können Patienten zudem ihre Vitaldaten vom *Health-Monitor* auslesen und Einstellungen anpassen.

Das MedG selbst wird auf Basis eines Mikrocontrollers oder Einplatinencomputers aufgebaut, welcher entweder direkt ohne dazwischenliegendes Betriebssystem oder aber mit spezialisierten aber freien Betriebssystemen wie Linux oder BSD ihre Funktion erfüllen. In diesem Fall wird angenommen, dass die Basis für das MedG ein Einplatinencomputer (wie beispielswei-

se ein ARM-basierter *Raspberry Pi*) mit einem zu diesem Zweck optimierten Linux-Kernel ist. Das MedG kommuniziert nicht über ein privates Patientengerät wie ein Smartphone, das als Hub genutzt wird, sondern direkt über ein integriertes GMS-Modul über das Mobilkommunikationsnetz mit der Krankenhaus-Infrastruktur. Die Kommunikation zum Rechenzentrum des Klinikums findet also – sowohl vom MedG als auch von privaten Patientengeräten – jeweils stets über öffentliche Internet-Infrastruktur statt.

3.4.3.2 Unterstützung durch ein FSN

Die Umsetzung der Infrastruktur des Szenarios als SN bietet einige Vorteile, um insbesondere die *a)* Flexibilität, die *b)* Zuverlässigkeit und *c)* die Managebarkeit des Netzes, der Dienste und Komponenten zu unterstützen. Abbildung 3.4 hebt Aspekte föderierter softwarebasierter Netze in der Infrastruktur des Szenarios hervor.

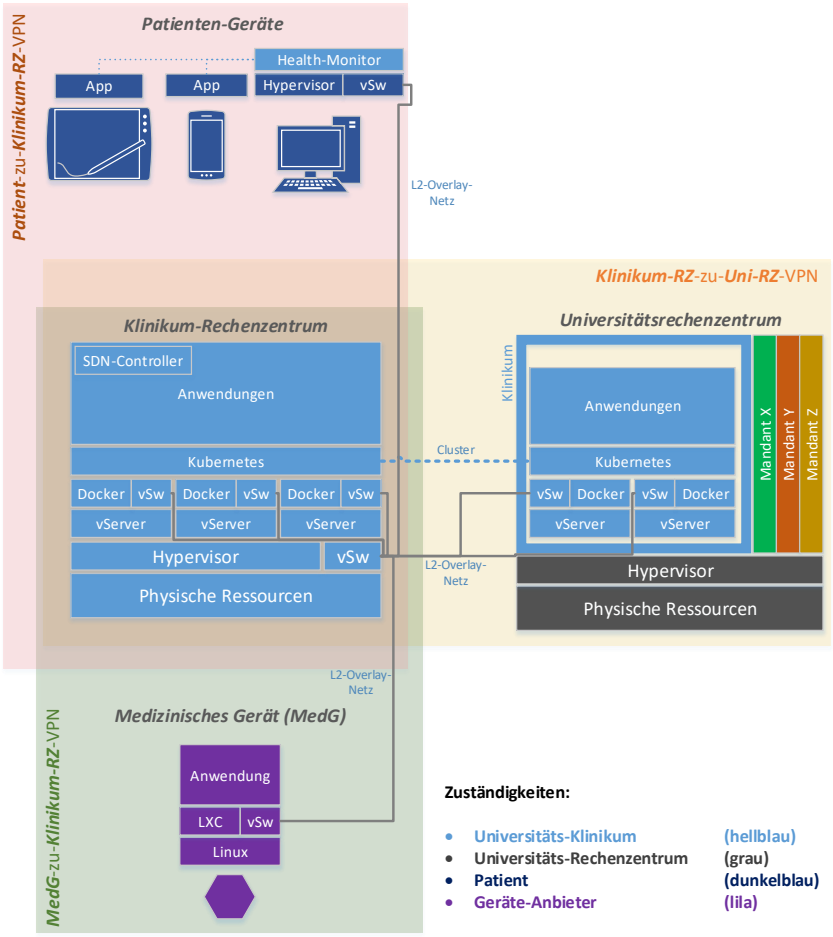


Abbildung 3.4: FSN-Infrastruktur und Komponenten am Beispiel eines konkreten Patienten.

Die Infrastruktur setzt sich aus IT-Ressourcen von unterschiedlichen Standorten zusammen. Im Konkreten sind das die jeweiligen Standorte des Universitätsklinikums, des Universitätsrechenzentrums, sowie des jeweiligen Patienten selbst und seiner privaten Geräte. Die unterschiedlichen verteilten Komponenten sind daher über jeweils ein eigenes VPN mit dem Standort des Universitätsklinikums als zentralem Knoten verbunden. Da die Einsicht der Informationen des MedG sowie dessen Konfiguration zentral über einen Dienst des Klinikums umgesetzt werden, sind Patienten-Geräte und das MedG in separaten VPNs vernetzt. Die Verwendung von getrennten VPNs erlaubt dabei einerseits die Trennung der Komponenten untereinander – so

können Patientengeräte nicht direkt auf ihr MedG zugreifen – andererseits bieten sie einen abgesicherten Tunnel, um eine sichere Kommunikation über öffentliche Internet-Infrastruktur zu realisieren.

Das Universitätsklinikum stellt für den Anwendungsfall der Patientenüberwachung bzw. -Versorgung eine Teilmenge seiner insgesamt im internen Rechenzentrum verfügbaren physischen Ressourcen bereit. Für eine feingranularere Einteilung der physischen Ressourcen werden sie über einen Hypervisor (z. B. KVM, d. h. entsprechend durch sog. Hardwarevirtualisierung) in mehrere virtuelle Ressourcen eingeteilt, welche zunächst unabhängig und isoliert voneinander virtuelle Server darstellen. Um die Skalierbarkeit der Anwendung zu vereinfachen, wird auf jedem virtuellen Server zudem ein System zur Containervirtualisierung (z. B. *Docker*) aufgesetzt. Alle Docker-Instanzen auf den jeweiligen virtuellen Servern werden daraufhin in ein gemeinsames *Kubernetes*-Cluster zusammengefügt. So kann das Kubernetes-Cluster bei Bedarf durch weitere Docker-Instanzen auf weiteren virtuellen Servern vergrößert werden – insbesondere auch über die im Universitätsrechenzentrum für den Anwendungsfall gemieteten Rechenkapazitäten.

Entsprechend würde dem Universitätsklinikum ein standortübergreifendes Kubernetes-Cluster zur Verfügung stehen, das je nach Bedarf weiter vergrößert, aber auch verkleinert werden kann. Durch die zentrale Verwaltung der insgesamt verfügbaren Ressourcen werden Dienste und Anwendungen auf dem Kubernetes-Cluster aufgesetzt und je nach Verfügbarkeit auf die IT-Ressourcen verteilt. Anwendungen können an bestimmte Ressourcen (insb. Docker-Instanzen) gebunden werden, so dass sie beispielsweise ausschließlich auf internen Systemen des Klinikums ausgeführt werden.

Auch das MedG basiert in diesem Fall auf Allzweck-Hardware – die eigentliche jeweilige Funktion des MedG wird auf einem Linux-basierten Kernel als virtualisierte Anwendung mittels Containervirtualisierung aufgebaut. Auf dem MedG wird eine zweckmäßigere Variante zur Containervirtualisierung über die *Linux-Container* (LXC) realisiert. Das bietet in der Praxis ähnliche Vorteile im Management wie durch *Network Functions Virtualization*: Die Funktion des jeweiligen MedG kann entsprechend der notwendigen Überwachung und Versorgung des Patienten so unter anderem

- als Komplettpaket inklusive Laufzeitumgebung, logisch getrennt von anderen (beispielsweise System-) Anwendungen, betrieben werden;
- als von der konkreten physischen Plattform unabhängige virtuelle Netzkomponente realisiert und behandelt werden;
- im Zuge von Softwareupgrades mit abgestimmter Laufzeitumgebung einfach und remote ausgetauscht werden;
- in ihrem jeweils aktuellen Zustand gesichert und wieder eingespielt werden oder als Fallback-Mechanismus verwendet werden können (z. B. bei fehlerhaften Updates).

Ähnliche Vorteile ergeben sich auch für den dem jeweiligen Patienten als virtuelle Instanz bereitgestellten *Health-Monitor*, welcher auf dem Heim-PC des Patienten läuft.

Der Netzbetrieb wird im Szenario 2 durch den Einsatz von SDN-fähigen und auf OpenFlow-basierenden Netzkomponenten (insbesondere Switches) gestützt. Zu diesem Zweck ist einerseits jeder (eigene und gemietete) virtuelle Server des Klinikum-Rechenzentrums mit jeweils einem virtuellen SDN-Switch (*vSw*) wie den *Open vSwitch* verbunden, andererseits sind innerhalb jedes virtuellen Servers alle Softwarecontainer ebenfalls jeweils über einen virtuellen Switch verbunden. Alle virtuellen Switches sind über einen Managementkanal mit einem zentralen OpenFlow-Controller verbunden. Der OpenFlow-Controller wird auch direkt auf dem

Kubernetes-Cluster als virtuelles System realisiert. Über diesen kann das Netz und die virtuellen Netzkomponenten mittels u. a. OpenFlow zentral gesteuert und überwacht werden. Beispielsweise wird der Netzverkehr für Daten aus dem MedG und andere kritische Dienste über eine Priorisierung bevorzugt behandelt.

Damit der im Szenario eingesetzte SDN-Controller ebenfalls die Topologie der installierten Switches erfassen und im Monitoring sowie der Steuerung des Netzverkehrs ausnutzen kann, ist über den virtuellen Switch ein Overlay-Netz auf Basis von VXLAN aufgesetzt. Entsprechend gibt es eine direkte Layer-2-Schicht-Verbindung zwischen allen virtuellen Switches. Generell ist es zudem denkbar und sinnvoll, für jeden Mandanten (in diesem Fall jedes MedG pro Patient) eine eigenes VXLAN zu definieren. Der SDN-Controller selbst ist als virtuelle Anwendung realisiert.

3.4.4 Organisation

Die Organisation des Szenarios unterscheidet sich wesentlich von der des vorherigen Szenarios. Sie wird stärker zentralisiert im Universitätsklinikum umgesetzt und ist weniger dynamisch. Auf die wesentlichsten Aspekte wird in den folgenden Abschnitten näher eingegangen.

3.4.4.1 Inter-Domänen-Organisation

Die Domänenstruktur von Szenario 2 ist in Abbildung 3.5 abgebildet. Sie zeichnet sich durch eine starke Ungleichverteilung von Parteien der unterschiedlichen Stakeholdergruppen aus. Das Universitätsklinikum sowie -Rechenzentrum sind in dem Szenario zentral aufgestellt. Demgegenüber gibt es mehrere Patienten (1 bis P) und Geräteanbieter (1 bis G). Manche Patienten werden voraussichtlich MedGs desselben Anbieters verwenden, andere wiederum die neuer Anbieter. Für MedG $G1$-GA (vgl. Domäne DOM0.3.G) bezeichnet G den Index des Geräteanbieters und A die Anzahl der von Geräteanbieter G angebotenen MedGs.

Auf darüberliegenden Ebenen 1 bis 3 ist dagegen eine Homogenität der Nutzer (verdeutlicht über die entsprechende farbliche Kodierung in Abbildung 3.5) in den einzelnen Domänen gegeben. Das Universitätsklinikum als zentraler Dienstbetreiber ist dafür zuständig, dass der angebotene Dienst der Patientenüberwachung und -Versorgung zuverlässig funktioniert. Entsprechend bestehen keine direkten Überschneidungen von administrativen Domänen und Parteien, jedoch indirekte vertikale Überschneidungen. Ein ineffektives Netzmanagement in Domäne DOM0.2 durch das Rechenzentrum beeinträchtigt beispielsweise die Zuverlässigkeit des gesamten Dienstes, da das darauf teilweise basierende *Kubernetes* und darüberliegende Anwendungen davon abhängig sind. Generell ist das Management des in Szenario 2 aufgespannten FSN zentral charakterisiert.

Im Szenario gibt es im föderationsweiten Netzmanagement unterschiedliche Ziele und daraus abgeleitete Verantwortlichkeiten:

- **Zuverlässigkeit der Dienst-Infrastruktur**. Die vom Universitätsklinikum für den Dienst der Patientenüberwachung und -Versorgung genutzte Infrastruktur (insbesondere Domänen DOM0.1, DOM1.1, DOM2.1, DOM3.1 sowie DOM3.2) muss zuverlässig funktionieren. Dazu gehört auch das standortübergreifende Kubernetes. Hierfür ist das **Universitätsklinikum** zuständig.

- **Dienst-Netz-Überwachung und -Konfiguration**. Das den Dienst unterstützende Netz, das VPN und das Overlay-Netz, müssen hinsichtlich eines korrekten und sicheren Betriebs überwacht und konfiguriert werden. Beispielsweise muss sichergestellt werden, dass die virtuellen Netzkomponenten zuverlässig und korrekt funktionieren. Auch hierfür ist das **Universitätsklinikum** zuständig.

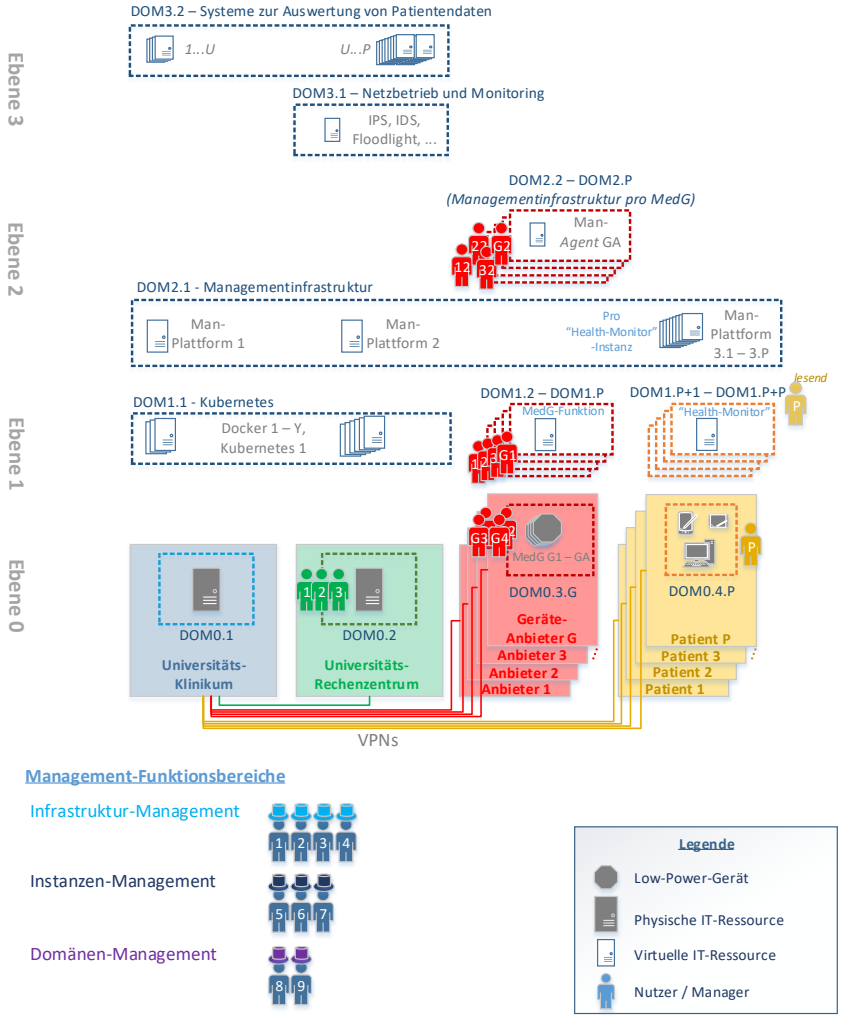


Abbildung 3.5: Domänenstruktur in Szenario 2.

- **Zuverlässigkeit und Sicherheit des Health-Monitors**. Die den Patienten bereitgestellte Anwendung als sofort nutzbare VM, der „Health-Monitor“, muss zuverlässig funktionieren und sicher umgesetzt sein. Das **Universitätsklinikum** als Anbieter ist für das Management dieser virtuellen Instanzen beim Patienten verantwortlich (Domänen DOM1.P+1 – DOM1.P+P).

- **Zuverlässigkeit der erweiterten Ressourcen**. Es muss sichergestellt werden, dass die vom Universitätsklinikum gemieteten Ressourcen (Domäne DOM0.2) bei Bedarf verfügbar sind und zuverlässig funktionieren. Als Anbieter muss das **Universitätsrechenzentrum** die Verfügbarkeit der Ressourcen sicherstellen.

- **Funktion des MedG**. Nicht nur die physische Plattform des MedG (d. h. ein Einplatinencomputer), sondern auch das darauf installierte Betriebssystem, Systemdienste und schlussendlich die Funktionalität des MedG muss zuverlässig funktionieren. Wie bereits beschrieben wurde, ist der **Geräteanbieter** für die generelle Funktionalität des MedG zuständig. Für die Verwaltung der MedG-Funktion **jedes MedGs** (**nicht** pro Geräteanbieter) ist auf Ebene 1 eine eigene Domäne (Domäne DOM1.2 – DOM1.P) aufgespannt

mit genau einem MedG pro Patient, die vom jeweiligen Anbieter umgesetzt wird. Die vom Universitätsklinikum verwaltete Managementplattform muss dem Geräteanbieter Funktionen zum Management der einzelnen MedGs bereitstellen.

- **Funktion Patienten-Geräte**. Auch die vom Patienten genutzten privaten Geräte, welche er zum Einsehen von Informationen und zur Konfiguration des MedG nutzt, müssen zuverlässig funktionieren. Das Management der privaten Geräte obliegt dem **Patienten** selbst.

Bereits hier wird deutlich, dass sich die Erfüllung der Ziele in diesem Szenario trotz gemeinsam genutzter Ressourcen voneinander trennen lassen. Überschneidungen administrativer Bereiche der Föderationspartner gibt es daher nicht und jeder Föderationspartner kann unabhängig voneinander handeln. Dabei besteht jedoch die Gefahr, dass unabgestimmte Änderungen den gesamten Dienst gefährden. Beispielsweise könnte das Universitätsrechenzentrum ein ungetestetes instabiles Update des verwendeten Hypervisors installieren, welches einen Ausfall des gesamten Dienstes nach sich zieht. Auch müssten Softwareupdates der MedG-Funktion selbst durch einen Geräteanbieter koordiniert sein, damit Patienten sich auf eine mögliche kurze Beeinträchtigung ihres MedG einstellen können und ebenfalls das Universitätsklinikum dies berücksichtigen kann.

3.4.4.2 Managementfunktionen und Rollen

Die Managementfunktionen in Szenario 2 haben einen anderen Schwerpunkt als beispielsweise im Szenario 1, wodurch sich ebenfalls andere Strukturen diesbezüglich ergeben:

FB1 **Infrastruktur-Management** bezogen auf die Funktionsfähigkeit im Sinne einer effektiven Konfiguration und ausreichender Verfügbarkeit von Ressourcen des Netzes. Im Zentrum steht hier das standortübergreifende Kubernetes-Orchestrierungssystem.

FB2 **Instanzen-Management** im Sinne der Konfiguration und Bereitstellung als auch Instandhaltung der eigentlichen Container und VMs, in denen die Anwendungen zur Auswertung von Gesundheitsinformationen und Steuerung der MedGs, aber auch Instanzen der Managementplattform, ausgeführt werden. Zusätzlich fallen die beim Patienten betriebenen Instanzen des Health-Monitors in diesen Funktionsbereich.

FB3 **Fehler-, Performanz- und Sicherheitsmanagement**. Wie auch in Szenario 1 ist im eigentlichen föderierten Netz kein Abrechnungsmanagement notwendig. Ein Abrechnungsmanagement muss allerdings vom Universitätsrechenzentrum durchgeführt werden, jedoch nicht als Teil des FSN. Auch Geräteanbieter werden für Wartung und Geräte-Vermietung (an Patienten) ein Abrechnungsmanagement benötigen, jedoch ebenfalls getrennt von der föderierten Infrastruktur und lediglich bezogen auf die jeweiligen MedGs.

FB4 **Domänen-Management**, welches aufgrund seiner Hintergrundfunktion im Szenario 2 ausgelagert wurde: Die manuelle Erstellung von Domänen ist nur eingeschränkt notwendig und kann für das Anlegen neuer oder dem Entfernen bestehender Geräteanbieter und Patienten im FSN einheitlich erfolgen.

An den funktionalen Bereichen orientieren sich die netzweit definierten Rollen der NMAN. Im föderierten Netz des Szenarios gibt es daher die Rolle des **Infrastrukturmanagers**, welcher die Aufgaben aus FB1 im Netz übernimmt, die des **Instanzen-Managers** für Aufgaben aus FB2, sowie **Domänenmanager** für FB4. Aufgaben aus FB3 werden jeweils für ihren Geltungsbereich und ihre Domänen von den Infrastrukturmanagern und Instanzen-Managern übernommen. Wie beschrieben kann ein Patient jedoch ausgewählte Netz- und Systeminformationen seines MedGs ausschließlich lesend einsehen (vgl. Abschnitt 3.4.3.1). Zu diesem Zweck wird im

Mandantenfähigkeit

Rolle	Beschreibung
Infrastrukturmanager	Erfüllt Aufgaben des Infrastruktur-Managements.
Instanzen-Manager	Erfüllt Aufgaben des Instanzen-Managements.
Domänenmanager	Verwaltung von Domänen im FSN.
Patient	Passive Rolle eines Patienten mit Zugriff auf ausgewählte Informationen und Funktionen des Managements.
MedG-Manager	Rolle der Geräteanbieter im FSN; Verwaltung der Medizinischen Geräte und ihrer Funktion.

Tabelle 3.3: Rollen im Netzmanagement in Szenario 2.

Management die Rolle **Patient** angelegt. Die Managementplattform muss diese funktionalen Bereiche und Rollen abbilden können. Des Weiteren wird zum Zweck der Managementaufgaben der Geräteanbieter – Management der jew. MedG-Funktion – die Rolle **MedG-Manager** (zuständig für Aufgaben des FB1 und FB2, aber ausschließlich beschränkt auf die MedGs) eingeführt. Eine Übersicht über alle Rollen ist Tabelle 3.3 gezeigt.

Durch die relativ eindeutige Aufgabentrennung sind in Abbildung 3.5 die zuständigen Personen nicht direkt den Domänen zugeordnet. Jede Person in einer Rolle, dargestellt mit *Hüten* auf den Personen und farblich passender Domänen-Eingrenzung, übernimmt dieselben Aufgaben in jeder Domäne. Beispielsweise kümmern sich alle Infrastrukturmanager (Mitarbeiter-Blau-{1-4}) gleichermaßen um die Aufgaben in den Domänen DOM0.1, DOM1.1, DOM2.1 sowie DOM3.1. Alle Instanzen-Manager (Mitarbeiter-Blau-{5-7}) kümmern sich um die Domänen DOM1.$P+1$ – DOM1.$P+P$. Domänenmanager sind nicht in klassische Domänen aufteilbar, sie verwalten die Struktur der Domänen (insb. DOM0.3.1 bis DOM0.3.G, DOM0.4.1 bis DOM0.4.P, DOM1.2 bis DOM1.P, DOM1.$P+1$ bis DOM1.$P+P$ sowie DOM2.2 bis DOM2.P). Die einzelnen Managementanwendungen, die zum Management der Umgebung von Szenario 2 notwendig sind, werden in Abschnitt 3.4.6 im Rahmen des Funktionsmodells beschrieben.

3.4.4.3 Managementinfrastruktur

Durch die verteilte Infrastruktur ist eine dezentral installierte Managementplattform notwendig. So wird eine Instanz im Rechenzentrum des Universitätsklinikums, eine weitere auf den gemieteten Ressourcen des Universitätsrechenzentrums und eine direkt in die den Patienten gegebenen Health-Monitor-Instanz integriert aufgesetzt.

Eine Besonderheit für das Management stellen die **MedG-Systeme** dar. Sie gehören zum Zuständigkeitsbereich des jeweiligen Geräteanbieters und sind insofern ein essenzieller Teil der Dienstinfrastruktur. Sie werden jedoch nicht (wie praktisch alle anderen Komponenten) zentral durch das Universitätsklinikum gemanagt. Ein Management der Netzfunktionen auf einem MedG, die Virtualisierungslösung und der virtuelle Switch, sind dennoch notwendig. Entsprechend gibt es in diesem Fall die Lösung, dass einerseits die Schnittstellen der Netzfunktionen von remote angesprochen werden und andererseits, dass zudem in Kooperation zwischen Klinikum und Geräteanbieter eine Managementplattform oder Agenten-Instanz in jedes MedG integriert wird. Letztere Variante geht mit mehr Vorteilen einher, so können beispielsweise Informationen und Ereignisse bei einem Verbindungsabbruch mit dem mobilen Gerät zwischengespeichert werden bzw. einfache Managementaufgaben automatisch auch ohne Netzverbindung ausgeführt werden. Daher wird im Szenario 2 diese Variante mit Einführung des Akteurs des Managementagenten verfolgt. Die Verantwortlichkeit für den Betrieb der Managementinfrastruktur auf den einzelnen MedG-Systemen liegt jedoch beim entsprechenden Geräteanbieter. Auch setzen MedG-Systeme auf einem Einplatinencomputer mit stark begrenzten Rechen-

Managementagenten

und Speicherressourcen auf. Auf ihnen können daher keine klassischen Managementplattform-Instanzen ausgeführt werden. Das hat insbesondere für das szenarienspezifische Kommunikationsmodell Auswirkungen, wie in Abschnitt 3.4.7 beschrieben wird.

Leichtgewichtigkeit der Plattform

Ähnliches gilt ebenfalls für Instanzen der Health-Monitore. Patienten haben oft (wenn überhaupt) PCs mit stark begrenzten Speicher- und Rechenressourcen und in der Regel sind private Geräte nicht mehr auf dem neuesten Stand der Technik. Auch sind diese Systeme alles andere als einheitlich, wodurch eine Managementplattform im herkömmlichen Sinne weitestgehend *plattformunabhängig* sein muss, d. h. auf gängigen Anwendersystemen nutzbar sein muss. Im Falle des Health-Monitors als *Trägersystem* für die Managementplattform betrifft das entsprechend vor allem auch genau diesen. Auch muss die Einrichtung des Health-Monitors und der darauf installierten Managementplattforminstanz sofort *Out-of-the-Box* und ohne komplexen Installationsprozess funktionieren. Auch bringen private Geräte eine weitere Sicherheitsanforderung des Integritäts- bzw. Manipulationsschutzes der Plattform mit sich, welcher trotz breitem Zugriff auf Plattform-Instanzen gewährleistet sein muss.

Plattformunabhängigkeit

Selbstkonfiguration

Manipulationsschutz

3.4.4.4 Organisation innerhalb Domänen

Die Zuteilung von Personen zu einzelnen Domänen ist im Fall des Szenario 2 nicht abhängig von ihrer Zugehörigkeit zu einer bestimmten Gruppe oder einem bestimmten Partner, sondern vielmehr an ihre jeweilige Funktion gebunden. Für die Infrastrukturmanager sind besonders die Domänen DOM0.1, DOM1.1, DOM2.1 und DOM3.1 von Bedeutung, für Instanzen-Manager hingegen die Aufgabenerfüllung in den Domänen DOM1.P+1 – DOM1.$P+P$ sowie die Domäne DOM3.2. Die Domänenmanager agieren nicht direkt in einer bestimmten Domäne, sondern sie verwalten die Einteilung von Ressourcen in neue Domänen bzw. die Auflösung bestehender Domänen. Auch die Aufgaben der Geräteanbieter und der Patienten sind klar festgelegt, ohne direkte Aufgabenüberschneidungen. Die Aufgabenverteilung wird dadurch, wie bereits beschrieben, deutlich homogener. Es gibt keine direkten Überschneidungen administrativer Domänen, sondern vor allem vertikale Abhängigkeiten, bedingt durch Virtualisierung der Infrastruktur.

Funktionsorientierte Zuweisung von Domänen

3.4.5 Bedeutung für das Informationsmodell

Wesentliche Unterschiede im Vergleich zu Szenario 1 ergeben sich für das Informationsmodell des Managements dieses Szenarios grob aus der Struktur der Vernetzung der Partner, der generierten und verarbeiteten Informationen, MOCs und MO-Typen, -Abhängigkeiten sowie -Funktionen.

3.4.5.1 Managementobjekte und Managementbeziehungen

Anders als in Szenario 1 befinden sich nicht alle Partner des FSN in einem gemeinsamen Netz und sind untereinander nicht vollständig verbunden. Wie bereits in vorherigen Abschnitten beschrieben stellt das Universitätsklinikum im Szenario eine zentrale Stelle dar, jede andere Partei ist einzeln über ein separates VPN damit verbunden. Dennoch gibt es im Domänenmodell (vgl. Abbildung 3.5) vertikale Abhängigkeiten, wie es leicht für Domäne DOM2.1 am Beispiel der einzelnen Patienten erkennbar ist. Die Managementplattform in jedem *Health-Monitor* ist von der Verfügbarkeit der privaten Host-Geräte des jeweiligen Patienten abhängig. Ein Ausfall betrifft immer auch den jeweiligen Health-Monitor.

Abhängigkeiten in Domänen auf Basis der Partnervernetzung

Besondere MOs in Szenario 2 stellen die medizinischen Geräte zur Patientenüberwachung und -Versorgung. Sie haben durch ihre Anbindung über das Mobilfunknetz die Eigenschaft, dass sie häufiger die Verbindung zum FSN verlieren und wiederaufnehmen als herkömmliche Server-Systeme in geographisch verteilten Datenzentren. Beim Dienst der Patientenversorgung ist es

Wiedererkennung MOs, Zustandswiederherstellung

notwendig, dass das Informationsmodell die Wiedererkennung von getrennten und wiederverbundenen MOs ermöglicht und den Zustand des Netzes sowie den des besagten MOs vor dem Verbindungsabbruch mit dem möglicherweise neuen Zustand des wiederverbundenen MOs zusammenführt.

MOC-Typen

Auch sind MedG in verschiedenen Typen vertreten. Zwar kann angenommen werden, dass MedGs auf Basis von Einplatinencomputern realisiert sind, jedoch kann zum einen die konkrete Hardware und das Betriebssystem, die Virtualisierungslösung, die virtualisierte Funktion und der damit verbundene Zweck eines MedGs variieren. Entsprechend können MedGs je nach ihrer Ausprägung Schnittstellen, Funktionen, generierte Informationen teilen, jedoch auch neue einführen. Entsprechend ist auch hier eine Einteilung nach MOC-Typen und Subtypen sinnvoll, beispielsweise ebenfalls nach genereller Funktion (z. B. *Automatisiertes Insulin-Abgabesystem*, Elektrokardiogramm), unter der Annahme, dass sich MedGs entsprechend ihrem Zweck in ihrem Funktionsumfang und Informationen ähneln. Ebenfalls zu berücksichtigen sind Instanzen des Health-Monitors auf den privaten Geräten der Patienten. Der Health-Monitor ist praktisch ein Gesamtsystem aus Dienst-Funktionalität für Patienten (d. h. das Auslesen von Gesundheitsinformationen sowie die beschränkte Konfigurierbarkeit des jeweiligen MedGs) und bereits lauffertiger Managementplattforminstanz. Ein Health-Monitor ist für genau ein MedG zuständig. Auch stellen die in Domäne DOM3.2 betriebenen virtuellen Instanzen zur Auswertung von aus den MedG gesammelten Gesundheitsdaten eine Besonderheit dar. Eine Instanz kapselt genau eine Anwendung zur Auswertung der Gesundheitsinformationen von genau einem MedG.

Dienst- und Managementbeziehungen

Abhängigkeiten von Diensten und Komponenten im Netzmanagement, auch über die bereits in Szenario 1 aufgezeigte klassischen Zusammenhänge in FSNs hinaus, werden so erkennbar. Ein Health-Monitor sowie eine Anwendung zur Auswertung von Gesundheitsdaten (welche wiederum auf orchestrierten IT-Ressourcen basiert) hängen von einem MedG ab. Eine Managementplattform muss solche szenarienabhängigen Abhängigkeiten abbilden und darstellen können, um beispielsweise die Auswirkung von Problemen und Vorfällen in der Infrastruktur auf die Funktionalität von Diensten sowie Gegenmaßnahmen aufzeigen zu können. Anders als eine Realisierungsabhängigkeit, wie sie bei VIRS zu VMs auftritt oder einer Steuerungsabhängigkeit, wie sie bei SDN-Controller zu SDN-Infrastruktur besteht, beschreibt diese Abhängigkeit mehr eine Dienstabhängigkeit im Allgemeinen. Managementbeziehungen müssen daher szenarienspezifisch definiert werden können.

3.4.5.2 Ereignisse und Zustände

Zusätzliche Ereignisse und Status

Auch Informationen wie Ereignisse und Zustandsdaten variieren im Gegensatz zu anderen Szenarien. Beispielsweise sind relevante Informationen bei mobilen MedGs zusätzliche Details wie aktuelle Akkuleistung und erwartete Restlaufzeit eines Geräts. Im Rahmen des Netzmanagements kann so beispielsweise unterschieden werden, ob ein MedG wegen mangelnder Stromversorgung, einem Verbindungsabbruch oder einem möglichen schwerwiegenden Softwarefehler nicht mehr erreichbar ist. Auch kann beispielsweise der maximale Datendurchsatz bei der Überwachung mobiler Geräte über das Mobilfunknetz im Management berücksichtigt werden. Ist beispielsweise nur eine minimale Bandbreite verfügbar (z. B. über GPRS), kann ein priorisierter Austausch von Informationen den wichtigen Produktivdaten (die Gesundheitsdaten des Patienten) die Zuverlässigkeit des Dienstes erhöhen. Beispielsweise werden nur Fehler- und Warnmeldungen an die zentrale Managementplattform gesendet, jedoch keine kritischen Status- oder Diagnostikmeldungen.

Priorisierter Austausch von Monitoring-Informationen

3.4.6 Bedeutung für das Funktionsmodell

Im Funktionsmodell unterscheidet sich Szenario 2 ebenfalls von Szenario 1. Insbesondere auf Managementfunktionen und notwendige Managementanwendungen, aber auch Eigenschaften des Funktionszugriffs wird im Folgenden eingegangen.

3.4.6.1 Managementfunktionen und Managementanwendungen

Zum Netzmanagement sind in Szenario 2 unterschiedliche Rollen und unterschiedliche Managementanwendungen notwendig. Anders als in Szenario 1 sind die Aufgaben des Netzmanagements jedoch relativ klar durch die definierten Domänen (vgl. Abbildung 3.5) getrennt, wodurch die Nutzung derselben Managementanwendungen durch unterschiedliche Partner praktisch nicht notwendig ist. Generell werden für die Aufgaben in den unterschiedlichen Funktionsbereichen unterschiedliche Managementanwendungen benötigt.

Für das **Infrastruktur-Management** (und folglich einen Infrastrukturmanager) muss es jeweils eine Managementanwendung geben, die

- den Status des zentralen Kubernetes als Gesamtsicht darstellt und Konfigurationsänderungen erlaubt;
- Informationen der Überwachung der physischen bzw. virtuellen Hostsysteme, inkl. Hypervisor (in Domäne DOM0.1), sowie deren Konfiguration erlauben;
- Funktionen der Netzfunktionen und SDN-Switches zur Überwachung und Konfiguration des Netzes anbieten.
- eine Zuordnung der zusammenhängenden Infrastruktur zwischen MedGs, Health-Monitors, Containern und Netzfunktionen zur Informationsauswertung (insbesondere zur Fehler- und Problem-Behandlung im Infrastruktur-Management) aufzeigen.
- Informationen und Funktionen aus netzweiten Sicherheitsfunktionen (insb. Domäne DOM3.1) anbieten (ebenfalls im Rahmen der Problembehandlung).
- Push-Benachrichtigungen über Probleme und Fehler im Netz an alle Infrastrukturmanager sendet.

Push-Benachrichtigungen

Das **Instanzen-Management** benötigt wiederum Managementanwendungen, die es erlauben,

- einzelne Container basierend auf der API von Kubernetes zu überwachen und zu steuern;
- Instanzen des Health-Monitors zu überwachen und zu steuern;
- eine Zuordnung von jeweils zusammenhängenden MedGs, Health-Monitors und Containern zur Informationsauswertung (insbesondere zur Fehler- und Problem-Behandlung im Instanzen-Management) aufzuzeigen;
- Instanzen-Logs mit unterschiedlichem Log-Level (z. B. *Fehler*, *Warnung*, *Information*) für alle Instanzen darzustellen;

Alarme

- per Push-Benachrichtigungen bzgl. Probleme und Fehler in virtuellen Instanzen an alle Instanzen-Manager sendet.

Im **Domänen-Management** sind hingegen Managementanwendungen mit folgender Funktionalität notwendig:

- Die Zuweisung und das Entfernen von Zusammenhängen zwischen Geräteanbieter, MedGs und jeweiligen Managementagenten.
- Die Zuweisung und das Entfernen von Zusammenhängen zwischen Patienten, MedGs und jeweiligen Health-Monitor-Instanzen.

Die Rolle **Patient** bekommt im Rahmen des Netzmanagements eine Managementanwendung zur

- Einsicht ausgewählter Informationen bzgl. Konnektivität, Datenverbrauch, Datendurchsatz seines Health-Monitors sowie MedGs,
- sowie die Möglichkeit, sich in Quasi-Echtzeit Push-Benachrichtigungen bzgl. Überwachungs- und Wartungsinformationen (z. B. Verbindungsabbruch oder geplanten Updates) schicken zu lassen.

Echtzeitbenachrichtigungen

3.4.6.2 Funktionszugriff

Anders als beispielsweise in Szenario 1 ist die Nutzung von Managementanwendungen im Kontext verschiedener Domänen in Szenario 2 kaum ausgeprägt. Die Managementanwendungen sollen vielmehr eine netzweite Perspektive darstellen.

Domänenübergreifende Informationen für Managementanwendungen

Neben den aufgezählten Managementanwendungen, welche mehr als Werkzeuge des Netzmanagements benutzt werden können, sind weitere im Kontext von Automatisierungsaspekten sinnvoll. Das FSN in Szenario 2 zeichnet sich auch durch die große Anzahl an Partnern aus, bei vergleichsweise wenig IT-Personal (z. B. im Universitätsklinikum), das Aufgaben aus dem Netzmanagement wahrnimmt. Bei Problemen sind jedoch kurze Reaktionszeiten notwendig, um den Dienst der Patientenüberwachung und -Versorgung und damit die Gesundheit von Patienten nicht zu beeinträchtigen. Auch ist es von besonderer Bedeutung, dass Überwachungs- und vor allem Steuerungsmechanismen unabhängig von schwankenden Bedingungen der Netzverbindung sein müssen: Automatisierte, für Patienten überlebenswichtige Aufgaben, beispielsweise zur Aufrechterhaltung der Funktion eines MedG und seine Konfiguration müssen auch ohne Verbindung zum zentralen Klinikum-Rechenzentrum möglich sein. So sind notwendige Managementanwendungen, welche auch automatisch auf spezifische Vorfälle reagieren und / oder betroffene Nutzergruppen darüber informieren zum Beispiel die Folgenden:

Netzunabhängige Steuerungsmechanismen Automatisierung

- Ein **Backup-Dienst**, welcher in periodisch kurzen Abständen ein Back-Up aller wichtigen Container und VMs, Anwendungen, virtueller Netzkomponenten und Managementplattforminstanzen erstellt.
- Der **Infra-Monitor** überwacht die Verfügbarkeit des Overlay-Netzes zwischen Klinikum und Universitätsrechenzentrum. Bei einem längeren (im Bereich von Minuten) Verbindungsabbruch übernimmt die Anwendung die automatische Wiederherstellung aller zu dem Zeitpunkt isolierten Container $U..P$ auf Ressourcen des Universitätsrechenzentrums (Domäne DOM3.2) auf den lokalen Ressourcen des Klinikums. So wird der Dienst im Falle eines Teilausfalls weiterbetrieben. Bei Wiederherstellung der Verbindung werden die Daten synchronisiert und der verteilte Normalbetrieb wiederhergestellt.
- Der **Netzpolizist** ist eine Managementanwendung, welche auf Meldungen des in Domäne DOM3.1 betriebenen IDS arbeitet und kompromittierte Systeme unmittelbar nach Auffälligkeit über den ebenfalls in Domäne DOM3.1 betriebenen SDN-Controller netzweit sperrt.
- Der **MedG-Verfügbarkeitsmonitor** überwacht die Konnektivität der einzelnen MedGs zum FSN. Bei zu langer Trennung eines MedG (z. B. ein Tag) wird automatisch der jeweilige Patient, als auch der zuständige Geräteanbieter beispielsweise per E-Mail und SMS informiert.
- Der **MedG-Anomaliedetektor** überwacht die Funktionsweise des Health-Monitors sowie des MedG auf sicherheitskritische Anomalien wie Änderungen von Dateiprüfsummen oder Netzverkehr. Bei Anomalien veranlasst die Anwendung die Rücksetzung auf ei-

nen vertrauenswürdigen Zustand. Die Anwendung muss hochzuverlässig und daher lokal an die in den Health-Monitor integrierte Managementplattforminstanz angebunden sein.

Netzunabhängigkeit der API

- Der **Health-Monitor-Monitor** überwacht die Konnektivität der Instanzen des Health-Monitors und benachrichtigt bei zu langer Trennung (z. B. ab wenigen Tagen) eines Health-Monitors vom FSN den jeweiligen Patienten sowie das Instanzen-Management-Team des Universitätsklinikums.

- Der **Partner-Monitor** ist eine Managementanwendung, welche die Konnektivität des Universitätsklinikums zu allen Partnern im FSN überwacht. Bei einem Verbindungsabbruch werden das Klinikum sowie der betroffene Partner per E-Mail informiert.

Der Fokus des Funktionsmodells im Szenario 2 liegt deutlich mehr auf der Automatisierung von Abläufen sowie der Maßnahmenergreifung.

3.4.7 Bedeutung für das Kommunikationsmodell

Das Kommunikationsmodell unterscheidet sich in Szenario 2 ebenfalls in einigen Punkten von den anderen Szenarien. Die Vernetzung der Partner ist sehr zentral orientiert, mit dem Universitätsklinikum als verbindenden Knoten des Netzes.

3.4.7.1 Systemarchitektur

Entsprechend müssen bei der Koordination der Instanzen der Managementplattform untereinander ebenfalls Informationsaustauschmodelle berücksichtigt werden. In dem vorliegenden Fall bietet es sich beispielsweise an, die Koordinationsoperationen ebenfalls zentralisiert auszurichten (z. B. Hub-and-Spoke-Modell, d. h. *Man-Plattform 1* in Domäne DOM2.1 als Hub nutzen) – ein stark dezentraler Peer-To-Peer-Ansatz (jede Instanz verteilt jede aggregierte/generierte Information an alle anderen Instanzen) dagegen wäre möglicherweise kontraproduktiv hinsichtlich der Netzbelastung. Die Vernetzung der Partner kann sich jedoch auch auf die zentrale Datenbank einer geeigneten Managementplattform auswirken. In Szenario 2 können beispielsweise eine Managementplattform mit zentraler Datenbank (d. h. zentral installiertem Datenbanksystem) Vorteile und vor allem auch Vereinfachungen mit sich bringen.

Informationsaustauschmodelle

Zentrale Datenbank

Auch wird in Szenario 2 die Möglichkeit der Nutzung von Agentensystemen auf einzelnen dafür geeigneten MOs (hier jedes MedG) eingeführt. Dadurch ändert sich nicht nur das Organisationsmodell (neue Rolle *Agent*), sondern auch das Kommunikationsmodell. Es legt die Kommunikation zwischen Managementplattform und Agent fest. Die Aufgabe des **Agenten** besteht hauptsächlich in der Sammlung und Zwischenspeicherung von Monitoring-Informationen aus sowie die Ausführung von Aktionen und Operationen des Managements auf den einzelnen MedGs. Die MedGs sind einzeln über ein VPN (und darauf beispielsweise VXLAN) mit dem Universitätsklinikum verbunden, wodurch die Kommunikation der Instanz Man-Plattform 1 mit dem jeweiligen Managementagenten notwendig ist. Die Steuerung erfolgt über die den Agenten nächstgelegene Managementplattforminstanz.

3.4.7.2 Eigenschaften der Kommunikation

Die Anbindung zwischen Managementplattform und dem jeweiligen Managementagenten der MedGs muss – trotz Einzelanbindung über ein VPN – zum Schutz sensibler Funktionen und Informationen über eine individuell abgesicherte Leitung (z. B. via TLS) realisiert werden. Eine Authentifizierung genauso wie Verschlüsselung und Integritätsschutz ist zwingend erforderlich. Der Agent verwaltet ein MO (ein MedG) und wird entsprechend zur gemanagten Umgebung gezählt. Die Anbindung erfolgt folglich über das SBI der Managementplattform.

Sicherheit SBI

Flexibilität SBI

Da SN-Komponenten auf MedGs (wie in Abschnitt 3.4.3.2 beschrieben) weitestgehend aus handelsüblichen Produkten bestehen, muss ein Agent die Schnittstellen dieser ansprechen können (ähnlich wie jede Instanz der Managementplattform zu den anderen MO ohne Agenten). Die Kommunikation zwischen Managementplattform und Agenten können hingegen durchaus auf Individuallösungen in Protokollen, Syntax und Semantik basieren und beispielsweise unnötigen Overhead von Mehrzweckansätzen wie HTTP, STIX und TAXII (vgl. Abschnitt 4.2.5.1) vermeiden, um die (ressourcentechnisch) *teure* Kommunikation über das Mobilfunknetz besser auszunutzen.

3.4.8 Einordnung des Szenarios

Wie bereits kurz in Abschnitt 3.4.2 beschrieben, wird die räumliche Dimension des regionalen Kerns der Föderation eines Universitätsklinikums, des Universitätsrechenzentrums und des Patienten dementsprechend eingestuft. Die Dauer der Föderation wird ausgehend vom Therapiezeitraum von Patienten als langfristig angesehen. Die Koordination besitzt durch das Universitätsklinikum als zentrale Stelle bzgl. Therapie und Diensterbringung einen expliziten Charakter. In dem speziellen Fall ist die Gruppenstruktur eher fest – dies ist jedoch anwendungsfallabhängig und beispielsweise für andere Fälle der Patientenüberwachung und -Behandlung möglicherweise auch flexibel. Das Vertrauen ist üblicherweise direkt; die Partner kennen sich in der Regel persönlich. Die Aufgabenzuordnung ebenso wie die Dynamik und Rollenvergabe ist in dem Fall statisch. Aufgaben überschneiden sich nicht, wodurch auch die Relation zwischen administrativen Domänen zueinander disjunkt ist. Die Zielsetzung ist gemischt: Das Klinikum und der Patient verfolgen in der Regel dasselbe Ziel, die Beziehung zwischen Klinikum und Universitätsrechenzentrum ist dagegen ein Anbieter-Kunden-Verhältnis mit komplementärem Ziel. Die Domänenstruktur ist global und insbesondere von Vorgaben aus dem Universitätsklinikum abhängig. Die föderierte Infrastruktur dient praktisch exklusiv dem Dienst der Patientenüberwachung und -Behandlung und ist daher praktisch asynchron und durch das Klinikum genutzt. Eine Übersicht der Erklärung ist in Tabelle 3.4 zusammengestellt.

Charakteristikum	Ausprägung in FSNs		
Räumliche Dimension	**regional**	überregional	gemischt
Dauer	kurzfristig	**langfristig**	
Koordination	implizit	**explizit**	
Gruppenstruktur	flexibel	**fest**	
Vertrauen	indirekt	**direkt**	gemischt
Aufgabenzuordnung	**statisch**	dynamisch	
Dynamik	**statisch**	dynamisch	
Rollenvergabe	**statisch**	dynamisch	
Relation zw. administrativen Domänen	überlappend	**disjunkt**	
Zielsetzung	überlappend	komplementär	**gemischt**
Domänenstruktur	**global**	hierarchisch	unabhängig
Art der Infrastrukturnutzung	symmetrisch	**asymmetrisch**	

Tabelle 3.4: Einordnung von Szenario 2.

3.5 Szenario 3: Bedarfsabhängige Erweiterung von IT-Ressourcen

Das dritte Szenario betrachtet eine ausschließliche Kunden-Anbieter-Konstellation am Beispiel der auch heutzutage besonders im Kontext von Cloud-Computing stark genutzten Mietung von IT-Ressourcen als Dienstleitung. Aspekte der IT-Infrastruktur spielen eine nachrangige Rolle

in diesem Szenario. Ausführliche Beispiele dazu werden in den vorherigen Szenarien beschrieben – Eigenschaften darin können auch auf dieses Szenario übertragen werden. Eine Illustration der wichtigsten Aspekte im Szenario wird in Abbildung 3.6 gezeigt, auf die sich die Beschreibungen der folgenden Abschnitte beziehen.

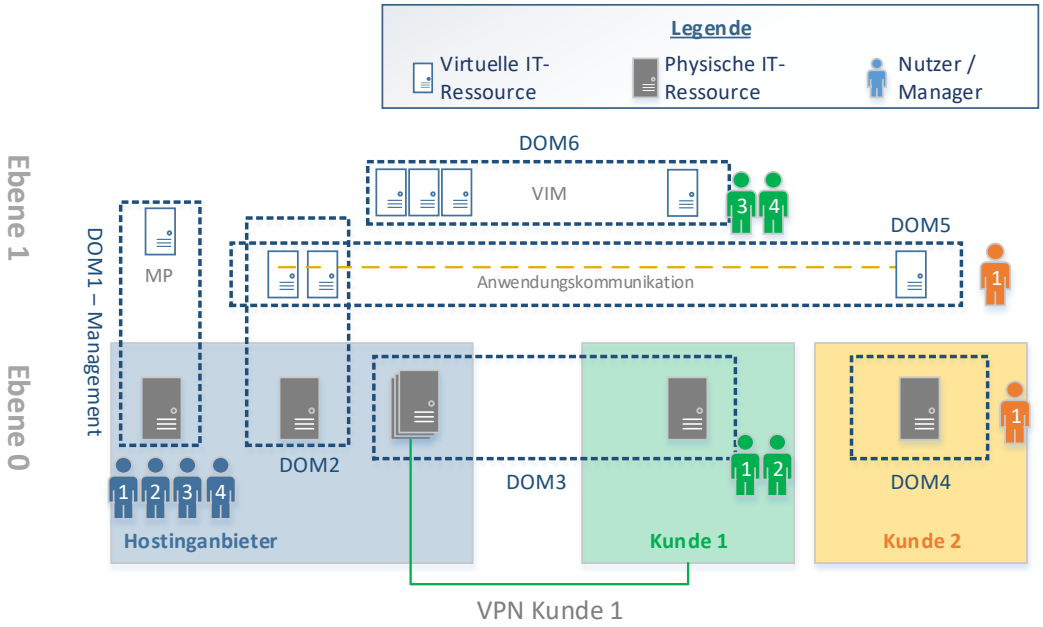


Abbildung 3.6: Übersicht über Föderations-, Infrastruktur- und Organisationsaspekte in Szenario 3.

3.5.1 Föderation und IT-Infrastruktur

Im Zentrum des in Abbildung 3.6 gezeigten Szenarios steht ein Hostinganbieter, der seinen Kunden IT-Ressourcen vermietet und je nach Dienstmodell einen Teil des Netzmanagements übernimmt. Die Kunden und entsprechend auch die Datenzentrenstruktur des Hostingproviders sind international aufgestellt. Eine Kündigung gemieteter IT-Ressourcen ist monatlich möglich und die meisten Kunden nutzen derartige Angebote, um ihre eigenen IT-Ressourcen im Bedarfsfall kurzfristig zu erweitern. Stellvertretend für unterschiedliche Dienstleistungen werden zwei unterschiedliche Kunden betrachtet. Eine Föderation besteht in diesem Szenario jedoch nur zwischen genau einem Kunden und dem Hostinganbieter, da die Kunden untereinander keine gemeinsame Ressourcennutzung aufweisen.

Datenzentrenstruktur

3.5.1.1 Föderationsbeziehungen

Der Hostinganbieter hat ausreichend IT-Personal um alle Managementaufgaben, die im Rahmen der Dienstleistung für von ihm betriebene IT-Ressourcen anfallen, zu erfüllen. Je nach Dienstmodell können Kunden selbst entscheiden, ob sie die gemieteten IT-Ressourcen weitestgehend selbst verwalten oder einen mehr oder weniger großen Teilbereich durch den Hostinganbieter verwalten lassen. Als weiteres Dienstangebot zur Unterstützung des Netzmanagements stellt der Hostinganbieter den Kunden zudem Zugriff auf die Netzmanagementplattform (in Abbildung 3.6 als *MP* bezeichnet) bereit, durch welche Managementaufgaben der gemieteten IT-Ressourcen technisch unterstützt werden. Eigene IT-Infrastruktur der Kunden ist jedoch nicht an die Managementplattform MP angebunden, sodass diese potenziell selbst jeweils eigene Netzmanagementplattformen betreiben.

Dritt-Managementplattformen

Domänenspezifische Aufgaben und Organisation

Kunde 1 hat ebenfalls ausreichend IT-Personal zur Verfügung und will das Netzmanagement für alle seine Ressourcen, auch gemietete, selbst übernehmen. Die von ihm in Anspruch genommene Dienstleistung umfasst Mehrzweckressourcen, auf denen er unterschiedliche eigene Dienste und Anwendungen betreibt. Nach der aus dem Cloud-Computing üblichen Kategorisierung nimmt Kunde 1 daher eine *Infrastructure-as-a-Service*-(IaaS)-Leistung in Anspruch. IaaS beschreibt gemäß der NIST-Definition von Cloud-Computing [37] ein Dienstmodell, in dem der Kunde selbst über umfangreiche Kontrolle über die Einrichtung von IT-Ressourcen wie Rechenleistung, Speicher oder Netzressourcen verfügt sowie ihr Betriebssystem und darauf betriebene Anwendungen selbst festlegen kann.

Kunde 2 ist hingegen ein kleines Unternehmen mit wenig IT-Personal, von dem sich lediglich ein Mitarbeiter mit dem Management der IT-Ressourcen befasst. Entsprechend ist das Ziel von Kunde 2, möglichst wenige Aufgaben des Netzmanagements selbst durchzuführen und viele Managementaufgaben dem Hostinganbieter zu überlassen. Das Ziel der Anmietung der zusätzlichen IT-Ressourcen für Kunde 2 ist der Betrieb einer eigenentwickelten verteilten Anwendung. Die Bereitstellung der dazu notwendigen Laufzeitumgebung ist Teil des Dienstangebots des Hostinganbieters. Gemäß dem Cloud-Computing-Dienstmodell der NIST [37] wird die Bereitstellung einer Laufzeitumgebung für Kundenanwendungen als *Platform-as-a-Service* (PaaS) bezeichnet. Demnach hat der Kunde an diesem Dienstmodell keinen Managementzugriff auf die eigentlichen niedrigebnigen IT-Ressourcen wie bei IaaS; diese werden weiterhin durch den Dienstanbieter gemanagt.

3.5.1.2 IT-Infrastruktur

Wie in Abbildung 3.6 gezeigt ist, kann die IT-Infrastruktur wie auch in den anderen Szenarien in mehrere Virtualisierungsebenen eingeteilt werden. Mindestens beim Hostinganbieter sind auf Ebene 0 eingebrachte Ressourcen jedoch meist bereits selbst virtuelle Systeme, die durch eine darunterliegende Cloud-Plattform realisiert und verwaltet werden. Die Cloud-Plattform ist nicht direkt Teil der Föderation. Auf Ebene 0 ist davon auszugehen, dass der Hostinganbieter weitestgehend homogene IT-Ressourcen im Dienstangebot bereitstellt und Kunden den Dienstanbieter entsprechend ihrer Anforderungen an diese aussuchen. Da sie das Angebot des Hostinganbieter nutzen, um ihre eigenen IT-Ressourcen zu erweitern, betreiben sie entsprechend selbst ähnliche Systeme. Für IaaS-Dienste kann zudem die Zahl der Ebene-0-Ressourcen relativ flexibel variieren, da sie der Kunde selbst über seinen Managementzugang einrichten und skalieren kann. Im Vergleich zu den zuvor beschriebenen Szenarien ist die grundlegende Art der Vernetzung der geographisch verteilten IT-Ressourcen unterschiedlich. Der Dienstanbieter stellt in diesem Fall lediglich die gemieteten Ressourcen bereit. Die Vernetzung wird durch die Kunden selbst und daher heterogen vorgenommen. So setzt beispielsweise Kunde 1 im Rahmen seiner IaaS-Dienstnutzung ein eigenes VPN zwischen den gemieteten Ressourcen und seinem eigenen Datenzentrum auf. Kunde 2 hat diese Möglichkeit im Rahmen seiner PaaS-Dienstnutzung nicht, die Kommunikation erfolgt vielmehr auf Anwendungsebene.

Heterogenität Vernetzung

Auf Ebene 1 umfasst das Szenario standortübergreifende Anwendungen, die je nach Art des in Anspruch genommenen Dienstes variieren. Beispielsweise werden IaaS-Hostingangebote, wie sie Kunde 1 einkauft, auf dieser Ebene über einen darüberliegenden VIM zusammengeführt und verwaltet. IT-Ressourcen aus dem PaaS-Dienstmodell, wie sie in diesem Szenario von Kunde 2 in Anspruch genommen werden, werden direkt genutzt. Auf dieser Ebene wird ebenfalls die Managementplattforminstanz MP betrieben. Der Hostinganbieter betreibt eine Managementplattforminstanz pro eigenem Datenzentrum, um die Überwachung und Steuerung der gemanagten Infrastruktur darin mit möglichst geringen Latenzen zu erreichen. Die Kunden bekommen Zugriff auf die Managementplattform in Form einer webbasierten Nutzeroberfläche und Programmierschnittstelle (API). Eine mandantenfähige Managementplattform ist in diesem Szenario daher unausweichlich.

Mandantenfähigkeit

Über Ebene 1 liegende Ebenen werden in diesem Szenario nicht explizit berücksichtigt. Durch verschiedene von Kunden betriebene VIMs oder andere Virtualisierungslösungen ist jedoch eine ähnliche Struktur wie in Szenario 1, mit verschachtelter Virtualisierung, SDN-gesteuerten Teilnetzen und heterogenen Netzfunktionen denkbar und bei hoher Kundenzahl wahrscheinlich.

3.5.2 Organisation

Durch die hier im Speziellen betrachtete allgemeine Anbieter-Kunden-Beziehung ergeben sich im Vergleich zu den vorherigen Szenarien Unterschiede in der Organisation. Auf diese wird im Besonderen eingegangen.

3.5.2.1 Managementfunktionen und -Anwendungen

Netzweite Vorgaben

Funktionsbereiche des Managements umfassen im Szenario aufgrund des breiten Einsatzzwecks der Infrastruktur die FCAPS-Elemente der ISO (vgl. Abschnitt 2.4.1.1). Zum einen hilft die Standardisierung der Managementfunktionen bei der Übertragung von in der Managementplattform daran ausgerichteten Strukturen (z. B. Aufgaben und Rollen) an Kundensysteme, da sich Kunden im Rahmen des Managements ihrer eigenen IT-Ressourcen mit hoher Wahrscheinlichkeit daran orientieren. Zum anderen sind in diesem Szenario alle Managementfunktionen von Bedeutung: Das in den vorherigen Szenarien nicht notwendige Abrechnungsmanagement ist in Cloud-Diensten von zentraler Wichtigkeit und es haben sich unterschiedlichste Abrechnungsmodelle, beispielsweise nach Nutzungszeit, Dienstvolumen, Quality-of-Service (QoS) und Ort herausgestellt [50]. Eine Abrechnung ist naheliegenderweise von Hostinganbieter zu Kunden notwendig und Kunden müssen die Möglichkeit haben, derartige nicht-MO-spezifische Informationen über einen Managementzugang einsehen und erfassen zu können. Zum anderen ist jedoch auch eine Abrechnung von Kunden zu ihren eigenen Kunden potenziell notwendig, da diese die Kosten der Anmietung von IT-Ressourcen für ihre Dienste möglicherweise selbst weitergeben.

Netzereignisse und -Informationen

Notwendige Managementanwendungen umfassen daher generell solche zur Überwachung und Steuerung von gemieteten IT-Ressourcen, ähnlich wie die bereits in den vorherigen Szenarien (siehe Abschnitte 3.3.2.5 und 3.4.6) beschriebenen Anwendungen. Auf diese wird nicht erneut explizit eingegangen. Szenarienspezifische Managementanwendungen sind die folgenden:

- Eine Anwendung zur Einsicht abrechnungsspezifischer Daten pro Dienst und Kunde, beispielsweise hinsichtlich verbrauchter Prozessorzeit und einer daraus resultierenden Kostenberechnung für einen Dienst.

- Eine Anwendung zur Einrichtung administrativer Domänen hinsichtlich darin enthaltener (insb. kundeneigener MOs), Zuständigkeiten und Rollen (siehe Abschnitt 3.5.2.3).

Parametrisierung von MAPPs

Die zentralen Managementanwendungen sind für alle Kunden nutzbar und werden entsprechend der betrachteten Zuständigkeiten in einer Domäne und Ressource parametrisiert.

3.5.2.2 Inter-Domänen-Organisation

Anders als in den vorherigen Szenarien sind administrative Domänen auf Ebene 0 nicht an das jeweilige von einer Partei betriebene Datenzentrum gebunden. Für IaaS-Dienste ist die Domänenstruktur vielmehr, wie am Beispiel der Domäne DOM3 (vgl. Abbildung 3.6) gezeigt wird, dass der jeweilige Kunde selbst dafür die Managementverantwortung hat. Bei PaaS-Diensten bleibt die Zuständigkeit für IT-Ressourcen der Ebene 0 hingegen beim Hostinganbieter, wie Domäne DOM2 verdeutlicht. Eine Besonderheit ist hier, dass sich Domäne DOM2 über mehrere Ebenen spannt und Teile der Zuständigkeiten von davon abgeleiteten IT-Ressourcen

beim Hostinganbieter bleiben. Domäne DOM2 überschneidet sich deshalb außerdem in PaaS-Ressourcen mit Domäne DOM5 durch die getrennten Verantwortlichkeiten dafür. So ist der Hostinganbieter für die Zuverlässigkeit des Systems bis hin zur Laufzeitumgebung zuständig, Kunde 2 hingegen für darauf betriebene Anwendungen.

Domänenüberschneidung

Ein Aspekt, der insbesondere bei kurzfristigen Föderationen nicht unverhältnismäßig aufwändig sein darf, ist die Einrichtung der Managementplattform. Die initiale Einrichtung der administrativen Domänen wird stets durch den Hostinganbieter als Betreiber der Managementplattform vorgenommen. Da dem Hostinganbieter die IT-Ressourcen der Kunden bei Initiierung des Vertrags nicht bekannt sind, würde die initiale Einrichtung (anders als in Abbildung 3.6 gezeigt wird) der Domänen DOM3 und DOM5 lediglich die von ihm bereitgestellten IaaS- und PaaS-Ressourcen umfassen. Die Domänen DOM4 sowie DOM6 sind bei Einrichtung der jeweiligen Föderationen zwischen dem Anbieter und den Kunden Kunde 1 und Kunde 2 nicht vorhanden: Die IT-Ressourcen in Domäne DOM4 sind nachträglich durch den Kunden angelegt und die in Domäne DOM6 zum Zeitpunkt der Initialisierung noch nicht erstellt. Den Kunden muss entsprechend im Management und der dazugehörigen Plattform MP die Möglichkeit gegeben werden, eigene Domänen anzulegen und zu verwalten. Ein Kunde darf jedoch nicht die Konfiguration der anderen Domänen beeinflussen können.

Dezentrale Domänenverwaltung

Auch Domäne DOM1, die Managementinfrastruktur umfassend, ist ebenenübergreifend. Sie weist die Zuständigkeit für die gesamte vom Hostinganbieter bereitgestellte Managementinfrastruktur ebendiesem zu. Die Konfiguration der Managementplattform obliegt ausschließlich ihrem Betreiber, dem Hostinganbieter.

3.5.2.3 Intra-Domänen-Organisation

Das Management innerhalb der Domänen unterscheidet sich in diesem Szenario stark untereinander, stets ausgehend vom jeweiligen Kunden. Je nach personellen Ressourcen, der Größe der administrativen Domäne (gemessen an der Anzahl darin befindlicher MOs), dem beim Kunden verfügbaren IT-Personal, Fachwissen und verfolgtem Ansatz können Zuständigkeiten, Gruppen, Rollen, Granularität der Aufgabenverteilung stark variieren. Auf der anderen Seite steht darüber hinaus stets der vom Hostinganbieter verfolgte Managementansatz, der nicht notwendigerweise auf jeden Kunden übertragbar ist. Eine Managementplattform und das durch sie implementierte Organisationsmodell muss unterschiedliche Ansätze der Parteien in einer gemeinsamen IT-Umgebung unterstützen können.

Heterogene Managementkonzepte

Wie bereits in Abschnitt 3.5.1.1 erklärt wurde, sieht der durch den Hostinganbieter angebotene Zugriff auf die Managementplattform MP nicht die Anbindung kundeneigener MOs an vor. Der Hostinganbieter will vielmehr aus Sicherheitsgründen möglichst wenig Schnittstellen der Plattform der Vielzahl an Kunden verfügbar machen, wodurch ausschließlich die API bereitgestellt wird. Die API ist deutlich einfacher kontrollierbar, da sie den Zugriff hinsichtlich Protokolle, Daten und Funktionen vorgibt. Das SBI hingegen richtet sich in erster Linie nach den von MOs bereitgestellten Schnittstellen. Ein Überblick über IT-Ressourcen in der Föderation ist für Kunden jedoch essenziell, wodurch der Hostinganbieter festgelegte Parameter (z. B. verfügbare CPU- und Speicherresourcen) über eine festgelegte API-Funktion bereitstellt. Kunden können nach einem vom Hostinganbieter vorgegebenen Schema ihre eigenen IT-Ressourcen in die Gesamtsicht der bereitgestellten Managementplattform MP eintragen. Ob die Daten dann von einzelnen Agenten auf den kundeneigenen Systemen oder eine weitere Managementplattform der Kunden eingepflegt werden, ist nicht durch den Hostinganbieter vorgegeben. Für diesen Funktionsumfang sind jedoch in ihren Privilegien zu diesem Zweck eingeschränkte Funktionskennungen zum Zugriff auf die API der Managementplattform wünschenswert, da Kunden für den Zweck nicht ihre Nutzerkennungen mit umfangreichen Privilegien nutzen dürfen, die bei einer Kompromittierung eines Systems zu Schäden führen können.

Zustandsoperationen durch MAPPs

Funktionskennungen, feingranulare Autorisierung

3.5.3 Bedeutung für das Informationsmodell

Aspekte des Informationsmodells, die sich aus dem Szenario ergeben unterstützen *a)* seine Notwendigkeit zur Flexibilität in Bezug auf erfassbare Daten und daraus generierten Informationen sowie *b)* die Berücksichtigung von örtlichen Gegebenheiten der IT-Infrastruktur.

Aspekt *a)* ergibt sich aus der in Abschnitt 3.5.2.1 beschriebenen Notwendigkeit, dass besonders auch nicht-MO-eigene Informationen, sondern in diesem Fall Nutzungs- und Abrechnungsinformationen für das Netzmanagement von Bedeutung sind und von Managementplattformen in ihrer Vielfalt unterstützt werden müssen. Zusammen mit in Szenario 2 (Abschnitt 3.4.5) oder Szenario 3 (siehe Abschnitt 3.3.3) beschriebenen Netzereignissen und -status ist diese Art der notwenigerweise zu berücksichtigenden Informationen je nach Szenario unterschiedlich und vor allem potenziell inkonsistent beschrieben.

Der Aspekt *b)* ergibt sich aus der Datenzentrenstruktur im Szenario. Der Hostinganbieter betreibt mehrere international verteilte Datenzentren, die zur Erfüllung desselben Dienstangebots dienen. Gleichzeitig muss er eine vereinbarte Dienstgüte erfüllen, was im Kontext ausreichender Performanz geringe Latenzen zu seinen Diensten (inkl. der angebotenen Managementplattform selbst) umfasst.

3.5.4 Bedeutung für das Kommunikationsmodell

Die in den vorherigen Abschnitten genannten organisatorischen Aspekte fließen auch in das Kommunikationsmodell ein. So wird in Abschnitt 3.5.1.1 eine Anbindung von kundeneigenen Managementplattformen an die durch den Hostinganbieter bereitgestellte Plattform MP beschrieben. Netzmanagementplattformen selbst bilden so einen zusätzlichen Akteur mit Managementplattformen in FSNs, der in bestimmten Szenarien Berücksichtigung finden und szenarienspezifisch geeignet angebunden werden muss. Im konkreten Fall würden kundeneigene Managementplattformen Informationen über kundeneigene MOs einpflegen, die Teil des FSN sind. Im SDN-Bereich werden solche Schnittstellen als *Eastbound-* oder *Westbound-Interfaces* bezeichnet [51]; sie verdeutlichen die Kommunikation auf gleicher Ebene.

Managementplattform als Akteur

Eastbound- / Westbound-Interface

Betreiber und Entwickler von Managementplattformen müssen die Möglichkeit haben, szenarienspezifische Funktionen der Managementplattform über die API an Nutzer durchreichen zu können. Der Zugriff auf die API ist im Szenario vor allem über das Netz notwendig.

Netzbasierte API

3.5.5 Bedeutung für das Funktionsmodell

Erweiterbarkeit Funktionsumfang

Die im Szenario auftretenden Anwendungsfälle für Funktionalitäten zur Beschreibung von MOs oder aber auch des Abrechnungsmanagements machen die funktionale Erweiterung der Managementplattform selbst erforderlich. Des Weiteren ist die Mandantenfähigkeit der Managementplattform aufgrund des Zugriffs mehrerer Parteien eine grundlegende Anforderung im Szenario. Eine Managementplattform muss folglich auch bei nachträglich erweiterten Funktionen einen einfachen mandantenfähigen Zugriff unterstützen.

Mandantenfähigkeit

Die in Abschnitt 3.5.1.2 beschriebene Möglichkeit zum Zugriff von externen Managementplattformen oder Agenten des Kunden über die API zur Beschreibung von MOs verdeutlicht die Notwendigkeit der Erweiterbarkeit von Plattformfunktionen.

Erweiterbarkeit Programmierschnittstelle

3.5.6 Einordnung des Szenarios

Szenario 3 vervollständigt in Bezug auf seine Charakterisierung die Spanne der in dieser Arbeit betrachteten FSN. Eine Zusammenfassung der Charakteristika ist in Tabelle 3.5 gezeigt. Die räumliche Dimension ist durch die international verteilten Datenzentren des Dienstanbieters und viele Kunden als gemischt anzusehen. Manche Kunden sind nahe einem Datenzen-

trum gelegen, andere nicht. Die Kunden nutzen die monatliche Kündigungsfrist, um ihre eigenen IT-Ressourcen zu erweitern, wodurch das Szenario vor allem kurzfristige Föderationen betrachtet. Die Koordination ist explizit durch das Dienstmodell des Hostinganbieters vorgegeben und teilt die Verantwortlichkeiten klar ein. Die Gruppenstruktur ist pro Anbieter-Kunden-Beziehung stets fest und ändert sich nicht, die Zielsetzung ist rein komplementär. Das Vertrauen ist in der Regel indirekt, da sich die Föderationsparteien üblicherweise nicht direkt kennen, sondern lediglich eine geschäftliche Beziehung eingehen. Die Aufgabenzuordnung, Dynamik und Rollenvergabe sind relativ statisch und weitestgehend durch das eingekaufte Dienstmodell bzw. den Kunden bestimmt. Die Relation zwischen Domänen ist in dem Szenario jedoch durchaus überlappend in IT-Ressourcen und muss feingranularer als nur systemspezifisch behandelt werden. Die Domänenstruktur ist im Fall des Szenarios hierarchisch aufgebaut: Die Grundstruktur wird durch den Hostinganbieter vorgegeben, kann aber durch die Kunden weiter verfeinert werden. Die Kunden sind zudem die ausschließlichen Endnutzer der föderierten Infrastruktur, wodurch die Nutzung als asymmetrisch klassifiziert wird.

Charakteristikum	Ausprägung in FSNs		
Räumliche Dimension	regional	überregional	**gemischt**
Dauer	**kurzfristig**	langfristig	
Koordination	implizit	**explizit**	
Gruppenstruktur	flexibel	**fest**	
Vertrauen	**indirekt**	direkt	gemischt
Aufgabenzuordnung	**statisch**	dynamisch	
Dynamik	**statisch**	dynamisch	
Rollenvergabe	**statisch**	dynamisch	
Relation zw. administrativen Domänen	**überlappend**	disjunkt	
Zielsetzung	überlappend	**komplementär**	gemischt
Domänenstruktur	global	**hierarchisch**	unabhängig
Art der Infrastrukturnutzung	symmetrisch	**asymmetrisch**	

Tabelle 3.5: Einordnung von Szenario 3.

3.6 Anforderungen an Frameworks für Managementplattformen in FSNs

Die in den vorherigen Abschnitten beschriebenen Szenarien verdeutlichen unterschiedliche Ausprägungen im Zweck, Aufbau und in der Organisation von FSNs und einer dafür geeigneten Managementplattform. In diesem Abschnitt werden Anforderungen an Frameworks für Teilmodelle von Managementplattformen in FSNs gestellt. Darüber hinaus ist die szenarienübergreifende Identifikation von Komponenten von Managementplattformen für FSNs notwendig, die szenarienspezifisch in der Implementierung der Frameworks angepasst werden müssen, die *Hot-Spots* (vgl. Abschnitt 2.3.2.1).

Die Anforderungen werden schließlich gemäß den vier Teilmodellen von Managementarchitekturen sowie nach nicht-funktionalen Anforderungen gruppiert. Jede Anforderungsbeschreibung besteht aus einem **eindeutigen Identifikator**, einer **Kurzbeschreibung**, einer **Gewichtung** hinsichtlich der Bedeutsamkeit der jeweiligen Anforderung, sowie den **Quellszenarien**, aus denen die Anforderung abgeleitet werden kann.

3.6.1 Gewichtung und Ableitung der Anforderungen

Die **Gewichtung** erfolgt dabei in einer dreistufigen Skala, welche die Notwendigkeit der Erfüllung einer Anforderung beschreibt.

- **Essenziell (3)**: Ein geeignetes Konzept eines Frameworks für den jeweils behandelten Teilaspekt von Managementplattformen in FSNs muss alle diese Anforderungen erfüllen. Andernfalls ist es als grundlegend ungeeignet einzustufen.

- **Wichtig (2)**: Die Erfüllung dieser Anforderungen ist notwendig, damit ein Framework für Managementplattformen in FSNs szenarienübergreifend eingesetzt werden kann. Ein Nichterfüllen wird die Eignung aus dem Framework abgeleiteter Managementplattformen voraussichtlich erheblich einschränken. In speziellen Szenarien kann eine darauf basierend implementierte Managementplattform jedoch noch zweckdienlich sein.

- **Empfehlenswert (1)**: Als *empfehlenswert* eingestufte, nicht erfüllte Anforderungen führen in der Regel zu unkritischen Einschränkungen im Management von (szenarienspezifischen und gleichartigen) FSNs, die im Allgemeinen vertretbar sind. Beispielsweise betreffen diese Anforderungen auch Optimierungen in der Umsetzung.

Die **Quellen**, aus denen eine Anforderung abgeleitet werden kann, können wie folgt dargestellt sein:

- **A**: Die Anforderung wurde aus den allgemeinen Charakteristika von FSNs abgeleitet. Die Herleitung dieser Anforderungen wurde bereits in Abschnitt 3.1.2.2 beschrieben.

- **S1**: Die Anforderung wurde (teilweise) aus Szenario 1 abgeleitet, der Kooperation mehrerer Organisationen im Rahmen eines gemeinsamen Projekts (Abschnitt 3.3).

- **S2**: Die Anforderung wurde (teilweise) aus Szenario 2 abgeleitet, einer Kooperation im Kontext eines medizinischen Dienstes (siehe Abschnitt 3.4).

- **S3**: Die Anforderung wurde (teilweise) aus Szenario 3 abgeleitet, der bedarfsabhängigen Erweiterung von IT-Ressourcen über einen Hostinganbieter (siehe Abschnitt 3.5).

3.6.2 Hot-Spot Analyse

Hot-Spots bezeichnen flexible Programmteile, welche in den Frameworks implementierbar sein müssen, um aus dem Framework abgeleitete Instanzen individuell an ein bestimmtes Szenario anzupassen. Aus den vorher beschriebenen Szenarien resultieren diesbezüglich mehrere Komponenten in Frameworks, welche als Hot-Spots realisiert sein müssen.

3.6.2.1 Hot-Spots des Funktionsmodells

Die funktionale Erweiterung einer Managementplattform erfolgt insbesondere über darauf aufbauende Managementanwendungen. Dennoch sind teilweise Funktionen der Managementplattform notwendig, die beispielsweise nicht einem bestimmten MO zugeordnet werden können, sondern vielmehr zur Einsicht und Konfiguration der Managementplattform selbst oder Strukturen des Managements dienen. Diese hier als **Plattformfunktionen** (abgrenzend zu *MO-spezifischen Funktionen*) bezeichneten Methoden erlauben beispielsweise das Anlegen und Ändern, aber auch das Entfernen von administrativen Domänen als grundlegende organisatorische Strukturen, von Nutzern und Rollen, die Konfiguration der API oder des SBI der Managementplattform.

Ein weiterer Hot-Spot ist die Beschreibung von **Managementfunktionen**. Wie in den Szenarien gezeigt werden konnte, können diese mit unterschiedlichem Schwerpunkt als Gruppierung für feingranularere Verantwortlichkeiten, Rollen und damit verknüpften Managementanwendungen definiert werden. Das Abrechnungsmanagement musste beispielsweise im Gegensatz zum dritten Szenario im ersten und im zweiten Szenario nicht berücksichtigt werden.

3.6.2.2 Hot-Spots des Kommunikationsmodells

Die Unterschiedlichkeit der Szenarien hat gezeigt, dass das NBI bzw. die **API** der Managementplattform nicht einheitlich sein kann. Unterschiede ergeben sich beispielsweise aus der Zusammensetzung der gemanagten Infrastruktur und darin konkret umgesetzter FSN-Konzepte. Beispielsweise kann SDN auf unterschiedliche Art und Weise umgesetzt sein, sei es mittels OpenFlow oder OVSDB, oder anderen Protokollen und Systemen. Jede Umsetzung bringt unterschiedliche Funktionen mit sich, welche sich auch in der API bzw. Funktionalität einer Managementplattform manifestieren. Die API muss entsprechend flexibel und erweiterbar sein.

Die Flexibilität des **SBI** ist zentraler Bestandteil für die Eignung einer Managementplattform für unterschiedliche Umgebungen. Je nach Szenario ist die Zusammensetzung der gemanagten Infrastruktur stark unterschiedlich und standardisierte APIs, über die MOs darin gemanagt werden, existieren praktisch nicht. Die Anbindung von MOs ist jedoch essenziell, weshalb nur ein erweiterbarer Ansatz hinsichtlich **Konnektoren und Schnittstellen** aber auch **Protokollen und Beschreibungsformaten** aufseiten des SBI erfolgreich sein kann. Dazu zählt beispielsweise auch ein geeigneter Umgang mit Parametern und Rückgabewerten, jedoch auch mit unterschiedlichen Authentifizierungsmechanismen (z. B. Basic-Auth, Token-basiert, usw.) der APIs von MOs. Derartige Aspekte müssen in den Frameworks berücksichtigt werden.

Auch die **Datenbankanbindung** muss je nach geeigneter Systemarchitektur einer Managementplattform ebenfalls flexibel sein. Beispielsweise bietet sich in Szenario 1 eine verteilte Datenbank an, in Szenario 2 dagegen eine zentralisierte. Je nach Szenario können so unterschiedliche Datenbanksysteme, beispielsweise anhand Performanzkriterien, Dauerhaftigkeit der Daten (z. B. transiente oder persistente Datenspeicherung) oder anhand von Verbindungskriterien (z. B. Zentralisierung der Kommunikation oder Zuverlässigkeit der Netzanbindung) geeignet sein.

Ein weiterer Aspekt, der besonders im Kommunikationsmodell zum Tragen kommt, ist eine flexible Art und Weise der **Authentifizierung** und **Authentisierung**. Je nach Anwendungsfall können für unterschiedliche Schnittstellen unterschiedliche Systeme herangezogen werden, beispielsweise über externe Verzeichnisdienste oder über eine direkt integrierte Nutzerverwaltung.

3.6.2.3 Hot-Spots des Informationsmodells

In den beschriebenen Szenarien wurde deutlich, dass unterschiedliche Bausteine des Informationsmodells Hot-Spots in FSNs darstellen. Zum einen muss eine Managementplattform für FSNs die Modellierung von **Managementobjektklassen** (MOCs) unterstützen. MOCs müssen nicht nur szenarienübergreifend, sondern auch innerhalb eines Szenarios stark heterogene MOs abbilden können.

Daneben gibt es in FSNs eine breite Spanne an **Managementbeziehungen**. SDN und NFV bringen vor allem Steuerungs- und Systembeziehungen (z. B. ein verteilter Dienst) zwischen MOs ein, System- und Netzvirtualisierung vor allem Realisierungsabhängigkeiten, die weiter unterteilt werden können. Eine Managementplattform muss ein adaptierbares **Abhängigkeitsmodell zwischen MOs** bieten, das ebenfalls damit verbundene funktionale und organisatorische Aspekte und Vorgaben implementieren kann. Beispielsweise wird eine VM im NFV-Paradigma nicht direkt über ihren Hypervisor, sondern über einen darüberliegenden VIM gesteuert.

Des Weiteren ist ein **Funktions- und API-Framework** für Systeme in FSNs notwendig. Dabei ist die Gleichartigkeit der Funktionalität von komponentenspezifischen FSN-Systemen zu beachten, die durch unterschiedliche Komponenten-Implementierungen dennoch nicht einheitlich nutzbar sind. Beispielsweise sind *Floodlight* und *Ryu* unterschiedliche SDN-Controller mit unterschiedlichen Attributen und Schnittstellen. Beide unterstützen jedoch die OpenFlow-Spezifikation eines SDN und teilen sich entsprechend einige darin spezifizierte Funktionen, wie die Installation eines statischen Flows.

Zwei weitere Hot-Spots betreffen nicht-MO-eigene Informationen: **Netzereignisse und -Zustände** sowie **Alarme**. Erstere lassen sich oft nicht als Attribut eines bestimmten MOs im Netz abbilden (z. B. *Status Netz-Topologie* oder *Ereignis eines netzbasierten DDoS-Angriffs*) und gehören somit nicht zum MOC, müssen jedoch ebenfalls als grundlegende Informationen, mindestens des Performanz- und Sicherheitsmanagements, berücksichtigt werden. Letztere sind ein wichtiger Bestandteil des Reportings: Alarme sind gemäß [52] fortbestehende Hinweise auf einen Fehlerzustand. Beide beschreiben szenarienspezifisch unterschiedliche Ausprägungen von Informationen im Netz.

3.6.2.4 Hot-Spots des Organisationsmodells

Im Organisationmodell gibt es unterschiedliche Ebenen, die abgebildet werden müssen. Die Szenarien haben verdeutlicht, dass **Föderationen und Föderationsbeziehungen** in FSNs unterschiedlich ausfallen. Diese beeinflussen Verantwortlichkeiten im Netzmanagement wesentlich (bspw. zentrales gegen dezentrales Management) und müssen in einem Modell in der Managementplattform geeignet dargestellt werden.

Innerhalb der Föderation beeinflussen dann **Domänenmodelle** auf nächster Ebene Verantwortlichkeiten am stärksten und sind ein potenziell geeigneter Ansatzpunkt zur Behandlung unterschiedlicher Organisationskonzepte der Partner. Eine Herausforderung im Umgang mit administrativen Domänen liegt auch in der Behandlung von Überschneidungen, die in FSNs durchaus auftreten.

Innerhalb von Domänen spielen **Nutzer- und Rollenbeschreibungen** und schließlich konkrete Verantwortlichkeiten bzw. **Managementaufgaben** eine konkrete Rolle. Diese unterscheiden sich in einer Föderation beispielsweise bezüglich des Umfangs der personellen Ressourcen (je weniger IT-Personal zur Verfügung steht, desto grobgranularer können Managementaufgaben vergeben werden), aber auch wiederum von bei einzelnen Parteien in der Föderation üblichen Managementansätzen.

3.6.3 Nicht-funktionale Anforderungen

Einige nicht-funktionale Anforderungen wurden bereits ausgehend von allgemeinen Charakteristika von FSNs am Anfang dieses Kapitels in Abschnitt 3.1.2.2 detailliert beschrieben. Weitere, auf Basis der Szenarien gestellte nicht-funktionale Anforderungen werden in den folgenden Absätzen erläutert. Eine Übersicht über alle nicht-funktionalen Anforderungen ist in Tabelle 3.6 mit jeweiliger Zuordnung zu einem der vier Teilmodelle zusammengestellt.

Verteilte Systemarchitektur Die gemanagten IT-Ressourcen sind in FSNs üblicherweise über mehrere geographische Standorte verteilt, über die sich auch die Managementplattform als verteiltes System erstrecken muss. Eine verteilte Managementplattform wurde in allen Szenarien berücksichtigt. Die Anforderung ist in den meisten Szenarien von FSNs relevant und daher *wichtig*.

Sparsamkeit der API-Kommunikation. Durch die Nutzung öffentlicher, oft durchsatzbeschränkter Infrastruktur, ist eine netzressourcenschonende Kommunikation zwischen Ma-

nagementplattform und -Anwendungen zu ermöglichen. Dieser Aspekt wurde in Szenario 1 in Abschnitt 3.3.5 begründet. Der Austausch möglichst weniger Daten zur Zweckerfüllung einer Managementanwendung ist *empfehlenswert* und ist in vielen Szenarien nicht ausgeprägt notwendig.

Komponentenagnostischer Zugriff auf MO-Funktionen Eine szenarienübergreifende Anforderung (insbesondere in Szenario 1 in Abschnitt 3.3.4 verdeutlicht) ist der typ- und komponentenagnostische Zugriff auf MO-Funktionen. Für einen Netzmanager steht in der Regel die eigentliche Funktion/Steuerungsmöglichkeiten des Netzes im Vordergrund, weniger hingegen die ausführende Komponente und ihr spezieller Typ (z. B. welche Art von SDN-Controller). Andernfalls könnte die Gefahr bestehen, dass derartige Informationen mehr Verwirrung als Nutzen bringen, weshalb diese Anforderung als mindestens *wichtig* eingestuft wird.

Plattformunabhängigkeit. In Szenario 2 (Abschnitt 3.4.4.3) wird die Notwendigkeit nach Plattformunabhängigkeit von Managementplattformen durch ihre Nutzung auf Patientengeräten und Einplatinencomputern herausgestellt. Eine Plattformunabhängigkeit ist hingegen nur in speziellen Szenarien wirklich notwendig und kann z. B. durch den Einsatz von Managementagenten umgangen werden. Die Anforderung ist daher *empfehlenswert*.

Leichtgewichtigkeit. In Szenario 2 (Abschnitt 3.4.4.3) wurde durch den notwendigen Einsatz einer Managementplattforminstanz auf leistungsschwächeren Einplatinencomputern ein ressourcenschonender Ansatz notwendig. Für ein Framework bedeutet das folglich, dass es eine grundlegend schlanke Ausgangsbasis und leichtgewichtige Ansätze zur Erweiterung bieten muss. Da diese Anforderung mehrere Szenarien betreffen kann, wird sie als *wichtig* eingestuft.

3.6.4 Funktionale Anforderungen

Anders als bei der Ableitung der nicht-funktionalen Anforderungen im vorherigen Abschnitt stammen die meisten Anforderungen an das Funktionsmodell aus den Szenarien. Eine Zusammenfassung bietet Tabelle 3.7. Die Anforderungen werden gemäß ihrer Berücksichtigung im Kontext von Managementfunktionen sowie hinsichtlich Anforderungen des Funktionsumfangs von Managementplattformen gruppiert.

3.6.4.1 Anforderungen im Kontext von Managementfunktionen

Anforderungen im Kontext von Managementfunktionen beziehen sich in erster Linie auf zu berücksichtigende Aspekte in den Kernprozessen des Netzmanagements. Diese wurden teilweise im Kontext allgemeiner Gegebenheiten in SNs in Abschnitt 3.1.2.2 beschrieben. Eine weitere betrifft die Beschreibung von Managementfunktionen:

Adaptierbarkeit Managementfunktionen. Managementarchitekturen berücksichtigen potenziell unterschiedliche oder unterschiedlich gruppierte Managementfunktionen, wie in allen Szenarien verdeutlicht wurde (vgl. Abschnitte 3.3.2.5, 3.4.4.2 sowie 3.5.2.1).

3.6.4.2 Anforderungen an Plattformfunktionen

Selbstkonfiguration. Wie zuvor in Szenario 1 (Abschnitt 3.3.1.2) sowie in Szenario 2 (vgl. Abschnitt 3.4.4.3) verdeutlicht wurde, ist eine einfache bzw. automatisierte Konfiguration von Managementplattforminstanzen notwendig. Ein Framework sollte diese Funktionalität entsprechend szenarienübergreifend unterstützen. Ein Fehlen der Selbstkonfigurierbarkeit beeinflusst jedoch das Netzmanagement nicht negativ, wodurch die Anforderung als *empfehlenswert* eingestuft wird.

ID	Kurzbeschreibung	Gewicht	Quelle
Anforderungen an das Funktionsmodell			
NF.1	Quasi-Echtzeitfähigkeit	3	A
NF.2	Nachvollziehbarkeit von Aktionen	1	A
NF.3	Komponentenagnostischer Zugriff auf MO-Funktionen	2	S1, S2, S3
NF.4	Plattformunabhängigkeit	1	S2
NF.5	Leichtgewichtigkeit	2	S2
Anforderungen an das Informationsmodell			
NF.6	Datenaktualität	3	A
Anforderungen an das Kommunikationsmodell			
NF.7	Logische Zentralisierung	3	A
NF.8	Agentenloses Management	3	A
NF.9	Skalierbarkeit	2	A
NF.10	Resilienz und Fehlertoleranz	3	A
NF.11	Verteilte Systemarchitektur	2	S1, S2, S3
NF.12	Sparsamkeit der API-Kommunikation	1	S1
NF.13	Sichere Kommunikation	3	A
Anforderungen an das Organisationsmodell			
NF.14	Mandantenfähigkeit	3	A
NF.15	Föderationsbeziehungen	2	A

Tabelle 3.6: Nicht-Funktionale Anforderungen an Managementplattformen.

Domänenverwaltung. Die Verwaltung von Domänen spielt in allen Szenarien eine wichtige Rolle zur Realisierung der Managementaufgabe in Föderationen. Explizit begründet wird die Anforderung in Abschnitt 3.5.2.2 in Szenario 3, ergänzt um wichtige Aspekte in Abschnitt 3.3.4 in Szenario 1. Die Verwaltung und Übersicht administrativer Domänen als eine der grundlegendsten organisatorischen Strukturen in FSNs ist *essenziell*.

Aufgabenbasierte Übersicht und Zugriffsbeschränkung. Ein Nutzer muss szenarienunabhängig eine Übersicht über seine Aufgaben und damit verbundene Managementanwendungen einsehen können. Eine Aufgabe im Management ist üblicherweise mit der Nutzung einer oder mehrerer Managementanwendungen verbunden, auf die ein berechtigter Nutzer (basierend auf seine Aufgaben) zugreifen können muss. So kann sichergestellt werden, dass Zugriffsberechtigungen im Management aktuell und sinngemäß sind. Diese Anforderung ist *essenziell* und nicht vernachlässigbar.

Anwendungsbereich von MAPPs. Eine aus Szenario 1 in Abschnitt 3.3.2.6 hergeleitet Anforderung ist die Berücksichtigung privater und föderierter Managementanwendungen. Erstere werden von Nutzern innerhalb der Föderation zur ausschließlich eigenen Nutzung entwickelt, Letztere zum allgemeinen Management für alle NMANs. Diese Anforderung ist potenziell in vielen Szenarien von Bedeutung und daher *wichtig*.

Parametrisierung von MAPPs. Um Managementanwendungen hinsichtlich der Erfüllung einer bestimmten (Teil-) Aufgabe des Managements von FSNs breit nutzen zu könnten, müssen sie an die Umgebung der jeweiligen Domäne anpassbar sein. Eine Unterstützung dieser Funktionalität in einem Framework für Managementanwendungen berücksichtigt insbesondere solche ähnlich zu Szenario 1 und Szenario 3 (vgl. Abschnitte 3.3.1.1 bzw. 3.5.2.2). Die Anforderung beeinflusst hingegen Szenarien mit homogeneren Domänen (z. B. Szenario 2) nicht wesentlich. Entsprechend ist diese Anforderung *wichtig*.

Zustandsoperationen durch Managementanwendungen. Insbesondere in Szenario 1 (vgl. Abschnitt 3.3.4) ist die Notwendigkeit zur Generierung von Netzereignissen und -Zuständen und in Szenario 3 (siehe Abschnitt 3.5.5) die Notwendigkeit zur Generierung von MO-Attributen beschrieben worden. Für eine Managementplattform muss diese Funktion uneingeschränkt und szenarienspezifisch umsetzbar sein. Diese Anforderung ist daher *essenziell*.

Priorisierter Informationsaustausch. Diese Anforderung spielt insbesondere basierend auf Szenario 2 (Abschnitt 3.4.5) und FSN-Umgebungen mit unsicherer und qualitativ schlechter Konnektivität geographisch verteilter Ressourcen eine Rolle. Durch eine Priorisierung im Austausch von Managementinformationen kann dann die Effektivität der Managementplattform besser aufrechterhalten werden, indem hochrelevante Informationen bevorzugt werden. Die Anforderung ist *empfehlenswert*, da sie vor allem auf eine Optimierung des Betriebs abzielt.

Manipulationsschutz der Plattform. Die Notwendigkeit zum Manipulationsschutz der Plattform ist in allen Szenarien/FSNs gegeben, in denen potenziell vertrauensunwürdige Partner (vgl. Anforderung *Vertrauensaspekte*) Teil der Föderation sind. Dies ist durch Szenario 2 (Abschnitt 3.4.4.3) beispielhaft verdeutlicht, in dem Patienten für Versorgungsdienste Teil der Föderation werden. Diese Anforderung ist daher *wichtig*.

ID	Kurzbeschreibung	Gewicht	Quelle
Anforderungen im Kontext von Managementfunktionen			
F.1	Automatisierung der Kernprozesse	3	A
F.2	Adaptierbarkeit Managementfunktionen	2	S1, S2, S3
Anforderungen an Plattformfunktionen			
F.3	Unterbrechungsfreier Betrieb	2	A
F.4	Selbstkonfiguration	1	S1, S2
F.5	Domänenverwaltung	3	S1, S2, S3
F.6	Aufgabenbasierte Übersicht und Zugriffsbeschränkung	3	S1, S2, S3
F.7	Anwendungsbereich von MAPPs	2	S1
F.8	Parametrisierung von MAPPs	2	S1, S2, S3
F.9	Zustandsoperationen durch Managementanwendungen	3	S1, S3
F.10	Priorisierter Informationsaustausch	1	S2
F.11	Manipulationsschutz der Plattform	2	S2

Tabelle 3.7: Anforderungen an das Funktionsmodell.

3.6.5 Anforderungen an die Umsetzung des Informationsmodells

Die Szenarien verdeutlichen verschiedene Arten von Informationen, die zum Management von FSNs notwendig sind. Frameworks des Informationsmodells müssen besonders die benutzerdefinierte Definition von MOCs, Managementbeziehungen und Netzereignisse sowie -zustände berücksichtigen. Eine Gesamtübersicht über Anforderungen an das Informationsmodell ist in Tabelle 3.8 gezeigt. Szenarienübergreifende Anforderungen an das Informationsmodell werden in den folgenden Abschnitten für das jeweilige Teilmodell beschrieben.

3.6.5.1 Anforderungen an MOCs

Modellierbarkeit FSN-spezifischer MO-Typen. Wie vor allem in Abschnitt 3.3.3 im Kontext von Szenario 1 näher beschrieben wurde, jedoch szenarienübergreifend gültig ist, ist die Modellierbarkeit von MOC-Typen wesentlich in FSNs. Wie in den Szenarien gezeigt werden konnte (siehe Abschnitte 3.3.3.1 und 3.4.5.1), sind MOs in FSNs nicht standardisiert und ein Komponententyp ist oft unterschiedlich implementiert. Eine Managementplattform muss mit dieser Heterogenität umgehen können. Die Anforderung ist daher *essenziell*.

Normalisierung von MO-Funktionen. Diese Anforderung ist mit der nicht-funktionalen Anforderung nach einem *komponentenagnostischen Zugriff auf MO-Funktionen* verbunden. Der Bedarf dazu wurde insbesondere in Szenario 1 beschrieben (Abschnitt 3.3.3), jedoch ist diese Anforderung dennoch szenarienübergreifend relevant und wird daher als *essenziell* eingestuft.

Abbildung von IT-Ressourcen-Besitz. Eine Voraussetzung zur Berücksichtigung von Vertrauensaspekten ist die Abbildung von Besitz-Abhängigkeiten zwischen IT-Ressourcen und Partnern. Beispielsweise wie in Szenario 1 (Abschnitt 3.3.2.2) gezeigt, ist es möglich, dass ein Partner bestimmte Anwendungen nur auf eigenen oder vertrauenswürdigen Systemen betreiben möchte. Diese Anforderung wird szenarienübergreifend von Bedeutung sein und ist daher *essenziell*.

MO-Register. Die Grundlage für die Managebarkeit eines Netzes, gerade in einem Szenario mit vielen Parteien, ist eine Übersicht über zu managende Dienste und Ressourcen (vgl. Abschnitt 3.3.1.2). Diese Anforderung ist szenarienübergreifend von hoher Bedeutung und daher *essenziell*.

3.6.5.2 Anforderungen an Managementbeziehungen

Modellierbarkeit von Managementbeziehungen. Wie über alle Szenarien deutlich wurde, bestehen unterschiedlichste Managementbeziehungen zwischen MOs. Dieser Aspekt wird darüber hinaus innerhalb eines Szenarios in Abschnitt 3.5.1.2 im Kontext des dritten Szenarios explizit erklärt. Managementbeziehungen müssen in einer Managementplattform an die Gegebenheiten im jeweiligen Szenario definiert werden können. Die Anforderung gilt szenarienunabhängig und ist *essenziell*.

Berücksichtigung der Abhängigkeit von Netzen und Netzkomponenten. Durch die in FSNs auftretende Vielschichtigkeit der Virtualisierung müssen diese *vertikalen Abhängigkeiten* erfasst werden. MOs und Netzfunktionen basieren nicht selten auf darunterliegenden Netzfunktionen, wodurch Störungen eines MOs alle darüberliegenden Schichten stark beeinflussen können. Insbesondere im Fehler- als auch im Sicherheitsmanagement sind derartige Abhängigkeiten von großer Bedeutung, welche im Informationsmodell darstellbar sein müssen. Diese Anforderung ist daher *essenziell*.

Berücksichtigung von Managementbeziehungen Die Abhängigkeit zwischen MOs spielt, wie durch die Szenarien verdeutlicht werden konnte, in FSNs eine besonders große Rolle. Virtuelle Systeme werden durch VIRS realisiert, durch wiederum andere Systeme in verschiedenen Ebenen orchestriert und je nach Funktion (zentralisiert) gesteuert. Das Informationsmodell muss derartige Aspekte szenariounabhängig abbilden können, um eine Grundlage für entsprechende Funktionalität ihrer Behandlung bereitzustellen. Diese Anforderung ist *essenziell*, da die Nicht-Berücksichtigung zu grober Fehlkonfiguration führen kann.

3.6.5.3 Anforderungen an Netzereignisse und -Zustände

Modellierbarkeit von Netzereignissen und -Zuständen. Ebenfalls szenariounabhängig *essenziell* ist die Notwendigkeit der Beschreibung von Ereignissen und Zuständen (zusammengefasst als Netzinformationen), die üblicherweise kein MO-Attribut darstellen. Ihre Modellierung muss anwendungsfallspezifisch unbedingt möglich sein.

Modellierbarkeit von Alarmen. Gleiches wie für Netzereignisse und -Zustände gilt ebenfalls für Alarme und Notifikationen (wie bereits in Abschnitt 3.6.2.3 beschrieben).

Normalisierung von Netzinformationen. Wie beschrieben, können Netz-Ereignisse und Zustände anwendungsfallspezifisch praktisch beliebig detailliert beschrieben werden. Beispielsweise können SDN-Controller oft die Netztopologie abbilden, stellen diese jedoch in unterschiedlichen Formaten dar. Ein Framework muss die Möglichkeit bieten, gleichartige Informationen in eine normalisierte Form zusammenzuführen. Diese Normalisierung bildet eine *essenzielle* Grundlage für darauf aufbauende einheitliche Zugriffe, Auswertungen und folglich automatische Abläufen.

Abhängigkeit von Ereignissen zu MOs. Eine weitere Anforderung hinsichtlich Netzinformationen, Ereignissen und Zuständen ist die Modellierbarkeit der Abhängigkeit zwischen diesen und MOs. Beispielsweise, auch wenn diese an sich Informationen unabhängig von MOs sind, helfen mögliche anwendungsfallspezifische Abhängigkeiten bei der Einordnung und Auswertung. Beispielsweise ist die *Quelle* eines Ereignisses wichtig, um den Kontext einzuschätzen. Diese Anforderung ist ebenfalls als *essenziell* eingestuft.

3.6.6 Anforderungen an die Umsetzung des Kommunikationsmodells

Anforderungen an das Kommunikationsmodell werden in an die Systemarchitektur einer Managementplattform, an ihre **API** gerichtete sowie an ihr **SBI** gerichtete Anforderungen gruppiert. Alle Anforderungen sind in Tabelle 3.9 zusammengefasst.

3.6.6.1 Anforderungen an die Systemarchitektur

Flexibilität der Architektur. Eine Managementplattform muss eine an das zu unterstützende Szenario geeignete Architektur aufweisen können. So ist oft eine Umsetzung als verteiltes System aufgrund der geographischen Verteilung sinnvoll (vgl. in Szenario 1, Abschnitt 3.3.5 bzw. Szenario 3 in Abschnitt 3.5.1.2). Die konkrete Umsetzung muss jedoch möglichst flexibel sein. Diese Anforderung wird für die meisten FSN-Szenarien von Bedeutung sein und ist daher *wichtig*.

Schnittstelle zu Managementanwendungen (API). Eine Managementplattform dient in erster Linie als geeignete Infrastruktur für Managementanwendungen – auch in FSNs. Sie ist daher *essenziell*. Anforderungen an eine geeignete Schnittstelle zu Managementanwendungen werden in Abschnitt 3.6.6.2 beschrieben.

ID	Kurzbeschreibung	Gewicht	Quelle
Teilmodellübergreifende Anforderungen			
I.1	Berücksichtigung von Ressourcenabhängigkeiten	3	A
I.2	Geographische Ressourcenverteilung	2	A
Anforderungen an MOCs			
I.3	Adressierbarkeit von MOs	3	A
I.4	Modellierbarkeit FSN-spezifischer MO-Typen	3	S1, S2, S3
I.5	Normalisierung von MO-Funktionen	3	S1, S2, S3
I.6	Abbildung von IT-Ressourcen-Besitz	3	S1, S2, S3
I.7	MO-Register	3	S1
Anforderungen an Managementbeziehungen			
I.8	Modellierbarkeit von Managementbeziehungen	3	S1, S2, S3
I.9	Berücksichtigung der Abhängigkeit von Netzen und Netzkomponenten	3	S1, S2, S3
I.10	Berücksichtigung von Managementbeziehungen	3	S1, S2, S3
Anforderungen an Netzereignisse und -Zustände			
I.11	Modellierbarkeit von Netzereignissen und -Zuständen	3	S1, S2, S3
I.12	Modellierbarkeit von Alarmen	3	S1, S2, S3
I.13	Normalisierung von Netzinformationen	3	S1, S2, S3
I.14	Abhängigkeit von Ereignissen zu MOs	3	S1, S2, S3

Tabelle 3.8: Anforderungen an das Informationsmodell.

Schnittstelle zur Infrastruktur (SBI). Auch wenn die Art der an eine Managementplattform angebundenen gemanagten Komponenten sich in FSNs im Gegensatz zu herkömmlichen Netzen geändert hat (d. h. stark zentralisiert, generell stark über Zwischensysteme), ist eine Schnittstelle zur gemanagten Infrastruktur unbedingt notwendig und daher *essenziell*. Die Anforderungen an das diese Aufgabe erfüllende SBI werden in Abschnitt 3.6.6.3 beschrieben.

Adaptierbarkeit des Informationsaustauschmodells. Wie in den Szenarien erkennbar, ist die Kommunikation zwischen geographisch verteilten Standorten in FSNs unterschiedlich: Beispielsweise ist sie in Szenario 1 eher querkommunikativ, hingegen in Szenario 2 stark zentralisiert (Klinikum-RZ als zentraler Punkt des Managements). Eine Managementplattform kann zur Optimierung der Kommunikation ebenfalls eine Synchronisation der einzelnen Instanzen gemäß dem gegebenen Austauschmodell umsetzen. Diese Anforderung ist daher als *empfehlenswert* anzusehen.

Integrierbarkeit von Agenten. Auch wenn Agenten in FSNs allgemein nicht mehr unbedingt notwendig sind, wird vor allem in Szenario 2 (vgl. Abschnitt 3.4.4.3) deutlich, dass *Kleinst-Rechenzentren* wie das MedG oft nicht genug Leistung mitbringen, um eine voll-umfangreiche Instanz einer Managementplattform auszuführen. In diesem Fall kann ein leichtgewichtigeres

Agentensystem diese Problematik beheben. Der Agent muss jedoch Teil der Managementinfrastruktur sein, das zur Überwachung und Steuerung genutzt werden kann. Auch wenn dieser Aspekt vor allem in Szenario 2 erkennbar ist, sind analoge Fälle für viele weitere Szenarien vor allem mit IoT-Fokus denkbar und die Anforderung folglich *wichtig*.

Flexibilität der Datenbankanbindung. Wie als Kontrast der Szenarien 1 und 3 mit einem idealerweise verteilten Datenbanksystem und Szenario 2 mit einem geeigneterweise zentralen Datenbanksystem verdeutlich wird, ist die Datenbankanbindung szenarienspezifisch geeignet umzusetzen. Allein gemäß der Hot-Spot-Analyse stellt die Datenbankanbindung einen flexiblen Teil dar, welcher durch das Framework geeignet unterstützt werden muss, wodurch die Anforderung als mindestens *wichtig* eingestuft werden muss.

Dritt-Managementplattformen. Eine in Abschnitt 3.5.1.1 durch Szenario 3 motivierte Anforderung an Managementplattformen in FSNs ist die Berücksichtigung von Dritt-Managementplattformen als Akteur im Netzmanagement. Die Anforderung ist jedoch nur in speziellen Szenarien von Bedeutung, wodurch sie als *empfehlenswert* betrachtet wird.

3.6.6.2 Anforderungen an die API

Die API als spezieller Teil des Kommunikationsmodells fungiert mindestens als Schnittstelle zwischen Managementplattform und Managementanwendungen. Anforderungen an die API sind als Teil der Tabelle 3.9 zusammengefasst.

Erweiterbarkeit der API. Je nach FSN-Umgebung ist ein unterschiedlicher Funktionsumfang einer Managementplattform notwendig, welcher sich ebenfalls in der API widerspiegeln muss, damit Managementanwendungen diese nutzen können. Dazu muss die API eines Frameworks für Managementplattformen je nach Anwendungsfall anpassbar bzw. erweiterbar sein. Diese Anforderung ist *essenziell* für die Eignung eines Designs für Managementplattformen in FSNs.

Netzzugriff auf API. Ein FSN zeichnet sich praktisch immer durch eine geographische Verteilung von Ressourcen aus, die ebenfalls in der Regel von unterschiedlichen Orten und Netzen aus genutzt werden: Beispielsweise von den jeweiligen Standorten der Partnerorganisationen. Eine szenarienübergreifende Größe im Management von FSNs ist daher eine breite netzbasierte Zugreifbarkeit auf die API der Managementplattform. Diese Anforderung ist *essenziell*.

Netzunabhängiger Zugriff auf API. In Szenario 2 (Abschnitt 3.4.6) hat sich jedoch noch die Notwendigkeit nach einer netzunabhängigen API-Lösung verdeutlicht. In einigen Szenarien und Anwendungsfällen wird eine zuverlässige und performante Anbindung zwischen MAPPs und einer Managementplattform benötigt. Beispielsweise bei kritischen zeitgesteuerten oder ereignisgesteuerten Kontrollmaßnahmen. Auch hier muss ein Framework geeignete Bausteine zur Realisierung einer netzunabhängige API bereitstellen können, weshalb diese Anforderung *wichtig* ist.

Push- und Pull-Mechanismen. Push- und Pull-Mechanismen sind ein fester Bestandteil vieler Szenarios – Push-basierte Kommunikation üblicherweise auch in Verbindung mit *Benachrichtigungen* (vgl. insb. Szenario 2 Abschnitt 3.4.6). Ein Framework muss beide Richtungen des Verbindungsaufbaus im Rahmen der Kommunikation zwischen Managementanwendungen und Managementplattform berücksichtigen und auch hinsichtlich der Erweiterbarkeit (vgl. Anforderung *Erweiterbarkeit der API*) unterstützen. Diese Anforderung ist daher *essenziell*.

ID	Kurzbeschreibung	Gewicht	Quelle
Anforderungen an die Systemarchitektur			
K.1	Flexibilität der Architektur	2	S1, S2, S3
K.2	Schnittstelle zu Managementanwendungen (API)	3	S1, S2, S3
K.3	Schnittstelle zur Infrastruktur (SBI)	3	S1, S2, S3
K.4	Adaptierbarkeit des Informationsaustauschmodells	1	S2
K.5	Integrierbarkeit von Agenten	2	S2
K.6	Flexibilität der Datenbankanbindung	2	S1, S2, S3
K.7	Dritt-Managementplattformen	1	S3
Anforderungen an die API			
K.8	Erweiterbarkeit der API	3	S1, S2, S3
K.9	Netzzugriff auf API	3	S1, S2, S3
K.10	Netzunabhängigkeit	2	S2
K.11	Push- und Pull-Mechanismen	3	S1, S2, S3
K.12	Zustandsbehaftete Kommunikation	1	S1
K.13	Bulk-Funktion	1	S1
K.14	Sichere Kommunikation	3	S1, S2, S3
K.15	Authentifizierung und Autorisierung API	3	S1, S2, S3
Anforderungen an das SBI			
K.16	Erweiterbarkeit hinsichtlich Protokolle	2	S1, S2, S3
K.17	Erweiterbarkeit hinsichtlich Austauschformate	2	S1, S2, S3
K.18	Push- und Pull-Mechanismen	3	S1, S2, S3
K.19	Authentifizierung und Autorisierung SBI	3	S1, S2, S3
K.20	Sichere Kommunikation	3	S1, S2, S3

Tabelle 3.9: Anforderungen an das Kommunikationsmodell.

Zustandsbehaftete Kommunikation. Die Anforderung nach zustandsbehafteter Kommunikation ist vor allem in Szenario 1 (Abschnitt 3.3.5) notwendig, um die Netzauslastung und Kommunikation zwischen MAPPs und Managementplattform geringer zu halten. Die Anforderung trägt zu einem ressourcenschonenden Ansatz bei und ist daher auch in anderen Szenarien nützlich. Ihre Umsetzung ist daher *empfehlenswert*.

Bulk-Funktion. Nicht nur zustandsbehaftete Kommunikation, sondern ebenfalls Bulk-Informationsaustausch und -Funktionen wurde in Szenario 1 (vgl. Abschnitt 3.3.5) für eine effiziente

Kommunikation identifiziert. Auch hier ist die Optimierung der Plattform im Vordergrund; die Anforderung ist folglich *empfehlenswert*.

Sichere Kommunikation API. Eine Anforderung, die insbesondere für breit über das Netz zugreifbare APIs wichtig ist, ist eine sichere Kommunikation. Dabei dürfen weder Informationen von unautorisierten Dritten eingesehen werden können (Vertraulichkeit), noch manipulierbar sein (Integrität). Diese Anforderung ist szenarienübergreifend notwendig und als *essenziell* eingestuft, da ausgetauschte Managementinformationen äußerst schützenswert sein können.

Authentifizierung und Autorisierung API. Zugriffe auf die API müssen authentifiziert und hinsichtlich ihrer Zugriffsrechte geprüft werden. Auch diese Anforderung ist szenarienübergreifend von großer Bedeutung, insbesondere in Szenarien mit sehr hohem Schutzbedarf der Informationen, wie in Szenario 2. Diese Anforderung ist *essenziell*, da eine sichere Authentifizierung eine Basis für die Mandantenfähigkeit eines Systems darstellt.

3.6.6.3 Anforderungen an das SBI

Die SBI verbindet eine Managementplattform mit der gemanagten Infrastruktur. Anforderungen an diese in FSN werden folgend aus den Szenarien abgeleitet. Eine Übersicht ist in Tabelle 3.9 gezeigt.

Erweiterbarkeit hinsichtlich Protokolle. Wie bereits in den Szenarien deutlich wurde, ist eine Erweiterbarkeit der SBI hinsichtlich nutzbarer Protokolle wichtig. MOs in FSNs nutzen in ihrer API meist kein standardisiertes Protokoll, wodurch das SBI unterschiedliche gängige und aber auch unübliche Protokolle unterstützen muss. Diese Anforderung ist *wichtig*, da sie nur in speziellen, sehr homogenen FSNs nicht notwendig ist.

Erweiterbarkeit hinsichtlich Austauschformate. Nicht nur die Erweiterbarkeit von Protokollen, sondern mindestens genauso wichtig ist die Erweiterbarkeit der SBI hinsichtlich Austauschformaten. Auch hier ist die Anforderung als *wichtig* zu erachten.

Push- und Pull-Mechanismen. In FSNs muss sich eine Managementplattform nach den Funktionen von MOs des gemanagten Netzes richten. Diese stellen üblicherweise eine fixe API oder anderweitige Schnittstelle bereit und bieten ganz unterschiedlich Umsetzungen einer Push- oder Pull-basierten Übertragung an. Entsprechend muss auch eine Managementplattform in FSNs das *Gegenstück* dazu anbieten können. Die Anforderung ist daher *essenziell*.

Authentifizierung und Autorisierung SBI. Die Sicherheit einer FSN-Umgebung hängt (wie beispielsweise in Anforderung 3.3.5 beschrieben, jedoch auch in den anderen Szenarien relevant) insbesondere auch von einer Anbindung mit authentifizierten MOs an die Managementplattform ab. Angreifer könnten ein kompromittiertes System in das gemanagte FSN integrieren und an die Managementplattform einbinden und gefälschte Informationen über den Zustand der FSN-Umgebung senden, wodurch automatisch ungeeignete Steuerungsmaßnahmen provoziert werden könnten. Auch wenn nicht alle MOs in FSNs Authentifizierungsmechanismen unterstützen, muss zumindest eine Managementplattform bereits solche anbieten und Autorisierungsmechanismen vorsehen. Die Anforderung ist daher als *essenziell* anzusehen.

Sichere Kommunikation SBI. Die Anbindung zwischen Managementplattform und MOs muss zudem im Sinne einer gewährleisteten Vertraulichkeit und Integrität sicher gestaltet sein, sofern MOs diese unterstützen. Auch diese Anforderung ist generell aus allen Szenarien motiviert, besonders jedoch für solche mit sehr sensiblen Informationen wie in Szenario 2 beschrieben. Die Anforderung ist als *essenziell* anzusehen.

3.6.7 Anforderungen an die Umsetzung des Organisationsmodells

Das Organisationsmodell ist unter speziellem Fokus auf Föderationen von besonderer Bedeutung im Management. Im Folgenden werden aus den Szenarien abgeleitete Anforderungen an ein szenarienübergreifendes Organisationsmodell in FSNs abgeleitet und in Tabelle 3.10 zusammengefasst. Die Anforderungen sind gruppiert nach Schwerpunkten in der Gesamtföderation als auch für domänenspezifische Aspekte.

ID	Kurzbeschreibung	Gewicht	Quelle
Anforderungen im föderierten Kontext			
O.1	Heterogene Managementkonzepte	2	A, S1, S3
O.2	Administrative Domänen	2	A
O.3	Zugriffssteuerung	3	S1, S2, S3
O.4	Nutzer- und Funktionskennungen	3	S1, S2, S3
O.5	Flexibilität	2	S1
O.6	Zugriffsrechte Managementanwendungen	3	S1, S2, S3
O.7	Funktionale Aufgabenorganisation	3	S1, S2, S3
O.8	Netzweite Vorgaben	2	S1, S3
O.9	Organisatorische Übersicht	2	S1
Anforderungen an Domänenmodelle			
O.10	Domänenüberschneidungen	3	A, S1
O.11	Domänenspezifische Rollen und Aufgaben	2	S1, S3
O.12	Koordination domänenübergreifender Aktivitäten	3	S1, S2, S3
O.13	Zentrale bzw. dezentrale Verwaltung von Domänen	2	S1, S2
O.14	Domänenspezifische Organisation	3	S1

Tabelle 3.10: Anforderungen an das Organisationsmodell.

3.6.7.1 Anforderungen im föderierten Kontext

Zugriffsteuerung. Eine Managementplattform verwaltet alle Managementinformationen und bietet alle notwendigen Managementfunktionen. Der Zugriff darauf muss eine in Föderationen geeignete Zugriffskontrolle durchführen und alle zentralen organisatorischen Aspekte wie Föderationsbeziehungen (z. B. Vertrauen zwischen Parteien, Kunden- und Anbieterbeziehung, etc.), Domänen und Zuständigkeiten berücksichtigen. Die Anforderung ist szenarienübergreifend notwendig und daher *essenziell*.

Nutzer- und Funktionskennungen. Die Anforderung nach Nutzer und Funktionskennungen begründet sich aus dem nutzerzentrierten Management in FSNs. Neben Nutzern stehen Anwendungen zur Automatisierung, beispielsweise dem zyklischen Versenden von Überwachungsdaten oder der Überprüfung und Forcierung von Richtlinien im Netz im Mittelpunkt. Für beide Nutzer, aber auch Funktionen, werden Entitäten benötigt. Die Anforderung ist *essenziell*.

Flexibilität. Die Organisation muss flexibel gestaltbar sein. Eine Zuständigkeitsverteilung in FSNs und Teilnetzen darin muss auch feingranular und schnell verteilt werden können, um lokal eingerichtete Schatten-IT (d. h., jeweils selbstbetriebene Dienste, vgl. Szenario 1 Abschnitt 3.3.2.2) durch einzelne Partner zu vermeiden. Die Anforderung ist nicht in allen Szenarien relevant und daher als *wichtig* eingestuft.

Zugriffsrechte Managementanwendungen. Die Zugriffsrechte von Managementanwendungen auf Funktionen der Managementplattform müssen fallspezifisch festlegbar sein. Die Zugriffsrechte sollten derart konfiguriert werden, dass Nutzer der Managementanwendungen die damit verbundenen Aufgaben im Management erfüllen können, ihre Rechte nicht aber die Zugriffsrechte eines Nutzers unautorisiert erweitern. Die Anforderung ist in allen Szenarien notwendig und daher *essenziell*.

Funktionale Aufgabenorganisation. Aufgaben und damit verbundene Berechtigungen müssen in großen FSNs, vor allem durch die weitere Ordnung durch Domänen einfach organisierbar sein. In den Szenarien wurde gezeigt, dass Aufgaben geeignet durch funktionale Bereiche gegliedert werden können. Jeder Nutzer kann dann, je nach Assoziation zum jeweiligen Aufgabenbereich, bestimmte damit verknüpfte MAPPs nutzen. Lediglich die Ausprägung der funktionalen Bereiche kann je nach Fokus variieren. Sie müssen variabel definierbar sein. Die Anforderung ist daher *essenziell*.

Netzweite Vorgaben. Organisationsstrukturen wie *Strukturen zur funktionalen Aufgabengliederung* müssen im Sinne einer gemeinsamen Nutzung des Netzes für alle Domänen vorgegeben werden können. Eine gewisse optionale Kontrolle des Managements ist daher trotz mehrerer Partner, Ziele und Aufgaben sinnvoll, wie beispielsweise in Szenario 1 beschrieben wurde (vgl. Abschnitt 3.3.1.1). Die Anforderung ist daher *wichtig*.

Organisatorische Übersicht. Gerade in komplexen FSNs ist für jeden Nutzer eine Übersicht über seine Verantwortlichkeiten, von ihm abhängige administrativen Domänen, Gruppen und Aufgaben notwendig, wie in Szenario 1 deutlich wurde (vgl. Abschnitt 3.3.2.3). In kleineren Szenarien (z. B. Szenario 3) ist diese Anforderung weniger relevant, wodurch die Anforderung als *wichtig* eingestuft ist.

3.6.7.2 Anforderungen an Domänenmodelle

Domänenspezifische Rollen und Aufgaben. Nicht jede Rolle und Aufgabe im Netzmanagement ist notwendigerweise in jeder Domäne vertreten. Wie besonders in Szenario 2 (Abschnitt 3.4.4.2) beschrieben, können sich Domänen auch stark in dem darin umzusetzenden Aufgabenbereich unterscheiden. Auch in Szenario 3 (Abschnitt 3.5.2.2) wurden Aufgaben eines MO auf mehrere Domänen aufgeteilt. Derartige Aspekte müssen in einer Managementplattform und auch in dafür geeignete Frameworks berücksichtigt werden. Da diese Anforderung nicht uneingeschränkt domänenübergreifend, jedoch in vielen Fällen relevant ist, ist sie als *wichtig* eingestuft.

Zentrale bzw. dezentrale Verwaltung von Domänen. Szenario 1 und Szenario 2 unterscheiden sich vor allem in dem Aspekt der Domänenverwaltung stark voneinander. In Szenario 1 (siehe Abschnitt 3.3.2.3) und Szenario 3 (siehe Abschnitt 3.5.2.1) können Domänen beispielsweise auch anwendungsfallabhängig für einen konkreten Nutzer oder einer kooperierenden Gruppe von Nutzern flexibel erstellt werden. In Szenario 2 ist die Struktur hingegen klar zentral und praktisch unveränderlich vorgegeben. Ein Framework / Design für Managementplattformen in FSNs muss daher beide Ansätze unterstützen können. Die Anforderung ist daher *wichtig*.

Koordination domänenübergreifender Aktivitäten. Wie bereits auch in den allgemeinen Charakteristika von FSNs am Anfang dieses Kapitels beschrieben, setzen sich FSN üblicherweise aus vielen Virtualisierungsschichten zusammen. Diese sind ebenfalls in den organisatorischen Aspekten (wie in Szenario 1 in Abschnitt 3.3.1.1 gezeigt) überaus relevant, da potenziell dadurch ebenfalls Domänen aufeinander aufsetzen und somit Abhängigkeiten schaffen. Der Ausfall einer virtuellen IT-Ressource auf einer Ebene kann potenziell darauf aufbauende Ressourcen und Domänen komplett *abschalten*. Vor allem für kritische Updates, Wartungsarbeiten, usw. muss daher eine domänenübergreifende Organisation stattfinden, um eine Kaskadierung von Problemen und z. B. Ausfällen handhabbar zu machen. Diese Anforderung ist *essenziell* und in allen Szenarien zu berücksichtigen.

Domänenspezifische Organisation. Gegenüber netzweiten Vorgaben sollte auch die Organisation und Verwaltung von Domänen über domänenspezifische Vorgaben beschreibbar sein. Teilweise gibt es in Szenarien einen Besitzer einer Domäne, der diese eingerichtet hat. Weitere nachträglich eingefügte Manager sollten daher beispielsweise in ihren Managementaufgaben eingeschränkt werden, um zu breit gestreute, unklar gestaltete Berechtigungen zu vermeiden, wie es in stark heterogenen Domänen wie in Szenario 1 notwendig ist (vgl. Abschnitt 3.3.2.6). Diese Anforderung ist entsprechend *essenziell*.

3.6.8 Anforderungen an die frameworkspezifische Umsetzung

In den letzten Abschnitten wurden zentrale Anforderungen betreffend eines geeigneten Designs von Managementplattformen für szenarienübergreifende Anwendungsfälle von FSNs beschrieben. Daneben wurden in Abschnitt 3.1.2.2 weitere Anforderungen betreffend der frameworkspezifischen Umsetzung beschrieben. Diese werden analog zu den vorherigen Kategorien von Anforderungen in Tabelle 3.11 zusammengefasst.

ID	Kurzbeschreibung	Gewicht	Quelle
U.1	Dokumentation	1	A
U.2	Unterstützung der Implementierung	1	A
U.3	Langlebigkeit des Frameworks	2	A
U.4	Erweiterbarkeit des Frameworks	2	A
U.5	Grad der Designfreiheit	1	A
U.6	Nutzerfreundlichkeit	2	A

Tabelle 3.11: Anforderungen an die Umsetzung der Frameworks.

3.6.9 Zusammenfassung der Kernanforderungen

In jedem der vier Teilmodelle wurden durch die allgemeinen Charakteristika des Betriebs und der Szenarien Anforderungsschwerpunkte herausgearbeitet, die im Management von FSNs im Vordergrund stehen. Anforderungen an das **Funktionsmodell** adressieren insbesondere eine Automatisierung und Echtzeitfähigkeit von Managementkernprozessen, die durch die Größe und Dynamik der Netze und darin zu verarbeiteten Daten aufkommen. Darüber hinaus ist die Adaptierbarkeit von Managementfunktionen notwendig, die sich szenarienspezifisch ändert. Im **Informationsmodell** erfordert die hohe Dynamik durch die vielen Parteien eine unbedingte Datenaktualität als Grundlage für Managemententscheidungen. Außerdem führt der Föderationsaspekt zur Erhöhung der Heterogenität der gemanagten IT-Infrastruktur, wodurch eine starre Definition von MOCs, Managementbeziehungen und Netzereignissen und -Status nicht szenarienübergreifend zweckdienlich ist. Sie müssen eine hohe und einfache Adaptierbarkeit

unterstützen und können kaum eingeschränkt werden – anders als beispielsweise Teilspezifika des Organisationsmodells. Das Informationsmodell nimmt daher eine Schlüsselrolle im Netzmanagement von FSNs ein, wie die allgemein sehr hohe Gewichtung der Anforderungen verdeutlicht. Im **Kommunikationsmodell** beeinflusst die Heterogenität ebenfalls Schnittstellencharakteristika: Das SBI muss die Anbindung heterogener MOs und die API die Bereitstellung von Management- und MO-Funktionen unterstützen. Ein weiterer Aspekt ist die Unterstützung einer flexiblen szenarienspezifisch geeigneten (in der Regel verteilten) Architektur einer Managementplattform. Damit verbunden ist auch die sichere Umsetzung von Schnittstellen. Im **Organisationsmodell** ist vor allem die Modellierbarkeit von organisatorischen Strukturen wie Parteien, Domänen, Aufgaben und Zuständigkeiten notwendig. Diese müssen auch entsprechend in Mechanismen zur Zugriffskontrolle in einem modellübergreifenden Ansatz zum Informationsmodell berücksichtigt werden können. Ein außerdem zu berücksichtigender Aspekt sind heterogene Managementkonzepte der Föderationsparteien. Das Organisationsmodell nimmt neben dem Informationsmodell einen weiteren Schwerpunkt im Management von FSNs ein, um den Aspekt der Föderation entsprechend zu berücksichtigen.

Kapitel 4

Gegenwärtiger Stand der Forschung und Technik

Inhalt

4.1 Kriterien zur Auswahl der untersuchten Vorarbeiten 96
4.2 Standards zu Netzmanagementframeworks 96
4.2.1 SNMP-Management-Framework 96
4.2.2 NETCONF und YANG 98
4.2.3 ETSI Zero Touch Network & Service Management (ZSM) 100
4.2.4 Standards aus dem Cloud-Computing: CIMI und OCCI 102
4.2.5 Standards für Netzereignisse und -Zustände 104
4.3 Managementplattformen für Software-Defined Networking 106
4.3.1 OpenDaylight (ODL) 106
4.3.2 Open Network Operating System (ONOS) 109
4.4 Managementplattformen für Network Functions Virtualization 113
4.4.1 Übersicht über NFV-Plattformen 113
4.4.2 Open Source MANO (OSM) 113
4.5 Paradigmenübergreifende Managementplattformen 118
4.5.1 XOS (CORD) 118
4.5.2 OpenStack 121
4.5.3 Open Network Automation Platform (ONAP) 127
4.6 Zusammenfassung der Auswertung 131

FSNs setzen sich aus unterschiedlichen Teilbereichen zusammen und werden in vielen Anwendungsfällen wie dem Cloud-Computing genutzt. Dementsprechend existieren zahllose verwandte Konzepte und Implementierungen von Plattformen zur Überwachung und Steuerung ähnlicher Netze. In Abschnitt 4.2 wird auf einige relevante Standards mit Relevanz für FSNs eingegangen. Sie behandeln meist nicht alle vier Teilmodelle von Managementarchitekturen, sondern nur einzelne Teilbereiche. Den Hauptteil dieses Kapitels bildet die Bewertung bestehender Managementplattformen aus dem Bereich SDN in Abschnitt 4.3, aus dem Bereich NFV in Abschnitt 4.4 sowie paradigmenübergreifend in Abschnitt 4.5. Eine Gegenüberstellung und Bewertung wird schließlich in Abschnitt 4.6 vorgenommen. Für die relevantesten bestehenden Plattformen wird die Erfüllung der Anforderungen in Tabelle 4.1 am Ende dieses Kapitels übersichtlich zusammengefasst.

4.1 Kriterien zur Auswahl der untersuchten Vorarbeiten

In dieser Analyse bestehender Vorarbeiten wird der Fokus auf bestehende Standards, besonders aber auf Konzepte und Plattformen gelegt. Grundvoraussetzung ist die Erfüllung der folgenden Kriterien:

1. Sie haben einen erkennbaren **ganzheitlichen Bezug** zu SNs.
2. Sie berücksichtigen **Föderationen** oder Teilaspekte darin.
3. Sie erfüllen grundlegende Charakteristika einer **Netzmanagementplattform**, insbesondere
   - ihre Funktion als Plattform und Infrastruktur für Managementanwendungen, sowie
   - die Unterstützung der Kernprozesse zur Überwachung und Steuerung der Infrastruktur.
4. Des Weiteren werden Konzepte und Systeme berücksichtigt, die aktuell durch eine allgemeine **Praxisrelevanz** Bedeutung im Netzmanagement haben.

In den folgenden Abschnitten werden dahingehend wichtige Standards und bestehende Managementplattformen auf ihre Erfüllung der im letzten Kapitel definierten Anforderungen untersucht. Auf relevante Vorarbeiten zu spezifischen Teillösungen wird direkt im Konzept in Kapitel 5 eingegangen.

4.2 Standards zu Netzmanagementframeworks

Standards beeinflussen nicht unwesentlich das Netzmanagement von FSNs. Insbesondere bei Schnittstellen- und Datendefinitionen zwischen Komponenten sind diese von Bedeutung.

4.2.1 SNMP-Management-Framework

Im Management von Netzkomponenten sind das *Simple Network Management Protocol* und -Framework sowie damit verbundene Strukturen wie *Management Information Bases* (MIB) und die *Structure of Management Information* (SMI) auch heute noch weit genutzte Standards. Das immer noch genutzte Grundkonzept ist inzwischen mehr als 30 Jahre alt; die letzte umfassendere Revision fand um das Jahr 2002 statt. Virtuelle Software-definierte Netze im heutigen Ausmaß gab es nicht. Das SNMP-Management-Framework (im Folgenden kurz als SNMP bezeichnet) ist besonders hinsichtlich seines Kommunikationsmodells und Informationsmodells von Bedeutung. Aspekte des Funktionsmodells sind in SNMP offengelassen. Auch das Organisationsmodell ist minimal und berücksichtigt beispielsweise Föderationen und administrative Domänen nicht.

4.2.1.1 Simple Network Management Protocol

Das Grundkonzept von SNMP ist in RFC 1157 [53] beschrieben und stellt eine Schnittstelle zur gemanagten IT-Infrastruktur dar (**K.3**). Demnach verfolgt es einen einfachen UDP-basierten Manager-Agent-Ansatz (**K.5**), der sich insbesondere durch einen sehr überschaubaren Satz an Operationen wie `get`, `get-next`, `set`, `trap` und `get-bulk` auszeichnet. Seit Version 3 unterstützt es Integritätsschutz, Verschlüsselung und Authentifizierung (vgl. [39]) (**K.19**, **K.20**). Seit SNMPv2 ist zudem auch Manager-zu-Manager-Kommunikation möglich, wodurch beispielsweise hierarchische Managementstrukturen, z. B. in großen Netzen, aufgebaut werden können (vgl. [54], [55]). Generell erlaubt SNMP Push–basierte Kommunikation (**K.18**).

In der Praxis ist die Bedeutung von SNMP für FSNs begrenzt. Virtuelle SDN-Switches und SDN-Controller nutzen in der Regel kein SNMP. Sehr breit angedachte Ansätze wie *OpenDaylight* (vgl. [56]) oder *ONOS* (vgl. [57]) bieten in begrenzter Form SNMP-Module. Im Bereich der Virtualisierung wird SNMP ebenfalls teilweise genutzt, beispielsweise für das Toolkit *libvirt* [58], das nicht nur viele gängige Hypervisors, sondern beispielsweise auch Containerlösungen wie *LXC* unterstützt.

4.2.1.2 Management Information Bases (MIB)

Das Management durch SNMP basiert auf Informationen mehrerer spezifischer MIB bzw. MIB-Module. Die aktuelle MIB-II beschreibt nach [59] essenzielle Elemente mit Fokus auf Fehler- und Konfigurationsmanagement. Eine Vielzahl weiterer MIBs für SNMP wird einerseits in RFCs, andererseits beispielsweise von Netzkomponentenherstellern definiert. Die MIBs-Abdeckung von SNMP für FSNs und darin zentrale Komponenten ist nicht vollständig und weist aktuell unterschiedliche Schwerpunkte auf. So gibt es im Bereich Virtualisierung für Hypervisors und VMs seit 2015 die verallgemeinernde RFC 7666-Spezifikation [60] ebendieser Komponenten, welche auf Basis unterschiedlicher herstellerspezifischer MIBs sowie der von libvirt erstellt wurde. MIBs für gängige Hypervisors sind vorhanden. Die Abdeckung im für das Netzmanagement wichtigen Bereich der SDN-Controller und NFV-MANO-Komponenten ist praktisch kaum vorhanden.

4.2.1.3 Structure of Management Information (SMI)

Die Sprache zur Beschreibung von Objekten einer MIB ist SMIv2 [61] aus dem Jahr 1999. Der Kern der Erweiterbarkeit ist dabei eng verwandt mit dem Konzept von MIBs: SMIv2 erlaubt die Beschreibung und Integration neuer Module in eine MIB-Baumstruktur, welche im *Object Identifier Tree* der ITU-T und ISO / IEC dauerhaft, eindeutig und geräteübergreifend adressierbar und in anderen Modulen wiederverwendbar sind. Jeder Knoten in einer MIB stellt ein MO dar. Der einzige weitere komplexe Datentyp (neben MOs selbst) sind Tabellen, auf deren Zeilen mittels eines Schlüssels oder einer Schlüsselkombination zugegriffen werden kann. Die Zugreifbarkeit auf ein MO kann in SMIv2 in geordneten Stufen festgelegt werden. Auch die Beschreibung des Status eines MO ist möglich, beispielsweise um ein Objekt als aktuell oder als veraltet zu kennzeichnen. Neben klassischen MOs unterstützt SMIv2 auch die Beschreibung von bedingten proaktiven Benachrichtigungen von Agenten zu Managern.

Die Beschreibung des Informationsmodells in FSNs durch SNMP ist ungeeignet. Es unterstützt eine eindeutige Adressierung von MOs über OIDs in einer MIB und die Adressierung von Instanzen über entsprechende Modelle. In der Praxis lassen sich diese Konzepte aber nicht gut auf FSNs übertragen, da die Fülle an bestehenden und neuen (Open-Source-) Komponenten in FSNs diese nicht berücksichtigen, beispielsweise entsprechende MIBs und OIDs nicht bereitstellen. Die Vielschichtigkeit von FSNs und Abhängigkeiten von MOs untereinander sowie geographische und föderale Aspekte werden dabei nicht explizit berücksichtigt und entsprechende MIBs sind nicht verfügbar. Über eine entsprechende Modellierung sind diese Aspekte aber denkbar (**I.6**, **I.10** und **I.2** jew. teilweise). Gleiches gilt für Netze und Netzkomponenten untereinander (**I.9** teilweise). Die Modellierung von Netzereignissen und -Zuständen wird nicht explizit berücksichtigt. Auch eine Normalisierung von Informationen ist nicht explizit gegeben und wird beispielsweise durch fehlende Objektorientierung erschwert.

4.2.1.4 Relevanz für Managementplattformen in FSNs

Die Rolle von SNMP ist in FSNs deutlich geringer als in herkömmlichen Netzen; vielmehr werden in FSNs andere Protokolle und Managementkonzepte umgesetzt. So bieten praktisch alle FSN-spezifischen Systeme eine eigene API und Agenten werden nicht mehr benötigt. Der Netzverkehr und Netzkomponenten werden über an SNs angepasste Protokolle wie OpenFlow, das OVSDB oder NETCONF (vgl. Abschnitt 4.2.2) konfiguriert. Wie in Abschnitt 4.2.2.2

erläutert wird, bestehen zudem Kompatibilitätsprobleme mit neueren Modellierungssprachen wie YANG, wodurch insgesamt eine Verwendung des SNMP-Informationsmodells für FSNs ausgeschlossen wird. Dennoch unterstützen viele Netzkomponenten (z. B. zur Weiterleitung von Netzverkehr) weiterhin SNMP, sodass es als Schnittstelle zur gemanagten IT-Infrastruktur auch in FSNs berücksichtigt werden muss.

4.2.2 NETCONF und YANG

Das *Network Configuration Protocol* (NETCONF) kann als einer der *Nachfolger* von SNMP angesehen werden. Es zielt vor allem auf die Unterstützung von Automatisierung und den Anspruch als ganzheitliche Lösung für das Management von Netzkomponenten (**K.2**) ab. YANG ist sowohl ein Modell als auch eine Sprache zur Abbildung von NETCONF-Daten. Der Fokus liegt auf dem Informations- und Kommunikationsmodell.

4.2.2.1 NETCONF

NETCONF definiert nach [62] einerseits Operationen zur Steuerung, Bearbeitung oder zum Entfernen von Konfigurationsattributen, andererseits zum Auslesen von schreibgeschützten Statusattributen auf Netzkomponenten. Es ist auf einer Client-Server-Architektur aufgebaut, wobei der NETCONF-Server auf den eigentlichen gemanagten Netzkomponenten installiert ist. Es unterscheidet beim Austausch generell Daten auf vier aufeinander aufbauenden Schichten in der Netzkommunikation: Die unterste Ebene *Secure Transport* umfasst den eigentlichen Transport der Nachrichten, beispielsweise über SSH oder HTTP bzw. SOAP über TLS mit Authentifizierung, Vertraulichkeit und Integritätsschutz (**K.19**, **K.20**). Die *Messages*-Schicht darüber beschreibt das Protokoll für den Austausch von RPC-basierten Steuernachrichten und von Notifications (**K.18**). Darüber setzt die *Operations*-Schicht mit NETCONF-Operationen auf. Die darüberliegende *Content*-Schicht umfasst schließlich Daten über den Status und die Konfiguration, sowie Benachrichtigungen von Netzkomponenten.

Das Konzept für die Konfiguration sieht mindestens einen Datenspeicher pro Netzkomponente vor: Alle NETCONF-Server müssen einen *running*-Datenspeicher haben, welcher den aktuellen Zustand und die aktuelle Konfiguration der jeweiligen Netzkomponente beschreibt. Daneben haben NETCONF-kompatible Netzkomponenten einen optionalen *candidate*-Datenspeicher, durch welchen der zukünftige Zustand beschrieben wird. Über die *commit*-Operation wird der *running*-Datenspeicher dem *candidate*-Datenspeicher angeglichen. Auch unterstützt NETCONF Sicherheitsmechanismen wie *confirmed-commits* mit notwendiger Bestätigung des Commits innerhalb einer vorgegebenen Zeitspanne und Zurücksetzen im negativen Fall. Ein dritter Datenspeicher ist der *startup*-Datenspeicher, gemäß dem eine Netzkomponente bei Neustart konfiguriert wird. In RFC 8342 [63] wurde darüber hinaus ein *intended*-Datenspeicher eingeführt, der Daten aus dem *running*-Datenspeicher bezieht und den nächsten Zustand einer Netzkomponente darstellt.

NETCONF unterstützt einen Satz einfacher Operationen. Neben dem Auslesen, Bearbeiten, Kopieren und Löschen eines ganzen Datenspeichers unterstützt es auch die Sperrung (*Locking*) eines Datenspeichers für eine Nutzer-Session für exklusiven Zugriff. Da sich über unterschiedliche Netzkomponenten (unterschiedliche Netzkomponententypen, aber auch Hersteller und Gerätemodelle) auch das Erscheinungsbild der Datenspeicher sowie unterstützte Funktionen ändern, werden unterstützte Modelle in Form von *Capabilities* ausgetauscht. Ein NETCONF-Client kann entsprechend Capabilities eines NETCONF-Servers abfragen.

4.2.2.2 YANG

YANG selbst hat nicht den Anspruch, alle Daten im Netzmanagement modellieren zu können, sondern es ist in erster Linie zur Modellierung von NETCONF-Daten ausgelegt. Es wird grundlegend in RFC 6020 [64] sowie im Rahmen von Erweiterungen im RFC 7950 [65] definiert.

Demnach entstand es auf Basis von Vorarbeiten, der *Next Generation Structure of Management Information* (SMIng). Sie zielen darauf ab, die in SNMP genutzte SMIv2 zu überarbeiten.

YANG unterstützt vor allem die Modellierung der Konfiguration und des Status von Netzkomponenten, die Definition von RPC-Operationen in NETCONF einer Netzkomponente (**I.5** teilweise) sowie Benachrichtigungen. Eine explizite Unterstützung von FSN-spezifischen Komponenten ist jedoch nicht gegeben (**I.4** teilweise). Die Erstellung eines Modells basiert auf einer hierarchischen Baumstruktur, welche von vorhandenen Strukturen abgeleitet wird. Die Grundstruktur von YANG-Modellen wird über versionierte Module und Submodule vorgegeben. Module und Submodule können Modelle aus anderen Modulen importieren. Als Baumstruktur ist ein YANG-Modell jedoch kreisfrei und gerichtet. Notifications können entweder alleinstehend innerhalb eines (Sub-)Moduls oder als Teil einer komplexeren Struktur definiert werden und setzen sich selbst wiederum aus Datentypen und Elementen zusammen (**I.11**, **I.12**, **I.14**). Zur Modellierung eigener Datentypen wie einer IP-Adresse, Ports oder CPU-Load stellt YANG eine Reihe an Basistypen bereit. Sie können in ihren Wertebereichen oder durch Mustervorgaben eingeschränkt werden. Um entsprechende Flexibilität bei der Wiederverwendung zu gewährleisten, erlaubt YANG eine Modifikation bestehender Datentypen und bedingte Erweiterung (z. B. gebunden an Attributwerte) von YANG-Modellen. Gemäß [64] können so herstellerspezifische Besonderheiten berücksichtigt werden. Ein Vorschlag zur Modellierung komplexerer Strukturen wie mehrschichtiger Netze wurde beispielsweise in RFC 8345 [66] beschrieben (**I.1**, **I.8**, **I.9**, **I.10**). In RFC 8641 [67] wird zudem eine YANG-Erweiterung für das Abonnement von Benachrichtigungen beschrieben. In der Überarbeitung der Sprache in [65] ist ein Sprachelement zur Beschreibung von Aktionen hinzugekommen. Im Gegensatz zu RPC-Aufrufen operiert eine Aktion auf einer spezifischen Objekt- oder Listenstruktur und ist an diese gebunden. Die Spezifikation von YANG berücksichtigt demnach ebenfalls eine Beschreibung von YANG-Modellen in einer verlustfreien XML-Notation, welche *YANG Independent Notation* (YIN) genannt wird. So können beispielsweise vorhandene und gängige XML-Parser mit YANG-Modellen umgehen. Zudem ist eine Abwärtskompatibilität zu SMIv2 berücksichtigt worden: Modelle, die in SMIv2 beschrieben wurden, sind ebenso mit YANG beschreibbar. Andersherum ist diese Eigenschaft nicht unbedingt gegeben.

4.2.2.3 Weitere Protokolle

Gemäß [65] hat sich YANG nicht ausschließlich im Kontext von NETCONF etabliert: *RESTCONF*, das *Constrained Application Protocol* (CoAP) sowie *Constrained Management Interface* (CoMI) verwenden YANG ebenfalls. Die letzteren beiden Ansätze dienen der Interaktion von IT-Systemen mit beschränkten Ressourcen (z. B. 8-Bit-Mikrocontroller mit geringer Netz-Bandbreite [68]) und weisen daher keine direkte Relevanz zur Thematik auf. RESTCONF ist in RFC 8040 [69] beschrieben. Das angestrebte Ziel ist eine geeignetere Integration derzeit vorzugsweise eingesetzter Webanwendungen in das Netzmanagement, insbesondere in die Abfrage und Konfiguration sowie Steuerung von Netzkomponenten. Webanwendungen bieten einige Vorteile, die insbesondere auch im Kontext von Managementanwendungen für das Netzmanagement nützlich sind. Beispiele sind eine hohe Reaktivität, zentrale Zugriffspunkte im Netz (z. B. über Webbrowser) und eine hohe Entwicklungsdynamik. Im Kontext von Webanwendungen wird in RESTCONF ein HTTP-basierter REST-Ansatz verfolgt. Eine Netzkomponente muss neben dem NETCONF-Server auch einen RESTCONF-Server betreiben. Der Funktionsumfang von RESTCONF ist dabei gegenüber NETCONF teilweise eingeschränkt.

4.2.2.4 Bewertung

NETCONF und RESTCONF werden insbesondere zur Kommunikation mit der gemanagten IT-Infrastruktur genutzt. Sie stellen daher vor allem eine Standardisierung des SBI dar, können aber nicht allein dazu verwendet werden. Darüber hinaus werden sie üblicherweise in SDN-Controllern mit sehr großem Funktionsspektrum wie OpenDaylight oder ONOS implementiert und haben somit kaum Berührungspunkte zu Netzmanagementplattformen. YANG

wird in OpenDaylight als zentrale Modellierungssprache genutzt. Eine Bewertung findet entsprechend in diesem Kontext in Abschnitt 4.3.1.2 statt. Durch seine Flexibilität ist YANG jedoch eine potenziell geeignete Basis zur Informationsmodellierung in FSNs.

4.2.3 ETSI Zero Touch Network & Service Management (ZSM)

Das von der ETSI beschriebene ZSM-Framework ist vor allem an Anwendungsfällen aus 5G ausgerichtet. Es zielt auf ein automatisiertes Ende-zu-Ende-(E2E)-Netzmanagement (**F.1**) in programmierbaren, software- und servicebasierten ganzheitlichen Netzarchitekturen ab [70]. ZSM wird in mehreren Spezifikationsdokumenten beschrieben; einige wie ETSI GS ZSM 008, 009-2 und -3 sind noch nicht zugänglich. ZSM berücksichtigt insbesondere NFV-basierte IT-Infrastruktur in einem föderierten Netz mit mehreren Betreibern. Der Fokus liegt vor allem auf der Festlegung einer Referenzarchitektur (Kommunikationsmodell) und notwendiger Managementdienste (Funktionsmodell) sowie organisatorischer Aspekte. Auf das Informationsmodell wird lediglich sehr abstrakt eingegangen. Die beschriebenen Komponenten sind häufig bereits auf Ebene eines Managementsystems und weniger auf der einer Managementplattform beschrieben. Gleichzeitig beschreiben die Spezifikationen eher Anforderungen an ZSM-konforme Umgebungen als konkrete Lösungen.

4.2.3.1 Kommunikationsmodell

Das Kernstück der Spezifikationen bildet die in [71] definierte ZSM-Referenzarchitektur. Darin besteht die gemanagte IT-Infrastruktur aus geographisch verteilten IT-Ressourcen (**I.2** teilweise) aus mehreren Managementdomänen, die über eine Kommunikationsinfrastruktur gekoppelt sind (**NF.11**). Für die gesamte IT-Infrastruktur gibt es eine domänenübergreifende *E2E Service Management Domain* (**NF.7**), in der E2E-Dienste verwaltet werden, die sich aus Diensten der einzelnen Managementdomänen zusammensetzen. Demnach sind notwendige Dienste auf Inter-Domänenebene beispielsweise solche zur Registrierung und Abmeldung und zum Finden von Managementdiensten, außerdem Wege zur Nutzung von Diensten mittels synchroner und asynchroner Kommunikation (**K.2**, **K.11**). Die konkrete Absicherung der Kommunikation ist nicht näher spezifiziert. Besonders die Integrität und Vertraulichkeit soll jedoch in einem ZSM-konformen System geschützt werden (**K.14**, **K.15**, **K.20**, **F.11**). Aspekte wie sparsame Kommunikation sind auf Anforderungsebene berücksichtigt (**NF.12**).

Ein Schwerpunkt in der Definition des ZSM-Frameworks liegt auf der Definition von Managementdiensten und damit verbundenen Prozeduren und Interaktionen zwischen Diensten und Nutzern, die zur Automatisierung des Netzmanagements notwendig sind. Das ZSM-Framework eignet sich daher weniger als Vorlage zur Implementierung von Managementplattformen, als mehr von Managementsystemen. Dabei wird auch auf die Absicherung des Zugriffs auf Managementdienste auf Anforderungsebene eingegangen.

4.2.3.2 Organisationsmodell

ZSM berücksichtigt in Szenarien inhärent Föderationen und die Kollaboration mehrerer Parteien wie mehrere Technologielieferanten und Kunden [72]. Technologielieferanten stellen beispielsweise Systeme des Zugangsnetzes (z. B. über Telekommunikationsinfrastruktur), aber auch dahinterliegender Transport- und Kernnetze sowie Dienste darin bereit. In ZSM stellt jeder der Lieferanten eine eigene Managementdomäne dar. Allgemein ist eine Managementdomäne in ZSM nach [71] ein getrennter Verantwortungsbereich; Managementdomänen können demnach zwar ineinander geschachtelt sein, jedoch wird in ZSM keine Überschneidung von Domänen behandelt.

Die Ausgangslage der betrachteten föderierten Szenarien in ZSM bildet nach [73] eine Vertrauensbeziehung zwischen den einzelnen Parteien bzw. Managementdomänen (**NF.15** teilweise). In dem Standard wird jedoch nicht wesentlich auf unterschiedliche Ebenen des Vertrauens

und damit verbundenen organisatorischen Maßnahmen eingegangen. Vielmehr wird impliziert, dass ohne bestehendes Vertrauen eine Föderation nicht möglich ist. Vertrauen muss demnach auf Basis einer Bedrohungs- und Risikoanalyse in Verbindung mit Sicherheitsmaßnahmen (z. B. Zugriffskontrolle) evaluiert und hergestellt werden. Eine derartige Sicht ist in den im letzten Kapitel definierten Szenarien nicht unbedingt stets anwendbar. In ZSM fehlt entsprechend eine differenzierte Möglichkeit zur Modellierung von Vertrauens- und anderen Föderationsbeziehungen sowie ihre Auswirkungen auf das Netzmanagement.

Ein weiterer in [73] berücksichtigter Aspekt ist die Mandantenfähigkeit des ZSM-Frameworks (**NF.14**). Demnach werden Managementdienste durch Nutzer mehrerer Mandanten verwendet, deren möglichen böswillige Ausnutzung entgegengewirkt werden muss, um sensible Daten des Managements zu schützen. ZSM empfiehlt zur Lösung einen richtlinienbasierten Ansatz: Demnach sollen mandantenspezifische Informationen und Richtlinien – genannt werden eine Sicherheitsrichtlinie sowie eine Richtlinie zur Isolation und zum Zugriff auf Ressourcen – in jeder Managementdomäne verwaltet werden. Die Managementmöglichkeiten eines Nutzers des jeweiligen Mandanten in einer Domäne werden entsprechend der definierten Richtlinien bereitgestellt. Die Form der Richtlinien wird nicht näher spezifiziert. Ein mit der Mandantenfähigkeit entsprechend eng verzahntes Thema ist die Zugriffskontrolle auf Managementdienste (**F.7**). Die Zugriffskontrolle soll demnach für jede Managementdomäne unterschiedlich, gleichzeitig jedoch im Gesamtzusammenhang berücksichtigt werden: Eine flexibel adaptierbare Zugriffskontrolle ist in ZSM notwendig. Auch die Änderung des Vertrauensverhältnisses zwischen Parteien wird hier beispielhaft als Fall genannt, der die Zugriffskontrolle beeinflussen kann. Eine wichtige Rolle spielt auch die Nachverfolgbarkeit: ZSM fordert die Anfertigung von Sicherheits-Logs für jeden Zugriff (**NF.2**).

4.2.3.3 Funktionsmodell

Der Schwerpunkt des ZSM-Frameworks liegt in der Automatisierbarkeit des Netzmanagements. Die Spezifikation zur Referenzarchitektur [71] definiert dazu, genauso wie andere Dokumente des Standards, notwendige konkrete Dienste. Spezifika einer Managementplattform werden dagegen weitestgehend auf Anforderungsebene beschrieben. In ZSM wird nach [74] eine Managementfunktion als Verallgemeinerung eines Managementdienstes in den Rollen eines Konsumenten oder Produzenten beschrieben. Ein Managementdienst hingegen realisiert demnach Managementfunktionalität und entspricht in dieser Arbeit daher einer Managementanwendung. Die ZSM-Referenzarchitektur [71] unterscheidet zwischen domänenspezifischen und domänenübergreifenden Managementdiensten des E2E-Servicemanagements. Es unterscheidet sich daher von dem in dieser Arbeit verfolgten Ansatz domänenparametrisierter Managementanwendungen. In ZSM besonders berücksichtigte Managementdienste sollen für das Performance-, Fault- und Configurationmanagement berücksichtigt werden, darüber hinaus die Sammlung von Daten zur Rückverfolgung, Log-, Topologie- und Inventardaten. Das Securitymanagement wird lediglich implizit adressiert durch die Beschreibung notwendiger Dienste zur Sammlung von Sicherheitsereignissen innerhalb der Managementdomänen und ihrer Auswertung im Intra- und übergreifenden Domänen-Kontext. Schwerpunkte domänenübergreifender Managementdienste im E2E-Dienstmanagement liegen auf der Orchestrierung und Entscheidungsfindung, der Anomalie-, Fehler-, Performanz- und Sicherheitsanalyse.

Die Automatisierung wird getrennt von der Referenzarchitektur in [75] beschrieben. Eine echtzeitfähige Überwachung wird in den Szenarien von ZSM betrachtet (vgl. [72]), in der Referenzarchitektur jedoch nicht konkretisiert (**NF.1** teilweise, **NF.6**). Mittel zur Automatisierung werden nicht eingeschränkt oder konkret vorgegeben. Vielmehr werden mögliche Ebenen aufgezeigt: Als niedrigste Ebene wird eine Policy-basierte Automatisierung beschrieben, in der automatisch auf definierte Auslöser (definierte Situationen) mit vorgegebenen Aktionen reagiert wird. Ein darauf aufbauender Automatisierungsmechanismus wird demnach durch Intent-basierte und modellgetriebene Automatisierung realisiert. Möglichkeiten der Automatisierung, wie sie im ZSM-Framework für das Netzmanagement beschrieben werden, sind

auf Ebene eines Managementsystems beschrieben (vgl. die Dienste zum Policy-Management in [71]). Auch in dieser Arbeit wird die Implementierung eines spezifischen Automatisierungsansatzes, wie Policy- oder Intent-basiertes Management, mehr als Teil des Managementsystems denn der Managementplattform aufgefasst. Diese Position verstärkt aber die Notwendigkeit einer Managementplattform zur Bereitstellung von grundlegenden Schnittstellen dafür.

4.2.3.4 Bewertung

Das ZSM-Framework kann zeitlich und inhaltlich als auf dem NFV-Standard aufbauend angesehen werden. Verglichen mit den in dieser Arbeit betrachteten FSNs, die durch Szenarien im letzten Kapitel beschrieben wurden, ist die im ZSM-Framework definierte Referenzarchitektur ein Spezialfall eines FSN. Einige in ZSM definierte Managementprinzipien (vgl. [71]) überschneiden sich stark mit denen in dieser Arbeit, beispielsweise *Extensibility*, *Scalability* (**NF.9**), *Closed-Loop Management Automation*, *Resilience* (**NF.10**) und *Automation* allgemein. Andere weniger, wie die *Separation of Concerns* (über Managementdomänen). Unter Berücksichtigung des letzteren Aspekts kann das ZSM-Framework als FSN ohne Domänenüberschneidungen angesehen werden. Weitere Managementprinzipien wie *Modularity*, *Model-driven, open interfaces* und *Intent-based interfaces* verdeutlichen bereits, dass sich das ZSM-Framework an Managementsysteme richtet; Aspekte von Managementplattformen werden häufig nur auf Ebene von Anforderungen beschrieben. Das ZSM-Framework kann entsprechend verwendet werden, um auf Basis darin beschriebener Funktionalität, Dienste, Prozesse und Verfahren ein Managementsystem für automatisiertes Netzmanagement zu entwickeln. Seine Eignung als Managementplattform in FSNs lässt sich jedoch nicht bewerten, da es auf einige Aspekte darin – beispielsweise die Modellierung der gemanagten IT-Infrastruktur, die Umsetzung von Managementfunktionen, die Modellierung der Organisation und die Behandlung heterogener Managementkonzepte sowie die Umsetzung von Kommunikationsschnittstellen nicht konkret eingeht.

4.2.4 Standards aus dem Cloud-Computing: CIMI und OCCI

Aufgrund der Überschneidungen von FSNs mit IT-Infrastrukturen des Cloud-Computings gibt es in diesem Bereich relevante Managementstandards: Im Cloud-Computing wichtige Anbieter-Nutzer-Beziehungen werden auch in dieser Arbeit als konkrete Spezialisierung einer Föderation angesehen. Zudem bilden IT-Ressourcen aus FSNs oft Kernkomponenten in IaaS-Infrastrukturen des Cloud-Computings.

4.2.4.1 Cloud Management Infrastructure Interface (CIMI)

Die *Cloud Management Infrastructure Interface* (CIMI) der DMTF ist nach [76] ein Ansatz für das Lifecycle-Management von IaaS-Umgebungen. Es besteht im Kern aus einem HTTP- bzw. REST-basierten Protokoll zur Kommunikation zwischen IaaS-Anbieter und -Nutzern sowie einem umfangreichen Modell zur Abbildung von IaaS-typischen Komponenten. In CIMI wird das Verhältnis zwischen IaaS-Anbieter zu -Nutzer als geschäftliche Beziehung angesehen und berücksichtigt daher sehr rudimentär Föderationsaspekte (**NF.15**). Der Hauptteil von CIMI bezieht sich auf das Informationsmodell und teilweise auch das Kommunikationsmodell im Netzmanagement.

Das in CIMI beschriebene Protokoll dient demnach insbesondere zur Ausführung von klassischen CRUD-Operationen (Create, Read, Update, Delete) durch IaaS-Nutzer auf Modellrepräsentationen von IaaS-Infrastrukturkomponenten eines Anbieters (**K.3**). Außerdem unterstützt das Protokoll weitere Funktionen wie ein einfaches Paging (bzw. die Begrenzung und Auswahl der übertragenen Menge an Elementen), der Einschränkung bezüglich Inhaltskategorien, der Auswahl von referenzierten Inhalten (d. h. Angabe welche Referenzen in Elementen detailliert beschrieben werden sollen, daher ist **NF.5** erfüllt) oder auch das Sortieren von *Collections* (vgl. folgender Absatz). Zudem unterstützt es sowohl eine synchrone als auch asynchrone Kommunikation (**K.18**).

Das in CIMI definierte Modell zur Beschreibung von Ressourcen in IaaS-Infrastrukturen beschreibt diese umfangreich in Form von Objekten mit jeweils essenziellen Attributen und üblichen Operationen (**I.13**, **I.4**, **I.5**). CIMI erlaubt auch die Modellierung von Beziehungen zwischen Ressourcen durch Referenzierung (**I.8**). Die darauf aufbauend modellierten Ressourcen überschneiden sich teilweise mit denen, die auch in SNs eingesetzt werden. Diese umfassen beispielsweise *System* (im Sinne einer Zusammenstellung von Cloud-Ressourcen als zusammenhängende Anwendung bzw. Ressourcen, daher **I.9** erfüllt) und virtuelle Maschinen, ein Netz sowie Kommunikationsendpunkte und spezielle Netzfunktionen (in CIMI als *Network-Service* bezeichnet). Hinzu kommen noch im Cloud-Computing übliche *Template*-Ressourcen (z. B. MachineTemplate). Auch werden Ressourcen aus dem Netzmonitoring wie eine Datenquelle, einfache Überwachungsmetriken sowie die Beschreibung eines Ereignisses berücksichtigt (**I.11**). Ein Ereignis ist in unterschiedliche Subtypen gegliedert: Es erlaubt die Spezialisierung als Zustand, Fehlermeldungen (**I.12**), Hinweise über Änderungen des Modells sowie Zugriffsmeldungen. Mehrere Ereignisse werden in einem *EventLog* zusammengefasst und beispielsweise mit einer anderen damit in Beziehung stehenden Ressource verknüpft (**I.14**). Eine besondere Ressource in CIMI stellt *ResourceMetadata* dar, durch welche IaaS-Anbieter fallspezifische Randbedingungen und Besonderheiten an IaaS-Nutzer kommunizieren können. Diese betreffen beispielsweise benutzerdefinierte Attribute oder Operationen genauso wie die Einschränkung ebendieser in ihrer Verfügbarkeit für bestimmte Nutzer. Darüber ist es ebenfalls möglich, die Notwendigkeit zur Unterstützung von, sowie Möglichkeit des Zugriffs auf ein bestimmtes Attribut zu spezifizieren (**O.3**). Modellseitige Erweiterungen erlaubt CIMI dabei nicht nur durch einen IaaS-Anbieter, sondern ebenfalls durch IaaS-Nutzer, welche eigene benutzerdefinierte Eigenschaften zu Ressourcen hinzufügen können, die durch IaaS-Anbieter hinterlegt und für spätere Zugriffe verfügbar gemacht werden.

CIMI ist für FSNs weniger von praktischer Bedeutung, jedoch eher durch umfangreiche Modellbeschreibung von Cloud-Ressourcen, die sich auch mit FSNs überschneiden. Für ein vollständiges Informationsmodell fehlt jedoch insbesondere die Berücksichtigung des SDN-Paradigmas sowie der Vielschichtigkeit von FSNs. Hinsichtlich der Nutzung ist das darin betrachtete Anbieter-Nutzer-Verhältnis im Kontext von IaaS-Umgebungen zu einfach gestaltet, wodurch für eine generalisierte Betrachtung ein spezifischeres Organisationsmodell (welches in CIMI praktisch nicht vorhanden ist) notwendig ist. In diesem Kontext wird beispielsweise auch nicht auf die im Cloud-Computing ebenfalls wichtige Mandantenfähigkeit im Detail eingegangen.

4.2.4.2 Open Cloud Computing Interface (OCCI)

Verglichen mit CIMI verfolgt das *Open Cloud Computing Interface* (OCCI) eine zunächst generische (**U.4** und **U.3**), einfache (**U.6**) und freie (**U.5**) Möglichkeit zur Modellierung von IT-Ressourcen, die insbesondere aus dem IaaS-Dienstmodell des Cloud-Computings stammen. Der zentrale konzeptionelle Inhalt in Form eines einfachen Kommunikationsmodells und eines grundlegenden durch Vererbung und Klassifikation erweiterbaren Informationsmodells ist in OCCI in [77] zusammengefasst (**U.1**). Insgesamt besteht OCCI demnach aus einer Vielzahl an Erweiterungen, die das Informations- und Kommunikationsmodell von IaaS-, jedoch auch z. B. PaaS-Infrastrukturen, konkretisieren.

Das in OCCI beschriebene Kernmodell zur Modellierung von IT-Ressourcen besteht gemäß [77] lediglich aus drei Wurzelelementen. Neben *Attribute* bietet es *Entity* zur Darstellung von Ressourcen und Verknüpfungen, sowie *Category* zur Klassifizierung von Entitäten. Von *Entity* abgeleitete Klassen sind *Resource* als Entsprechung einer IT-Ressource sowie *Link*, das zur Herstellung einer Abhängigkeit (**I.10**) zwischen IT-Ressourcen dient. Je nach Anwendungsfall werden davon notwendige spezifischere Entitäten im Informationsmodell abgeleitet (**I.4**, **I.8**, **I.9**). OCCI bietet durch die Beschreibung dieser Kernelemente des IaaS-Dienstmodells aufgrund der Paradigmenüberschneidung eine gute Grundlage auch für ähnliche Elemente in FSNs. Weitere Cloud-spezifische Ressourcen-Profile sind darüber hinaus in [78] beschrieben. Attribute und Operationen (**I.5**) werden über die Klassifikation durch Zuweisung von Kindklassen von *Ca-*

tegory zu einer konkreten *Entity* hinzugefügt. Attribute sind entsprechend an *Kind* und *Mixin* gebunden, die die jeweils zugewiesene Instanz von *Entity* verwendet. Auf diese Weise wird gemäß [77] eine Erweiterbarkeit zur Laufzeit erreicht (**NF.3** teilweise erfüllt, da bezogen auf das Informationsmodell). Das Informationsmodell ist zudem einfach erweiterbar: Die Ableitung und Beschreibung konkreter MOs würde durch Spezialisierung von *Resource* und *Link* stattfinden; die Klassen *Kind* und *Mixin* werden explizit nicht spezialisiert, sondern geeignet instanziiert und mit Attributen sowie Aktionen verknüpft und schließlich mit Instanzen der Kindklassen von *Entity* assoziiert. Auch zusätzliche Attribute werden durch die gleichnamige Klasse immer einheitlich mittels Instanziierung beschrieben. OCCI erlaubt die Einschränkung des Wertebereichs von Attributen und Default-Werten. Daneben berücksichtigt OCCI zumindest teilweise eine Unterstützung von automatisierten Managementabläufen: In der Erweiterung zur Beschreibung von Service-Level-Agreements in [79] erlaubt OCCI die Festlegung von Arten und Grenzwerten von Richtlinien, sowie eine Referenzierung von Gegenmaßnahmen (auch in Form von ausführbaren Dateien) bei Verletzung einer Richtlinie. Eine Adressierung von Ressourcen findet über pro Cloud-Anbieter eindeutige URIs statt, wodurch Föderationsaspekte hier nicht berücksichtigt sind.

Das Kommunikationsmodell in OCCI wird in [77] nur stark abstrahiert dargestellt. Demnach betrachtet es die hauptsächlich HTTP-basierte Kommunikation zwischen Cloud-Anbieter und -Nutzer. Die Kommunikation zwischen Cloud-Plattform und den eigentlichen Ressourcen wird nicht näher konkretisiert, sodass diese Schnittstelle sich eher auf eine API einer Managementplattform und Managementanwendungen übertragen lässt (**K.2**). Die durch OCCI standardisierte HTTP-basierte Schnittstelle wird in [80] näher beschrieben (**K.9**). Sie bietet REST-Funktionen wie Paging oder Filterung von Informationen. Eine Absicherung über TLS und Authentifizierung wird lediglich empfohlen (daher **K.14** und **K.15** teilweise erfüllt). Die Darstellung von OCCI-Inhalten bei der Übertragung wird über sog. *Renderings* beschrieben (z. B. eine Serialisierung über JSON [81]).

4.2.5 Standards für Netzereignisse und -Zustände

Als Netzereignisse und -Zustände werden in dieser Arbeit Informationen des Netzmanagements bezeichnet, die keine Netzkomponenten darstellen. Beispiele sind Ereignismeldungen über einen mit Malware infizierten Host im Netz oder ein Modell der Topologie des gemanagten Netzes zu einem bestimmten Zeitpunkt. Im Folgenden werden aktuelle Standards mit Föderationsbezug bewertet. Sie befassen sich vor allem mit der Modellierung derartiger Informationen (Informationsmodell) und ihrem Austausch (Kommunikationsmodell).

4.2.5.1 STIX und TAXII

Die *Structured Threat Information Expression* (STIX) und *Trusted Automated Exchange of Intelligence Information* (TAXII) beschreiben zwei offene, zusammenhängende Standards des *OASIS Cyber Threat Intelligence Technical Committee* zur maschinenlesbaren Modellierung und dem Austausch von sicherheitsrelevanten Informationen. Vor allem mit Fokus auf einen interorganisationalen Austausch von Informationen adressiert **STIX** erweiterbare Modellierungsaspekte für Informationen. Die aktuellste Spezifikation von STIX ist in Version 2.1 in [82] beschrieben. Die Nutzung bestehender standardisierter Teilkomponenten ist gegenüber benutzerdefinierter Komponenten empfohlen, da diese von vielen Organisationen nutzbar sind (**I.11** teilweise, **I.13**). Auch wenn der Fokus von STIX nicht explizit Netzmanagement (in FSNs) ist, behandelt es ein Teilgebiet des Informations- und Kommunikationsmodells mit Fokus auf Sicherheitsinformationen. Demnach dient STIX der Beschreibung, der Verarbeitung und dem Austausch von Bedrohungs- und damit verknüpften Inhalten. STIX-Modelle sind dazu konzipiert, einfach zwischen Organisationen ausgetauscht zu werden. Dafür ist in STIX Version 2.1 JSON als Transportformat für die Serialisierung der Modelle vorgesehen. STIX-Modelle bestehen aus unterschiedlichen Objektgruppen zur Beschreibung von Bedrohungen und Angriffen, Netzressourcen und Beziehungen zwischen beiden vorherigen Gruppen. Auf diese Weise entstehen zusammenhän-

gende Graphen zur Beschreibung von Bedrohungsinhalten (z. B. einem Botnetz). Die Standardisierung erfolgt auch durch ein vorgegebenes Vokabular.

Gemäß [83] ist **TAXII** für den Austausch von STIX-kodierten Modellen ausgelegt, jedoch nicht ausschließlich darauf beschränkt. Generell zielt es auf einen skalierbaren Austausch von Informationen ab und lässt sich eher dem NBI von Managementplattformen zuordnen als dem SBI (**K.2**). TAXII bietet dazu zwei unterschiedliche Mechanismen: Zum einen *Collections* für einen Request-Response-Ansatz (Pull), zum anderen *Channels* mit einem Publish-Subscribe-Ansatz (Push) (**K.11**). TAXII baut auf HTTP auf und basiert wie STIX auf JSON (**K.9**). TAXII bietet Funktionen wie das Filtern oder Sortieren und Gruppieren ausgetauschter Informationen, oder die Limitierung einer Anfrageantwort (**NF.12**, **K.12**). Ein TAXII-Server und -Client muss TLS Version 1.2 verwenden. Eine Authentifizierung über das HTTP Basic-Authentication-Verfahren wird serverseitig empfohlen, ist jedoch nicht zwingend notwendig, anders als clientseitig, bei der zumindest eine entsprechende Unterstützung unbedingt notwendig ist (**K.14**, **K.15**). Hingegen ist eine Verifikation von Server- oder Client-Zertifikaten nicht zwingend vorgeschrieben, sondern lediglich eine clientseitige Verifikation des Serverzertifikats empfohlen. TAXII ist erweiterbar und erlaubt Nachrichten, um beliebige benutzerdefinierte vorgabenkonforme Attribute zu ergänzen. Diese können jedoch von TAXII-Servern abgelehnt, oder, wie auch von TAXII-Clients, ignoriert werden (**K.8** teilweise, da nur eingeschränkt).

4.2.5.2 Cloud Auditing Data Federation (CADF)

Cloud Auditing Data Federation (CADF) ist nach [84] ein Standard der CADF zur ressourcenschonenden Modellierung (**NF.5** teilweise, da nur für das Informationsmodell gültig) von Netzereignissen aus dem Cloud-Computing. Es dient der Übertragung von Netzereignissen (**I.11**) von Cloud-Anbietern zu Cloud-Nutzern zum Zweck der Auditierung der Konformität bereitgestellter u. a. Netzinfrastruktur zu vereinbarten Policies (z. B. aus SLAs). Das CADF beschreibt ein Datenmodell sowie ein einfaches HTTP-basiertes Kommunikationsmodell. Die Modellierung von Konfigurationsdaten ist demnach explizit nicht berücksichtigt, genauso Aspekte des Alerting und Policy Enforcements.

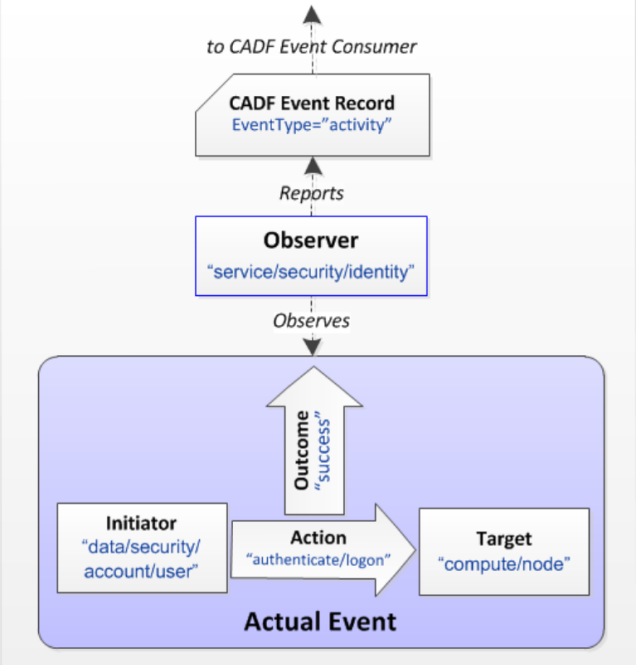


Abbildung 4.1: Genereller Aufbau eines CADF-Ereignisses mit Beispielbelegungen [84].

Die Übersicht über den generellen Aufbau eines CADF-Ereignisses ist in Abbildung 4.1 dargestellt. Ein Ereignis wird durch mehrere *Resource*-Komponenten beschrieben: Der *Initiator* unternimmt eine *Action* auf ein *Target*, wodurch ein *Outcome* entsteht. Diese werden potenziell durch ein *Measurement* (zusätzliche Informationen des *Target*) sowie ein *Result* ergänzt. Durch weitere Komponenten *Reporter* und *Reporterchain* werden Systeme beschrieben, die dann bei der Beschreibung oder dem Transport eines Ereignisses in Form eines *CADF Event Record* beigetragen haben. *Observer* ist beispielsweise ein spezieller *Reporter*, der ein Ereignis beobachtet und initial über einen *CADF Event Record* beschrieben hat. Ein wesentlicher Teil der Standardisierung liegt in der Definition von Taxonomie-Gruppen und komponentenspezifischem Vokabular. Die in Abbildung 4.1 belegten Beispielwerte der unterschiedlichen Komponenten entsprechen Elementen dieser Taxonomien. Die Taxonomie ist für die Teilbereiche in FSNs nur teilweise geeignet. So sind Ressourcen der Virtualisierung gut abgebildet, SDN-spezifische Ressourcen fehlen dagegen. Gleiches gilt für die CADF Action Taxonomy, die auch Aktionen für Virtualisierungskomponenten (z. B. `deploy`, `enable`, `disable`, usw.) beschreibt (**I.5** teilweise). Die Taxonomien sind erweiterbar konzipiert, verlieren dabei jedoch ihren Standardisierungscharakter. In Verbindung mit dem Grundmodell für CADF-Ereignisse erfolgt so eine Normalisierung von Ereignissen (**I.13**). CADF berücksichtigt die Adressierbarkeit (**I.3**) und geographische Verteilung von IT-Ressourcen (**I.2**).

4.2.5.3 Bewertung von STIX, TAXII und CADF

Seit STIX 2 wurde die Beschreibung von Bedrohungsinhalten mittels JSON deutlich übersichtlicher und einfacher gestaltet als in vorherigen Versionen. Die Modellierung von Inhalten wird jedoch nur unzureichend unterstützt und Konzepte wie Vererbung können nicht genutzt werden, wodurch es als primäres Modellierungsformat in Managementplattformen nicht ideal ist. STIX kann als Vorlage zur Modellierung von Informationen aus dem Securitymanagement in einer Managementplattform dienen. Andere Informationen, beispielsweise aus dem Fault- oder Performancemanagement werden nicht bereitgestellt. Einsatzmöglichkeiten für Managementplattformen sind am SBI oder East- / Westbound-Interface einer Managementplattform. TAXII wird daher eher als eines mehrerer zu unterstützender Protokolle angesehen denn als universell und einziger Kommunikationskanal.

Die Praxisrelevanz des CADF ist in FSNs gemäß der Übersicht aus [85] relativ gering. Das einzige bereitgestellte Profil ist in diesem Fall für OpenStack in [86] definiert und seit einigen Jahren nicht mehr überarbeitet worden. OpenStack selbst wird hochaktiv weiterentwickelt. Diese Form der Standards zeigt am praktischen Beispiel die Nachteile derartiger umfangreicher Standards in Form einer schnellen Alterung und aufwändigen Überarbeitung.

4.3 Managementplattformen für Software-Defined Networking

Auch wenn SDN lediglich ein Teilaspekt von FSNs ist, entstanden in diesem Kontext SDN-Controller, die über den Anspruch der bloßen Steuerung von Netzen hinausgingen und vielmehr ein ganzheitliches Management von Netzen und Netzinfrastruktur versprechen. Im Folgenden werden OpenDaylight und ONOS als zwei der allgemeinsten Plattformen hinsichtlich ihrer Verwendung für das Netzmanagement in FSNs bewertet.

4.3.1 OpenDaylight (ODL)

Eine als SDN-Controller entstandene, jedoch inzwischen in einem bei weitem breiteren Managementkontext einsetzbare Plattform ist OpenDaylight (ODL). Die Besonderheit von ODL ist einerseits die modellgetriebene Programmierbarkeit und andererseits der modellgetriebene Managementansatz [87]. ODL strebt entsprechend die Bereitstellung einer zentralisierten, programmierbaren Überwachung und Steuerung von Netzkomponenten an (**NF.7**, **U.4**) [88]. ODL

stellt zwar Möglichkeiten zur Überwachung und Steuerung von Netzen bereit, jedoch fehlt eine Möglichkeit zur Automatisierung dieser Kernprozesse (**F.1** teilweise).

4.3.1.1 Kommunikationsmodell

ODL ist ein weitestgehend plattformunabhängiges Java-basiertes System aus unterschiedlichen zusammenarbeitenden Microservices, die auf Apache Karaf als OSGi-Laufzeitumgebung aufgesetzt sind (**NF.4**). Karaf erlaubt dabei unter anderem die dynamische Einbindung von Funktionserweiterungen von ODL zur Laufzeit (**F.3**) aus unterschiedlichen (lokalen oder entfernten) Repositories (insb. Maven) und das Management der einzelnen Microservices. Karaf bietet auch bereits einige Sicherheitsfunktionen wie RBAC-basierten Zugriffsschutz [89]. Ebenfalls ist innerhalb der Karaf-Umgebung der Zugriff auf bereitgestellte Schnittstellen der Microservices möglich, wodurch eine (jedoch nicht abgesicherte) netzunabhängige API bereitsteht (**NF.10**).

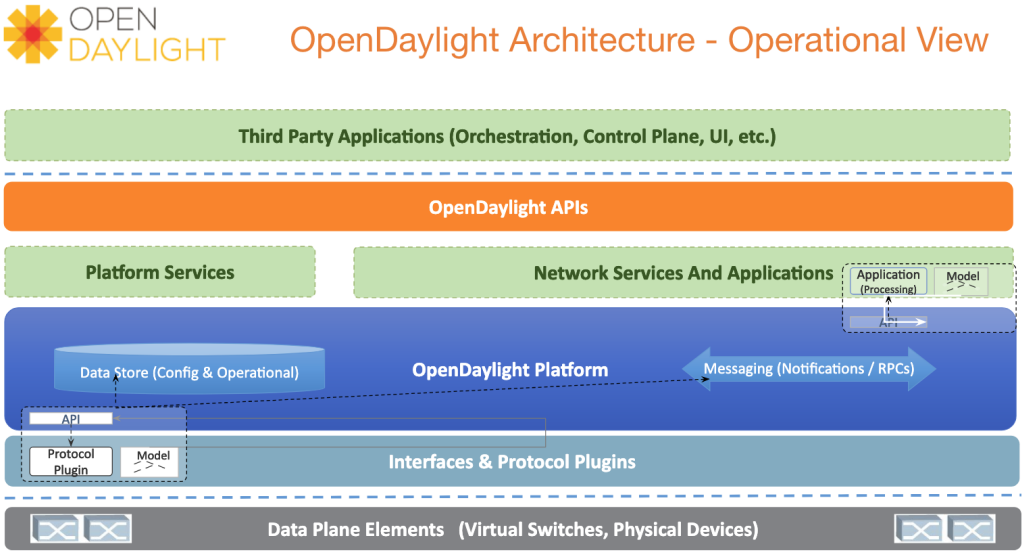


Abbildung 4.2: Architektur von OpenDaylight 12 [90].

Eine Übersicht über die grobe Architektur von ODL ist in Abbildung 4.2 gezeigt. Demnach handelt es sich bei ODL um eine agentenlose Plattform (**NF.8**), welche im Kern aus einem Datenspeicher sowie einem Framework zur Kommunikation von Plugins, Anwendungen und Plattformdiensten besteht. In der Abbildung wird deutlich, dass ODL die Einbindung zum einen von Schnittstellen und Plugins zur gemanagten Infrastruktur, zum anderen von Plattform- und Netzdiensten sowie Netzanwendungen unterstützt (**K.2**, **K.3**). Es stellt einen modellbasierten Ansatz für die Definition von Schnittstellen und APIs bereit. Eine detaillierte Beschreibung der modellgetriebenen Programmierung in ODL wird im nächsten Absatz vorgenommen. Die Schnittstellen und REST-basierte API (**K.9**) erlauben den Zugriff externer Anwendungen auf die von ODL bereitgestellten Funktionen. Sowohl die API als auch das SBI sind flexibel erweiterbar (**K.8**, **K.16**, **K.17**, **K.5**). Generell erlaubt ODL auch Push-und-Pull im SBI (**K.18**), in der REST-basierten API liegt der Fokus hingegen mehr auf einem Pull-basierten Ansatz via HTTP. Das Abonnement asynchroner Benachrichtigungen über die API auf Basis von Websockets ist jedoch auch möglich (**K.11**) [91].

Gemäß [92] ist der zentrale Datenspeicher in ODL, der *Data Tree*, entsprechend seiner Bezeichnung als Baumstruktur aufgebaut. Zustandsinformation von Netzkomponenten bzw. dem Netz selbst werden darin in Form von Baumelementen und Teilbäumen dargestellt. Es wird grundlegend zwischen zwei Teilbäumen unterschieden: Der *Operational Data Tree* beschreibt den Ist-Zustand von System- und Netzkomponenten, wohingegen der *Configuration Data Tree* deren

Soll-Zustand beschreibt. Der Zugriff auf die Datenspeicher wird über ODL-Plugins (bzw. Microservices) ermöglicht, die wiederum entsprechende Funktionen an Managementanwendungen implementieren können (**F.9**). Jedes Element und jeder Teilbaum im Data Tree ist durch einen eindeutigen Identifikator adressierbar (**I.3**). Der Datenspeicher kann transient als auch persistent umgesetzt werden. Generell besteht auch die Möglichkeit, einen benutzerdefinierten Datenspeicher zu verwenden (**K.6**).

Der Informationsaustausch zwischen Anwendungen wird über unterschiedliche Kommunikationsmechanismen realisiert. ODL unterstützt nach [92] ein einfaches Rollenmodell, in dem zwischen einem Daten-*Producer* und einem Daten-*Consumer* unterschieden wird. Eine einfache Unicast-Kommunikation erfolgt über eine Anfrage von einem Consumer an einen Producer mit asynchroner Antwort durch den Producer. Daneben unterstützt ODL Multicast-Austausch in Form von *Notifications* (direkt von Producer zu Consumer) sowie *Data Change Events*, welche anders als Notifications von dem in ODL integrierten *Data Broker* (koordiniert den Zugriff auf den Datenspeicher) für definierte Änderungen im Datenspeicher direkt an alle Abonnenten geschickt werden. Die Parallelisierung erlaubt eine asynchrone Kommunikation (**NF.1**, **NF.6**).

ODL unterstützt die Möglichkeit zum Clustering, d. h. dem Betrieb mehrerer ODL-Instanzen als zusammenhängendes System (**NF.11**, **K.1**, **NF.9**). Nach [93] wird Clustering durch ODL genutzt, um hohe Skalierbarkeit und Verfügbarkeit zu erreichen und Datenverlust durch einen Systemausfall zu vermeiden. Die Datenverwaltung wird in sogenannten *Shards*, d. h. Partitionen von Daten, welche in einem Cluster redundant ausgelegt sein können (**NF.10**), organisiert. Das Clustering selbst kann mit Hilfe von bereitgestellten Skripten initialisiert werden (**F.4** teilweise). Gemäß [94] ist die Absicherung der Clustering-Schnittstellen ein noch bekanntes offenes Problem, da weder Verschlüsselungs- noch Authentifizierungsmechanismen in die Kommunikation integriert sind (**NF.13** teilweise, da ohne Clustering nur interne Kommunikation). Anders sieht es generell bei der API aus. Hier bietet ODL für die API Authentifizierungsmechanismen und die Unterstützung von TLS. Das OpenFlow-Modul erlaubt beispielsweise eine TLS-Verbindung mit gegenseitiger Authentifizierung und NETCONF eine über SSH abgesicherte Verbindung (**K.14**, **K.15**, **K.19**, **K.19**).

4.3.1.2 Informationsmodell und Erweiterbarkeit

Die Kernkomponente zur Erweiterbarkeit von ODL ist der sogenannte *Model-Driven Service Adaptation Layer* (MD-SAL). Gemäß [92] dient er als Infrastruktur für benutzerdefinierte Daten und Schnittstellen zwischen ODL-Diensten. Die Besonderheit des MD-SAL ist, dass er über YANG (beschrieben in Abschnitt 4.2.2) modelliert werden kann und auch die Modellierung von FSN-spezifischen MOs (**I.4** teilweise sowie **I.2** teilweise, da nicht explizit unterstützt), Managementbeziehungen (**I.6**, **I.8**, **I.9**, **I.10**), Netzereignissen und -Zuständen (**I.11**, **I.13**, **I.14**) und Alarme erlaubt (**I.12**). ODL konzentriert sich jedoch hauptsächlich auf Steuerungsabhängigkeiten und weniger auf Realisierungsabhängigkeiten aus der Systemvirtualisierung (**I.1** teilweise). Auch die Modellierung und Normalisierung von MO-Funktionen ist möglich (**I.5**). In YANG definierte Schnittstellen werden direkt über einen RESTCONF-Connector auf der API zugänglich gemacht [92]. Sie müssen jedoch zuvor als Dienst implementiert werden, wobei die Entwicklung praktisch nicht eingeschränkt wird. Ein komponentenagnostischer Zugriff auf MO-Funktionen ist dadurch nur teilweise möglich, da das Durchreichen von MO-Funktionen von ODL so nicht vorgesehen ist, jedoch pro MO-Typ umgesetzt werden kann (**NF.3**).

4.3.1.3 Organisationsmodell

Die Organisation des Managements über ODL wird durch den Dienst *Authentication, Authorization and Accounting* realisiert, der wiederum auf *Apache Shiro* basiert (**O.3**, **O.6**, **O.8**). Nach [95] bildet es ein Framework zur Forcierung von Zugriffsregeln auf Ressourcen durch Nutzer. Die Authentifizierung und Autorisierung kann dabei flexibel (**O.5** teilweise) umgesetzt werden und ODL unterstützt demnach fünf verschiedene Implementierungen:

- *TokenAuthRealm*: Erlaubt die Modifikation von Zugriffsrichtlinien pro ODL-Knoten.
- *ODLJndiLdapRealm*: Nutzt zur Authentifizierung Daten eines LDAP-Dienstes, in dem Nutzergruppen in ODL-Rollen übersetzt werden.
- *ODLJndiLdapRealmAuthNOnly*: Wie *ODLJndiLdapRealm*, nur ohne Übersetzung von Rollen.
- *ODLActiveDirectoryRealm*: Verwendet den *ActiveDirectoryRealm*, der von Apache Shiro unterstützt wird.
- *KeystoneRealm*: Verwendet den auch in OpenStack (vgl. Abschnitt 4.5.2) genutzten *Keystone*-Dienst zur Authentifizierung.

Die Organisation über den Keystone-Dienst berücksichtigt die meisten Elemente, die auch in Föderationen vorkommen, wie sie in dieser Arbeit betrachtet werden. Demnach kann sich ein Nutzer über seine Kennung (**O.4**) und ein Passwort authentifizieren. Zudem wird nach administrativer Domäne unterschieden. Zugriffsregeln werden entweder dateibasiert für jeden ODL-Knoten einzeln oder MD-SAL-basiert konfiguriert. Demnach sind Ressourcen keine MOs, sondern ein URL-Muster mit jeweils für eine Rolle erlaubte HTTP-Aktionen (z. B. POST, GET, usw.). Zum einen bezieht sich das so umgesetzte Autorisierungskonzept lediglich auf API-Endpunkte (**F.7**) und ist nicht direkt auf MOs übertragbar, zum anderen spielen administrative Domänen in Autorisierungsrichtlinien keine Rolle mehr und sind daher nicht ausreichend berücksichtigt (**O.2** teilweise). Aspekte wie Mandantenfähigkeit sind nicht direkt durch ODL vorgesehen, könnten jedoch über den AAA-Dienst feingranular umgesetzt werden (**NF.14** teilweise).

4.3.1.4 Bewertung

ODL bietet eine sehr umfangreiche Dokumentation über unterschiedliche Webquellen an. Viele davon sind sehr versionsspezifisch, redundant und unübersichtlich (**U.1** teilweise erfüllt). Dazu existieren viele Versionen und Softwarebibliotheken – vor allem zusammenarbeitende Maven-Archetypen zur Unterstützung der Implementierung (**U.2** teilweise erfüllt) – sodass die Entwicklung und Adaptierung von ODL unintuitiv und langwierig ist. ODL als auf Apache Karaf aufgebaute Anwendung hat doch einige unmittelbare Vorteile: Karaf ist ein gut gepflegtes Framework, was insbesondere auch die Langlebigkeit von ODL unterstreichen kann (**U.3**). Auch unterstützt Karaf die Erweiterbarkeit der Funktionalität auch im laufenden Betrieb und das Management von Erweiterungen. Der Grad der Designfreiheit in ODL ist gut ausgewogen über einerseits eine praktisch beliebige Erweiterbarkeit, andererseits einem bereits bestehenden Satz von Standardanwendungen wie einem OpenFlow- und vielen weiteren Plugins (**U.5**). Die Freiheit, die ODL Entwicklern in der Implementierung lässt, ist sehr groß: Das gesamte System kann praktisch beliebig erweitert werden, wodurch die angesprochene Problematik der schwierigen Einarbeitung in ODL verstärkt wird. Daher ist eine Einschränkung und Fokussierung der erweiterbaren Schnittstellen von Vorteil für Frameworks für Managementplattformen in FSNs. ODL mangelt es in Hinblick auf Netzmanagement in FSNs vor allem an geeigneten organisatorische Strukturen für Föderationen. Hinsichtlich SNs hat ODL aber auch Defizite im Informationsmodell (z. B. Abhängigkeiten von Netzen, Netzebenen und ihren Komponenten).

4.3.2 Open Network Operating System (ONOS)

ONOS ist wie ODL eine hauptsächlich in Java und auf Apache Karaf und OSGi entwickelte microservicebasierte Plattform mit ähnlichen Architekturaspekten (**NF.4**, **U.3**, **U.4**, **NF.1**, **NF.8** erfüllt sowie **U.5** teilweise erfüllt). Gemäß [96] zielt ONOS auf das Management des gesamten Netzes und seiner Netzkomponenten ab. Es unterstützt Erweiterbarkeit der Funktionalität zur Laufzeit (**F.3**) und bietet Möglichkeiten zur Überwachung und Steuerung des Netzes. Es fehlt jedoch an Möglichkeiten der Automatisierung (**F.1** teilweise).

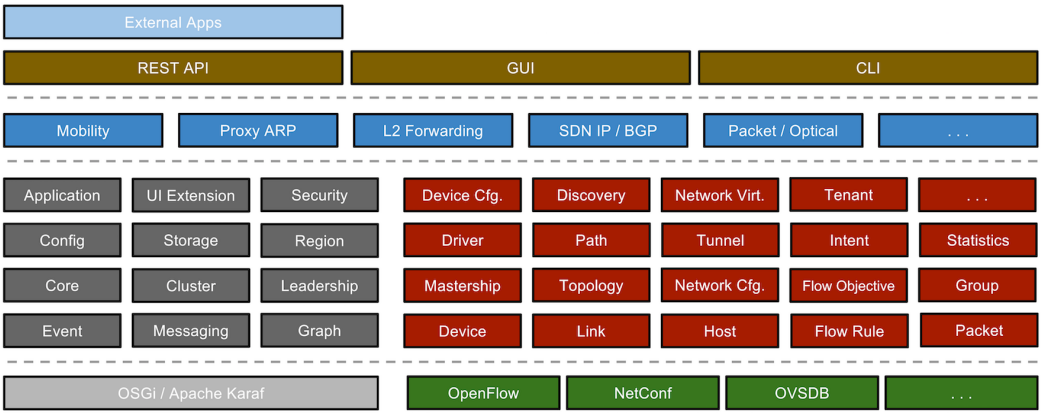


Abbildung 4.3: Übersicht über die Architektur und Dienste von ONOS [97].

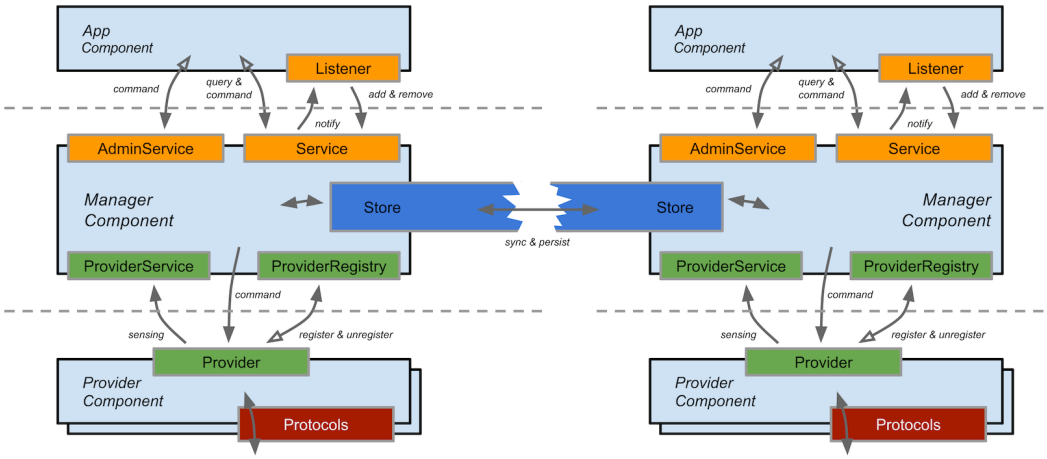


Abbildung 4.4: ONOS Subsysteme und Kommunikationswege [97].

4.3.2.1 Kommunikationsmodell

ONOS setzt sich nach [97] aus mehreren Diensten (auch als *Subsysteme* bezeichnet) zusammen, die unterschiedliche Funktionen implementieren. Eine Übersicht der einzelnen ONOS-Dienste ist in Abbildung 4.3 gezeigt. Beispielsweise ist das *Device Subsystem* für das Management von Infrastrukturgeräten, das *FlowRule Subsystem* hingegen für das Management von Flowregeln auf Geräten zuständig. Die Hauptschnittstelle zur Erweiterung von ONOS ist eine Java-API (**K.10**).

Die ONOS-Dienste sind durch Komponenten aus unterschiedlichen Schichten realisiert, wie in Abbildung 4.4 gezeigt wird. Nach [97] gibt es *Provider* zur Kommunikation mit der gemanagten IT-Infrastruktur (**K.3**), die das SBI bilden. Darüber liegen *Manager*-Komponenten des Systemkerns, die eine weitere Verarbeitung der Informationen übernehmen und darüber *App*-Komponenten, welche wiederum diese Informationen verarbeiten und potenziell Maßnahmen ergreifen können.

Wie in [98] beschrieben wird, unterstützt ONOS auch ein Clustering mehrerer verteilter Instanzen (**NF.9**, **NF.11**, **K.1**). Die Einrichtung eines Clusters ist über bereitgestellte Tools unterstützt (**F.4** teilweise). In einem Cluster sind einige Operationen wie die Verwaltung von ONOS-Applikationen logisch zentralisiert [99]. Durch die Umsetzung erlaubt ONOS darüber hinaus eine gewisse Fehlertoleranz gegenüber ausgefallenen Knoten in einem Cluster, die lediglich

neu gestartet werden müssen und sich anschließend selbst synchronisieren (**NF.10**). Durch die Java-API als Hauptschnittstelle zur Erweiterung von ONOS in Kombination mit einer via TLS absicherbaren Clusterkommunikation (vgl. [100]) kann die Gesamtarchitektur auf sichere Weise erweitert werden (**NF.13**).

Die Erweiterung des SBI durch *Provider*-Komponenten ist flexibel hinsichtlich Protokollen und Austauschformaten gestaltet (**K.16**, **K.17**, **K.18**, **NF.6**, **K.5**) und erlaubt die Umsetzung einer sicheren Kommunikation (**K.20**, **K.19**).

Der Zugriff auf ONOS ist über eine REST-API, eine webbasierte GUI und ein CLI möglich (**K.2**, **K.9**, **I.7**) [101]. Die REST-API basiert auf der Spezifikation der *Jakarta RESTful Web Services* (JAX-RS), wie aus der Implementierung[IV] von ONOS hervorgeht (Stand Mai 2022, ONOS 2.7). Sie ist entsprechend erweiterbar und flexibel gestaltbar (**K.9**, **K.8**, **K.13** teilweise). An der API gibt es keinen geeigneten Autorisierungsmechanismus und Nutzer müssen sich lediglich authentifizieren (**K.15** teilweise) [102]. Die interne Kommunikation zwischen ONOS-Diensten erlaubt dabei sowohl einen Push- als auch einen Pull-basierten Ansatz. In den Standard-API-Funktionen von ONOS ist lediglich eine Pull-basierte Kommunikation vorgesehen (vgl. [103]), eine Push-basierte Kommunikation wird jedoch von JAX-RS im Allgemeinen unterstützt (**K.11** teilweise).

4.3.2.2 Organisationsmodell

ONOS Organisationsmodell ist sehr einfach gehalten und generell nicht für föderiertes Netzmanagement angepasst. Im Kontext von Anwendungen beschreibt es einige organisatorische Strukturen wie eine *TenantId* zur Zuweisung von OpenFlow-Agents oder von virtuellen Netzen (**NF.14**, **I.6** teilweise). Auch definiert ONOS Domänen, denen Infrastrukturgeräte zugewiesen werden können (**O.2** teilweise). Diese Elemente dienen in ONOS jedoch hauptsächlich zur Strukturierung und haben keinen direkten Einfluss auf Zuständigkeiten oder Aufgabenbereiche.

Das Berechtigungsmodell von ONOS ist ebenfalls einfach gestaltet. Auf Ebene der REST-API gibt es keine Autorisierungsmechanismen, sondern lediglich eine Nutzerauthentifizierung (**O.4**) [102]. Im Kontext der Java-API bietet ONOS hingegen einen *Security-Mode*, durch den der Zugriff von ONOS-Diensten untereinander durch Richtlinien eingeschränkt werden kann. Der Security-Mode wird jedoch nicht mehr unterstützt und ist nicht ohne Weiteres lauffähig [104]. ONOS unterstützt entsprechend keine feingranulare Organisation des Managements, sondern erlaubt in gewissem Maß netzweite Vorgaben durch einfache Autorisierungsmechanismen (**O.8**).

4.3.2.3 Informationsmodell

ONOS bietet ein zentrales Informationsmodell im Java-Package `org.onosproject.net`, das wesentliche Elemente in Form von Java-Klassen beschreibt und entsprechend zur Entwicklung von ONOS-Diensten und zur Integration mit anderen ONOS-Diensten genutzt werden kann. Die bereitgestellten Modelle sind jedoch sehr generisch, beispielsweise für ein gemanagtes Netzinfrastrukturgerät, für ein an das Netz angeschlossenes Endgerät oder für eine Netzverbindung und nicht an typischen MOs, wie sie in FSNs vorkommen (vgl. Abschnitt 3.1.2.1) ausgerichtet. Da ONOS in erster Linie dem SDN-Paradigma zugeordnet werden kann und selbst eine SDN-Controller-Implementierung bereitstellt, ist jedoch im ursprünglichen Konzept ein Management von Dritt-SDN-Controllern nicht vorgesehen, wodurch sie nicht in Form einer MOC beschrieben sind. FSN-spezifische MOs sind jedoch in ONOS modellierbar (beispielsweise im Kontext eines benutzerdefinierten ONOS-Dienstes), jedoch fehlt eine geeignete Unterstützung

[IV] `https://github.com/opennetworkinglab/onos/tree/onos-2.7`

der Modellierung (**I.4** teilweise). Dasselbe gilt für Managementbeziehungen (**I.8** sowie **I.10** teilweise). Eine Adressierbarkeit von MOs (**I.3**), ihre geographische Verteilung (**I.2**) sowie Abhängigkeiten zwischen Netzkomponenten und Netzen (**I.9**) werden jedoch bereits unterstützt. Realisierungsabhängigkeiten oder Systemabhängigkeiten werden in ONOS nicht modelliert (**I.10** und **I.1** teilweise, da Netze unterstützt).

Auch bietet ONOS ein Konzept zur Modellierung von MO-Funktionen. Diese werden nach [105] durch einzelne Treiber mit jeweils *Behaviours* implementiert. Behaviours zielen demnach auf die Behandlung von ähnlichem Funktionsumfang von Geräteklassen ab und implementieren jeweils einzelne Aspekte des Funktionsumfangs. Durch Vererbung können Behaviours entsprechend für mehrere Geräte derselben Produktfamilie genutzt werden (**I.5**).

ONOS unterstützt kein vereinheitlichtes strukturiertes Modell für Netzereignisse und Netzzustände. Einzelne Implementierungen sind in Unterpaketen im Package `org.onosproject.net`, beispielsweise eine *Topology* bzw. ein *TopologyGraph* oder CPU-Load; diese haben jedoch kein zentrales Elternelement, das Netzereignisse oder -zustände generell beschreibt. Derartige Informationen müssen entsprechend von Entwicklern ohne Unterstützung angelegt werden (**I.11** und **I.13** teilweise, **I.14**). ONOS unterstützt jedoch die Modellierung von Alarmen (**I.12**, vgl. Java-Doc[V]).

4.3.2.4 Funktionsmodell

ONOS fungiert nach [96] insbesondere als *Netzbetriebssystem* mit zentraler Steuerungsmöglichkeit. Es ist weniger darauf ausgerichtet, andere Komponenten wie SDN-Controller zu managen, die eine bestimmte Funktion im Netz bieten. Vielmehr ist ONOS darauf ausgerichtet, derartige Funktionalität selbst zu implementieren. Aspekte der Bereitstellung eines komponentenagnostischen Zugriffs auf MO-Funktionen sind daher durch die Flexibilität möglich, jedoch nicht primär so vorgesehen (**NF.3** teilweise). Durch den modularen Ansatz von ONOS ist jedoch eine allgemeine Nutzung und Erweiterung der Plattform für unterschiedlichste Zwecke möglich.

Aus organisatorischer Sicht unterstützt ONOS keine Orientierung an Managementfunktionen und Aufgaben. Aufgrund der Defizite im Organisationsmodell ist daher auch eine funktionale Unterstützung in der Konfiguration, beispielsweise einer fehlenden Domänenverwaltung oder aufgabenbasierte Übersicht und Zugriffsbeschränkung, nicht möglich.

4.3.2.5 Erweiterbarkeit und Bewertung

Die Erweiterbarkeit von ONOS ist ebenfalls ähnlich zu der von ODL, da beide auf Apache Karaf basieren. Im Gegensatz zu ODL verfolgt ONOS jedoch keinen vorrangig modellgetriebenen Ansatz. Eine Weiterentwicklung unterstützt ONOS mit entsprechenden Softwarewerkzeugen wie dem Tool *onos-create-app*, mit dem ein Maven-Projekt mit initialer Skelett-Struktur eingerichtet werden kann (**U.2**). Die Dokumentation von ONOS ist darüber hinaus sehr zentralisiert und klar versioniert gehalten, was die Verständlichkeit deutlich einfacher macht (**U.1**, **U.6**). Davon abgesehen ist der Entwickler eines Moduls relativ frei in der Gestaltung der Anwendung (**U.5**).

ONOS lässt sich, wie in den vorherigen Abschnitten beschrieben wurde, nicht direkt als Managementplattform für FSNs einsetzen. Die größten Defizite liegen in der Umsetzung des Organisationsmodells, darin fehlender Strukturen wie die Modellierung einer Föderation, die Berücksichtigung unterschiedlicher Managementkonzepte der Parteien oder die Koordination des Managements in administrativen Domänen. Diese und Lösungen zu weiteren Defiziten in den drei anderen Teilmodellen lassen sich jedoch durch die Modularität von ONOS auch nachträglich integrieren, wodurch ONOS eine mögliche geeignete Basis für Managementplattformen in FSNs bieten kann.

[V] `http://api.onosproject.org/2.7.0/apidocs/org/onosproject/alarm/package-summary.html`

4.4 Managementplattformen für Network Functions Virtualization

Komponenten zum Management und zur Orchestrierung von NFV-Netzen sind in der NFV-Referenzarchitektur im gleichnamigen MANO-Block vorgesehen (vgl. Abschnitt 2.1.3.1). Implementierungen von NFV-MANO sehen meist auch SDN-Komponenten zur Konfiguration von Netzverbindungen vor. In den folgenden Abschnitten wird ihre Nutzbarkeit im Kontext von FSNs untersucht.

4.4.1 Übersicht über NFV-Plattformen

Das Netzmanagement als Ganzes steht in NFV-MANO-Plattformen meist nicht im Vordergrund. Sie fokussieren sich in der Regel auf einen Teil des Konfigurations- und Lebenszyklusmanagements von NFV-kompatiblen Diensten, den virtuellen Netzfunktionen (VNF). In erster Linie existiert eine Vielzahl von Implementierungen von Systemen aus der NFV-Referenzarchitektur (vgl. Abschnitt 2.1.3). Beispiele sind VIMs (später näher am Beispiel von OpenStack in Abschnitt 4.5.2 erläutert) oder Systeme der Virtualisierungsschicht. Eine Sonderstellung im NFV-Ökosystem nimmt die *Open Platform for NFV* (OPNFV) ein, in der weder NFV-MANO-Aufgaben noch Netzmanagement im Vordergrund stehen. OPNFV ist vielmehr eine Plattform zur Integration, Installation und zum Testen bestehender Teillösungen und -Systeme der NFV-Referenzarchitektur [106]. Entsprechend liegt OPNFV nicht im Fokus dieser Arbeit.

Aufseiten NFV-MANO gibt es ebenfalls mehrere Implementierungen. Eines der am besten gepflegtesten und aktivsten Projekte stellt *Open Source Mano* (OSM) dar. Im folgenden Abschnitt wird OSM daher im Detail auf seine Eignung zum Management von FSNs untersucht. *Open Baton*[VI] verfolgt ein vergleichbares, jedoch limitierteres Konzept im Kontext von FSNs. Beispielsweise ist die Organisation von Rollen und Nutzern sowie die Definition von Ereignissen und Alarmen in Open Baton unflexibler. Die Weiterentwicklung von Open Baton wurde gemäß der Versionsverwaltung[VII] des Projekts im Jahr 2019 eingestellt (Stand Mai 2022). Andere MANO-Implementierungen wie *OpenMANO* sind beispielsweise selbst Teil einer anderen MANO-Plattform geworden. OpenMANO wurde eingestellt und Erkenntnisse daraus flossen in die Entwicklung von OSM ein [107]. Das *SONATA*-Projekt stellte gemäß [108] den ersten integrierten Ansatz für Dienstentwicklung, -Tests und -Installation dar. Es wurde Ende 2017 eingestellt [109]. Das Vorhaben wurde jedoch im Projekt *5Gtango* bis 2020 weitergeführt (vgl. [110]) und die SONATA-Plattform in diesem Kontext weiterentwickelt. Bis auf kleinere Anpassungen ist die Weiterentwicklung der SONATA-Plattform[VIII] jedoch seit Anfang 2021 inaktiv (Stand Mai 2022).

4.4.2 Open Source MANO (OSM)

OSM ist ein bei der ETSI untergebrachtes Projekt zur Entwicklung einer NFV-kompatiblen Management- und Orchestrierungsplattform. Nach [111] implementiert es die Funktionalität des NFVO sowie eines VNFM. OSM ist anders als viele vergleichbare Projekte immer noch aktiv mit halbjährlichem Releasezyklus. Die einzelnen Komponentenimplementierungen von OSM sind öffentlich[IX] einsehbar.

[VI] `https://openbaton.github.io/`
[VII] `https://github.com/orgs/openbaton/repositories`
[VIII] `https://github.com/orgs/sonata-nfv/repositories`
[IX] `https://osm.etsi.org/gerrit/`

4.4.2.1 Funktionalität der Plattform

Die Funktionalität von OSM kann in zwei größere Bereiche unterteilt werden:

- Zum einen bietet OSM nach[111] primär die Funktionalität als **End-to-End Network Service Orchestrator**,
- zum anderen bietet es teilweise Funktionen des **Netzmanagements**.

Demnach unterstützt es die Verwaltung von Netzdienstmodellen (vgl. Abschnitt 4.4.2.3), ihre Modellierung und ihr *Onboarding* (d. h. die Einführung von Dienstmodellen in den Dienstkatalog) sowie die Erstellung, den Betrieb und die Finalisierung von Netzdiensten. Gemäß [112] erfolgt der Betrieb eines Netzdienstes in drei Phasen, den sogenannten Tagen bzw. *Days*.

- Die Day-0-Konfiguration stellt die Managebarkeit eines (virtuellen oder physischen) Systems her, indem beispielsweise SSH-Schlüssel hinterlegt, Benutzer angelegt und die Netzkonfiguration durchgeführt wird.
- Die Day-1-Konfiguration richtet das System für seinen Dienst ein, beispielsweise durch die Installation und Konfiguration von Softwarepaketen.
- Die Day-2-Konfiguration bezieht sich auf die Dienst- und Managementaktualisierung, beispielsweise auch die Sicherung von Logs oder Daten.

Die Konfiguration von VMs und Anwendungen geschieht dabei demnach durch sogenannte *Charms* über die VNF-Manager. Charms sind Programme oder Skripte, die auf den Systemen ausgeführt werden können. Die Funktionalität umfasst auch die Konfiguration der Vernetzung von NFV-Ressourcen über SDNs. Ein Anwendungsfall von OSM ist beispielsweise die Unterstützung unterschiedlicher Dienstmodelle in 5G-Netzen, sodass es auch Network Slicing berücksichtigt.

Aus Sicht des Netzmanagements unterstützt OSM nach [112] Funktionalität mit Fokus auf Performancemanagement und Faultmanagement; andere Managementfunktionen dagegen nicht explizit. OSM erlaubt die Definition von Metriken in Ressourcen-Deskriptoren (vgl. Abschnitt 4.4.2.3) für die gemanagte IT-Infrastruktur. Die Metriken werden dann durch VIMs oder auch den OSM-Dienst *VNF Configuration and Abstraction* (VCA) automatisch gesammelt und in der Standardkonfiguration in einer Prometheus-Datenbank gespeichert. Die Visualisierung erfolgt über Grafana. Die Metriken können zudem über das NBI von OSM abgefragt werden. Im Faultmanagement unterstützt OSM die Definition von Alarmen über Thresholds. Ausgelöste Alarme werden über das Policy-Modul (POL) ausgewertet und können automatisch darauf reagieren: Entweder über die Ausführung eines benutzerdefinierten *WebHooks*, oder über ein in OSM implementiertes automatisches Skalieren von Ressourcen (**F.1**). Mittels eines angegebenen unteren Grenzwerts, eines oberen Grenzwerts, beispielsweise bezüglich der CPU-Auslastung, sowie einer Grenzüberschreitungszeitdauer, kann OSM virtuelle Maschinen automatisch (in einem vorgegebenen Bereich) erstellen oder entfernen.

Die Installation von OSM ist nach [113] über ein Installationsskript, eine Charm-basierte Installation über Juju oder auch als fertige Vagrant-Konfigurationsdatei. Mit einer angegebenen minimal notwendigen Konfiguration von zwei CPUs, sechs Gigabyte RAM und 40 Gigabyte Sekundärspeicher kann OSM für eine Managementplattform nicht als ressourcenschonend angesehen werden. Die Installation erfolgt auf Basis virtualisierter Maschinen, wodurch sie über eine Virtualisierungsebene plattformunabhängig erfolgen kann (**NF.4** teilweise). Im Installationsumfang ist zudem eine webbasierte graphische Oberfläche enthalten, durch die das Management der NFV-Komponenten realisiert wird. Jeder Nutzer hat für jedes Projekt eine Übersicht über darin enthaltene MOs (**I.7**) Rollen und damit verbundene Berechtigungen (**O.9**).

4.4.2.2 Kommunikationsmodell

Die Architektur von OSM ist in Abbildung 4.5 gezeigt. Der Kern von OSM ist ein Kafka-Bus, der die einzelnen OSM-Dienste verbindet. Nach [114] sind die wichtigsten OSM-Dienste die Folgenden:

- NBI: Ein REST-basiertes Northbound-Interface gemäß der Spezifikation ETSI GS NFV-SOL 005.
- LCM: Ein Modul, durch das das Lifecycle-Management realisiert wird.
- VCA: *VNF Configuration & Abstraction*, zum Lebenszyklusmanagement von Charms [115].
- RO: Ein Dienst zur Umsetzung von Ressourcenorchestrierung und VIM-Installation.
- POL: Der Policy-Management-Dienst zur Forcierung von Richtlinien (vgl. vorheriger Abschnitt).
- MON: Ein Monitoring-Dienst zur Sammlung von Ereignissen und Metriken der gemanagten Infrastruktur.

Weitere Dienste können flexibel in OSM über den Kafka-Bus oder die Microservice-Architektur angebunden werden (**K.6**).

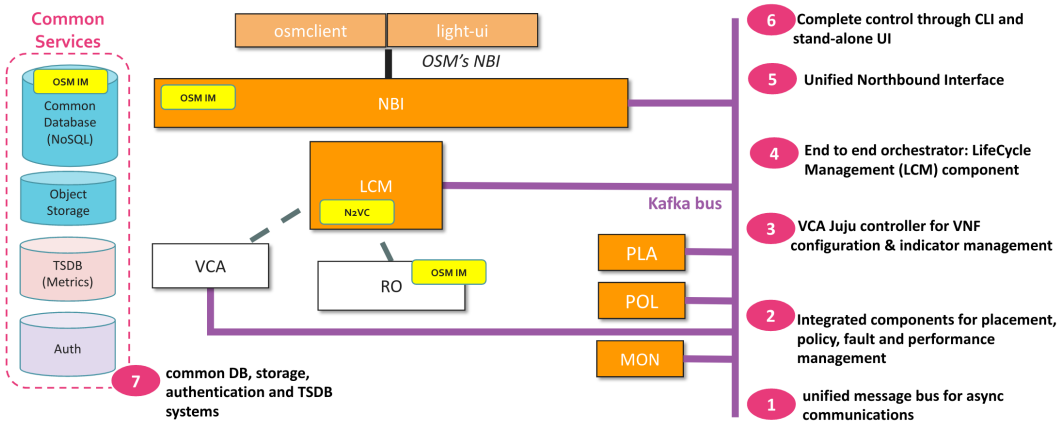


Abbildung 4.5: OSM-Systemarchitektur und -Dienste [116].

Das SBI von OSM ist über den N2VC- (innerhalb des LCM-Dienstes) und RO-Dienst implementiert (**K.3**). Der N2VC-Dienst stellt die Verbindung und Kommunikation zu VCA-Diensten her. Gemäß [38] unterstützt OSM Juju und Helm als Komponenten zur VCA. Komponenten wie VIMs oder SDN-Controller sind über den RO-Dienst angebunden. Demnach werden unterschiedliche Public-Cloud-Lösungen wie Azure oder AWS, aber auch IaaS-Plattformen wie OpenStack oder OpenVIM unterstützt. Aufseiten von SDN-Controllern werden Floodlight, OpenDaylight und ONOS unterstützt (**NF.8**). Gemäß dem Quelltext im entsprechenden Git-Repository[X] des RO-Dienstes verfolgt OSM eine lose Normalisierung von MO-Funktionen nach ihrem entsprechenden Typ (**NF.3**, **I.5**). Über das SBI abgerufene Daten erfassen immer den letzten Zustand der gemanagten Infrastruktur (**NF.6**). Des Weiteren kann OSM mit einer Installation über mehrere verteilte Datenzentren hinweg umgehen und erlaubt die Anbindung von *WAN Infrastructure Managers* (WIM), die Vernetzungstechnologien wie VPN oder DynPaC (zwischen VNFs) darstellen (**I.2**) [115]. Weitere Komponenten können am SBI über eine Pluginarchitektur flexibel

[X] `https://osm.etsi.org/gerrit/osm/RO.git`

angebunden werden [117]. Die einzelnen SBI-Plugins sind als Pythonskripte realisiert (**K.16**, **K.17**,**K.18**,**K.19**,**K.20**).

OSMs NBI (**K.2**) wird über den gleichnamigen Dienst implementiert (**NF.7**). Gemäß [117] dient das NBI insbesondere zur Bereitstellung der Funktionen von OSM an *Operations*- und *Business-Support-Systeme* (OSS / BSS). Das NBI ist in OSM als OpenAPI-Modell mittels des Werkzeugs Swagger modelliert, sodass die Anbindung von OSS / BSS oder über Gateway-Systeme einfacher umgesetzt werden kann. Nach [118] implementiert das NBI die Spezifikation ETSI GS NFV-SOL 005 durch eine HTTP-basierte REST-Schnittstelle, welche YAML und JSON unterstützt (**K.9**). Die Kommunikation ist über TLS abgesichert (**NF.13**) und erlaubt wahlweise eine tokenbasierte Authentifizierung (Standardkonfiguration), eine *Basic Authentication* genauso wie einen unauthentifizierten Zugriff (**K.14**, **K.15**). Neben Pull-Zugriffen unterstützt das NBI nach [112] auch Push-Benachrichtigungen (**K.11**). *Subscriber* können sich registrieren und einen HTTP-Endpunkt angeben, an den vom Nutzer gefilterte Benachrichtigungen geschickt werden (**I.11**, **I.12**, **I.13**, **I.14**, **K.12**). Demnach wird eine Bearbeitungsdauer nahe der Echtzeit berücksichtigt (**NF.1**). OSM bietet des Weiteren eine als East- und Westbound-Interface bezeichnete Schnittstelle. Diese ist demnach für die Anbindung von Inventarisierungssystemen sowie Systeme im Fault- und Performancemanagement vorgesehen [117]. Die konkrete Ausprägung des East- / Westbound-Interface geht aus dem Dokument nicht eindeutig hervor.

4.4.2.3 Informationsmodell

Nach [119] ist das Informationsmodell von OSM gänzlich mit der ETSI-Spezifikation GS NFV-SOL 006 in Version 2.6.1. kompatibel. Wesentliche in der Implementierung[XI] berücksichtigte MO-Modelle umfassen demnach einen Teilbereich derer aus FSNs:

- Virtual Network Function Descriptor (VNFD): Ein VNFD beschreibt eine Konfigurationsvorlage einer virtuellen Netzfunktion hinsichtlich ihrer Bereitstellung und Verhaltens [120].

- Virtual Network Function Record (VNFR): Der VNFR ist nicht näher in NFV-SOL 006 beschrieben. Nach [121] beschreibt ein VNFR im Kontext von NFV den Zustand einer VNF.

- Network Service Descriptor (NSD): Ein Template zur Beschreibung der Bereitstellung eines Netzdienstes, seiner VNFs und ihren Abhängigkeiten untereinander [120].

- Network Service Record (NSR): Der NSR wird nicht näher beschrieben. Anhand der Implementierung kann angenommen werden, dass ein NSR (analog zum VNFR) den Zustand eines Netzdienstes beschreibt.

- Network Slice Template (NST): Das NST ist nicht in NFV-SOL 006 definiert. Gemäß der Implementierung wird ein NST unter anderem durch mehrere Netzdienste und einen Network-Slice-Typ – *enhanced Mobile Broadband slice*, *Ultra Reliable Low Latency Communication slice* oder *massive Machine Type Communications slice* – beschrieben (vgl. auch [117]).

- Network Slice Instance (NSI): Der Zweck eines NSI wird ebenfalls nicht konkret beschrieben. Gemäß der Spezifikation ETSI / 3GPP-Spezifikation zur 5G-Systemarchitektur [122] bezeichnet ein NSI eine Menge von Netzfunktionen mit ihren notwendigen Ressourcen, die einen *Network Slice* bilden.

Die Elemente des Informationsmodells werden mittels YANG modelliert. Gemäß [115] kann OSM auch *Physical Deployment Units* (PDUs) handhaben und zwischen drei unterschiedlichen

[XI] `https://osm.etsi.org/gerrit/osm/IM`

Arten von Netzfunktionen unterscheiden: Virtuelle Netzfunktionen (VNF), physischen Netzfunktionen (PNF) und hybriden Netzfunktionen (HNF). Letztere sind eine Kombination der ersten beiden. In [123] wird darüber hinaus die Handhabbarkeit von Kubernetes-Netzfunktionen (KNF) und in [38] auch Container-Netzfunktionen (CNF) angegeben. Im Informationsmodell von OSM sind alle Ressourcen adressierbar und managebar (**I.3**). Ressourcenabhängigkeiten werden dabei nur zwischen den im Informationsmodell definierten Komponenten berücksichtigt, aber nicht aus dem SDN- und NFVI-Bereich (**I.1**). Dabei wird jedoch berücksichtigt, dass NFVI-Komponenten wie Hypervisors gemäß Konzept durch VIMs gemanagt werden, welche wiederum durch OSM gemanagt werden. SDN-Komponenten werden jedoch nicht explizit im Modell berücksichtigt (**I.4** teilweise).

Auch werden in OSM zwei Managementbeziehungen berücksichtigt, wie in [111] verdeutlicht ist: Einerseits eine *isComposedOf*-Beziehung, die insbesondere Bausteine von MOs verdeutlicht, wie die Zusammensetzung eines *Network Slice* aus mindestens einem *Network (Slice) Subnets*. Andererseits werden *isAShareOf*-Beziehungen berücksichtigt, durch welche eine gemeinsame Verwendung beschrieben wird. Beispielsweise kann diese Beziehung verwendet werden, um zu beschreiben, dass ein *Network Slice Subnet* potenziell von mehreren Netzdiensten geteilt wird. In VNFR-Modellen werden PNFs und VNFs zudem mit ihrem jeweiligen VIM in Verbindung gebracht, was als Steuerungsabhängigkeit interpretiert werden kann. Eine direkte Realisierungsabhängigkeit von einem Virtualisierungssystem zur jeweiligen VM wird jedoch nicht berücksichtigt (**I.9**, **I.10**). Die Ebenen unter einem VIM, die NFVI inkl. der Virtualisierungsschicht und den Virtualisierungssystemen, liegt generell nicht im Fokus von OSM.

4.4.2.4 Organisationsmodell

Das Organisationmodell von OSM ist vergleichsweise einfach gehalten und verwaltet die Ressourcen und den Zugriff darauf nach [115] in einem mandantenfähigen RBAC-basiertem Ansatz (**NF.14**). Projekte erfüllen dabei die Funktion administrativer Domänen (**O.2** teilweise, **F.5**, **F.8**), berücksichtigen aber keine Überschneidungen. Demnach können in OSM benutzerdefinierte Rollen angelegt und mit Berechtigungen belegt werden und somit auch in gewissem Umfang unterschiedliche Managementkonzepte mehrerer Mandanten umgesetzt werden (**O.1** teilweise). Rollen sind jedoch global gültig und nicht nur auf einzelne Domänen bzw. Projekte beschränkt (**O.13** teilweise, **O.14** teilweise). Aufgaben werden nicht explizit vorgesehen, können jedoch teilweise durch Rollen verwirklicht werden (**F.6** teilweise). Die Berechtigungen werden für einzelne Rollen im JSON-Format beschrieben und teilen sich demnach einerseits in Variablen wie *force*, *admin* oder *public* mit definierter Bedeutung, deren Zutreffen für eine Rolle über einen booleschen Wert angegeben wird. Andererseits können REST-Pfade, d. h. URIs des NBIs, hinsichtlich ihrer Zugreifbarkeit für einen Nutzer eingeschränkt werden (**O.3** teilweise, da keine Föderationsaspekte berücksichtigt; **O.6**). Diese REST-Zugriffspunkte werden nach [118] hierarchisch organisiert, sodass bei Fehlen einer definierten Berechtigung die unmittelbar nächste Berechtigung für eine Zugriffsentscheidung herangezogen wird.

Die Organisation der gemanagten IT-Infrastruktur erfolgt im Kontext von *Projekten* [115]. Nutzer (**O.4**) sind einem Projekt zugeordnet und haben im Rahmen dieses festgelegten Geltungsbereichs entsprechend der ihnen zugewiesenen Rollen Zugriffsrechte auf Funktionen von OSM im Umgang mit ihnen (oder auch organisatorischen Strukturen wie die Konfiguration von Rollen und Nutzern selbst).

4.4.2.5 Erweiterbarkeit und Bewertung

OSM ist durch seine Microservice-Architektur (**NF.11** teilweise, da einzelne Dienste nur an einem Ort vorhanden sind) vergleichsweise einfach erweiterbar und flexibel (**K.1**). Ein unterbrechungsfreier Betrieb ist jedoch nur teilweise gegeben, beispielsweise beim modellbasierten Ansatz zur Erweiterung des Informationsmodells (**F.3** teilweise). Die Nutzerdokumentation ist zentral abgelegt und ausführlich beschrieben. Daneben ist ebenfalls eine Dokumentation

für Entwickler vorhanden, sie bezieht sich aber überwiegend auf die Einrichtung einer Entwicklungsumgebung und den Umgang mit Entwicklerwerkzeugen (**U.1** teilweise). Auf einzelne Aspekte wie die Entwicklung eines SDN- oder VIM-Plugins am RO-Dienst wird ebenfalls eingegangen. In der Dokumentation wird auf Konventionen hingewiesen, beispielsweise, dass bei der Entwicklung eines RO-Plugins die Methodennamen und -Parameter nicht geändert werden dürfen, sondern lediglich die Implementierung für ein spezifisches MO angepasst werden darf [124]. Erweiterungen des Organisationsmodells sind in OSM lediglich hinsichtlich benutzerdefinierter Rollen möglich. Strukturen zur Unterstützung von Föderationen scheinen nicht vorgesehen zu sein. Das Informationsmodell ist hingegen über YANG auch für Benutzer erweiterbar, jedoch nicht in der Entwicklerdokumentation vorgesehen. Der Anwendungsbereich von OSM zielt bewusst auf NFV und insbesondere für NFV-Orchestrator relevante MOs und Funktionen ab. So wird auch die Erweiterung des Funktionsmodells nicht explizit unterstützt. Die Einarbeitung in den bestehenden Quelltext ist einerseits durch eine klare Struktur des OSM-Projekts unkompliziert. Andererseits ist die Dokumentation des Quelltexts de-facto nicht oder nur in sehr geringfügigem Umfang vorhanden.

OSM ist weniger ein Framework, sondern vielmehr eine über Plugins und Microservices erweiterbare Managementplattform. Der Anwendungsbereich geht dabei einerseits kaum über das NVF-Paradigma hinaus und berücksichtigt andererseits keine Föderation von mehreren Parteien, sondern nur einzelne Teilaspekte davon (beispielsweise Domänen ohne Überschneidungen oder eine geographische Verteilung der gemanagten IT-Infrastruktur).

4.5 Paradigmenübergreifende Managementplattformen

In diesem Abschnitt werden zusätzlich Managementplattformen untersucht, die nicht einem einzigen Paradigma zugeordnet werden können. Alle betrachteten Plattformen weisen jedoch einen starken Bezug zu Anwendungsfällen aus (föderierten) SNs auf.

4.5.1 XOS (CORD)

XOS ist eine von der *Open Networking Foundation* (ONF) unterstützte Plattform. Sie verfolgt eine zentrale Operationalisierung und Steuerung vieler verteilter und heterogener Dienste (vgl. [125]). Es ist ein wesentlicher Bestandteil des Projekts *Central Office Re-architectured as a Datacenter* (CORD) und bildet darin den zentralen Controller zur Zusammenführung und dem Management einer SDN- und NFV-konformen Datenzentrumsarchitektur. XOS weist einige Frameworkbestandteile auf und legt Schwerpunkte auf das Informations- und Kommunikationsmodell. Das Organisationsmodell ist ebenfalls flexibel gestaltet und eng mit dem Informationsmodell verzahnt. Das Funktionsmodell ist weitestgehend offengelassen.

4.5.1.1 Informationsmodell

XOS stellt nach [126] ein Framework zur Modellierung der gemanagten IT-Infrastruktur bereit. Den Kern darin bildet die eigens dafür ausgelegte Modellierungsprache *xproto*, die auf Googles *Protocol Buffers* basiert und vor allem die Serialisierbarkeit der dadurch beschriebenen Modelle unterstützt. Demnach erlaubt xproto eine vererbungsbasierte erweiterbare Modellierung von MOs und Policies. Der Fokus liegt besonders auf der Modellierung eines MO und Attribute, die es beschreiben. Die Modellierung von MO-Funktionen wird dagegen nicht direkt durch xproto unterstützt. Sie können aber über zusätzlichen benutzerdefinierten Python-Code definiert werden, durch den auch die Anpassung des Datenmodells (z. B. Erstellen eines Nutzers), weniger aber die Ausführung einer bestimmten MO-Funktion der gemanagten Infrastruktur erfolgen kann. MO-Funktionen werden in XOS vielmehr über das Kommunikationsmodell abstrahiert und nicht im Informationsmodell berücksichtigt. Attribute eines Modells können in

xproto über bereitgestellte Eigenschaften gesteuert werden, beispielsweise das Festlegen eines Minimal- und Maximalwertes, die Einschränkung valider Werte (vergleichbar mit einer Enumeration) und das Festlegen eines Defaultwertes. Ähnliche Kriterien können nicht nur auf Attribut-, sondern auch auf Modellebene definiert werden.

Im Grundkonzept wird in XOS zwischen drei verschiedenen Attributtypen unterschieden:

- *declarative*, als erwarteter Soll-Zustand eines MOs, gemäß dem ein MO konfiguriert wird;
- *feedback*, als Betriebs- oder Ist-Zustand eines MO;
- *bookkeeping*, als für die Verarbeitung notwendiger interner Zustand.

Auf dieser Unterscheidung stützt auch das Kommunikationsmodell der *Goal-oriented synchronization* in XOS, wie im folgenden Abschnitt erläutert wird. XOS stellt in seinen *Core Models* [127] bereits eine MO-Grundstruktur von Modellen bereit, die sich stark an Anwendungsfällen aus dem CORD-Projekt orientiert. Sie definieren Elemente aus dem Cloud-Computing (z. B. *Image* und *Flavor*) und Network-Slicing aus der 5G-Architektur. Über die vorgegebenen Dienst-, Slice- und Instanzenmodelle wird so eine mandantenfähige Modellierung geschaffen (**NF.14**). Konkrete Netzkomponenten aus der SDN- und NFV-Architektur sind nicht vordefiniert oder unterstützt, können aber über xproto benutzerdefiniert modelliert werden (**I.4** teilweise, **I.3**, **I.6**).

Die Modellierung von MOs über xproto bietet auch die Möglichkeit, Managementbeziehungen zu definieren (**I.1** teilweise, da nur sehr generell festgelegt): Attribute eines Modells können auf Attribute eines anderen Modells bidirektional referenzieren. Für jede Referenz unterstützt xproto die Festlegung von Mengenbeziehungen. Auf diese Weise können beispielsweise auch Managementbeziehungen in FSNs modelliert werden, wie Beziehungen zwischen NFV-Komponenten oder auch Steuerungsbeziehungen zwischen SDN-Controller und -Infrastruktur (**I.8**, **I.9**).

Netzereignisse und -Zustände, die direkt von MOs der gemanagten IT-Infrastruktur kommen, sind per se nicht als Modell in XOS vorgesehen. Im Fokus steht vielmehr der Soll- und Ist-Zustand von Netzkomponenten. Auch sind derartige Modelle nicht in den Core-Models bereitgestellt. Netzereignisse und -Zustände sind jedoch generell durch xproto beschreibbar (**I.11** teilweise, **I.14**). Die Möglichkeiten, Alarme zu generieren, werden ebenfalls nicht explizit im Konzept beschrieben. Zum einen könnte jedoch eine Modellierung über xproto möglich sein, durch die Definition entsprechender Alarmzustände. Zum anderen erlaubt XOS das Logging von Fehlerzuständen (vgl. [128]), die jedoch nicht direkt durch die API abrufbar sind (**I.12** teilweise).

4.5.1.2 Kommunikationsmodell

XOS verfolgt nach [129] eine Microservice-Architektur und bietet eine Installation auf Kubernetes über den Paketmanager *Helm* an (**NF.11**, **K.1**). Durch den Betrieb auf Kubernetes ist XOS skalierbar umgesetzt (**NF.9**). Die Dienste sind hauptsächlich in Python geschrieben, wodurch XOS plattformunabhängig betrieben werden kann (**NF.4**); der Kern von XOS hat nur wenige Megabyte Quelltext (**NF.5**). Zur Entwicklung von Komponenten zur Erweiterung der Funktionalität von XOS wird eine eigene Softwarebibliothek, die *xosapi* mit Schwerpunkt des *Synchronizer-Frameworks* bereitgestellt (vgl. [130]).

Das Synchronizer-Framework stellt eine koordinierte Schnittstelle zwischen dem SBI und dem von XOS gehaltenen Datenmodell dar (**K.3**). Gemäß [131] verfolgt XOS einen Ansatz der *Goal-*

oriented synchronization: Der Ist-Zustand gemanagter Systeme wird überwacht (*feedback states*) und schrittweise durch Aktionen über das SBI dem Soll-Zustand (*declarative states*) angepasst. Durch diesen Ansatz soll das Netzmanagement auch weniger fehleranfällig sein, da alle Operationen hin zu einem bestimmten Zielzustand ausgeführt werden (**NF.10**) [131].

Der Prozess wird durch das Zusammenspiel von vier XOS-Komponenten realisiert:

- *Synchronizer Actuators* werden über Änderungen des Zustands eines Modells informiert und übersetzen die Änderungen in eine Konfiguration des entsprechenden Dienstes.
- *Event Steps* warten auf Push-Benachrichtigungen von MOs und pflegen sie entsprechend in das Datenmodell ein.
- *Pull Steps* fragen aktiv den Zustand der gemanagten IT-Infrastruktur ab und passen gemäß der Antwort das Datenmodell an (**NF.6**, **K.18**, **NF.8**).
- *Model Policies* machen Vorgaben zu gültigen Abhängigkeiten zwischen Modellen und setzen diese um.

Die Implementierung der Steps muss explizit durch einen Entwickler stattfinden, erlaubt daher aber eine hohe Flexibilität (**K.16**, **K.17**, **K.19**, **K.20**, **K.5**). Der Soll-Zustand kann durch Policies und Event- sowie Pull-Steps verändert werden. Der Ist-Zustand eines Modells kann durch Actuators und ebenfalls Event- und Pull-Steps verändert werden. In XOS können mehrere Synchronizer implementiert werden und mehrere Modelle können durch einen Synchronizer verwaltet werden. Ein Modell wiederum kann durch genau einen Synchronizer verwaltet werden. In XOS wird auch die Abhängigkeit von Modellen untereinander berücksichtigt: Die Einhaltung der Reihenfolge voneinander anhängiger Änderungen am Datenmodell ist demnach stets garantiert (**I.10**). Demgegenüber gibt es keine Garantie, wann der Soll-Zustand eines Modells auf einem MO der gemanagten IT-Infrastruktur umgesetzt wird, wodurch unbedingte Echtzeitanforderungen nicht immer erfüllt werden können.

XOS bietet unterschiedliche APIs an (**K.2**): Gemäß [126] bildet einerseits eine gRPC-API die Hauptschnittstelle zur Kommunikation mit XOS. Darüber hinaus bietet XOS eine TOSCA-API, die im Kontext des CORD-Projekts genutzt wird (**K.9**). Demnach wird die API auf Basis der xproto-basierten Modelle über ein in XOS dafür bereitgestelltes Tool generiert. Die API ist folglich hinsichtlich ihrer Funktionalität flexibel, beschränkt sich aber aufgrund des modellbasierten Ansatzes in XOS im Wesentlichen auf den lesenden und schreibenden Zugriff des Datenmodells (**NF.3** teilweise, **F.9**). Die API kann jedoch um benutzerdefinierte Funktionen von Synchronizern erweitert werden [132] (**K.8** teilweise, **K.14**, **K.15**, **NF.13**). Die gRPC-basierte API unterstützt jedoch keine Push-Benachrichtigungen (**K.11** teilweise).

4.5.1.3 Organisationsmodell

Auch das Organisationsmodell basiert genauso wie das Informationsmodell auf der eigenen Modellierungssprache xproto (**O.5**, **O.8**). Nach [133] erlaubt xproto die Definition von *Security Policies*, die schließlich an der API von XOS forciert werden (**O.6**). Eine Policy ist demnach ein logischer Ausdruck, der für ein Modell oder auch für die komplette Umgebung angewendet werden kann. In Kombination mit einer an der API durchgeführten Authentifizierung erlaubt XOS daher die Anpassung von Autorisierungskonzepten (**O.3**). In den Core-Models dazu gegebene relevante Modelle sind *User* (**O.4**) zur Beschreibung eines Nutzers, *Privilege* zur Beschreibung eines bestimmten Lese-, Schreib- oder Berechtigungsprivilegs und *Principal* zur Beschreibung von Identitäten für Rechenressourcen (vgl. [127]), eine geographische Trennung von IT-Ressourcen durch das Modell *Site* (**I.2**) sowie *TrustDomain* (**NF.15** teilweise). Eine *TrustDomain* beschreibt einen Namensraum zur Isolierung von Ressourcen [127]. Auch wenn XOS

somit bereits Aspekte in Föderationen definiert und in die Organisation des Netzmanagements einfließen lassen kann, weist es gerade in FSNs einige Lücken auf: So erlaubt es nicht die Integration heterogener Managementkonzepte mehrerer Partner, und administrative Domänen werden ebenfalls nicht explizit berücksichtigt, sondern sind nur im Kontext mehrerer Mandanten in Kombination mit geeigneten Security Policies realisierbar. Die Behandlung sich überschneidender administrativer Domänen bleibt ebenfalls unberücksichtigt.

4.5.1.4 Funktionsmodell

Durch den stark modellgetriebenen Ansatz von XOS beschränkt sich seine Hauptfunktionalität im Auslesen oder der Modifikation von Attributen des XOS Datenmodells. Dazu bietet XOS verschiedene APIs sowie eine Nutzeroberfläche (**NF.7**, **I.7**), einerseits über die XOS-Shell (vgl. [134]) als CLI und andererseits als webbasierte graphische Nutzeroberfläche[XII]. Die Weiterentwicklung von XOS kann weitestgehend im Rahmen eines unterbrechungsfreien Betriebs (**F.3**) durchgeführt werden: Synchronizer werden als eigenständige Microservices in die XOS-Architektur eingebunden [128]. Das Management über XOS ist nicht in klassische Managementfunktionen wie FCAPS eingeteilt und auch nicht aufgabenorientiert. Die Kernprozesse des Managements, die Überwachung und Steuerung sind in dem verfolgten Ansatz der *Goal-oriented synchronization* automatisiert umgesetzt (**F.1**).

4.5.1.5 Erweiterbarkeit und Bewertung

XOS bietet im Informationsmodell und Organisationsmodell das xproto- sowie im Kommunikationsmodell das Synchronizer-Framework. Eine geführte Adaption der Managementplattform an unterschiedliche gemanagte IT-Umgebungen ist daher möglich. Die Frameworks sind angemessen dokumentiert (beispielsweise zu implementierende Callback-Funktionen in Synchronizern), unter anderem mit Beispielen, Tutorials und einer Konzeptbeschreibung übersichtlich beschrieben (**U.1**, **U.2**) und bieten genug Flexibilität, um langlebig nutzbar zu sein (**U.3**). XOS unterstützt die Generierung von Modellen (vgl. [126]) durch eine eigene Toolchain und die Entwicklung von Synchronizern durch bereitgestellt Images für Docker-Container (**U.6**). Der modellgetriebene Ansatz zum Angleich des Ist-Zustands zum Soll-Zustand schränkt die Erweiterbarkeit des Frameworks teilweise ein und unterstützt einen funktionalen Ansatz zum Zugriff auf die IT-Infrastruktur nicht explizit (**U.4** teilweise), sondern lediglich durch Einbindung von benutzerdefiniertem Python-Quelltext. Der Grad der Designfreiheit ist dennoch hoch (**U.5**).

Wie im Projektverzeichnis[XIII] der XOS-Plattform zu erkennen ist, wurde ihre Weiterentwicklung vor ca. zwei Jahren eingestellt (Stand Mai 2022). Durch die beschriebene Flexibilität eignet es sich jedoch als Basis für die Entwicklung von Managementplattformen. Die aktuell größten Einschränkungen weist es in Hinblick geeigneter organisatorischer Strukturen für das Management von FSNs auf: Zwar besteht eine Trennung von Diensten für die gesamte gemanagte IT-Umgebung (*Service*) und auf einzelne Mandanten eingeschränkte Dienste (*ServiceInstance*), eine geeignete Umsetzung administrativer Domänen und die Behandlung ihrer Überschneidungen ist aber nicht berücksichtigt. Auch ist das Management nicht aufgabenorientiert und hauptsächlich auf Basis von Nutzern und Privilegien aufgebaut. Es berücksichtigt entsprechend auch kaum Föderationsbeziehungen zwischen föderierten Parteien oder heterogene Managementkonzepte. XOS kann jedoch durch seine Teilframeworks als Plattform zur Erweiterung um geeignete organisatorische und funktionale Aspekte geeignet sein.

4.5.2 OpenStack

OpenStack ist nach [135] eine *Infrastructure-as-a-Service*-(IaaS)-Plattform. Entsprechend dient sie primär zur Verwaltung von CPU-, Speicher- und Netzressourcen gemäß den Prinzipien des Cloud-Computings. Im Kontext von NFV wird OpenStack oft als VIM betrachtet (bspw. in OSM).

[XII] `https://github.com/opencord/xos-gui`
[XIII] `https://github.com/orgs/opencord/repositories`

Die Funktionalität von OpenStack geht jedoch deutlich über die eines VIM hinaus. Es bietet auch einige Dienste des Netzmanagements, auf denen in der folgenden Beschreibung und Bewertung ein deutlicher Fokus gelegt wird.

4.5.2.1 Funktionalität

OpenStack ist keine monolithische Softwareanwendung, sondern setzt sich aus einer Vielzahl von interagierenden Microservices zusammen. Der Funktionsumfang hängt entsprechend von den jeweils in einer Umgebung installierten Diensten ab. Die Installation von OpenStack ist daher vergleichsweise kompliziert und kann viele Ausprägungen haben (**NF.11**, **K.1**). Eine automatisierte Installation ist eingeschränkt über verschiedene Tools wie Helm, Ansible oder Puppet (vgl. [136]) möglich und benötigt stets ein Unix-System als Plattform. Eine plattformunabhängige Installation ist daher nur teilweise möglich (**NF.4** teilweise). In Anleitung [135] wird eine Installation über zwei Rechnerknoten mit insgesamt zwei CPUs, sechs Gigabyte Primär- sowie 15 Gigabyte Sekundärspeicher gefordert. In dieser Installation bietet OpenStack jedoch kaum Funktionen des Netzmanagements – d. h. zur umfangreichen Überwachung und Steuerung der Infrastruktur. Es wird die Bereitstellung von deutlich mehr Ressourcen empfohlen. Entsprechend bietet OpenStack keinen ressourcenschonenden Ansatz. Alle OpenStack-Dienste werden auf einer zentralen Website[XIV] aufgelistet. Im Folgenden wird die Funktionalität im Netzmanagement bewertet und weniger auf die Kernfunktionalität von OpenStack, das Lebenszyklusmanagement von IT-Ressourcen eingegangen.

- Der Dienst *Ceilometer* dient nach [137] zur Sammlung und Normalisierung von Telemetriedaten aus OpenStack-Diensten. Demnach erlaubt er die Verarbeitung von einerseits Notifikationen und andererseits aktiv abgefragten Daten von den OpenStack-Diensten. Die gesammelten Daten werden schließlich üblicherweise in einer Zeitreihendatenbank gesammelt und für andere Dienste zugreifbar gemacht.

- Der Dienst *Monasca* bietet nach [138] eine mandantenfähige, performante und fehlertolerante Lösung zur Überwachung von Infrastrukturkomponenten und Anwendungen. Er ist selbst als Microservice-Anwendung umgesetzt. Zur Überwachung eines Systems (z. B. einer VM) kommt eine anwendungsspezifische Agentensoftware zum Einsatz, die Daten sammelt und auf einen zentralen Kafka-Bus legt (**K.5**). Die Nachrichten auf dem Bus werden schließlich von anderen Monasca-Teildiensten ausgewertet. Monasca bietet zudem die Funktion, Alarme und Benachrichtigungen zu generieren und zu verwalten. Alarme werden über von den Nutzern definierte Ausdrücke zur Auswertung (beispielsweise Thresholds) und unter Berücksichtigung der Auftretenshäufigkeit oder zeitlicher Aspekte von Events festgelegt. Für die mandantenfähige Verwaltung von anwendungsspezifischen Daten und Funktionen bietet Monasca eine eigene API sowie eine Integration in die OpenStack Web-UI *Horizon*. Die Fehlertoleranz besteht in Monasca insbesondere darin, dass es verarbeitete Daten kennzeichnet und daher im Falle eines Systemfehlers den letzten Stand wiederaufnehmen kann, ohne dass gesammelte Ereignisse unberücksichtigt bleiben.

- *Vitrage* ist gemäß [139] ein Dienst zur *Root Cause Analysis*, d. h. der Ursachenanalyse im Fehlerfall. Als Basis dafür dient demnach eine Abbildung von physischen auf virtuelle Ressourcen, über die Zusammenhänge modelliert werden (**I.1** und **I.10** teilweise, da beispielsweise Steuerungsbeziehungen oder Systembeziehungen nicht explizit modelliert werden). Wie in [140] beschrieben wird, kann in Vitrage über sogenannte *Templates* die Ableitung von Alarmen und Ressourcenzuständen konfiguriert werden. Demnach werden innerhalb der Templates über eine eigene Syntax Szenarien modelliert, die Realisierungsabhängigkeiten gemeinsam mit Alarmzuständen (über boolesche Ausdrücke verknüpft) modellieren und auf dieser Basis Aktionen ausführen können. Aktionen umfas-

[XIV] `https://www.openstack.org/software/project-navigator/openstack-components`

sen beispielsweise ebenfalls die Generierung eines Alarms, die Änderung eines Zustands einer Ressource im Modell, oder auch die Ausführung benutzerdefinierter Aktionen (beispielsweise von Shell-Skripten) über OpenStacks Mistral-Dienst oder über Webhooks. Vitrage bietet genau wie Monasca und viele andere OpenStack-Dienste eine mandantenfähige API.

- Ein weiterer dedizierter Dienst zur Alarmierung in OpenStack ist *Aodh*. Wie in [141] beschrieben wird, erlaubt auch dieser Dienst die regelbasierte Ausführung von Aktionen als Reaktion auf Netzereignisse und -Zustände, die von dem Dienst Ceilometer gesammelt werden. Demnach erlaubt auch Aodh Webhook-basierte Aktionen und darüber hinaus das Generieren eines Logeintrags sowie die Benachrichtigung über den OpenStack-Dienst Zaqar.

Ein weiterer Dienst, der insbesondere zur Fehlerbehandlung beiträgt, ist *Blazar* [142] zur Ressourcenplanung durch -Reservierung und automatisierter Wiederherstellung von Ressourcen bei Ausfall eingeplanter Infrastruktur. Der Dienst *Masakari* stellt zudem fehlerhafte VMs wieder her [143]. Aus funktionaler Sicht sind insbesondere die aufgezeigten Überschneidungen durch unterschiedliche Dienste auffällig und für OpenStack als Managementplattform ein Nachteil. Ein vollintegrierter Ansatz in der Umsetzung ist nicht gegeben, sondern Funktionen und Strukturen wie Alarme müssen potenziell mehrfach modelliert werden. Dienste wie Monasca, Vitrage oder Aodh unterstützen jedoch eine automatisierte Überwachung und Steuerung (**F.1**). Hinsichtlich der Nachvollziehbarkeit unterstützen die einzelnen OpenStack-Dienste in der Regel ein dateibasiertes Logging (**NF.2**).

4.5.2.2 Kommunikationsmodell

Die Architektur von OpenStack nach [135] ist in Abbildung 4.6 dargestellt. Gezeigt ist jedoch nicht die vollständige Architektur, sondern lediglich einige Kerndienste, anhand derer das Architekturprinzip erläutert wird. Jeder Dienst in OpenStack weist selbst eine eigene, meist auf Microservices basierende Architektur auf. Dienste implementieren und bieten jeweils eine eigene API, über die neben Nutzerzugriffen auch die interne Kommunikation der Dienste untereinander stattfindet. Jeweilige Prozesse innerhalb eines Dienstes kommunizieren demnach oft über einen AMQP-Dienst. Die Architektur bietet einige Vorteile wie Flexibilität (**K.6**), eine uneingeschränkte Erweiterbarkeit der Funktionalität und eine gute Skalierbarkeit (**NF.9**, **NF.11**) sowie über Redundanz von Diensten einen teilweise unterbrechungsfreien Betrieb (**F.3** teilweise). Gleichzeitig zeigen sich aber auch Nachteile wie eine fehlende quelltextgestützte Anleitung bei der Erweiterung, eine hohe Komplexität und ein größer ausgeprägtes Fehler- sowie Schwachstellenpotenzial, der Einsatz redundanter Dienste (wie Datenbanken) und uneinheitliche Datenmodelle (vgl. nächster Abschnitt). Die Absicherung von APIs ist in OpenStack über TLS empfohlen [144] (**NF.13**, **K.14**).

Wie aus der Abbildung deutlich wird, bieten die jeweiligen Dienste eigene APIs (**K.2**). Die APIs sind netzbasiert (**K.9**) und in der Regel über einen HTTP-Dienst realisiert (**K.12**). Durch die APIs der Dienste ist OpenStack umfangreich steuerbar (**F.9**). OpenStack bietet jedoch eine zentrale webbasierte Nutzeroberfläche über den Dienst Horizon (**NF.7**). Einige Dienste wie *Vitrage* (vgl. [139]) und *Monasca* (vgl. [138]) verfügen über eine Oberflächenintegration. Generell erfüllen gerade die im letzten Abschnitt betrachteten Dienste zum Netzmanagement jedoch Push- und Pull-Funktionalität (**K.11**).

Von OpenStack berücksichtigte Komponenten der gemanagten Infrastruktur umfassen insbesondere Dienste zur Verwaltung von CPU-, Speicher- und Netzressourcen. Der Zugriff auf MOs ist über die Dienste abstrahiert und normalisiert (**NF.3**, **NF.8**, **I.5**). Die Dienste stellen das SBI zu den gemanagten Ressourcen bereit (**K.3**). Nach [145] fallen darunter die vom Dienst *Nova* unterstützten Hypervisor wie KVM, QEMU oder XEN, aber auch die Containerlösung LXC. Das Management von Netzkomponenten und Netzen übernimmt der Dienst *Neutron*. SDN-

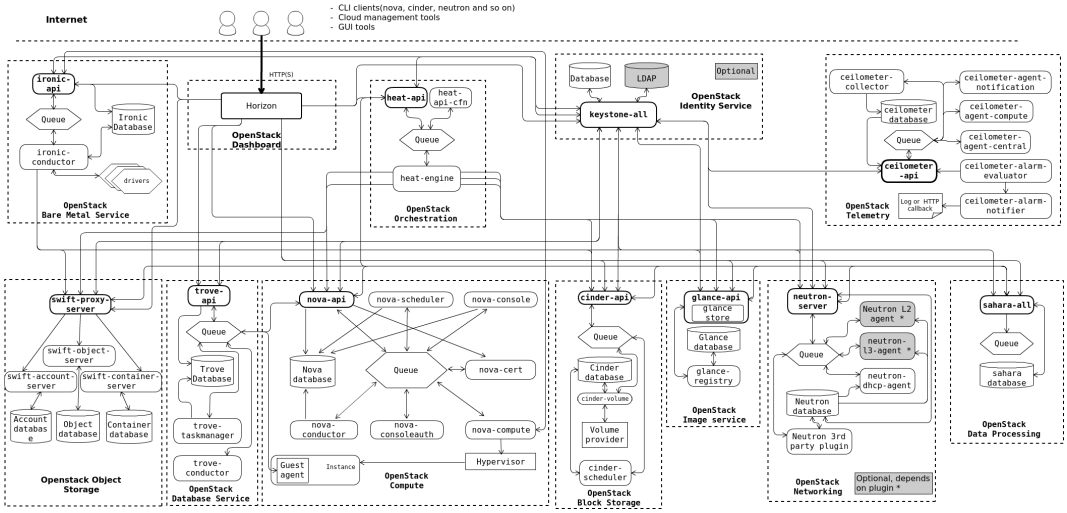


Abbildung 4.6: Logische Sicht der Architektur von OpenStack am Beispiel einiger Dienste [135].

Controller wie OpenDaylight (vgl. [146]) können über Plugins an Neutron angebunden, überwacht und gesteuert werden. Eine weitere Anbindung an die gemanagte IT-Infrastruktur ist je nach OpenStack-Dienst unterschiedlich. Beispielsweise setzt Monasca eigene Softwareagenten zur Überwachung von VMs ein. Das Management beliebiger NFV-MANO-Komponenten wie VIMs, NFV-Orchestrator oder VNF-Manager ist in OpenStack nicht vorgesehen. Vielmehr werden derartige Funktionen über eigene Dienste integriert: Die Grundfunktionalität (über Nova, Neutron und Cinder) entspricht der eines VIM: Eine Orchestrierung ist über *Heat* (vgl. [147]) möglich und der Dienst *Tacker* (vgl. [148]) implementiert einen NFV-Orchestrator und einen VNF-Manager. Generelle Schnittstellen zu derartigen Komponenten anderer Hersteller fehlen jedoch, wodurch die Heterogenität der Komponenten in diesem Bereich nicht behandelt werden kann. Das SBI kann flexibel durch die Entwicklung benutzerdefinierter Dienste und teilweise Plugins erweitert werden (**K.16** bis **K.20**). Die Erweiterung ist jedoch entsprechend aufwändig und ungeführt.

4.5.2.3 Informationsmodell

Das in OpenStack berücksichtigte Informationsmodell spiegelt sich im Wesentlichen im Datenmodell des dafür entwickelten Software Development Kit (SDK) wider. In der SDK-Dokumentation [149] wird darauf hingewiesen, dass das Datenmodell nicht starr ist und darin beschriebene Elemente uneingeschränkt um benutzerdefinierte kontrolliert Attribute erweiterbar sind. Das Datenmodell beschreibt daher zwei Arten von Attributen: Zum einen solche, die als fix für jedes Element angenommen werden können und in jedem Elementobjekt enthalten sind, und zum anderen benutzerdefinierte Attribute, die in einem Schlüssel-Werte-Paar-Format zusätzlich unter einem eigens dafür vorgesehenen Attribut *properties* angegeben werden können. Auf diese Weise findet eine Normalisierung mit flexibler Erweiterbarkeit statt. Zu im Datenmodell beschriebenen Elementen gehören demnach neben der Beschreibung von Lokalisationsdaten (**I.2**) besonders auch cloudspezifische Strukturen wie *Flavor*, *Image* und *Volume* auch Strukturen des Betriebs (in dieser Arbeit auch als Netzzustände bezeichnet) wie *Security Group*, *ComputeLimits* und *ComputeUsage* oder *ServerUsage* und viele weitere. Die relevanten Objekte sind im SDK[XV] für jeden OpenStack-Dienst getrennt gruppiert. Im FSN-Kontext dient zur Beschreibung einer VM als zentrales MO in OpenStack das Element *Server*. Auch ein Element zur Beschreibung eines Bare-Metal-Systems, eines Hypervisors, eines Netzdienstes oder einer Netzressource oder eines Routers ist definiert, die letzteren beiden jedoch ohne konkrete Parameter. OpenStack verfolgt hier einen objektorientierten Ansatz, wichtige FSN-Komponenten

[XV] `https://github.com/openstack/openstacksdk`

wie SDN-Controller, VIMs oder andere Komponenten aus dem NFV-Paradigma fehlen (**I.4** teilweise). Ein *Server* genauso wie ein Bare-Metal *Node* können einen Besitzer haben (**I.6**). Jedes Element in OpenStack hat eine eindeutige Adresse und kann in der Umgebung inklusive der Netzstruktur abgerufen werden (**I.3**, **I.7**, **I.9**).

Durch andere Dienste wird das Informationsmodell von OpenStack um weitere Elemente erweitert. So definiert Ceilometer nach [137] Zähler (original: *Meter*), die entweder kumulativ, diskret oder veränderlich und insbesondere benutzerdefiniert (**I.11**) sein können. Darüber hinaus unterscheidet Ceilometer zwischen *Events* (Objektzustand zu einem Zeitpunkt) und *Samples* (einfache Werte). Die Hauptinformation in Events wird über eine Menge an Schlüssel-Werte-Paare beschrieben. Dabei wird ebenfalls bereits eine Basis für Mandantenfähigkeit geschaffen: Jedes Event verweist nach Möglichkeit auf das damit jeweils verbundene Projekt, den Mandanten und Nutzer. Darüber hinaus besteht die Möglichkeit, von einer Netzinformation auf die entsprechend betroffene Ressource zu verweisen (**I.14**). Auch Monasca bietet nach [150] in seinem Datenmodell vergleichbare Elemente wie *Metric* (ein Vektor aus Informationen) und *Measurement* (eine *Metric* mit Zeitstempel), darüber hinaus aber Alarm-Definitionen und Alarme (**I.12**). Alarm-Definitionen sind demnach Richtlinien zur Generierung eines Alarms. Sie werden insbesondere grenzwert- und vergleichsbasiert unter Verwendung boolescher Ausdrücke beschrieben und generieren Alarme mit vier unterschiedliche Schweregraden. Im Dokument wird darauf hingewiesen, dass Alarme lediglich einmal pro Minute ausgewertet werden, sodass eine Echtzeitverarbeitung bei Monasca nicht gegeben ist. In Aodh wird nach [141] ein anderes Alarmierungskonzept genutzt und Alarm-Definitionen werden mittels JSON, aber ebenfalls über Thresholds und boolesche Ausdrücke realisiert. Genauso wie in Monasca erfolgt demnach auch hier die Auswertung der Alarme einmal pro Minute. Die Datenaktualität ist daher nur teilweise und nicht für alle Dienste gegeben (**NF.6** teilweise).

Das flexible Konzept von OpenStack als Menge interagierender Dienste mündet in einer offensichtlichen Schwäche im darin verfolgten Informationsmodell: Es besteht aus einigen sich teilweise überlappenden Ansätzen ohne zentrale Vorgaben. Der Umgang mit Informationselementen wird auf diese Weise erschwert, da eine generelle Normalisierung fehlt und lediglich in Teillösungen besteht (**I.13** teilweise).

4.5.2.4 Organisationsmodell

Die Organisation in OpenStack wird nach [151] zentral über den Dienst *Keystone* realisiert, der für die Authentifizierung und Autorisierung zuständig ist (**K.15**). Darüber hinaus bietet der Dienst ein zentrales Register über in OpenStack genutzte Dienste (z. B. Nova oder Neutron). Zur organisatorischen Strukturierung der gemanagten IT-Infrastruktur und der OpenStack-Dienste selbst bietet Keystone demnach fünf zentrale Einheiten:

- Eine *Domain* dient demnach zur logischen Aufteilung von Projekten, Nutzern und Rollen via Namespaces. Beispielsweise können Domains genutzt werden, um eine OpenStack-Umgebung über mehrere Organisationen logisch zu unterteilen und beispielsweise unterschiedliche Authentifizierungsmechanismen wie LDAP und AD in Domänen einzusetzen. Unter Einbeziehung von Nutzerrollen können durch Domänen heterogene Managementkonzepte mehrerer Parteien einer Föderation berücksichtigt werden (**O.1** teilweise, **O.8**).

- Ein *Project* dient zur Einteilung der gemanagten IT-Infrastruktur. Ein Project in OpenStack entspricht daher eher einer administrativen Domäne, wie sie in dieser Arbeit betrachtet wird (**O.2**). Mandantenfähigkeit ist in OpenStack durch Berücksichtigung des jeweiligen Project-Kontexts für alle relevanten Dienste gegeben (**NF.14**). Projekte lassen sich als Ressourcen in OpenStack direkt verwalten (**F.5**, **O.13**). Domänenspezifische organisatorische Vorgaben sind jedoch nicht festlegbar und können nur in geringfügigem Ausmaß über das Rollenkonzept gesteuert werden (**O.14** teilweise).

- Ein *User* ist eine Entität mit Anmeldedaten und kann einer oder mehreren Domains sowie einem oder mehreren Projects zugeordnet sein (**O.4**).

- Eine *Group* ist eine Menge von *User*-Instanzen und kann ebenfalls Domains und Projects zugeordnet sein.

- Eine *Role* kann auf Ebene einer Domain oder einem Project definiert sein. Rollen können von Nutzern angelegt, mit Berechtigungen assoziiert und Users oder Groups zugewiesen werden.

Ortsinformationen werden über eine *Region* berücksichtigt (**I.2**); sie beschreiben eine Aufteilung einer OpenStack-Installation und können als Baumstruktur durch mehrere Sub-Regions eingeteilt werden. Die wichtigsten Elemente wie Projects, Roles und User sind je nach Berechtigung einsehbar (**O.9** teilweise, da ohne konkrete Berechtigungen).

Die Authentifizierung über Keystone kann mittels eines pluginbasierten Ansatzes flexibel, beispielsweise über *Time-based One-time Password* (TOTP), Mehrfaktorauthentifizierung oder LDAP erfolgen. Auch ist die Nutzung einer föderierten Identität möglich, beispielsweise über SAML 2.0 oder *OpenID Connect*. Das ist auch der einzige Kontext, in dem eine Föderation von Infrastruktur in OpenStack berücksichtigt wird. Die Autorisierung ist in Keystone durch den Betreiber festlegbar. Ein *Endpoint* eines OpenStack-Dienstes kann in drei Zugriffsebenen *admin*, *internal* (für Inter-Dienst-Kommunikation) oder *public* eingeteilt werden, die eine grobe Zugriffsbeschränkung festlegen. Eine Definition von Berechtigungen erfolgt gemäß [152] für jeden OpenStack-Dienst separat über einen regelbasierten Ansatz. Der Zugriff kann demnach bis auf Ebene von Endpoints – also einzelnen Funktionen wie das Erstellen einer VM, das Anlegen einer Rolle, etc. – der einzelnen OpenStack-Dienste gesteuert werden und ist daher sehr feingranular und explizit möglich (**F.7**, **F.8**, **O.6**). Eine aufgabenbasierte Zugriffsverwaltung ist daher indirekt über die Steuerung von Endpunkten für Rollen möglich (**F.6** teilweise). Die Definitionen müssen auf einem Knoten über eine entsprechende Datei *policy.json* im JSON-Format definiert werden. Der Autorisierungsansatz von OpenStack ist nicht in Abhängigkeit von Elementen der Organisation abhängig und darin integriert (**O.3** teilweise).

4.5.2.5 Erweiterbarkeit und Bewertung

Die Erweiterbarkeit von OpenStack ist generell durch die Entwicklung neuer Dienste gegeben, wenn auch ohne Unterstützung im Sinne eines Frameworks. Die Dokumentation ist ausführlich (**U.1**) und OpenStack stellt SDKs für unterschiedliche Programmiersprachen bereit, wodurch die Implementierung teilweise erleichtert wird (**U.2** teilweise). Die Nutzerfreundlichkeit ist jedoch durch die hohe Komplexität – auch durch die Microservice-Architektur – eingeschränkt (**U.6** teilweise).

OpenStack kann potenziell in einigen föderierten Szenarien eine geeignete Netzmanagementplattform darstellen. Es ist jedoch viel mehr als das und stellt nicht nur das Management, sondern ebenfalls die IT-Infrastruktur bereit. Als generell einsetzbare Managementplattform in FSNs weist es jedoch einige Schwächen auf: Aufseiten organisatorischer Aspekte ist OpenStack vor allem durch die Vernachlässigung von Beziehungen zwischen Föderationspartnern (z. B. Vertrauensaspekte) ungeeignet. Diese sind nicht modelliert und fließen nicht in Managemententscheidungen ein. Auch das Autorisierungskonzept sieht zwar eine feingranulare Konfigurierbarkeit vor, jedoch in einem umständlichen expliziten Ansatz unter Außerachtlassung organisatorischer Modelle. Des Weiteren können auch heterogene Managementkonzepte über Domains realisiert werden, dieser Ansatz ist jedoch unflexibel und ebenfalls zu explizit. Das Funktionsmodell ist umfangreich, aber im Wesentlichen auf die Grundfunktion von OpenStack als

VIM ausgerichtet. Netzmanagement spielt nur eine untergeordnete Rolle und ist durch wenige teilweise zueinander inkompatible oder auch redundante Dienste realisiert. Das Kommunikationsmodell ist weitestgehend für ein allgemeines Netzmanagement nutzbar, es fehlt jedoch vor allem an einer Unterstützung der Erweiterbarkeit der API innerhalb der Dienste. Das Informationsmodell berücksichtigt zum einen nicht die Unterstützung der Modellierung zentraler Komponenten aus FSNs (z. B. SDN-Controller, SDN-Infrastruktur und VIMs). Zum anderen bietet es kein umfangreiches zentrales Modell für Managementbeziehungen zwischen MOs.

4.5.3 Open Network Automation Platform (ONAP)

Die *Open Network Automation Platform* (ONAP) ist nach [153] eine Orchestrierungs-, Management- und Automatisierungsplattform für Netze und Dienste. Sie richtet sich insbesondere an IT-Netzbetreiber wie ISPs und Betreiber von Telekommunikationsnetzen, Cloud-Betreiber und Unternehmen. ONAP orientiert sich an mehreren überwiegend aus dem Telekommunikationsbereich stammenden Standards wie TM Forum, 3GPP, ETSI NFV, ETSI ZSM (vgl. Abschnitt 4.2.3) und OASIS TOSCA [154]. Netzmanagement an sich wird eher implizit betrachtet, der Schwerpunkt liegt stärker auf dem Lebenszyklusmanagement von IT-Ressourcen und Diensten.

4.5.3.1 Kommunikationsmodell

Wie in Abbildung 4.7 gezeigt ist, besteht ONAP aus einer Vielzahl miteinander über Netzschnittstellen kommunizierender Microservices (**NF.11**, **K.1**). Durch die Microservice-Architektur wird vermeintlich hohe Verfügbarkeit (**NF.10**), Skalierbarkeit (**NF.9**), Sicherheit sowie ein einfaches Management der Plattform selbst angestrebt [153]. Auch wenn die Aspekte der hohen Verfügbarkeit (durch Redundanz) und Skalierbarkeit auf die Microservice-Architektur zutreffen, bringen sie hinsichtlich der Sicherheit und einem einfachen Management der Plattform eher Nachteile. Durch die netzbasierte Kommunikation der einzelnen Dienste ist zum einen ein erheblicher Aufwand zur Absicherung notwendig und erfordert außerdem ein aufwändiges Credential-Management. Dazu nutzt ONAP seit neueren Versionen den Service-Mesh-Dienst *Istio* (**I.13**) [155]. Die netzbasierte Architektur kann eine Quelle für mögliche Performanzprobleme sein, da sie auf der Zuverlässigkeit der Netzinfrastruktur gründen. Die Performanz ist daher beträchtlich von der anwendungsfallspezifisch gewählten Topologie abhängig (**NF.1** teilweise). Ein Vorteil, der sich aus der Architektur ergibt ist, dass Dienste einfach im laufenden Betrieb ausgetauscht werden können (**F.3**).

Der Betrieb von ONAP erfordert einen enormen Einsatz von IT-Ressourcen, den potenziell viele FSN-Betreiber nicht einsetzen können oder wollen. Dieser Aspekt wird ebenfalls durch die für eine komplette Installation von ONAP angegebenen minimalen Hardwareanforderungen von 224 GB Arbeitsspeicher und 112 (virtuellen) CPUs für das Cloud-Setup deutlich [156]. Eine minimale Installation benötigt demnach 16 GB Arbeitsspeicher und beschränkt sich im Wesentlichen auf den Inventarisierungsdienst A&AI [157]. Trotz großem Ressourceneinsatz ist die Minimalinstallation nicht praktikabel einsetzbar.

Der Installationsprozess von ONAP gestaltet sich durch die vielen zusammenwirkenden Dienste und sehr hohen Ressourcenbedarf als aufwändig und fehleranfällig. Zur Vereinfachung wird daher ein Skript zur Installation auf Basis des Paketmanagers Helm bereitgestellt. Die Installation ist aufgrund mehrerer überschneidender Installationsanleitungen und Versionsprobleme nicht trivial und muss in der Praxis manuell je nach ONAP-Version angepasst werden (**F.4**). Aufgrund des hohen Aufwands ist eine über die Werkzeuge teilunterstützte Installation am realistischsten, sodass eine Plattformunabhängigkeit eingeschränkt ist (**NF.4** teilweise).

ONAP bietet gemäß [158] eine umfangreiche API an (**K.2**). Die Hauptschnittstelle basiert auf einer REST-API (**K.9**), über die auf ONAP-Funktionen aus den unterschiedlichen Diensten zugegriffen werden kann. Auch unterstützt ONAP demnach das Abonnement einiger Ereignisse. Ereignisse, die über den zentralen asynchronen Kommunikationsbus *DMaaP* ausgetauscht

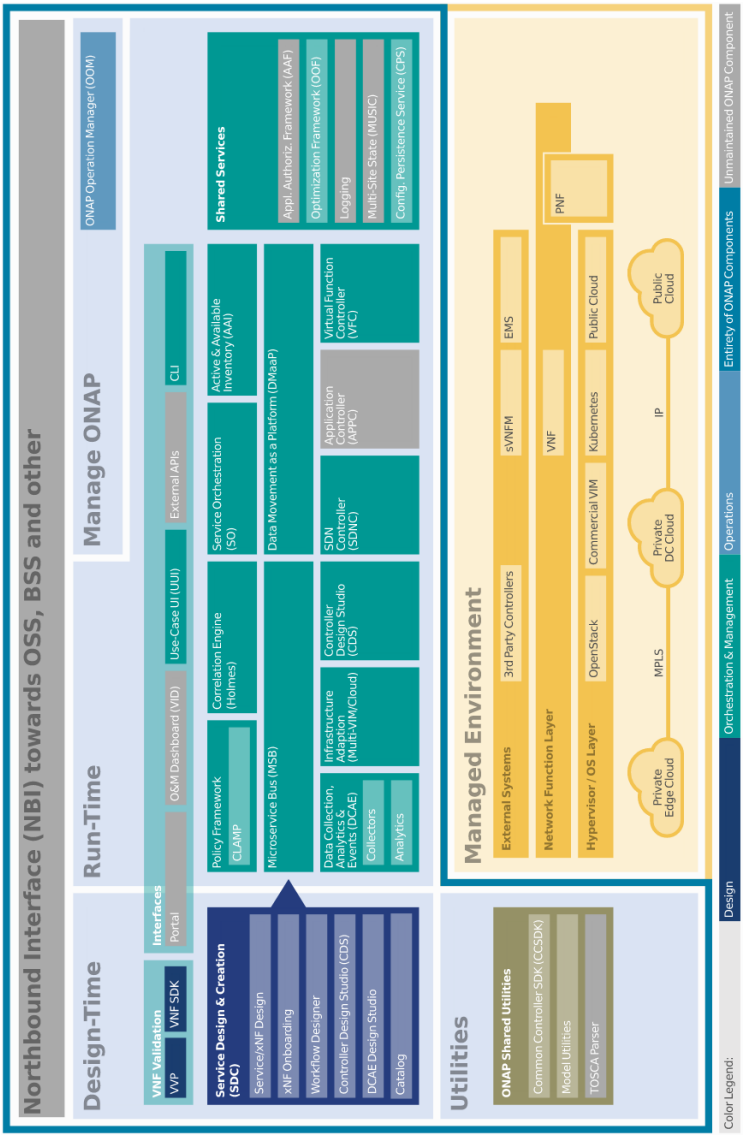


Abbildung 4.7: Architektur von ONAP, basierend auf [153].

werden, können ebenfalls abonniert oder verwaltet werden (**K.11**, **K.9**, **K.12**) [159]. Eine einfache Erweiterbarkeit des NBI ist nicht vorgesehen. Die API von ONAP ist über TLS abgesichert (**K.14**, **K.15**) [160]. Durch zentrale Zugriffspunkte ist ONAP logisch zentralisiert (**NF.7**).

Das SBI wird in ONAP insbesondere über die Komponente *MultiCloud* realisiert, welche gemäß [161] ein Kommunikationsplugin pro VIM-Modell (z. B. OpenStack oder Kubernetes) zur Kommunikation bereitstellt (**K.3**, **K.16** sowie **K.17** und **NF.3** teilweise, da nur für wenige VIMs). Demnach ist MultiCloud eine Vermittlungsebene für ONAP-Dienste zum Zugriff auf die gemanagte IT-Infrastruktur – beispielsweise für die verschiedenen Controller (u. a. SDN-Controller) oder auch den VM-Placement-Dienst (OOF) und DCAE. Eine Broker-Komponente ist in MultiCloud für die Weiterleitung von API-Anfragen an das entsprechende Adapterplugin zuständig, das mit einem VIM der gemanagten IT-Infrastruktur kommuniziert. Adapterplugins zur gemanagten IT-Infrastruktur sind implementierbare Microservices (**K.19**, **K.20**). MultiCloud Adapterplugins setzen offenbar vorwiegend Pull-Zugriffe von ONAP zur gemanagten IT-Infrastruktur um. Push-Kommunikation wird dagegen überwiegend über *Collectors* in DCAE behandelt (**K.18**, **K.8**, **K.5**) [162].

Eine Besonderheit in der Architektur von ONAP ist die Integration von Controllern wie *SDN-Controller* (SDNC) sowie *Virtual-Function-Controller* (VFC) an zentraler Stelle. Somit liegt ein wesentlicher Unterschied zu in dieser Dissertation betrachteten FSNs vor, in der das Management grundlegend heterogener Komponenten betrachtet wird. Durch die zentrale Positionierung dieser Komponenten werden sie in ONAP homogenisiert in einer Föderation betrachtet.

4.5.3.2 Informationsmodell

ONAPs Informationsmodell ist objektorientiert und kompositionsbasiert. Es setzt sich aus mehreren Teilmodellen zusammen. Die Basis bildet das *Root Model* (vgl. [163]) und beschreibt grundlegende Elemente wie IT-Ressourcen, Domänen und Datentypen. Darauf aufbauend werden Modelle unter anderem zur Beschreibung virtueller und physischer Netzfunktionen und von Netzdiensten bereitgestellt. NFV- und SDN-spezifische Komponenten wie VIMs oder SDN-Controller werden hingegen vernachlässigt und stellen nicht direkt MOs der gemanagten IT-Infrastruktur dar, sondern werden vielmehr als Teil von ONAP angesehen. Auch MOC-Funktionen gehören nicht zum Datenmodell von ONAP. Da in ONAP der Zugriff auf die gemanagte Infrastruktur insbesondere über die integrierten Controller vorgesehen ist, fällt eine Beschreibung von Funktionen von Komponentenklassen hier potenziell weg, wodurch das Informationsmodell von ONAP jedoch lückenhaft ist und eine modellbasierte Nutzung von Funktionen inklusive einer Normalisierung von MOC-Funktionen nicht ermöglicht wird. Funktionen der zentralen Controller sind nicht im Modell abgedeckt. Die Beschreibung von MO-spezifischen Gegebenheiten der Mehrschichtigkeit von Netzen, der geographischen Verteilung über ein *Location Model* (**I.2**) und der Ressourcenadressierung (**I.3**) erfüllt ONAP. Einem Cloud-Standort kann zudem ein Cloud-Provider zugewiesen werden (vgl. *Infrastructure*-Modell in Modeling-Projekt[XVI], Stand Mai 2022), wodurch Ressourcenbesitz beschrieben werden kann (**I.6**). ONAPs Informationsmodell ist stark an ETSI NFV (vgl. [164]) ausgerichtet; eine benutzerdefinierte Erweiterung ist nicht explizit im Konzept beschrieben (**I.4** sowie **I.8** teilweise). Die Nutzung ist daher nur begrenzt möglich.

Managementbeziehungen zwischen MOs berücksichtigt ONAPs Informationsmodell in Form virtueller Kommunikationsverbindungen zwischen VNFs [165]. Das *NetworkServiceDescriptor-Model* erlaubt die Beschreibung von Netzdiensten im Sinne eines *Deployment Templates* inklusive damit verknüpfter virtueller oder physischer Netzfunktionen [166]. Eine Erweiterung stellt das *Enhanced Nested Service Model* [167] dar, das lediglich eine Verschachtelung von Diensten, aber nicht von IT-Ressourcen beschreibt, wodurch beispielsweise einfache in FSNs immer vorkommende Realisierungsbeziehungen oder Kontrollbeziehungen nicht abgebildet werden. Konnektivitätsbeziehungen werden auf Ebene von virtuellen Netzfunktionen beschrieben (vgl. [165]). Durch die Außerachtlassung zentraler SN-Komponenten wie Hypervisor oder VIMs werden generelle Ressourcenabhängigkeiten im Datenmodell nur teilweise betrachtet (**I.1** sowie **I.10** teilweise).

Die Modellierung von Netzereignissen und -Zuständen wird in ONAP über das *Virtual Functions Event Streaming Model* (VES) unterstützt. Dessen Knotenelement stellt die Klasse `VESEvent` dar, von der resultierende Informationen in domänenspezifischen Gruppen (z. B. *Measurement*, *Fault*, *Perf3gpp*, usw.) abgeleitet werden (vgl. [168]). Als Domäne wird hier keine administrative Domäne bezeichnet, sondern vielmehr eine inhaltliche Gruppierung. Wie in [162] beschrieben wird, erlaubt ONAP die Sammlung anderer Formate wie SNMP-Traps oder auch RESTCONF. Die Daten werden über Mapper in das VES-Format überführt. Eine eingeschränkte Erweiterbarkeit ist in VES nach [169] über einige dafür vorgesehene Felder möglich (**I.11** teilweise, **I.13**, **I.14**). Umfangreichere benutzerdefinierte Erweiterungen sind jedoch nicht vorgesehen. Alarme werden ebenfalls in VES insbesondere durch das *Fault*-Element dargestellt (**I.12**).

[XVI] `https://git.onap.org/modeling/modelspec`

4.5.3.3 Organisationsmodell

In [153] wird explizit die Unterstützung eines zwischen Anbieter und Kunden kollaborativen Managements von Netzen und Komponenten adressiert. In ONAPs Lizenzmodell werden Kunden, Serviceanbieter oder auch Hersteller zum Zweck der Modellierung von Dienstleistungsvereinbarungen berücksichtigt [170]. Daneben werden in einem *Business Interaction Model* [171] Vereinbarungen zwischen Parteirollen, Ressourcenrollen und Kundenkonten (**NF.15**) beschrieben. Diese Modellkomponenten kommen der Beschreibung einer Föderation in ONAP am nächsten. Sie decken aber wichtige Beziehungen zwischen Parteien, insbesondere Vertrauensbeziehungen im Rahmen einer Föderation, nicht flexibel genug ab. Insgesamt scheint das Modell von ONAP hinsichtlich der Beschreibung von Föderationen nicht explizit eine Kooperation mehrerer gleichrangig agierender Parteien zu unterstützen. Vielmehr fokussiert es sich auf die Beschreibung der Anbieter-Kunden-Beziehung.

Das Organisationsmodell von ONAP sieht administrative Domänen als Sammlung von Deskriptor- oder Netzkomponenten-Entitäten vor, die einem gemeinsamen Zweck dienen [163]. Diese Sichtweise ist für FSNs jedoch nicht ausreichend, da in jedem Fall bestimmte Nutzer ebenfalls diesen Domänen angehören müssen. Ebenfalls fehlen geeignete Konzepte zur Behandlung von sich überschneidenden oder in anderer Hinsicht miteinander in Beziehung stehenden Domänen, darüber hinaus die Berücksichtigung globaler und Domänen-lokaler Organisationsaspekte wie Zuständigkeiten und damit verbundenen Aufgaben innerhalb einer Domäne und domänenübergreifend.

ONAP zielt auf eine breite mandantenfähige Implementierung ab (**NF.14**, **F.7** teilweise, da nicht in Domänenkonzept vorgesehen) [155]. Aufgaben und Zuständigkeiten werden in ONAP nicht explizit, sondern im Rahmen von Berechtigungen modelliert (**O.5**, **O.8**). In früheren ONAP-Versionen war die zentrale Komponente zur Authentifizierung und Rollen-(RBAC)-basierten Zugriffskontrolle der Dienst *Application Authorization Framework* (AAF) [172]. Seit dem ONAP-*Istanbul*-Release wird, wie in [155] beschrieben wird, die Authentifizierung und Autorisierung durch Istio gelöst (**O.3**). In ONAP wird die Authentifizierung demnach über den Dienst *Keycloak* realisiert, der im Hintergrund unterschiedliche Dienste wie LDAP oder auch das AAF nutzen kann. ONAP unterstützt einen rollenbasierten Autorisierungsansatz, der den Zugriff von Diensten untereinander und von Nutzern auf Dienste bestimmt (**O.6**, **O.4**). Eine Einbeziehung administrativer Domänen des Informationsmodells ist jedoch nicht vorgesehen (**O.2** teilweise), genauso wie fehlende Föderations- und Domänenbeziehungen oder Domänenüberschneidungen. Dennoch ist eine feingranulare *manuelle* Definition und Verwaltung von Rollen und Befugnissen nach Namensraum möglich, wodurch bestimmte domänenspezifische Aufgaben für die ganze Föderation nachgebildet werden können (**O.11** teilweise).

4.5.3.4 Funktionsmodell

Die Kernfunktionen von ONAP umfassen gemäß [153] die Folgenden:

1. Die Unterstützung der Modellierung von Diensten und IT-Ressourcen, sowie von Artefakten aus dem Management (z. B. Policies) im Rahmen eines bereitgestellten *Design-Frameworks*.

2. Ein *Analyse-Framework* zur Überwachung von IT-Ressourcen auf Basis der bereitgestellten Modelle und Informationen.

3. Ein *Orchestrierungs- und Steuerungsframework* zur Verwaltung von IT-Ressourcen auf Basis der bereitgestellten Modelle, inklusive einem automatisierten *Closed-Control-Loop* (**F.1**).

ONAPs Kernfunktionen finden sich auch in der Architektur, wie in Abbildung 4.7 gezeigt wird. Aus der Perspektive des Netzmanagements sind insbesondere die Überwachung und automatisierbare Rekonfiguration der gemanagten IT-Infrastruktur von Bedeutung. Die Überwachung wird insbesondere durch den DCAE-Dienst realisiert (**NF.6**), dessen Auswertungen darüber hinaus die Basis für automatisierte Control-Loops bilden. Control-Loops können über den Dienst CLAMP und darin notwendige Policies in TOSCA modelliert werden, die für XACML-, Drools- und APEX-*Policy Decision Points* konvertiert werden (vgl. [173]). Die Funktionalität ist nicht primär an Managementfunktionen (z. B. FCAPS) ausgerichtet.

Über seine NBI-Schnittstelle bietet ONAP weitreichende Funktionalität für die drei zuvor genannten Kernaufgaben. Gemäß [174] gehört es ebenfalls zu den Aufgaben von ONAPs Inventarisierungsdienst A&AI, Beziehungen zwischen Elementen zu erfassen und abrufbar zu machen, um Auswirkungen (z. B. bei Komponentenausfall) ermitteln zu können (**I.9**).

4.5.3.5 Erweiterbarkeit und Bewertung

ONAP bietet einige Frameworkbestandteile, beispielsweise zur Modellierung von Diensten und Closed-Control-Loops. Diese Aspekte sind jedoch expliziter Teil von ONAPs Kernfunktionen. Eine Erweiterbarkeit, die darüber hinaus geht, ist hingegen nur beschränkt möglich (**U.5** teilweise). Das Organisations- und Informationsmodell in ONAP ist daher einerseits sehr breit und umfangreich, andererseits jedoch eher starr. Trotz sehr ausführlicher Dokumentation (**U.1**) ist die Übersichtlichkeit aufgrund vieler zusammenspielender Einzeldienste und mehrerer ONAP-Releases nur eingeschränkt gegeben (**U.6** teilweise). Da die Implementierung sich stark an Standards wie ETSI NFV orientiert, kann sie als durchaus langlebig angesehen werden (**U.3**).

Die größten Einschränkungen von ONAP hinsichtlich eines Einsatzes als Managementplattform in FSNs liegen bereits in seiner generellen Architektur: Komponenten, die in FSNs als grundlegend heterogen unterstützt werden müssen, sind mit dem in ONAP integrierten SDN-Controller und Virtual-Function-Controller (sowie dem nicht mehr gepflegten Application-Controller) vereinfacht und homogenisiert an zentraler Stelle platziert. Daraus folgen Einschränkungen im Informationsmodell sowie im Kommunikationsmodell. Das Organisationsmodell unterscheidet zwar zwischen Dienstbetreibern und Kunden als Föderationspartner, Konzepte für ein gemeinsames Management oder unterschiedliche Domänen und Domänenüberschneidungen fehlen jedoch weitestgehend. Das Föderationsmodell ist entsprechend nicht flexibel genug. In der Praxis ist der Betrieb von ONAP durch die Notwendigkeit sehr vieler Ressourcen und einer fehleranfälligen Installation weiter beschränkt.

4.6 Zusammenfassung der Auswertung

Wie in Abschnitt 4.2 gezeigt werden konnte, gibt es einige Standards mit Relevanz für FSNs. Sie beschreiben unterschiedliche Aspekte des Netzmanagements:

- Standards wie SNMP, NETCONF mit YANG, CIMI sowie OCCI beschreiben Schnittstellenlösungen zur gemanagten IT-Infrastruktur. Sie können auch genutzt werden, um Netzkomponenten zu beschreiben.

- STIX mit TAXII sowie CADF beschreiben auch Schnittstellen zur gemanagten IT-Infrastruktur, jedoch mit Fokus auf Netzereignissen, -Zuständen sowie Alarmen.

- Managementframeworks wie ZSM tragen dagegen im Rahmen von Prozess- und Architekturaspekten bei, sind jedoch in der Regel zu abstrakt, um als Basis für Managementplattformen nutzbar zu sein. Darüber hinaus werden darin Managementanwendungen stark thematisiert, die jedoch auf Ebene eines Managementsystems und nicht einer Managementplattform betrachtet werden.

Standards können kaum zur Implementierung von Frameworks für Managementplattformen beitragen und betrachten nur einen (teilweise zueinander konkurrierenden) Teilbereich von FSNs. Frameworks für Managementplattformen müssen eine generische Lösung zur Implementierung unterschiedlicher Standards vorsehen und diese in FSNs zusammenführen können. Die Nutzung von Standards als Lösung für eines der vier Teilmodelle in FSNs widerspricht grundlegend der notwendigen Flexibilität von Managementplattformen: Standards sind auf Vereinheitlichung und möglichst wenig benutzerdefinierten Inhalt ausgelegt, FSNs erfordern jedoch höchste Flexibilität in der Festlegung der vier Teilmodelle.

Eine Zusammenfassung der wichtigsten Managementplattformen aus den unterschiedlichen Bereichen ist in Tabelle 4.1 gezeigt. Standards werden aus zuvor genannten Gründen nicht explizit aufgeführt. Aktuelle Managementplattformen decken bereits einige Anforderungen ab; alle weisen jedoch Stärken in ähnlichen Bereichen auf mit ebenfalls gemeinsamen Defiziten in Bezug auf meist einzelne in den vier Teilmodellen. **Dunkelblau** markierte Anforderungen werden vom jeweiligen System erfüllt und **hellblau** markierte Anforderungen teilweise erfüllt. **Weiß** markierte Anforderungen werden nicht berücksichtigt oder erfüllt.

Im **Funktionsmodell** sind demnach die folgenden **Defizite** herausgearbeitet worden:

- Dem Netzmanagement kommt gerade in Lösungen aus dem NFV-Bereich wenig Relevanz zu. Die Ansätze betrachten vielmehr die Orchestrierung und das Management von Diensten.

- Auch, wenn sich einzelne Managementplattformen an Managementfunktionen orientieren, v.a. am Fault-, Configuration-, Performanz- und für Cloud-Computing-nahe Lösungen auch Accountingmanagement, ist die Abdeckung sehr unterschiedlich und unflexibel.

- Managementaufgaben lassen sich in den bestehenden Ansätzen nicht szenarienabhängig anpassen. Die Organisation des Managements ist darüber hinaus in keiner der Lösungen an den Funktionsbereichen orientiert.

- Auch die Unterstützung organisatorischer Aufgaben wie dem Domänenmanagement wird nicht flächendeckend gelöst.

- Es fehlen teilweise ausreichend flexible und einfache Konzepte zur Automatisierung der Kernprozesse. Lösungen sind teilweise nicht einheitlich (z. B. in OpenStack) oder unnötig komplex für Managementplattformen (z. B. ONAP).

Die notwendige Flexibilität im **Informationsmodell** ist in vielen Lösungen bereits gegeben und ein wichtiger Aspekt deren Konzepte. Dennoch bestehen auch noch **Defizite**:

- Keine der Lösungen sieht die Modellierung von Komponenten aller drei Paradigmen in FSNs, der Virtualisierung, SDN und NFV vor. Auch wenn sie teilweise hinreichend flexible Modellierungsansätze bieten, unterstützen sie die Modellierung nicht ausreichend.

- Alle betrachteten Lösungen modellieren in ihrem Konzept nur Teile der Ressourcenabhängigkeiten. Darin vorkommende Mehrschichtigkeit durch Virtualisierung und die Berücksichtigung notwendiger Komponentenabhängigkeiten (z.B. Realisierungsabhängigkeiten oder Steuerungsabhängigkeiten) fehlen meist.

- Managementbeziehungen zwischen MOs sind in den meisten Lösungen nicht flexibel modellierbar. Wie im zuvor erwähnten Punkt benötigt es daher einen flexiblen Ansatz, der gleichzeitig durch Grundmodelle die Implementierung unterstützt.

Das **Kommunikationsmodell** ist gerade aufseiten des SBI von Managementplattformen durch aktuelle Ansätze sehr gut unterstützt. **Defizite** in diesem Bereich sind insbesondere im NBI und dem East-/Westbound-Interface verortet:

- Managementplattformen außerhalb des SDN-Paradigmas stellen oft eine zu anwendungsfallspezifische API mit wenigen Konfigurationsmöglichkeiten bereit. Management- oder MO-Funktionen können dann nicht an den Nutzer oder Managementanwendungen durchgereicht werden.

- Die Kommunikation zu anderen bestehenden Managementplattformen ist nicht ausreichend vorgesehen und szenarienabhängig notwendig.

Im **Organisationsmodell** gibt es gemäß der gezeigten Übersicht die größten Lücken in bestehenden Ansätzen. **Defizite** darin sind besonders die Folgenden:

- Die Modellierung einer Föderation mit ihren Partnern und Föderationsbeziehungen fehlt komplett oder ist nur sehr anwendungsfallspezifisch umgesetzt (bspw. in ONAP).

- Heterogene Managementkonzepte mehrerer Parteien in einer Föderation werden von keiner Managementplattform ausreichend berücksichtigt. Mit Ausnahme von OSM oder OpenStack, die diesen Aspekt teilweise über heterogene Rollensätze in Domänen erfüllen, betrachtet keine Managementplattform diese Anforderung hinreichend.

- Ein durch administrative Domänen strukturierte Organisation ist in den meisten Plattformen nicht vorgesehen. Beziehungen zwischen administrativen Domänen und ihre Auswirkungen auf Verantwortlichkeiten wird von keiner der Lösungen betrachtet.

- Eine Unterscheidung zwischen einem föderationsweiten Management und einem organisations- oder domänenspezifischen Management wird nicht berücksichtigt. Dazu gehört ebenfalls, dass Domänenüberschneidungen nicht behandelt werden.

- Ein geeignetes Zugriffsbeschränkungsverfahren auf Basis von Föderations-, Domänen- und Zuständigkeitsmodellen wird in keinem der bestehenden Ansätze verfolgt.

In Hinblick auf die flexible Erweiterbarkeit der Managementplattformen verfolgen diese unterschiedliche Konzepte. So sind OSM, OpenStack und ONAP vergleichsweise starr und unterstützen vor allem jeweils definierte Anwendungsfälle gut. Ihre Erweiterung ist daher nur eingeschränkt möglich und kaum unterstützt. OpenDaylight, ONOS und XOS sind hingegen sehr flexibel erweiterbar. Sie bilden entsprechend eine potenziell geeignete Grundlage, um als Basis für Managementplattformen in FSNs genutzt werden zu können.

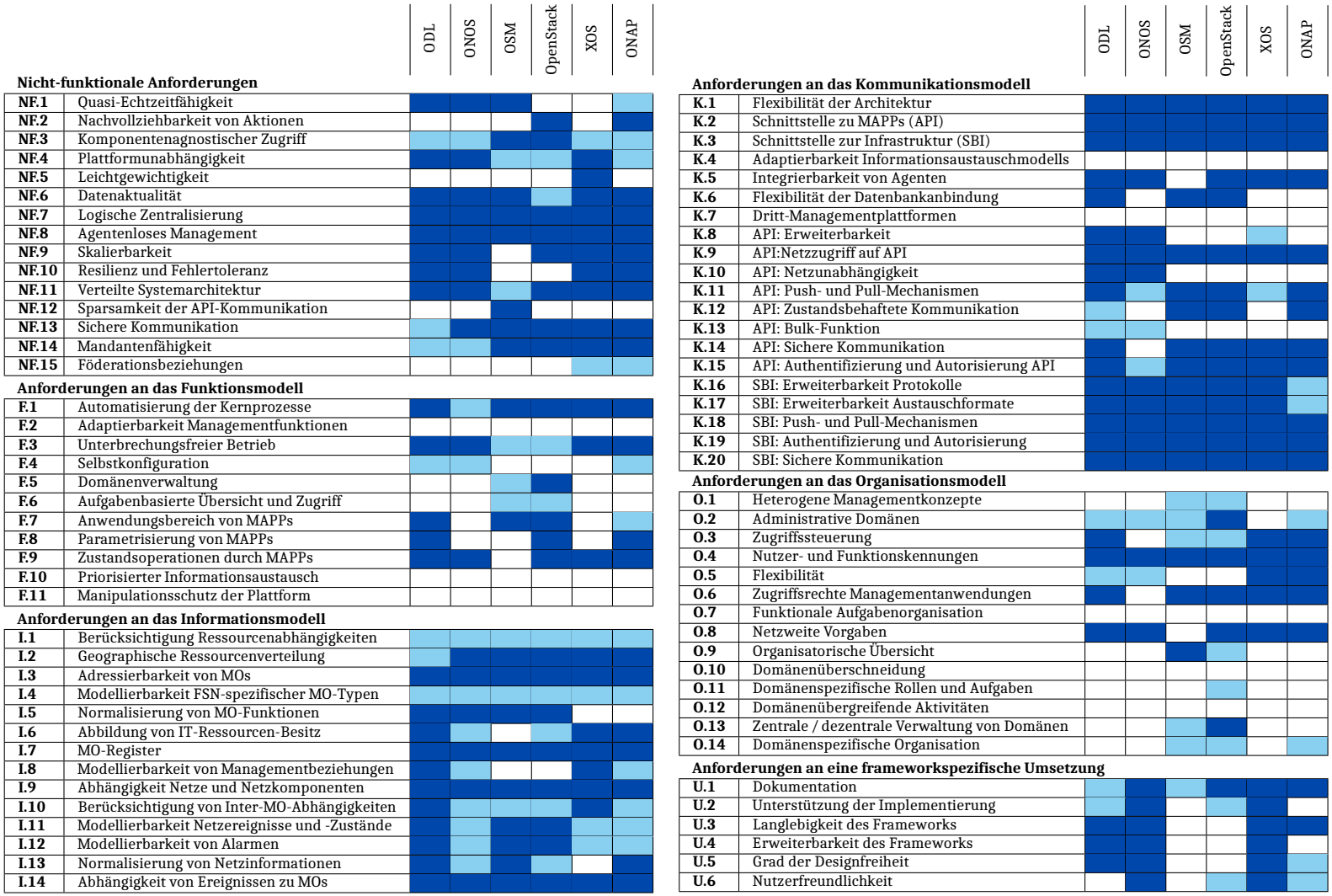


Tabelle 4.1: Bewertung bestehender Ansätze auf Basis der Anforderungen an Managementplattformen in FSNs (Dunkelblau: erfüllt; Hellblau: teilweise erfüllt; Weiß: nicht erfüllt).

Kapitel 5

Framework-Konzepte für Managementplattformen in FSNs

Inhalt

5.1 Operative Managementprozesse . . . 136
5.1.1 Überwachung . . . 137
5.1.2 Steuerung . . . 138
5.2 Darstellung von Modellierungskonzepten . . . 139
5.3 Modellübergreifende Komponenten und Hilfselemente . . . 140
5.3.1 Hilfselemente . . . 140
5.3.2 Tagging von Elementen . . . 141
5.4 Informationsmodell . . . 141
5.4.1 Framework zur Modellierung von Managementobjektklassen . . . 142
5.4.2 Framework für Managementbeziehungen . . . 148
5.4.3 Framework für MOC-Funktionen . . . 153
5.4.4 Framework für Netzereignisse und -Zustände . . . 159
5.4.5 Framework für Alarme . . . 162
5.4.6 Unterscheidung von Ist- und Soll-Zustand . . . 164
5.5 Organisationsmodell . . . 164
5.5.1 Framework für Föderationsmodelle und -Beziehungen . . . 165
5.5.2 Framework für administrative Domänen . . . 167
5.5.3 Nutzer- und Rollen-Framework . . . 171
5.5.4 Framework für Managementaufgaben . . . 172
5.5.5 Mandantenfähiger Zugriff auf Informationselemente . . . 174
5.6 Kommunikationsmodell . . . 181
5.6.1 Southbound-Interface . . . 181
5.6.2 Northbound-Interface und Inter-Plattform-Kommunikation . . . 189
5.6.3 Datenbankanbindung . . . 202
5.6.4 Zentrale Komponenten des Kommunikationsmodells . . . 207
5.7 Funktionsmodell . . . 209
5.7.1 Funktionale Managementbereiche . . . 210
5.7.2 Erweiterung der Managementplattformfunktionalität . . . 211
5.7.3 Automatisierung der Kernprozesse . . . 213
5.7.4 Verwaltung von Managementanwendungen . . . 215
5.8 Zusammenfassung der Kernkonzepte für FSNs . . . 215

Die **Zielsetzung** des in diesem Kapitel beschriebenen Konzepts von Frameworks für Teilmodelle von Managementplattformen in FSNs umfasst in erster Linie die folgenden Aspekte:

a) Die Ausarbeitung geeigneter Modellkomponenten, die zur Modellierung konkret betrachteter FSNs genutzt werden können.

b) Die Unterstützung wichtiger Kernprozesse der Überwachung und Steuerung im Management, wie sie in Abschnitt 5.1 veranschaulicht werden.

c) Die Entwicklung von Funktionalität und klaren Schnittstellen zur direkten Einbindung der Modelle und Informationen in den Managementprozess.

d) Möglichst einfache und geleitete Lösungen, um eine bessere Nutzbarkeit der Frameworks zu unterstützen.

> Die in diesem Konzept vorgestellten **Frameworks** sind grundlegend derart gestaltet, dass sie **jeweils für sich stehend** genutzt werden können und jeweils einen isolierten Teilbereich in den vier Teilmodellen des Netzmanagements betrachten. Gleichzeitig weisen sie **definierte Schnittstellen untereinander** auf, um auch für einen ganzheitlichen Lösungsansatz nutzbar zu sein.

5.1 Operative Managementprozesse

Als Grundlage für die notwendige Funktionalität und dahinterstehende benötigte Modelle dieses Konzepts für Managementplattformen in FSNs dient die Beschreibung und im Laufe dieses Abschnitts nähere Untersuchung des operativen Managementprozesses, wie er in Abbildung 5.1 gezeigt wird.

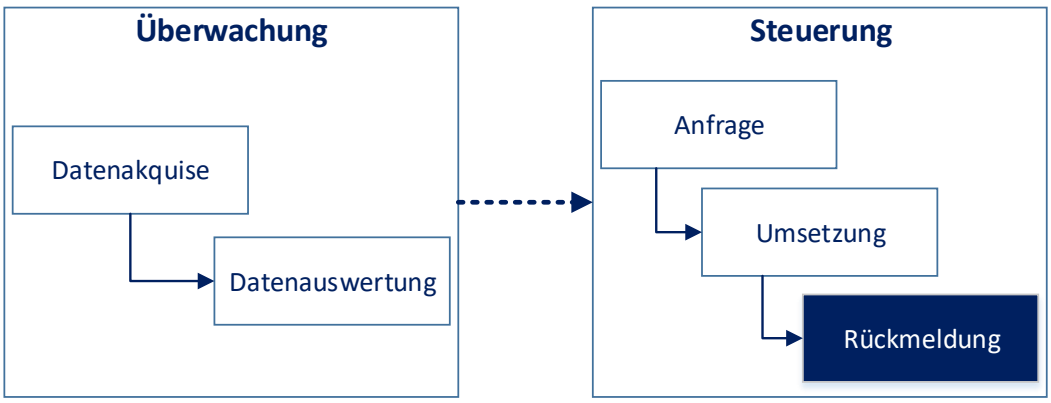


Abbildung 5.1: Kernprozesse des Managements (Weiße Box: Teil-Prozess; Blaue Box: Aktivität).

Wie bereits in Abschnitt 2.4 beschrieben wird, setzt sich Netzmanagement generell aus den Prozessen der **Überwachung** sowie **Steuerung** zusammen. Beide können auch unabhängig voneinander durchgeführt werden. Im üblichen Fall setzt der Prozess der Steuerung jedoch auf Informationen aus dem Prozess der Überwachung auf. Besonders hinsichtlich der Automatisierung des Netzmanagements greift der Prozess der Steuerung auch zeitlich unmittelbar nach dem Prozess der Überwachung ein, um beispielsweise Probleme (z. B. eine detektierte Kompromittierung eines MO) unmittelbar und selbstständig zu beheben.

5.1.1 Überwachung

Der Überwachungsprozess ist bereits in einigen bestehenden Arbeiten betrachtet worden. So beispielsweise auch für SDN- und NFV-Netze in 5G-Telekommunikationsnetzen in [175]. Die Überwachung erfordert demnach die Sammlung, Normalisierung und Auswertung der Daten. Diese Aktivitäten werden auch in dieser Arbeit als Grundlage genutzt. Der Prozess der Überwachung lässt sich daher in zwei Teilprozesse unterteilen, wie in Abbildung 5.1 gezeigt wird: einerseits die **Datenakquise** und andererseits die **Datenauswertung**.

5.1.1.1 Datenakquise

Im Prozess der Datenakquise als Teilprozess der Überwachung im operativen Managementprozess werden alle Aktivitäten mit dem Ziel der Erfassung des Zustands der gemanagten Infrastruktur durchgeführt (vgl. Abbildung 5.2). Die Datenakquise beginnt mit dem **aktiven Abfragen** (Pull) oder dem **passiven Horchen** und Bekommen (Push) von Rohdaten am SBI, entweder direkt von der **gemanagten Umgebung** (d. h. mittelbar oder unmittelbar von MOs) oder von **Managementanwendungen**. Letzteres beispielsweise in Form eines Ergebnisses einer weiterverarbeiteten Form bestehender Daten (vgl. Abschnitt 5.1.1.2). In diesem Schritt ist eine **Authentifizierung und Autorisierung** zwischen Managementplattform und der jeweiligen Gegenstelle notwendig, mindestens jedoch empfohlen, um etwaige Manipulationen (z. B. Übermittlung gefälschter Daten durch unautorisierte Systeme) zu verhindern. Das erfordert aufseiten der APIs von MOs jedoch die Unterstützung entsprechender Funktionen.

Abbildung 5.2: Prozess der Datenakquise innerhalb des Überwachungsprozesses (Weiße Box: Teil-Prozess; Blaue Box: Aktivität).

Da in FSNs Austauschformate und Informationen grundsätzlich unterschiedlich ausfallen können, ist das **Parsen** der Rohdaten für die Weiterverarbeitung notwendig. Dabei wird zunächst das Objekt in seiner **Vollständigkeit** und ohne Informationsverlust erstellt. Da jedoch unter Umständen nicht der gesamte Inhalt der übertragenen Nachricht für das Netzmanagement relevant ist, erfolgt in der nächsten Aktivität eine darauf aufbauende Normalisierung. In der Aktivität der **Normalisierung** werden daher zuvor geparste Daten in ein **einheitliches Format** innerhalb der Plattform gebracht. Die Notwendigkeit dazu besteht, da unterschiedliche Systeme (z. B. zwei unterschiedliche SDN-Controller) Ereignisse und Zustände unterschiedlich hinsichtlich Format und Inhalt beschreiben. Ihre Vergleichbarkeit im Rohformat ist daher eingeschränkt. Beispielsweise gibt es allein unterschiedliche Formate und Granularitätsgrade der Beschreibung eines Zeitpunkts; auch sind andere Informationen unterschiedlich kodiert (bspw. kann *1* genauso wie *true* je nach Kontext dieselbe boolesche Bedeutung haben). Entsprechend dient die Normalisierung der Herstellung von Vergleichbarkeit gleichartiger Informationsobjekte und -Attribute. Auf die Aktivität der Normalisierung folgt die **Speicherung** des jeweils generierten Informationsobjekts. Eine permanente Speicherung von Informationsobjekten ist notwendig, um Veränderungen in der gemanagten Umgebung nachzuvollziehen und beispielsweise auch darauf aufbauend entsprechende Auswertungen durchführen zu können. Nachdem die Daten gespeichert sind, können sie an beliebige verschiedene Managementanwendungen zur Auswertung gegeben werden.

5.1.1.2 Datenauswertung

Die Datenauswertung (siehe Abbildung 5.3) ist in diesem Konzept keine Aufgabe der Managementplattform, sondern die von Managementanwendungen. Die Hauptaufgabe der Managementplattform besteht in der Bereitstellung der zuvor akquirierten Informationen an jeweils anfragende Managementanwendungen. Hier müssen zwei Fälle berücksichtigt werden. Einerseits, dass Managementanwendungen Informationen selbst **abfragen** (Pull), andererseits, dass Managementanwendungen ein **Abonnement** von Informationen abschließen (Push). Im ersten Fall muss die Managementplattform geeignete Schnittstellen zur Informationsabfrage bereitstellen, im letzteren Fall müssen Funktionalität und Schnittstellen zur Verwaltung von Informationsabonnements durch eine Managementplattform realisiert sein. Aspekte der **Mandantenfähigkeit** sowie (**Authentifizierung und Autorisierung**) müssen dabei berücksichtigt werden, damit sensible Informationen nicht an unberechtigte Personen gelangen.

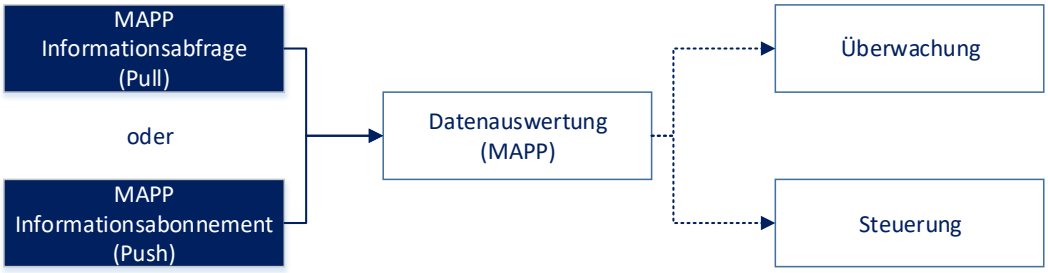


Abbildung 5.3: Prozess der Datenauswertung innerhalb des Überwachungsprozesses (Weiße Box: Teil-Prozess; Blaue Box: Aktivität).

Die in Managementanwendungen durchgeführte **Datenauswertung** kann bis zu zwei nachgelagerte Prozesse nach sich ziehen: Einerseits können Ergebnisse einer Datenauswertung selbst wieder in den übergeordneten Prozess der **Überwachung** als abgeleitete Informationen einfließen und für nachgelagerte Datenauswertungen genutzt werden. Andererseits können auf Basis von Ergebnissen einer Datenauswertung automatisierte oder manuelle **Steuerungsaktionen** ausgeführt werden.

5.1.2 Steuerung

Der Prozess der **Steuerung** bildet neben der Überwachung den zweiten *großen* Teilprozess des gesamten Managementprozesses. Er selbst setzt sich aus drei Hauptaktivitäten zusammen (vgl. Abbildung 5.1): Die **Anfrage** einer Steuerungsfunktion durch eine Managementanwendung, die **Umsetzung** der Anfrage in der gemanagten Infrastruktur sowie schließlich einer **Rückmeldung** hinsichtlich Ergebnisse und Erfolg von Steuerungsaktionen. Üblicherweise resultieren Steuerungsfunktionen in der **Änderung der Konfiguration** der gemanagten Umgebung.

5.1.2.1 Anfrage

Ein **Funktionsaufruf**, wie er in Abbildung 5.4 gezeigt ist, erfolgt im Rahmen einer Anfrage einer Steuerungsfunktion über eine **Managementanwendung** (bspw. auch eine graphische Nutzeroberfläche). Die Schnittstelle zur Steuerungsfunktion wird über das **NBI** der Managementplattform bereitgestellt. Bereits hier ist der Einsatz geeigneter **Authentifizierungsmechanismen** sowie die Berücksichtigung von **Mandantenfähigkeit** und **Autorisierung** unbedingt notwendig.

Anschließend an den Funktionsaufruf wird die Anfrage einer Steuerungsfunktion einer **Validierung** unterzogen: Nachdem die Authentifizierung bereits vorher durchgeführt wurde, müssen in dieser Aktivität feingranulare **Autorisierungsaspekte** (bzgl. Domänen- und durch Auf-

Abbildung 5.4: Prozess der Anfrage innerhalb des Steuerungsprozesses (Weiße Box: Teil-Prozess; Blaue Box: Aktivität).

gaben implizierte Berechtigungen) sowie die Stimmigkeit übergebener **Parameter** für die jeweils aufgerufene Funktion validiert werden. Schließlich folgt die **Transformierung** der durch API der Managementplattform angebotenen Steuerungsfunktion in potenziell mehrere **Funktionsaufrufe** über die Schnittstellen von MOs der gemanagten IT-Infrastruktur.

5.1.2.2 Umsetzung

Im Rahmen der Umsetzung (siehe Abbildung 5.5) des Steuerungsprozesses werden die zuvor identifizierten Schnittstellen und **Funktionsaufrufe auf MOs** entsprechend einer vorgegebenen Reihenfolge ausgeführt. Im Anschluss folgt die **Auswertung** der **Rückgabe** der jeweils aufgerufenen Funktionen auf MOs. Zunächst wird eine Aggregation der Ergebnisse vorgenommen.

Abbildung 5.5: Aktivitäten des Prozesses der Umsetzung innerhalb des Steuerungsprozesses.

5.1.2.3 Rückmeldung

Im Rahmen der Rückmeldung des Steuerungsprozesses werden die zuvor im Umsetzungsprozess aggregierten Rückgabewerte zu einem **Gesamtergebnis** der jeweiligen auf der Managementplattform aufgerufenen Steuerungsfunktion zusammengefasst. Dazu zählt beispielsweise, ob die Steuerungsfunktion erfolgreich war oder bei Misserfolg, welche konkreten Aktionen fehlgeschlagen sind. Auch können detailliertere fallbezogene Rückgabewerte übergeben werden (z. B. IDs und Attribute veränderter MOs).

5.2 Darstellung von Modellierungskonzepten

Die Beschreibung der Kernkonzepte in diesem Kapitel wird weitestgehend durch Modelle der objektorientierten Programmierung beschrieben. Die Darstellung ist im Text derart gewählt, dass die Art der **Modellelemente** eindeutig ist:

- Klassennamen im Text werden durch `Schreibmaschinenschrift` gekennzeichnet. Im Rahmen des Konzepts wird dann von **Klassen** oder **Elementen** synonym gesprochen.

- Eine Klasseninstanz bzw. ein Klassenobjekt wird als **Instanz** bezeichnet. Die jeweils instanziierte Klasse ist im Text eindeutig ersichtlich, beispielsweise als Präfix in Schreibmaschinenschrift vorangestellt. Elemente und ihre Entsprechungen werden nach Kontext austauschbar verwendet (z. B. die Klasse `SystemInstanz` für eine Systeminstanz).

Illustrationen, die Konzepte und Modellelemente unterstützend beschreiben, sind an den Konventionen von Klassen- und Sequenzdiagrammen der *Unified Modeling Language* orientiert. Aus Gründen der Übersichtlichkeit werden sie jedoch vereinfacht verwendet und beispielsweise auf die Darstellung von Attributen in Klassendiagrammen verzichtet. Ein Modellierungsbeispiel wird in Abbildung 5.6 gezeigt. Wesentliche Aspekte bei der Darstellung umfassen die Folgenden:

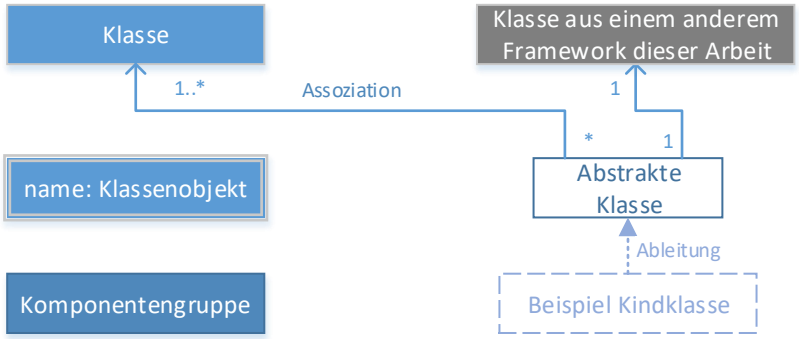


Abbildung 5.6: Modellierungsbeispiel mit Bedeutung der verwendeten Elemente.

- **Hellblaue Boxen** stellen Klassen dar.
- **Weiße Boxen** stellen abstrakte, nicht-instanziierbare Klassen dar.
- **Dunkelblaue Boxen** stellen Komponentengruppen dar.
- **Graue Boxen** beschreiben Klassen oder Komponenten aus einem anderen Teilframework, die in jeweils anderen Abschnitten detailliert beschrieben werden.
- **Boxen mit einem doppelten Rahmen** und einem Doppelpunkt-Präfix und optional vorangestelltem Instanzennamen stellen Klasseninstanzen dar.
- **Gestrichelte Elemente** stellen Beispielerweiterungen dar, die nicht Teil der Frameworks sind.

Anders dargestellte Objekte werden entsprechend beschrieben. Oft werden Frameworkkonzepte durch **Pseudocode** beschrieben. Durch den stark objektorientierten Charakter der Frameworks und ihrer Beschreibung ist auch der Pseudocode an einer objektorientierten Programmiersprache orientiert – in diesem Fall **Java**.

5.3 Modellübergreifende Komponenten und Hilfselemente

In den im Folgenden beschriebenen Frameworks sind einige modellübergreifende Komponenten notwendig, auf die im Folgenden eingegangen wird.

5.3.1 Hilfselemente

Hilfselemente beschreiben Attribute von Klassen der Frameworks. Bei der Vielzahl der in den folgenden Abschnitten beschriebenen Frameworkelementen sind **Identifikatoren** zur föderationsweit eindeutigen Adressierbarkeit von Elementen des Netzmanagements notwendig. Beispielsweise für MOs, Managementbeziehungen, Nutzer und Rollen, Aufgabenbereiche und administrative Domänen. Auf die konkrete Umsetzung föderationsweit spezifischer IDs wird in diesem Konzept nicht im Detail eingegangen. Sie werden vielmehr als anwendungsfallspezifisch implementierbar betrachtet. Eine Besonderheit von Identifikatoren ist, dass sie zum geeigneten Datenaustausch de- / serialisierbar gestaltet sein müssen. Entsprechende Funktionalität muss ebenfalls bei Definition einer Identifikatorklasse implementiert werden.

Ein weiteres Hilfselement sind **Zeitstempel**. Zeitstempel dienen der Beschreibung eines Zeitpunktes, beispielsweise des Auftretens einer Netzinformation oder der Generierung eines

Alarms. Auf die notwendige Genauigkeit eines Zeitstempels und seine Umsetzung wird in diesem Konzept ebenfalls nicht näher eingegangen. Im Rahmen der Frameworks ist eine anwendungsfallspezifisch geeignete Darstellung eines Zeitstempels zu implementieren. Auch Zeitstempel müssen de- / serialisierbar umgesetzt sein: Beispielsweise müssen Netzinformationen oder Alarme häufig mit Nutzern über das NBI ausgetauscht werden.

Darüber hinaus sind im Konzept **Standort**-Informationen notwendig. Diese beschreiben beispielsweise den Standort eines Datenzentrums. Genauso wie die vorherigen Hilfselemente sind auch Standortinformationen stark anwendungsfallabhängig beschreibbar. Eine Serialisierung von Standortinformationen hat nicht denselben Stellenwert wie die von Identifikatoren oder Zeitstempeln. Falls notwendig, ist derartige Funktionalität jedoch auch hier vorzusehen.

5.3.2 Tagging von Elementen

Das Tagging von Elementen ist in diesem Konzept insbesondere zur Regelung des mandantenfähigen Zugriffs auf Informationselemente (Abschnitt 5.5.5) notwendig. Im Konzept wird zwischen der Kennzeichnung von Klassen und von Klasseninstanzen unterschieden. Elemente, die durch Tags gekennzeichnet werden, umfassen alle Elemente der im Folgenden beschriebenen Frameworks, die durch den Prozess der Zugriffsentscheidung gesteuert werden. Diese umfassen vor allem Elemente des Informationsmodells (z. B. Repräsentationen von MOs und Managementbeziehungen), aber potenziell auch Elemente des Organisationsmodells (z. B. Nutzer- und Rollenrepräsentationen), auf die über das NBI zugegriffen werden kann. Die Umsetzung eines **Tags** wird in diesem Konzept nicht konkret vorgegeben. Beispielsweise sind Tags denkbar, die durch benutzerdefinierte selbsterklärenden Zeichenketten beschrieben werden (z. B. „föderationsweit-zugreifbar“), andererseits kann bei der Erstellung von Tags auch jede andere benutzerdefinierte Kodierung verwendet werden. Ein Tag muss eine **Vergleichsoperation** bereitstellen, durch die die Gleichheit zu einem anderen Tag bestimmt werden kann. Diese Operation muss je nach Tag durch den Entwickler implementiert werden.

5.3.2.1 Tagging von Klassen

Tagging von Klassen dient insbesondere der Kennzeichnung von Gruppen von Elementen. Das Ziel einer durch Tags gekennzeichneten Klasse ist das implizite Tagging aller Instanzen dieser Klasse. Auf diese Weise kann das Tagging benutzerfreundlicher umgesetzt werden, ohne dass jede einzelne Klasseninstanz einzeln durch ein Tag gekennzeichnet werden muss. Eine Klasse kann allgemein durch beliebig viele Tags markiert werden, um in verschiedenen Kontexten der Zugriffsbestimmung differenziert behandelt zu werden: Beispielsweise im Kontext der gesamten Föderation oder in einzelnen Domänen.

5.3.2.2 Tagging von Klasseninstanzen

Analog zu dem im vorherigen Abschnitt beschriebenen Tagging von Klassen funktioniert auch das Tagging von Klasseninstanzen. In diesem Fall hat ein Tag jedoch keinen Einfluss auf andere Instanzen derselben Klasse. Tags für Instanzen müssen an den föderationsweit eindeutigen IDs der jeweiligen Klasseninstanzen hängen, da sich Objektadressen ändern können – beispielsweise beim Laden eines Objekts aus einer Datenbank.

5.4 Informationsmodell

Das Informationsmodell wird über **fünf Teilframeworks** realisiert:

- Ein Framework für **Managementobjektklassen (MOC)**,
- ein Framework für **Managementbeziehungen**,

- ein Framework für **MOC-Funktionen**,
- ein Framework für **Netzereignisse und -Zustände** sowie
- ein Framework für **Alarme**.

Über die Trennung ist für das jeweilige Teilmodell eine geeignete fokussierte Erweiterung möglich.

5.4.1 Framework zur Modellierung von Managementobjektklassen

Die Modellierung von MOCs beschränkt sich in diesem Framework auf ihre Beschreibung über **Attribute**. Funktionen von MOCs in FSNs werden in einem separaten Modell für MOC-Funktionen in Abschnitt 5.4.3 beschrieben. Auch wenn sich FSNs in vielen Aspekten unterscheiden, sind sie ab einer gewissen Abstraktionsebene unter anderem bezüglich generischer MOCs gleichartig. Die Ähnlichkeit begründet einen **vererbungsbasierten** Ansatz. Generische MOCs lassen sich in FSNs einerseits gut in Unterklassen einteilen. Andererseits werden einige Attribute in allen MOCs benötigt. Ein **Klassifizierungssystem**, wie es beispielsweise durch OCCI (siehe Abschnitt 4.2.4.2) verfolgt wird, bildet dieses Charakteristikum besser ab als ein reiner vererbungsbasierter Ansatz. Gleichzeitig erlaubt es einfachere Änderungen zur Laufzeit. Entsprechend wird in dieser Arbeit ein kombinierter Ansatz beider Prinzipien vorgeschlagen.

5.4.1.1 Managementobjektklassen und Klassifikation

Dieses Konzept unterscheidet grundlegend zwischen eigentlichen Managementobjektklassen `FSNMOC`, und ihrer Klassifikatorklassen `FSNMOCK`.

> Durch `FSNMOC`-Klassen werden in diesem Konzept **Trägersysteme für Netzfunktionen** modelliert. Trägersysteme sind entweder virtuelle oder physische Systeminstanzen oder Anwendungen.
>
> Die eigentliche **Netzfunktion** wird in diesem Konzept über `FSNMOCK`-Klassen modelliert. Sie beschreiben entsprechend beispielsweise SDN-Controller, VIMs, Hypervisor oder andere Komponentenklassen in FSNs. Eine `FSNMOC`-Klasse kann durch mehrere `FSNMOCK`-Klassen klassifiziert sein: Bspw. implementiert eine Anwendung einen SDN-Controller und ein VIM.

Abbildung 5.7 illustriert das Gesamtkonzept zur Modellierung von MOCs. In diesem Konzept wird zunächst von jedem MOC als zentrale Systeminstanz oder aber als FSN-spezifische Anwendung ausgegangen. Eine Systeminstanz ist klassischerweise eine virtuelle oder physische Maschine mit Rechen- und Speicherkapazitäten. Eine Anwendung hingegen läuft auf einer Systeminstanz (vgl. Managementbeziehungen in Abschnitt 5.4.2). Beide sind von der Klasse `FSNMOC` abgeleitet. `FSNMOC` selbst ist generisch und nicht instanziierbar.

Ähnlich wie in OCCI aus dem Cloud-Computing-Bereich wird eine Netzfunktion durch Assoziation von Klassifikationen (im Folgenden als MOC-Klassifikation bzw. **MOCK** bezeichnet) an das jeweilige Trägersystem, eine Systeminstanz oder Anwendung, erfolgen. Die Klassifikation wird über die separate Klasse **Klassifikator** durchgeführt, wodurch Klassifizierungen auch zur Laufzeit dynamisch zuweisbar sind.

Eine Übersicht wichtiger MOCK wurde in dieser Arbeit bereits im Rahmen der Anforderungsanalyse in Tabelle 3.1 gezeigt. Sie umfasst verschiedene Komponenten aus dem NFV-Paradigma (VIM, NFVO und VNFM), aus dem SDN-Paradigma (SDNC und SDNA), Virtualisierungssysteme (VIRS) und nicht-FSN-spezifische Systeme (NFSN) und Managementanwendungen (MAPP).

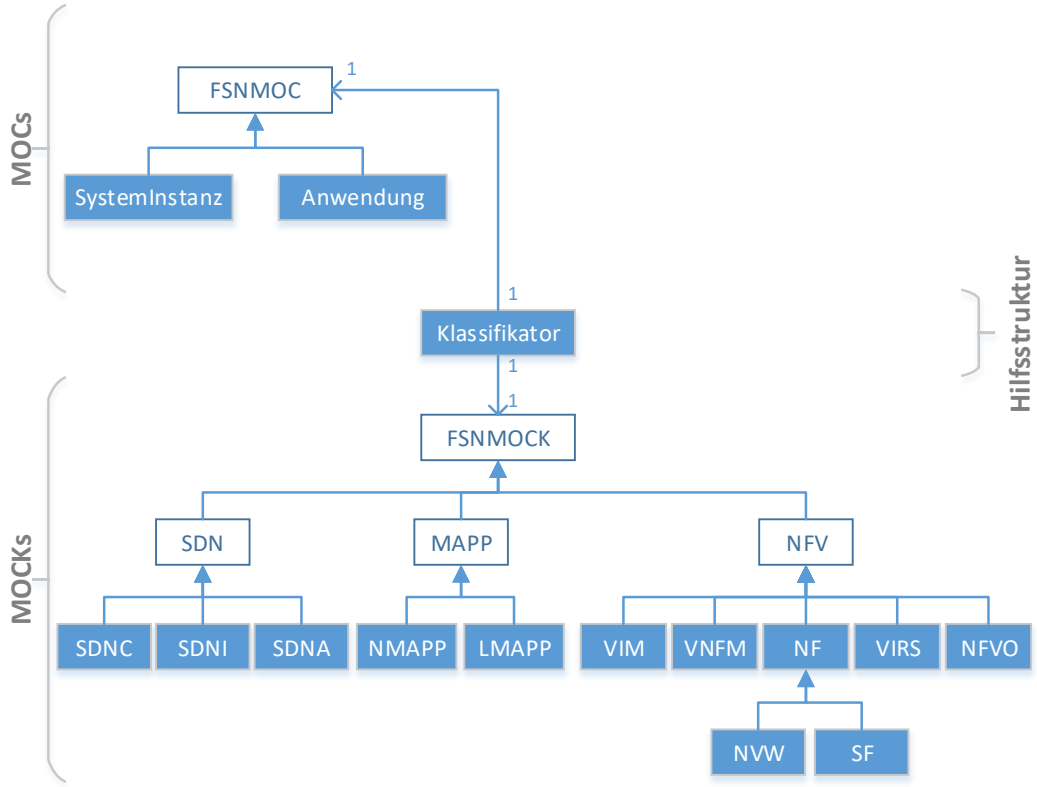


Abbildung 5.7: Managementobjektklassen (MOCs) und MOC-Klassifikationen (MOCKs) mit Ableitungsbeziehungen. Nur nicht-abstrakte MOCKs können zur Klassifikation von MOCs genutzt werden. Die Klassifikation wird durch dynamische Zuweisung über eine Verbindungsklasse `Klassifikator` durchgeführt.

NFSN sind in diesem Modell eine nicht durch eine der Unterklassen von SDN oder NFV klassifizierte Systeminstanz, können jedoch genauso von der Klasse `FSNMOCK` abgeleitet werden. Auch wenn SDN-Infrastrukturkomponenten (SDNI) aus Sicht einer Managementplattform nicht direkt zu Akteuren zählen, ist es dennoch sinnvoll, ihren Zustand über ein entsprechendes MOCK zu beschreiben. SDNI ist zunächst allgemein gehalten, SDN-Switches und weitere SDN-fähige Infrastrukturkomponenten müssen davon abgeleitet werden. Alle SDN-spezifischen MOCKs werden von einem MOCK `SDN` abgeleitet. Managementanwendungen lassen sich zudem in netzgebundene Managementanwendungen (NMAPP) und lokal angebundene Managementanwendungen (LMAPP) unterteilen. Als weitere MOCKs müssen Netzfunktionen (NF) im Allgemeinen vorgesehen sein. Sie sind dem Bereich NFV zugeordnet, da er die Trennung von Funktion und Trägerplattform beschreibt. `NF` kann selbst weiter spezialisiert werden: Zum einen in Komponenten zur Netzverkehrweiterleitung (NVW), zum anderen in Sicherheitsfunktionen (SF). Erstere umfassen klassischerweise z. B. Switches, Router, Load-Balancer oder Netz-Proxies und Letztere beispielsweise Firewalls, NIDS und Systeme zur Deep-Packet-Inspection.

Da die Klassifizierung eines MOC als `SDN`, `NFV` oder `MAPP` wenig sinnvoll ist, ist sie in diesem Frameworkkonzept nicht erlaubt. Eine Klassifizierung kann erst mit MOCKs ab der jeweils dritten Ebene vorgenommen werden. Dieses Konzept ist zusammen mit einer Übersicht über alle MOCs und MOCKs in Abbildung 5.7 illustriert. Die beschriebenen MOCKs sollen einerseits direkt assoziierbar sein, andererseits bilden sie ein Basissystem zur Ableitung spezifischerer MOCKs. Beispielsweise kann bei Bedarf ein MOCK `OpenFlow-Controller` von der MOCK `SDNC` abgeleitet werden und weitere komponentenspezifische Attribute umfassen. Auch können von der Klasse `FSNMOCK` weitere Gruppen zur Klassifikation abgeleitet werden.

5.4.1.2 Attribute

Attribute von MOs werden in den Klassifikationsklassen beschrieben, die von FSNMOCK abgeleitet sind. Sie müssen erweiterbar und kontrolliert modifizierbar sein. Eine a priori gegebene vollumfängliche Sammlung ist in der Regel nicht sinnvoll bzw. möglich, da Attribute für unterschiedliche Zwecke anwendungsfallspezifisch unterschiedlich dargestellt werden. Attribute werden selbst über einen objektorientierten Ansatz mit einem zentralen parametrisierbaren Attributtyp `Attribut` (siehe Pseudocodeimplementierung in Listing 5.1) erweitert. Die Klasse `Attribut` ist abstrakt und nicht instanziierbar. Ein Attributtyp muss sich zunächst neben dem eigentlichen Werte-Typ `T` durch ein **Name**-Feld sowie eine informelle **Beschreibung** des Zwecks auszeichnen, damit ein Nutzer des Attributtyps, beispielsweise bei der Modellierung eines neuen MOCK, dieses sinnvoll einsetzen kann.

```
1  abstract class Attribut<T> {
2
3    // Referenz auf Zugriffsprüfer
4    ZugriffsPrüfer zPrüfer;
5
6    T Wert;
7    String Name;
8    String Beschreibung;
9    ZugriffsLevel Zugriff;
10
11   abstract boolean prüfeWert(T w);
12
13   public void setzeWert(T w) {
14     if (zPrüfer.kannSchreiben(Zugriff) and prüfeWert(w, index)) {
15       Wert = w;
16     }
17   }
18
19   public T holeWert() {
20     if zPrüfer.kannLesen(Zugriff) {
21       return Wert;
22     }
23   }
24 }
```

Listing 5.1: Pseudocodedarstellung des Attributtyps `Attribut`.

Attribute müssen unterschiedlich zugreifbar sein. Um komplexere Zugriffssysteme wie das in SNMP bzw. SMIv2 abbilden zu können, wird die Art des Zugriffs über einen benutzerdefinierbaren Aufzählungstyp `ZugriffsLevel` kontrolliert. Dafür hat jeder Attributstyp eine Referenz auf eine Instanz der Klasse `ZugriffsPrüfer`, in dem Entwickler für anwendungsfallspezifisch erstellte Zugriffslevel eine **zentrale Zugriffssteuerung** auf Attributebene ermöglichen. Die Klasse `ZugriffsPrüfer` muss immer Methoden zum lesenden und schreibenden Zugriff implementieren, da auf diese bereits in `Attribut` in den entsprechenden Methoden `setzeWert` und `holeWert` zugegriffen wird. Die Klasse `ZugriffsPrüfer` kann um beliebige Prüfmethoden erweitert werden. Diese Form der Berechtigungsmodellierung und -Überprüfung reguliert lediglich allgemeine Lese- und Schreibmöglichkeiten eines Attributs: Es legt fest, **ob und wie** ein Attribut verwendet werden kann, jedoch **nicht von wem**. Eine allgemeine Zugriffskontrolle von Elementen des Informationsmodells für Nutzer der Managementplattform wird in Abschnitt 5.5.5 im Organisationsmodell behandelt. Der Zugriff auf Attribute wird stets über Methoden des Frameworks für MOC-Funktionen in Abschnitt 5.4.3 forciert. Bei der Erstellung eines neuen Attributs muss die abstrakte Templatemethode `prüfeWert` implementiert werden. Durch sie kann ein Entwickler **gültige Wertebereiche** oder Muster beschreiben, die ein Attributstyp annehmen kann (z. B. ein regulärer Ausdruck zum Prüfen einer korrekten IP-Adresse). Neben der Berücksichtigung von **skalaren Typen** wie Integer, String und Boolean, sowie darauf aufbauenden Datentypen (z. B. IP-Adresse, Hostname, oder eine OpenFlow DataPath ID) müssen auch **Referenzen auf MOs** (z. B. Liste von VMs, welche von einem Hypervisor realisiert werden) berücksichtigt werden.

```
abstract class Komplex<U,V> {

  // Referenz auf Zugriffsprüfer
  ZugriffsPrüfer zPrüfer;

  Tabelle<U,V> Werte;
  String Name;
  String Beschreibung;
  ZugriffsLevel Zugriff;

  abstract boolean prüfeEintrag(U key, V value);

  public Integer einträge() { return Werte.size();}

  public void setzeWert(U key, V value) {
    if (zPrüfer.kannSchreiben(Zugriff) and prüfeEintrag(key, value)) {
       Werte[key] = value;
    }
  }

  public V holeWert(U key) {
    if zPrüfer.kannLesen(Zugriff) {
      return Werte[key];
    }
  }

  public void entfWert(U key) {
    if (zPrüfer.tabEntferne(Zugriff)) {
      Werte.lösche(key);
    }
  }
}
```

Listing 5.2: Pseudocodedarstellung des Attributtyps `Komplex`.

```
// Schlüssel in Stapel sind vom Typ Integer
abstract class Stapel<V> extends Komplex<Integer, V> {

  abstract boolean prüfeEintrag(Integer key, V value);

  public void push(V value) {
    // key selbst generieren
    Integer key = this.einträge();
    this.setzeWert(key, value)
  }

  public V pop() {
    Integer key = this.einträge();
    if (key > 0) {
      this.holeWert(key-1);
      this.entfWert(key-1);
    }
  }
}
```

Listing 5.3: Pseudocodedarstellung der beispielhaften Ableitung des Attributtyps `Stapel` von `Komplex`.

Zusammengesetzte Datentypen wie Arrays oder Tabellen sind auch in allen gängigen Modellen notwendig. Dazu wird ein gesonderter Attributstyp `Komplex` (vgl. Listing 5.2) eingeführt, welcher aufgrund ihrer Zugriffsoptimiertheit und Flexibilität (andere Datentypen lassen sich damit einfach nachbauen) auf **assoziativen Arrays** (im Folgenden als *Tabelle* bezeichnet) basiert. Der Zugriff kann auf Tabellen als Ganzes gesteuert werden. Eine weitere Methode `einträge` gibt die Anzahl der Elemente in der genutzten Tabelle zurück. Über die abstrakte Methode `prüfeEintrag` können auch Restriktionen in Schlüsseln oder Werten implementiert werden.

Auch müssen Elemente aus der Tabelle entfernbar sein (Methode `entfWert`). Der zusammengesetzte Datentyp `Komplex` kann entweder direkt als Tabelle verwendet werden oder als Basis für die Ableitung neuer Datentypen dienen. Beispielsweise ist zur Beschreibung von Managementinformation oft ein **Stapel** (d. h. eine Liste mit *Last-In-First-Out*-Zugriff) sinnvoll. Dieser beispielhafte `Stapel` kann, abgeleitet von `Komplex`, um entsprechende Funktionen `push` und `pop` erweitert werden, wie das Beispiel in Listing 5.3 zeigt.

5.4.1.3 Fixe FSN-spezifische Attribute

In Abschnitt 5.4.1.1 wurden MOCs und MOCKs des Frameworks eingeführt. MOCs als eigentliche Träger einer Netzfunktion in Form einer Systeminstanz oder Anwendung haben genauso wie MOCKs **Standardattribute** zur Beschreibung der eigentlichen Netzfunktion. Sie werden an alle jeweiligen Unterklassen vererbt. Eine Übersicht zu notwendigen Attributen der beschriebenen generischen MOCs sowie MOCKs ist in Tabelle 5.1 zusammengefasst. Anwendungsfallspezifische Attribute können nachträglich hinzugefügt werden.

MOC-Bezeichnung		**Standard-Attribute**
Managementobjektklassen		
FSNMOC		ID, Kurzbeschreibung, Dienstzugriffspunkt
	SystemInstanz	Typ (physisch, virtuell), Systemarchitektur, Anzahl CPU-Kerne, Primärspeicher, Sekundärspeicher, Online-Status
	Anwendung	
MOC-Klassifikationen		
FSNMOCK		Version, Kurzbeschreibung, Liste von `MOCFunktion`, abstrakte-Klasse-Markierung, Liste obsoleter Attribute

Tabelle 5.1: Standardattribute generischer MOCs sowie MOCKs in FSNs. Einrückungen entsprechen einer Vererbungshierarchie.

5.4.1.3.1 FSNMOC Alle MOs müssen einen minimalen Satz an Attributen haben, damit sie managebar sind. Diese Attribute werden in der Klasse `FSNMOC` gekapselt und über Vererbung an alle davon abgeleiteten MOCs weitergegeben. Dazu zählt eine föderationsweit eindeutige **ID**, um auf Instanzen der MOCs zugreifen und diese adressieren zu können. Eine **Kurzbeschreibung** erlaubt für jedes MO, z. B. im Rahmen von Benachrichtigungen oder Nutzeroberflächenelementen, eine kontextuelle Einordnung. Jedes MO wird zudem durch einen **Dienstzugriffspunkt** beschrieben, über den *MOC-Funktionen* zum Management des jeweiligen MOs zugreifbar sind. Diese MOC-Funktionen werden in Abschnitt 5.4.3.2 beschrieben. Weitere wichtige Informationen, beispielsweise zur Organisation (u. a. ein Besitzer, Datenzentrum oder involvierte Domänen) einer Systeminstanz werden dynamisch in den Frameworks des Organisationsmodells ergänzt (vgl. Abschnitt 5.5) und sind nicht direkt an das jeweilige MO gebunden.

Eine **Systeminstanz** erweitert diesen Satz um das Attribut **Typ**, da sie entweder eine *virtuelle* oder eine *physische* Instanz ist. Darüber hinaus müssen entsprechende Systeminformationen beschrieben werden, mindestens die **Systemarchitektur**, die **Anzahl der CPU-Kerne**, die Größe des verfügbaren **Primär-** sowie **Sekundärspeichers**. Zudem ist als eine der grundlegendsten Eigenschaften der Überwachung die Erreichbarkeit bzw. der **Online-Status** eines MO im Netz notwendig. Netzinformationen (insb. Netze, Adapter, Adressen) werden nicht direkt in dieser MOC-Beschreibung geführt, sondern separat im Abhängigkeitsmodell beschrieben (vgl. Abschnitt 5.4.2.2).

Eine FSN-spezifische **Anwendung** wird aufgrund der Spannweite der Umsetzungsmöglichkeiten durch keine Standardattribute beschrieben. Das Element erbt jedoch alle Attribute der Klasse `FSNMOC`.

5.4.1.3.2 FSNMOCK Netzfunktionen müssen unter anderem durch ihre hohe Heterogenität, insbesondere durch ihre jeweilige **Version** unterschieden werden. Diese beeinflusst potenziell anwendungsfallspezifische Attribute und MOC-Funktionen. Jede Klasse wird darüber hinaus durch eine informelle **Kurzbeschreibung** verständlich beschrieben. Das Element `FSNMOCK` und alle davon abgeleiteten Elemente enthalten zudem eine Liste von unterstützten **MOC-Funktionen**. Diese werden in Abschnitt 5.4.3.1 näher beschrieben. Im Kontext von MOC-Funktionen ist die Markierung als **abstrakte** (hinsichtlich MOC-Funktionen nicht-instanziierbare) Klasse notwendig. Wie in Abschnitt 5.4.1.5 beschrieben wird, erhält jedes MOCK zudem eine Liste zur Kennzeichnung obsoleter Attribute.

Eine Grundlage zur konkreten anwendungsspezifischen Modellierung von NFV-Komponenten und -Ressourcen kann das von der ETSI entwickelte NFV-Informationsmodell bieten (vgl. referenzierte Archivdatei in Anhang A von ETSI NFV-IFA 015 [176]).

5.4.1.4 Erweiterung des MOC-Modells

Die Erweiterung des MOC-Modells ist über **drei Fälle** vorgesehen:

1. Ableitung spezifischerer `FSNMOC`-Klassen,
2. Ableitung spezifischerer `FSNMOCK`-Klassen, und
3. Erweiterung eines bestehenden MOCKs um ein **Attribut**.

Im **ersten Fall** werden Trägersysteme für Netzfunktionen näher spezifiziert. Beispielsweise ist eine Unterscheidung von herkömmlichen Systeminstanzen und Low-Power-Systeminstanzen denkbar.

Im **zweiten Fall** werden bestehende MOCKs bzw. Netzfunktionen weiter spezialisiert. Ein Entwickler identifiziert dafür das geeignete Eltern-MOCK im Vererbungsbaum und leitet ein neues Element davon ab. Auch ist eine **Mehrfachzuweisung** von MOCKs zu MOCs denkbar und möglich. Auf diese Weise ist die Problematik hinsichtlich einer teilweise schwierigen Trennung von Systemen in FSNs behandelbar. Beispielsweise könnten SDN-Controller (Klasse `SDNC`) in mehrere Subklassen eingeteilt werden, `OpenFlowC` für OpenFlow-basierte Controller und `OVSDBC` für OVSDB-basierte Controller. Auch gibt es SDN-Controller wie OpenDaylight, die mehrere Konzepte unterstützen und folglich von beiden und weiteren Subklassen abgeleitet werden müssten. Gegenüber einem klassischen reinen (baumförmigen-) Vererbungsansatz von MOCs hat das auf dem OCCI-basierten Klassifikationssystem den Vorteil, dass solche Mehrfachzuweisungen inhärent möglich sind. Eine eindeutige Trennung bei Mehrfachvererbung wird – ähnliche wie in der YANG Modellierungssprache durch Modul-*Prefixes* – durch Angabe des **vollen Attributsnamens**, hier beschrieben durch *MOCKID*:*Attributname* erwirkt. *MOCKID* steht für den Namen des entsprechenden MOCK, in dem das Attribut **ursprünglich definiert** wird. Bei Fremdzugriff auf ein anderes MO wird die Eindeutigkeit durch Einbeziehung der ID der MOC-Instanz `FSNMOC.ID` hergestellt: *FSNMOC.ID*:*MOCKID*:*Attributname*.

Im **dritten Fall**, der Erweiterung eines MOCK um ein Attribut, müssen auch alle abgeleiteten MOCKs das **neue Attribut erben**. Aus praktischer Sicht ist es wünschenswert, dass diese Vererbung **unmittelbar** und **automatisch** vorgenommen wird, ohne dass ein Entwickler beispielsweise jeden Kind- und Kindeskind-Knoten manuell identifizieren und neu kompilieren muss. Durch das im vorherigen Absatz beschriebene Prinzip der **Attributsnamensreferenzierung** ist diese unmittelbare Vererbung jedoch bereits berücksichtigt und abgeleitete MOCs müssen **nicht nachträglich angepasst** werden.

5.4.1.5 Versionen von MOCs

Netzkomponenten haben in FSNs genauso wie in herkömmlichen Netzen **unterschiedliche Versionen**. Neue Versionen von Netzkomponenten können *a)* **neue Attribute** mit sich bringen oder auch zu *b)* **Obsoleszenz von Attributen** früherer Versionen führen. Der erstere Fall *a)* lässt sich durch den beschriebenen objektorientierten und vererbungsbasierten Ansatz einfach umsetzen, indem neue Versionen von bestehenden MOCKs abgeleitet, mit entsprechendem Identifikator benannt (z. B. `OpenFlowC1_5`) und um entsprechende neue Attribute erweitert werden. Der zweite Fall macht die Einführung eines optionalen **Registers obsoleter Attribute** von Vorfahren-Klassen in jedem MOCK notwendig. Das Register wird bei Ableitung von MOCKs nicht vererbt, sondern von Eltern-MOCK zu Kinder-MOCKs kopiert und kann für jedes MOCK individuell angepasst werden.

5.4.1.6 Verwaltung von MOC-Instanzen

Instanzen von MOCs als Abbildungen des aktuellen Zustands tatsächlicher MOs einer gemanagten IT-Infrastruktur haben entsprechend genau eine gültige Version im Speichermodell der Managementplattform. Dennoch ist eine **Historie** von Zuständen der MOC-Instanzen potenziell szenarienabhängig notwendig, beispielsweise zum Nachvollziehen von Änderungen des Zustands eines MO im Rahmen einer Fehlersuche im Netz.

Die Verwaltung der Daten muss hinter dem MOC-Modell szenarienspezifisch in einer der Managementplattform nachgelagerten Datenbank umgesetzt werden. Vor allem die Suche und der Zugriff auf eine MO-Instanz muss dabei durch die Managementplattform szenarienspezifisch unterstützt werden. Beispielsweise ist in Abschnitt 5.6.1.1 die Identifikation einer MO-Instanz auf Basis seiner IP-Adresse und des jeweiligen Datenzentrums, auf dem es installiert ist, möglich.

Das Speichern des **aktuellen Zustands** von MOC-Instanzen deckt darin einen Standardfall ab und ist über Speicheroperationen einfach realisierbar. Der zweite Fall, das Speichern einer **Historie von Zuständen** von MOC-Instanzen, ist genauso realisierbar und eine Frage der Umsetzung der Speicheroperation und der Daten der Datenbank. Ist im ersteren Fall dazu weder in den Speicheroperationen als Parameter noch in der Datenbank selbst ein zeitlicher Parameter notwendig, so müssen Zeitstempel in letzterem Fall hinzugefügt werden. Auch werden in ersterem Fall Daten in der Datenbank direkt überschrieben, in letzterem Fall ersetzt, der vorherige Zustand jedoch aufgehoben.

5.4.2 Framework für Managementbeziehungen

Eine ebenfalls mit der MOC-Modellierung zusammenhängende Thematik ist die Modellierung von Beziehungen oder Abhängigkeiten zwischen MOs untereinander. Wie in den Szenarien in Kapitel 3 verdeutlicht wurde, sind diese Abhängigkeiten essenziell bei der Überwachung und Steuerung eines FSN, da die Konfiguration von Systemen von anderen darüberliegenden Systemen beeinflusst oder vorgenommen wird.

Eine sinnvolle Definition und Verallgemeinerung von Managementbeziehungen kann zunächst auf Basis identifizierter Abhängigkeiten zwischen MOs in FSNs gestaltet werden. Als Grundlage der folgenden Auflistung dienen auch unterschiedliche Vorarbeiten: Beispielsweise gemäß [177] das Bestreben, Abhängigkeiten in NFV-Netzen zu formalisieren oder auch ein aus [178], ein feingranulares Abhängigkeitenmodell zur Fehlersuche im Netz. Demnach und gemäß den in dieser Arbeit gewählten Szenarien sind folgende Abhängigkeiten zu berücksichtigen:

- **Kontrollabhängigkeiten**: Einerseits kontrollieren beispielsweise VIMs mehrere Virtualisierungssysteme, welche wiederum VMs kontrollieren. Andererseits werden, wie z. B.

in [179], mehrere SDN-Infrastrukturkomponenten von einem SDN-Controller kontrolliert. VIMs wiederum werden teilweise, SDN-Controller jedoch gemäß SDN-Paradigma grundsätzlich von Anwendungen kontrolliert. Dieselbe Abhängigkeit besteht auch zwischen Managementanwendungen und einer Managementplattform. Hierdurch ergeben sich auch transitive Beziehungen. Diese Art von Abhängigkeiten müssen in der Steuerung eines MO berücksichtigt werden.

- **Realisierungsabhängigkeiten**: Gerade in SNs sehr häufig und mehrschichtig (d. h. transitiv) vorkommende Abhängigkeiten bestehen bezüglich der Realisierung eines Systems. Da die meisten Systeme virtuelle Instanzen (oft auch mehrfach geschachtelt) sind, sind sie vom zuverlässigen Funktionieren der Systeme darunterliegender Virtualisierungsebenen abhängig. Hierzu kann jedoch auch gezählt werden, dass Systeme Ressourcen einbinden (z. B. VMs nutzen Sekundärspeicher aus einem Speichersystem), ohne deren Verfügbarkeit die Systeme selbst ebenfalls nicht mehr funktionieren. Und auch Netzverbindungen mit diesen Abhängigkeiten existieren, beispielsweise realisiert ein physischer Link oft (jedoch nicht immer, z. B. bei systeminternen virtuellen Links) eine Vielzahl virtueller Verbindungen. Darüber hinaus zählen auch Beziehungen zwischen Anwendungen und Systeminstanzen (vgl. Abschnitt 5.4.1.1) hinzu. Die *Network Markup Language* (NML) [180] ist beispielsweise ein Ansatz einer Sprache zur Beschreibung von mehrschichtigen Netzen über mehrere Domänen und Abhängigkeiten darin. NML bezieht sich jedoch explizit auf die Modellierung der Datenebene und lässt die Kontrollebene (d. h. Controller und Managementanwendungen) außen vor.

- **Netzabhängigkeiten**: Die Verbindung zwischen zwei MOs wird in diesem Konzept ebenfalls als Abhängigkeit modelliert. So sind beispielsweise Hosts bzw. Instanzen über eine Netzverbindung zu Switches an das Netz angebunden. Bei Ausfall einer Netzkomponente oder einer Verbindung fällt ebenfalls die jeweilige durch Instanzen realisierte Funktion im Netz weg. Auch Komponenten zur Netzverkehrsweiterleitung sind miteinander verbunden, z. B. Router und Switches untereinander.

- **Systemzugehörigkeitsbeziehung**: Besonders aufgrund der geographischen Verteilung in FSNs operieren Systeme darin teilweise selbst als vernetztes Cluster wie es beispielsweise im SDN-Controller OpenDaylight der Fall ist. Aber auch davon unabhängig stellen oft mehrere verteilte Systeme einen Dienst gemeinsam bereit.

Abhängigkeiten, wie sie oben gruppiert und beispielhaft erläutert wurden, zeigen dabei unterschiedliche, bei der Modellierung zu berücksichtigende Charakteristika:

- **Transitivität** von Abhängigkeiten ist häufig anzutreffen (z. B. bei Kontroll- oder Realisierungsabhängigkeiten).

- Die **Richtung** ist entscheidend und muss eindeutig sein. Abhängigkeiten können **unidirektional** (z. B. bei SDN-Controller *kontrolliert* SDN-Switch) oder auch **bidirektional** (z. B. bei Switch-zu-Switch-Netzverbindungen) sein.

- **Kardinalität**: Ein MO kann mehrere Abhängigkeiten zu anderen MOs aufweisen.

5.4.2.1 Abhängigkeitselemente

Abhängigkeitselemente sind Elemente, zwischen denen Abhängigkeiten in einem Modell beschrieben werden. Im Informationsmodell sind das **Instanzen** der Klasse `FSNMOC` und alle davon abgeleiteten Klassen (vgl. Abschnitt 5.4.1 und Unterabschnitte). Diese bilden daher die Schnittstelle zwischen dem MOC-Modellierungs-Framework und dem hier beschriebenen Managementbeziehungsframework. Im Kontext einer konkreten Abhängigkeit werden die Elemente in `Subjekt` und `Objekt` eingeteilt, um die Abhängigkeitsrichtung eindeutig abbilden zu

können. Eine Abhängigkeit kann praktisch als **Prädikat** angesehen werden, wobei nach dem Muster ***Subjekt Prädikat Objekt*** Abhängigkeiten formuliert werden können. Beispielsweise *SDNC steuert SDNI*, wodurch eine Konsistenz in der Definition von Abhängigkeiten erreicht wird. Passiv definierte Abhängigkeiten (z. B. *SDNC wird_gesteuert SDNI*) werden ausgeschlossen. Auch muss automatisch erkennbar sein, welche Komponente von in der definierten Abhängigkeit von der anderen abhängig ist. Dies ist jedoch nach Abhängigkeitstyp unterschiedlich: Beispielsweise ist eine VM vom darunterliegenden VIRS abhängig (z. B. *VIRS realisiert VM*), hingegen ein Switch in einem Netz nicht unbedingt von einem anderen (z. B. *Switch ist_verbunden_zu Switch*). Diese Aspekte müssen je nach Abhängigkeitsbeziehung definiert sein, wie im nächsten Abschnitt erläutert wird.

5.4.2.2 Abhängigkeitsbeziehungen

Abhängigkeitsbeziehungen müssen einerseits erweiterbar sein, da je nach gemanagter Umgebung potenziell unterschiedliche Beziehungen von Bedeutung sind. Andererseits müssen sie in einer Managementplattform einen Zweck bzw. eine Funktionalität erfüllen (z. B. Ereignisauslösung bei Zustandsänderung). Aus diesem Grund werden Managementbeziehungen selbst als Klassen mit implementierbarer Funktionalität beschrieben. So sind sie an sich unabhängig von anderen Komponenten verwaltbar und gleichzeitig flexibel modifizierbar.

In diesem Konzept werden Abhängigkeiten mit vertikalem als auch horizontalem Charakter unterschieden. Abhängigkeiten mit **vertikal transitivem Charakter** erfordern beispielsweise Funktionalität zum **Durchlaufen** der Beziehungen in eine **bestimmte Richtung**. Beispielsweise soll, ausgehend von einer virtuellen Instanz, die jeweilige VIRS-Kontrollinstanz gefunden werden, von welcher aus wiederum die jeweilige VIM-Komponente (oder auch hier wiederum darüberliegende VIM-Komponenten) ermittelt werden muss, um die jeweilige virtuelle Instanz korrekt zu steuern. Kontroll- und Realisierungsabhängigkeiten haben vertikalen Charakter. Netzabhängigkeiten genauso wie Systemzugehörigkeitsbeziehungen haben dagegen einen deutlich **horizontal transitiven Charakter**. Ein Durchlaufen zusammenhängender Abhängigkeiten ist auch hier eine sinnvolle Funktion, jedoch stets auf **derselben Ebene**. Entsprechend ergibt sich daraus eine konzeptionelle Trennung im Modell zur Beschreibung von Abhängigkeitsbeziehungen, wie in Abbildung 5.8 gezeigt ist. Ähnlich wie in OCCI (vgl. Abschnitt 4.2.4.2) wird auch in diesem Konzept jegliche Art von Beziehung zwischen MOs über ein gemeinsames Klassenobjekt modelliert.

5.4.2.2.1 Klassenstruktur Eine Wurzelklasse `Abhängigkeit` wird durch die Klassen `AVertikal` sowie `AHorizontal` spezialisiert, durch die zwischen horizontalen und vertikalen Abhängigkeiten unterschieden wird. Sie unterscheiden sich über ihre Funktionen. Die Klasse `Abhängigkeit` hat, neben einer **ID**, zwei Attribute **Subjekt** und **Objekt** (vgl. vorheriger Abschnitt). Ein **Prädikat** beschreibt die Art der Abhängigkeit und drückt sich über den Klassennamen selbst aus. So wird `AVertikal` durch die Klassen `AVSteuerung` für eine Steuerungsabhängigkeit und `AVRealisierung` für eine Realisierungsabhängigkeit spezialisiert. Analog wird `AHorizontal` durch `AHNetz` als Netzabhängigkeit und `AHSystem` als Systemabhängigkeit spezialisiert. Kernmerkmale von Elternklassen werden zur Verdeutlichung von Merkmalen jeweils als Teil des Namens von Kindklassen behalten. Eine beispielhafte Erweiterung wird an den Netzabhängigkeiten `TAHNetz` demonstriert: Ethernet-Verbindungen `AHNEthernet` (z. B. inkl. MAC-Adressen und VLAN-IDs) oder IP-Verbindungen `AHNIP` (mit IP-Adressen) stellen Spezialisierungen von Netzabhängigkeiten dar.

5.4.2.2.2 AVertikal Die Klasse **AVertikal** bietet und vererbt die Funktionen `hoch` sowie `runter`. Sie schreiten entweder eine Ebene hoch oder runter im durch verkettete Abhängigkeiten geschaffenen Abhängigkeitsgraphen. Der Begriff der *Ebene* unterscheidet sich je nach Abhängigkeit, beispielsweise im Kontext der Steuerung oder Realisierung. Die Richtung der Funktionen `hoch` / `runter` muss dabei jeweils stets einheitlich sein, entweder zum jeweiligen

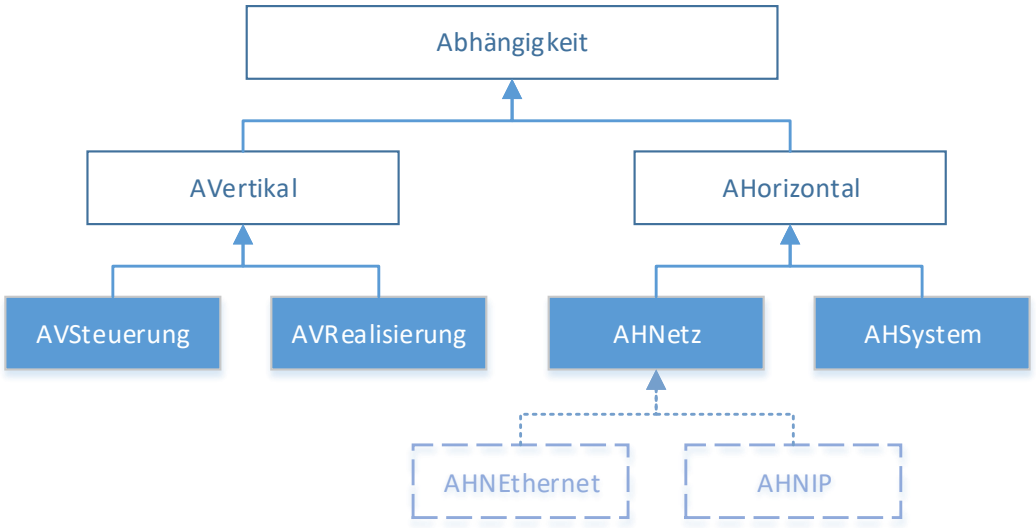
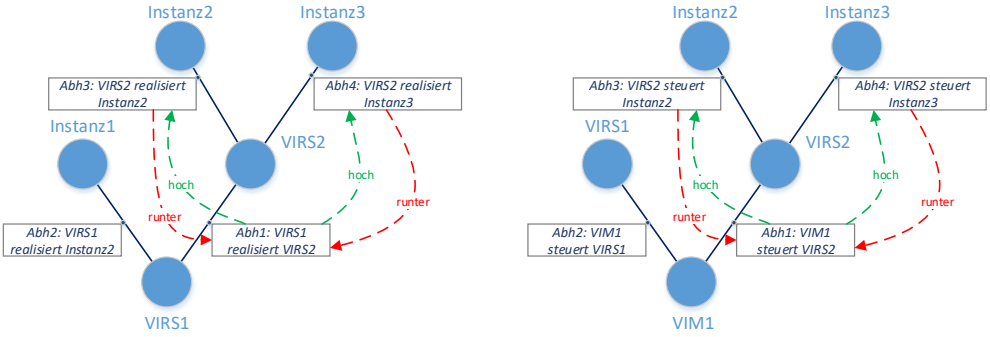


Abbildung 5.8: Übersicht über das vererbungsbasierte Modell für Managementbeziehungen. Die Klassen `Abhängigkeit`, `AVertikal` und `AHorizontal` können nicht direkt instanziiert werden. Die Klassen `AHNEthernet` und `AHNIP` gehören nicht zum Framework, sondern stellen Beispielableitungen dar.

nächsten `Subjekt` oder `Objekt`. Daher soll in vertikalen Abhängigkeiten per **Richtungskonvention** das **Subjekt** das vom Objekt unabhängige MO beschreiben und stets als niedrigere Ebene angesehen werden; Elemente auf im Abhängigkeitskontext *höheren Ebenen* bauen auf darunterliegenden Elementen auf.



(a) Beispiel für AVRealisierung.

(b) Beispiel für AVSteuerung.

Abbildung 5.9: Vertikales Durchlaufen mit Funktionen `hoch` / `runter` in Kind-Klassen von `AVertikal`. Die Kanten repräsentieren die Abhängigkeitsobjekte.

Die Funktionsweise von `hoch` und `runter` wird in Abbildung 5.9 gezeigt. Demnach kann praktisch direkt für Realisierungs- bzw. Steuerungsabhängigkeiten ein beinahe identisches Modell verwendet werden. Beispielsweise wird mit Fokus auf Realisierungsabhängigkeiten in Abbildung 5.9a bei Aufruf der Methode `hoch` in Abhängigkeit *Abh1* die Abhängigkeiten *Abh3* sowie *Abh4* zurückgegeben. Bei Aufruf der Methode `runter` auf Abhängigkeit *Abh3* oder *Abh4* wird in gezeigten Fall Abhängigkeit *Abh1* zurückgegeben. Analoges gilt für Steuerungsabhängigkeiten in Abbildung 5.9b. Die Bindung der Funktionen an die einzelnen Abhängigkeitsklassen selbst (und nicht z. B. an MOCs/MOs) erlaubt die effiziente Generierung eines Abhängigkeitsbaums für einen konkreten Abhängigkeitstypen. Parallele, nicht im Fokus stehende Abhängigkeitspfade (z. B. hier Abhängigkeit *Abh2* in beiden Abbildungen) können so gezielt ignoriert werden.

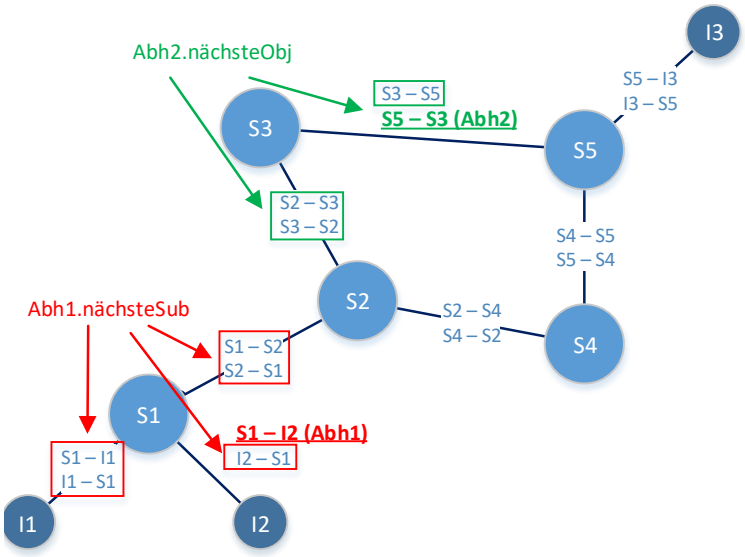


Abbildung 5.10: Illustration der Funktionen `nächsteSub` und `nächsteObj` am Beispiel von Netzabhängigkeiten `AHNetz` zwischen Switches und Instanzen. Die Kanten verbildlichen die Abhängigkeitsobjekte.

5.4.2.2.3 AHorizontal Die Klasse **AHorizontal** bietet im Vergleich aufgrund ihres horizontalen Charakters andere Funktionen. Es müssen weniger one-to-many- als vielmehr **many-to-many**-Abhängigkeiten berücksichtigt werden: Eine Komponente im Netz ist beispielsweise oft mit mehreren anderen verbunden oder ein System besteht aus potenziell mehreren Instanzen. Des Weiteren sind die identifizierten Abhängigkeiten **ungerichtet**: Netz- bzw. Verbindungsabhängigkeiten eines Switches zu anderen sind genauso andersherum gültig. Die Richtung einer Abhängigkeit ist jedoch abhängig von der Interpretation im jeweiligen Anwendungsfall und kann gleichermaßen als gegeben angenommen werden. Funktionen wie in `AVertikal`, die auf einer strikten Richtung basieren, machen jedoch so nicht immer Sinn. Daher stellt `AHorizontal` zwei andere Funktionen zum Durchschreiten eines Netzes bereit. Die Iteratorfunktionen `nächsteSub` sowie `nächsteObj` erlauben die Selektion der jeweils *nächsten* gleichartigen Abhängigkeit in die entsprechend das `Subjekt` bzw. `Objekt` (dann selbst wiederum in einer der beiden Rollen) involviert ist. Darüber hinaus eine Methode `wähle(k, index)`, durch welche eine bestimmte Abhängigkeit für einen Knoten `k` (jew. Subjekt oder Objekt) via Index `index` gezielt gewählt werden kann. Das Beispiel in Abbildung 5.10 verdeutlicht anhand der Modellierung von Abhängigkeiten in einem Netz unter Berücksichtigung von Switches *S1* bis *S5* und Instanzen *I1* bis *I3* die Funktionen `nächsteSub` und `nächsteObj`. Der Aufruf ersterer Funktion für die Abhängigkeit *Abh1* gibt alle gleichartigen (AHNetz-)Abhängigkeiten für das Subjekt (markiert durch rote Boxen), in diesem Fall Switch *S1*, zurück. Der Aufruf der Methode `nächsteObj` für Anhängigkeit *Abh2* gibt alle Abhängigkeiten des Objekts, in diesem Fall Switch *S3* zurück (markiert durch grüne Boxen). Bei horizontalen Abhängigkeiten ist die Richtung oft nicht eindeutig ersichtlich. So hängt kein Switch direkt vom anderen ab, sondern vielmehr das Funktionieren eines Systems (z. B. das Netz) von gleichrangigen Einzelkomponenten (z. B. Netzkomponenten und -Verbindungen). Entsprechend ist eine Richtungskonvention nicht sinnvoll.

5.4.2.3 Erweiterbarkeit

Die Erweiterung von Abhängigkeitsbeziehungen, wie sie im letzten Abschnitt erläutert werden, ist durch Erweiterung der bestehenden Beziehungen vorgesehen. So setzen sich die zum Zweck der Konzeptbildung definierten Abhängigkeitsgruppen aus unterschiedlichen, feingranulareren Abhängigkeiten zusammen, wie in Abschnitt 5.4.2 beschrieben wurde. Speziellere Kontroll-

abhängigkeiten können von der Klasse `AVSteuerung` und speziellere Realisierungsabhängigkeiten von `AVRealisierung` abgeleitet werden. Speziellere Netz- und Systemabhängigkeiten können von den Klassen `AHNetz` respektive `AHSystem` abgeleitet werden. Betrifft die Erweiterung eine Abhängigkeit, die zu keiner der vier in diesem Konzept speziellsten Klassen passt, so ist sie durch Ableitung von `AVertikal` oder `AHorizontal` entweder in vertikale oder horizontale Abhängigkeiten einzuordnen, wodurch sie gleichzeitig die entsprechenden Funktionen und Konventionen erbt. Ist eine neue Abhängigkeit weder durch die Funktionen in `AVertikal` noch `AHorizontal` beschreibbar, so kann sie, abgeleitet von der Wurzelklasse `Abhängigkeit`, eine neue Gruppe von Abhängigkeiten eröffnen, die im Rahmen dieses Konzepts mit Berücksichtigung allgemeiner FSN-spezifischer Charakteristika so nicht abgebildet wird.

5.4.3 Framework für MOC-Funktionen

In den vorherigen Abschnitten dieses Konzepts wurde ein Framework zur Beschreibung von MOCs, ihren Attributen und Managementbeziehungen zwischen MOCs in FSNs definiert. In diesem Abschnitt wird ein dazu abgestimmtes Framework zur Modellierung von **Funktionen von MOs** in MOCs beschrieben. Die Kernkonzepte des Frameworks sind im Rahmen von Vorarbeiten dieser Dissertation entstanden und in [181] veröffentlicht worden. Das darin beschriebene Framework wird in diesem Abschnitt in den Gesamtkontext der anderen Frameworks gesetzt und in einem integrierten Ansatz beschrieben.

Die Implementierung von MOC-Funktionen konkreter Netzfunktionen wie SDN-Controllern oder VIMs ist anwendungsfallabhängig. Eine geeignete Vorlage bieten beispielsweise bereits bestehende Managementplattformen, die nach Paradigma gruppiert in Kapitel 4 untersucht wurden, oder aber Standards zur Beschreibung von Schnittstellen. Im NFV-Bereich betrifft das vor allem die Schnittstelle zwischen VIMs und NFV-Orchestrator (ETSI NFV-IFA 005), VIMs und VNF-Manager (ETSI NFV-IFA 006) sowie dem NFV-Orchestrator und VNF-Manager (ETSI NFV-IFA 007).

5.4.3.1 Organisation von MOC-Funktion

Wie auch in [181] beschrieben wird, weisen MOs in FSNs eine hohe Heterogenität auf, haben jedoch gleichzeitig gemeinsame Charakteristika:

1. FSN-spezifische Netzkomponenten eines bestimmten Typs teilen sich oft Funktionen, haben jedoch unterschiedliche Schnittstellen und Austauschformate. Beispielsweise bieten OpenFlow-basierte SDN-Controller, wie *Floodlight* und *OpenDaylight*, Operationen aus der OpenFlow-Spezifikation, bieten aber unterschiedliche Schnittstellen. Analoges gilt für unterschiedliche Hypervisor. Ein **vererbungsbasierter Ansatz** bietet sich entsprechend zur Modellierung an.

2. Der Satz unterstützter Funktionen einer Netzkomponente in FSNs ist von der Version der Netzkomponente abhängig. Dabei kommen nicht ausschließlich neue Funktionen hinzu, sondern es werden teilweise bestehende Funktionen älterer Versionen derselben Netzkomponente nicht mehr unterstützt. Daher muss nach **Versionen einer Netzkomponente differenziert** werden.

3. Netzkomponenten können Funktionalität aus mehreren Spezifikationen unterstützen. Beispielsweise bietet der im ersten Punkt erwähnte SDN-Controller *OpenDaylight* nicht nur OpenFlow-Funktionen, sondern auch das *Open vSwitch Database*-Protokoll [182] und viele weitere. In [181] wurde dieser Aspekt durch **Mehrfachvererbung** von Elementen gelöst.

Im Rahmen der Einbettung der Modellierung von MOC-Funktionen im Gesamtkontext dieser Arbeit bedarf es geringfügiger Anpassungen hinsichtlich der Beschreibung aus der Publikation.

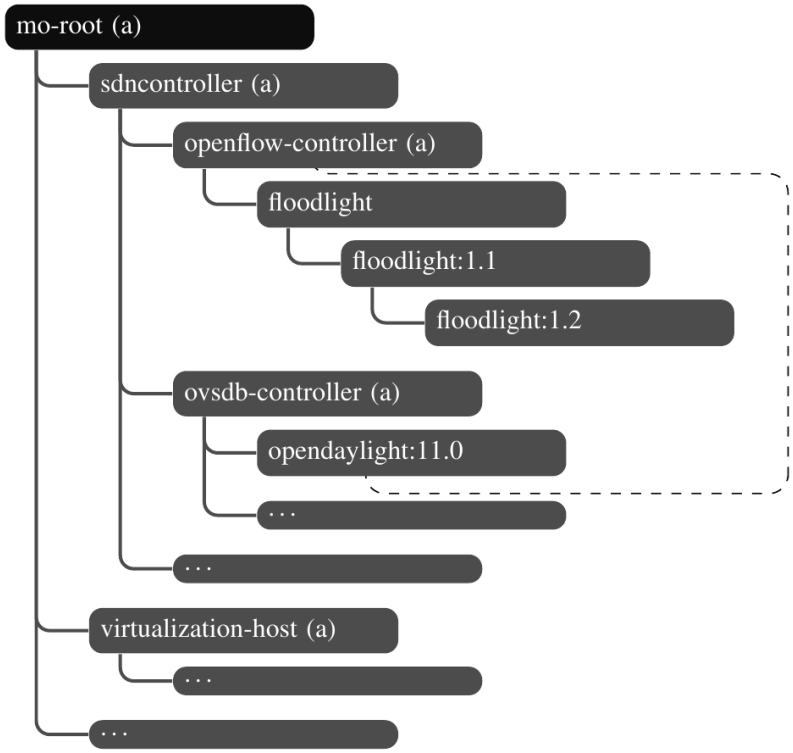


Abbildung 5.11: Darstellung des MOCTypeModels aus [181]. Durchgezogene Linien implizieren primäre Eltern-Kind-Vererbung, Strichlinien sekundäre Vererbungsbeziehungen.

Darin wurde die Struktur der Modellierung durch das **MOCTypeModel** (ein Vererbungsbaum mit Mehrfachvererbung) realisiert. Abgeleitet von einem gemeinsamen Wurzelelement werden Netzkomponenten nach Paradigma und schließlich gemeinsamen Funktionen geordnet. Ein Beispiel des MOCTypeModel ist in Abbildung 5.11 gezeigt. Jedes Element des MOCTypeModels besteht demnach aus folgenden Informationen:

- Ein Elementbezeichner (z. B. generisch „sdncontroller“ oder spezifisch „floodlight“), mit optional daran angehängter Version („z. B. 1.1“).
- Eine Liste von Elternelementen aus dem MOCTypeModel zur Realisierung der Mehrfachvererbung. Diese kann im integrierten Ansatz dieses Konzepts vernachlässigt werden.
- Eine Liste an MOC-Funktionen, auf die in Abschnitt 5.4.3.2 näher eingegangen wird.
- Eine Markierung für abstrakte Klassen. Derartige Klassen dienen nur als Gruppierungsknoten für MOC-Funktionen für davon abgeleitete Elemente. Sie implementieren die Funktionen jedoch nicht. Auch dieser Aspekt wird in Abschnitt 5.4.3.2 erläutert.

Im integrierten Ansatz dieses Konzepts wird die Mehrfachvererbung durch **Klassifikation** auf Basis von `FSNMOCK`-Elementen ersetzt. Mehrfachvererbung wird folglich durch Mehrfachklassifikation ersetzt, sodass eine klare funktionale Trennung der Netzkomponenten gewährleistet wird. **Abstrakte** Knoten werden auf Ebene von `FSNMOCK`-Instanzen unterstützt. Wie in Abbildung 5.12 gezeigt wird, sind Instanzen der Klasse `MOCFunktion` direkt mit dem jeweiligen `FSNMOCK`-Element verknüpft.

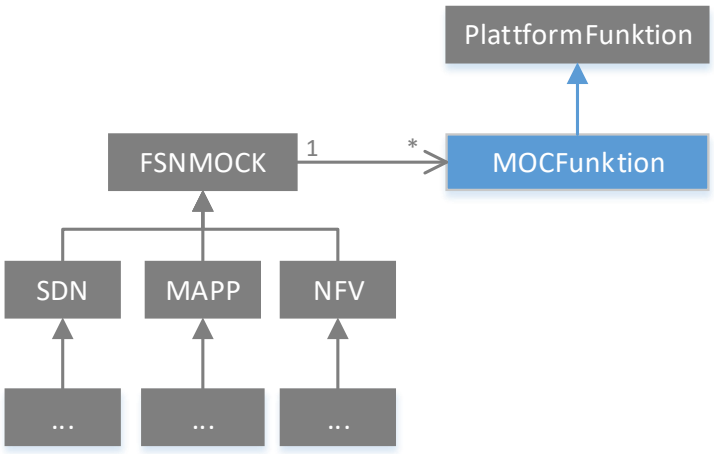


Abbildung 5.12: Klassifizierung von FSNMOCK-Instanzen mit MOCFunktion-Instanzen.

In diesem Konzept wird zwischen **zwei Arten von MOC-Funktionen** unterschieden:

- MOC-Funktionen, die eine tatsächlich von einer MO bereitgestellte Funktionalität beschreiben. Diese werden auf der jeweiligen Netzkomponente ausgeführt. Im Konzept werden diese Funktionen als **reale MOC-Funktionen** bezeichnet. Ein Beispiel ist die Funktion eines Hypervisors zum Starten einer bestimmten VM.

- MOC-Funktionen, die keine eigentliche Funktion auf einem MO ausführen, sondern vielmehr auf das Modell der Managementplattform lesend oder schreibend zugreifen. Durch diese, in diesem Konzept **virtuelle MOC-Funktion** genannten Funktionen, kann auf Attribute einer bestimmten FSNMOC-Instanz und ihrer Klassifikationen zugegriffen werden.

Beide Arten sind für eine bessere Handhabbarkeit zwischen den Teilframeworks von einer gemeinsamen Elternklasse MOCFunktion abgeleitet. Diese stellt wiederum ein Kindelement der Klasse PlattformFunktion aus dem Funktionsmodell dar (vgl. Abschnitt 5.7.2.1).

5.4.3.2 Reale MOC-Funktionen

Reale MOC-Funktionen sind im Gegensatz zu den virtuellen Pendants auch Fokus in [181]. Diese werden genau wie virtuelle MOC-Funktionen von der Klasse MOCFunktion abgeleitet. Von ihrer Elternklasse PlattformFunktion erbt sie die Attribute einer eindeutigen **ID**, durch die auf eine Funktion eines MO zugegriffen werden kann, eine Liste an **Typen für Parameter**, die zum Aufruf der Funktion notwendig sind und eine Liste an **Typen der Rückgabe** (beides über Klasse FunktionsTyp) der Funktion. Eine reale MOC-Funktion wird durch eine **Treiberfunktion** MOCFRTreiber realisiert. Treiber für MOC-Funktionen werden, wie in Abschnitt 5.7.2.1 beschrieben wird, von Treiberklassen von Plattformfunktionen abgeleitet.

Die Klassen MOCFunktion und MOCFunktionReal sind in Listing 5.4 gezeigt. Neue Datentypen für MOC-Funktionen müssen durch die Implementierung der generischen Klasse FunktionsTyp erstellt werden. Die Klasse selbst wird parametrisiert und die Methode checkTyp(MOCFunkParam p) implementiert. Letztere gleicht ab, ob ein Parameter- oder Rückgabewert einer Plattformfunktion (implementiert durch Klasse FunktionsWert) übereinstimmt. Anders als in [181] bekommt durch Tabellen **jeder Parameter** und **jeder Rückgabewert** einer MOC-Funktion einen **Namen**, wodurch im hier integrierten Ansatz die Eindeutigkeit dieser Werte gegeben ist. Eine sinnvolle Benennung der Parameter und Rückgaben erleichtert außerdem deren Dokumentation.

```
1  class MOCFunktion extends PlattformFunktion {
2    //Attribute geerbt von Elternklasse PlattformFunktion
3    //Identifikator  fID;
4    //String  fName;
5    //Tabelle<String, FunktionsTyp> fParameterTypen;
6    //Tabelle<String, FunktionsTyp> fRückgabeTypen;
7  }
8
9
10 class MOCFunktionReal extends MOCFunktion {
11   // [..] Konstruktor, Getter, Setter
12 }
13
14 // Klasse zur Definition von Typen. Eigentlich Teil des Funktionsmodells.
15 abstract class FunktionsTyp<T> {
16   String   fDescription;
17   T        fTyp;
18
19   // Pro Parametertyp sind Typ-Checker und Konditionen
20   // (z.\,B. valide Eingaben) zu prüfen.
21   abstract boolean checkTyp(FunktionsWert p);
22
23   // [..] Konstruktor, Getter, Setter
24 }
25
26 // Container-Klasse für Werte. Eigentlich Teil des Funktionsmodells.
27 abstract class FunktionsWert<T extends FunktionsTyp<V>, V> {
28   V     fWert;
29   T     fTyp;
30 }
```

Listing 5.4: Pseudocode-Implementierung der Klassen `MOCFunktion`, `MOCFunktionReal` und dazugehörigen Klassen `FunktionsTyp`, `FunktionsWert`.

Die beschriebenen MOC-Funktionen haben den Zweck, dass sie, wie auch in [181] erwähnt wurde, der **Normalisierung ähnlicher Funktionen** dienen. Beispielsweise haben die meisten Hypervisor eine Funktion *startVM* zum Starten einer virtuellen Maschine. Diese Funktion verlangt zudem hypervisorübergreifend üblicherweise dieselben Parameter (z. B. die ID der jeweiligen zu startenden VM) und gibt ein ähnliches Ergebnis zurück – z. B. einen booleschen Wert, der angibt, ob die durchgeführte Aktion Erfolg hat. Entsprechend wird die Funktion *startVM* beispielsweise dem von `FSNMOCK` abgeleiteten Element `VIRS` zugewiesen und somit an alle davon abgeleiteten Klassen spezifischerer Virtualisierungssysteme vererbt. Die dadurch stattfindende Vereinheitlichung von Funktionen kann im Management zu einer generellen Automatisierbarkeit beitragen, da Funktionen MO-übergreifend gleich genutzt werden können. In vielen bestehenden Managementplattformen (vgl. bestehende Systeme aus Kapitel 4) fehlt die Abstraktionsebene der MOC-Funktionen, wodurch eine Normalisierung für ähnliche Netzkomponenten in anderen Ansätzen nicht möglich ist.

Die **eigentliche Ausführung** ist stark abhängig vom jeweiligen Netzkomponententyp und seinen spezifischen Schnittstellen. In diesem Beispiel wäre `VIRS` eine abstrakte Klasse, d. h. sie selbst kann die Funktion *startVM* nicht implementieren, sondern nur vererben. Eine MOC-Funktion wird durch einen MOC-Funktionstreiber implementiert. Der **MOC-Funktionstreiber** ist eine abstrakte Klasse, die entweder *a)* **von Eltern-`FSNMOCK` vererbt** (falls vorhanden) wird, oder, im Falle einer Inkompatibilität mit Kindknoten, *b)* für diesen neu implementiert werden muss. Ein Beispiel für Fall *a)* ist, dass eine Netzkomponenten-Implementierung, z. B. ein fiktiver SDN-Controller *sdnc1.1*, in einer neuen Minor-Version *sdnc1.2* eine neue Funktion erhalten hat, alte Funktionen jedoch bestehen bleiben. Die `FSNMOCK`-Klasse `sdnc1.2` kann dann von `sdnc1.1` abgeleitet werden und erbt die weiterhin funktionierenden MOC-Funktionstreiber. Die neue Funktion kann ausschließlich für den `sdnc1.2`-Knoten angefügt und um den entsprechenden neuen Treiber erweitert werden. Fall *b)* ist notwendig, wenn sich über Versionen beispielswei-

se Schnittstelleninformationen ändern. Entsprechend ist ein MOC-Funktionstreiber nicht nur mit einer MOC-Funktion verknüpft, sondern auch mit einer bestimmten FSNMOCK- oder eine davon abgeleitete Klasse, wie in Abbildung 5.13 gezeigt wird. Die Klasse MOCFunktionTreiber steht für Treiber sowohl für reale und für virtuelle MOC-Funktionen. Die zuvor gezeigte Zuweisung von MOC-Funktionen zu einem MOCK bleibt jedoch bestehen, da die Treiberzuweisung nur für nicht-abstrakte MOCK Verwendung findet.

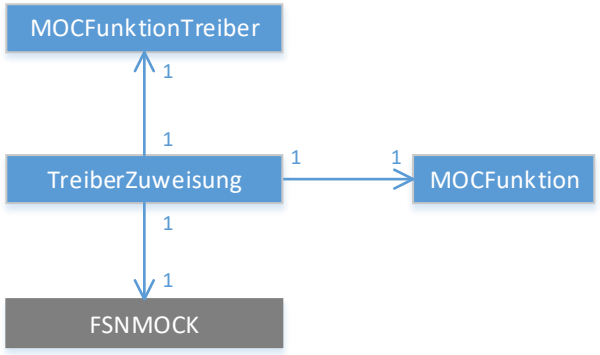


Abbildung 5.13: Zusammenhang zwischen einem MOCK, einer MOC-Funktion und einem MOC-Funktionstreiber. Der MOC-Funktionstreiber implementiert eine MOC-Funktion für ein MOCK und ist für davon abgeleitete MOCKs anwendbar.

```
1  abstract class MOCFRTreiber {
2
3    Set<Normalisierungsmarke>  fNormMarken;
4
5    static <K extends Konnektor> Konnektor holeKonnektor(Class<K> konnektor) {
6      // Methode zum Anfragen einer Konnektorinstanz über das NBI-Register
7      // Rückgabe ist Instanz von Konnektor der Klasse K, die von der Konnektor-
8      // Klasse abgeleitet werden kann.
9    }
10
11   abstract Tabelle<String, FunktionsWert> ausführen(Tabelle<String,
         FunktionsWert> parameter,
12                   FSNMOC mo,
13                   Dienstzugriffspunkt dzp,
14                   Verarbeitungsgrad verarbeitungsgrad,
15                   Boolean bypass,
16                   Boolean priority);
17 }
18
19 // Dienstzugriffspunkt zur Ausführung einer MOC-Funktion
20 // auf einer Netzkomponente
21 abstract class Dienstzugriffspunkt<T> {
22   abstract T getDienstzugriffspunkt();
23 }
24
25 // Beeinflusst den Verarbeitungsgrad der Rückgabe
26 // der aufgerufenen MOC-Funktion
27 enum Verarbeitungsgrad {
28   RAW, PARSED, NORMALIZED;
29 }
```

Listing 5.5: Pseudocode-Implementierung der Klassen MOCFRTreiber, Dienstzugriffspunkt und der Aufzählung Verarbeitungsgrad.

Ein MOC-Funktionstreiber (vgl. Listing 5.5) implementiert für reale MOC-Funktionen die abstrakte Klasse MOCFRTreiber und insbesondere die darin beschriebene Templatefunktion ausführen. Die Funktion erwartet die zuvor beschriebenen benannten Parameter parameter

entsprechend der MOC-Funktionsdefinition, die `FSNMOC`-Instanz `mo`, auf der die Funktion aufgerufen wird und den zu nutzenden `Dienstzugriffspunkt` des MOs. Die Parameter des `Verarbeitungsgrad grad`, `bypass` und `priority` beeinflussen den Rückgabewert und die Art der Behandlung der Rückgabe der MOC-Funktion, wie in Abschnitt 5.6.1.3 zusammen mit der Behandlung am SBI beschrieben wird. Darin wird auch die Verarbeitung der Rückgabe des Funktionsaufrufs beschrieben. Die Variable `fNormMarken` der Klasse `Normalisierungsmarke` dient der Unterstützung der Auswahl eines geeigneten Normalisierers (d. h. die Überführung der Rückgabe in ein `Netzinformation`-Element), wie in Abschnitt 5.6.1.2 beschrieben wird.

Abbildung 5.14 zeigt eine schematische Ausführung einer MOC-Funktion über einen MOC-Funktionstreiber. Schnittstellenklassen zum Aufrufen bzw. der Rückgaben von einem zum nächsten Element werden durch die grünen Boxen illustriert. Eine generische Klasse `T` ruft dabei insbesondere durch Eingabe der benannten Parameter über eine Tabelle, wie zuvor beschrieben, die Methode `ausführen` auf dem jeweiligen MOC-Funktionstreiber auf. Der MOC-Funktionstreiber fragt über die Methode `holeKonnektor` wiederum zunächst vom SBI-Register den passenden Konnektor einer angegebenen Konnektorklasse ab, generiert eine serialisierte Darstellung der übergebenen Parameter und gibt diese Darstellung an den zuvor vom SBI-Register ermittelten passenden Konnektor weiter, welcher die entsprechende Funktion am eigentlichen MO aufruft. Da das SBI-Register alle Schnittstellen verwaltet, wird die Rückgabe (Schritt 5) direkt darüber gehandhabt.

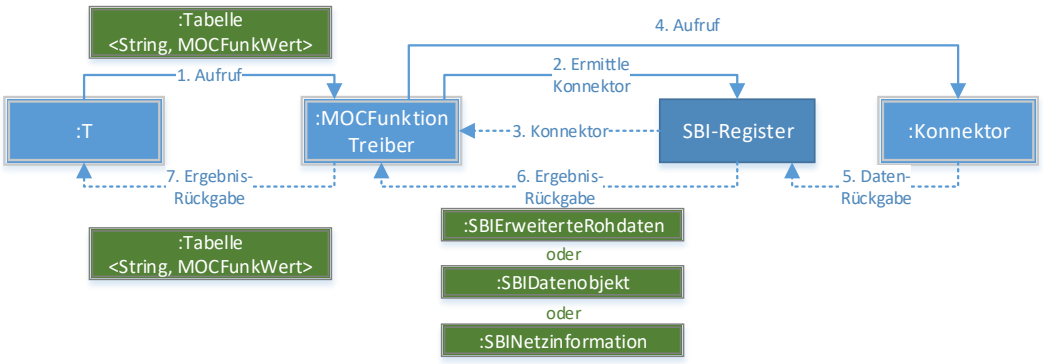


Abbildung 5.14: Darstellung vom Funktionsaufruf am MOC-Funktionstreiber für eine generische Klasse `T` über das SBI-Register zum jeweiligen Konnektor. Die grünen Elemente stellen Schnittstellenklassen zum Aufruf bzw. der Rückgabe der jeweiligen Komponente dar.

Wie in Abschnitt 5.6.1.3 beschrieben wird, kann der MOC-Funktionstreiber den Grad der verarbeiteten Information als Rückgabe wählen: Zwischen `SBIErweiterteRohdaten`, `SBIDatenobjekt` und `SBINetzinformation`, welche im MOC-Funktionstreiber wiederum weiterverarbeitet und gemäß definiertem benannten `FunktionsWert`-Schema zurückgegeben wird.

Die Auswahl des jeweils geeigneten Konnektors wird über das SBI-Register koordiniert. Der MOC-Funktionstreiber fragt eine Instanz einer Konnektorklasse am SBI-Register an, welcher eine verfügbare Instanz dieser Klasse zurückgibt. Beispielsweise würde die Anfrage einer implementierten Klasse `HTTPKonnektor` einen Konnektor für HTTP-Verbindungen zurückgeben, der durch das SBI referenziert wird. Der Mechanismus wird in Abschnitt 5.6.4.2 behandelt.

5.4.3.3 Virtuelle MOC-Funktionen

Virtuelle MOC-Funktionen unterscheiden sich von *realen MOC-Funktionen* insofern, dass sie nicht auf der eigentlichen Netzkomponente ausgeführt werden, sondern entweder lesend oder schreibend auf das in der Managementplattform von der gemanagten Infrastruktur gehaltene Modell zugreifen. Der Vorteil der Einführung virtueller MOC-Funktionen ist, dass der Zugriff

auf das Modell einheitlich wie bei realen MOC-Funktionen über das NBI an Managementplattformen und Nutzer zugänglich gemacht werden kann. Auch kann auf diese Weise die Autorisierung des Zugriffs einheitlich behandelt werden, wie in Abschnitt 5.5.5 näher für alle Elemente des Informationsmodells erläutert wird.

Das Konzept ist analog wie bei realen MOC-Funktionen: Die Funktion wird durch eine entsprechende Klasse `MOCFunktionVirtuell` beschrieben. `FSNMOCK`-Klassen können damit klassifiziert werden, wodurch sich die Funktionen auch an Kind-Klassen vererben. Die eigentliche Funktionalität wird durch einen Treiber implementiert.

Wie in Listing 5.6 gezeigt wird, weicht die Spezifizierung von `MOCFunktionVirtuell` kaum von der einer realen MOC-Funktion ab. Lediglich der Treiber wird durch eine angepasste Klasse `MOCFVTreiber` ersetzt. Der Treiber kann deutlich einfacher implementiert werden, da es weder einen MO-spezifischen Dienstzugriffspunkt gibt, noch die Steuerung der Rückgabe aus dem SBI notwendig ist. Lediglich der Zugriff auf die entsprechende `FSNMOC`-Instanz ist notwendig. Die Zuweisung erfolgt jedoch wie im letzten Abschnitt beschrieben bzw. wie in Abbildung 5.13 gezeigt wird, analog zu realen MOC-Funktionen.

```
1 class MOCFunktionVirtuell extends MOCFunktion{
2   // [..] Konstruktor, Getter, Setter
3 }
4
5 abstract class MOCFVTreiber {
6   abstract Tabelle<String, FunktionsWert> ausführen(
7                 Tabelle<String, FunktionsWert> parameter,
8                 FSNMOC mo);
9 }
```

Listing 5.6: Pseudocode-Implementierung der Klasse `MOCFunktionVirtuell`.

5.4.3.4 Serialisierung von MOC-Funktionen

Die MOC-Funktionen stellen an sich einen serialisierbaren Datentyp dar, die Treiber dagegen nicht. Eine Serialisierung der Funktionalität ist beispielsweise zum Zweck der Synchronisation von Managementplattform-Instanzen notwendig und kann durch Austausch von noch zu kompilierbarem Quellcode oder den Austausch direkt ausführbaren Codes realisiert werden. Letzteres wird beispielsweise in [181] in Form von ausführbarem Lua-Code vorgeschlagen.

5.4.4 Framework für Netzereignisse und -Zustände

Netzereignisse und -Zustände sind insofern von klassischen MOCs zu trennen, als dass sie keine direkten Attribute oder Eigenschaften von MOCs darstellen. Beispielsweise referenziert ein Zustand zur Momentaufnahmenbeschreibung der Topologie des Netzes zu einem bestimmten Zeitpunkt meist auf mehrere `FSNMOC`-Instanzen, zwischen denen logische Verbindungen bestehen. Im Mittelpunkt von Netzereignissen und -Zuständen stehen entsprechend die **Ereignis- und Zustandsinstanzen** selbst, welche im Folgenden unter den Begriff ***Netzinformationen*** zusammengefasst werden. Netzinformationen können entweder von **extern** (über Sensoren wie IDS, Firewalls, usw.) oder aber **intern** durch die Managementplattform oder -Anwendungen (bspw. die Netztopologie) selbst **generiert** werden.

Netzinformationen dienen im Netzmanagement neben der Beschreibung des Netzes auch dem Zweck der **Problemermittlung** mittels **Korrelationsverfahren** in Managementanwendungen. Entsprechende unterstützende Funktionen und Datenstrukturen zum effizienten Zugriff werden auch im Framework berücksichtigt. Einige Aspekte und Konzepte zur Modellierung von Netzinformationen sowie dem Informationsaustausch in interorganisationalen IT-Infrastrukturen wurden in Vorarbeiten zu dieser Dissertation erstellt und in [183] veröffentlicht.

5.4.4.1 Charakteristika von Netzinformationen

Ereignisse genauso wie Zustände sind stark an einen **bestimmten Zeitpunkt** gebunden und können üblicherweise von zeitlich folgenden Ereignissen oder Zuständen *entwertet* werden. Beispielsweise wird ein Ereignis über einen Ausfall einer VM durch ein folgendes Ereignis, das die Verfügbarkeit derselben VM anzeigt, obsolet gemacht. Ein Zustand über die Netztopologie zum Zeitpunkt *A* wird durch einen unmittelbar darauf folgenden gleichartigen Zustand über die Netztopologie obsolet gemacht. Obsolet meint hier, dass entsprechende Netzinformationen nicht mehr den **aktuellen Zustand** des Netzes darstellen (wie es beispielsweise auch in STIX über die `revoked`-Eigenschaft von Elementen berücksichtigt wird, vgl. [82]).

Als **Netzzustände** werden hier Informationen bezeichnet, die tatsächlich statische **Momentaufnahmen** von Teilaspekten des Netzes darstellen. Oft können sie daher aus mehreren Einzelinformationen zusammengestellt werden, die in einer Managementplattform bereits erhoben werden. Auch werden sie meist von Zuständen **desselben Typs** obsolet gemacht. **Netzereignisse** hingegen beschreiben **Ereignisse** im Netz, die vorwiegend durch Ereignisse eines **anderen Typs** obsolet gemacht werden, teilweise jedoch auch gar nicht. Klassischerweise stellen in der Praxis beispielsweise auch von allen Produktivsystemen generierten *Log-Daten* meist Elemente dar, die im Rahmen dieses Konzepts als Netzereignis bezeichnet werden. Aufgrund des zeitlichen Zusammenhangs werden Netzinformationen meist in einer Zeitreihendatenbank gespeichert, wie es beispielsweise im Ceilometer-Projekt von OpenStack (siehe Abschnitt 4.5.2.1) oder in OSM (siehe Abschnitt 4.4.2.1).

5.4.4.2 Attribute

Netzereignisse und Netzzustände unterscheiden sich in Hinblick auf unbedingt notwendige generelle Attribute nicht wesentlich. Eine Übersicht grundlegender Attribute ist in Tabelle 5.2 gezeigt. Alle Netzinformationen stellen einen Zustand zu einem bestimmten Zeitpunkt, wodurch Attribute in Netzinformationen mit Ausnahme weniger Felder grundlegend **schreibgeschützt** sind und nur bei Erstellung eines Objekts festgelegt werden können. Ein flexibles Attributsystem wie für MOCs ist daher nicht notwendig.

Für einen effizienten Zugriff benötigt jede Netzinformation eine föderationsweit eindeutige **ID**. Außerdem sind generell die Komponenten als **Hauptquellen** eine unbedingt notwendige Information, um den Gesamtkontext zu erfassen: Beispielsweise ist die Quellkomponente einer Meldung über eine detektierte Kompromittierung essenziell, um das betroffene Teilnetz zu identifizieren. Die Quelle wird als Referenz auf ein eindeutiges MO gemäß MOC-Modell in Abschnitt 5.4.1 umgesetzt. Teils stammen Netzinformationen auch aus mehreren Quellen, welche angegeben werden können. Neben den Quellen können darüber hinaus **verknüpfte MOs** referenziert werden.

Zeitstempel sollten einerseits möglichst genau dem Auftreten des eigentlichen Ereignisses bzw. dem Zeitpunkt der Erstellung eines Zustands entsprechen. In verwandten Ansätzen wie in Ceilometer wird in Ereignissen beispielsweise nur der Zeitstempel des Quellsystems explizit berücksichtigt [137]. Dieser ist zwar zu bevorzugen, jedoch potenziell nicht immer vorhanden. In diesem Konzept wird daher der Empfangszeitpunkt als alternativer Zeitstempel berücksichtigt. Der Umstand der Generierung des Zeitstempels wird außerdem über das Attribut **Zeitstempelquelle** festgehalten mit vordefinierten Werten *original* für einen originalen Zeitstempel oder *plattform* für einen nachträglich generierten Zeitstempel (d. h. das Ereignis/der Zustand traten *spätestens* bei hinterlegtem Zeitstempel auf).

Die Speicherung des detaillierten **Informationstyps** dient der Klassifizierung und nachträglichen Filterung von Ereignissen/Zuständen. Dieser Detailtyp ist benutzerdefiniert und muss möglichst konkret den entsprechend geeigneten Typ eines Netzereignisses oder -Zustands beschreiben (z. B. *Intrusion-Detection-Ereignis*, *Netztopologie*, usw.). Ein damit verwandtes

Attribut	Beschreibung
ID	Eine eindeutig referenzierbare ID innerhalb der Föderation.
Hauptquellen	Eine Menge von Quell-MOs der erstellten Netzinformation.
Verknüpfte MOs	Eine Menge von MOs, die mit Netzinformation in Verbindung stehen.
Zeitstempel	Ein Zeitstempel des Auftretens oder alternativ der Erstellungszeitpunkt.
Zeitstempelquelle	Quelle des Zeitstempels: Entweder aus dem Originalinhalt (`original`, z. B. einem Sensor im Netz) oder bei Fehlen durch die Managementplattform (`plattform`) generiert.
Informationstyp	Der Detailtyp der jeweiligen Information (entspricht meist Klassentyp).
Aufgabenbereiche	Zur Einordnung der Information in Aufgabenbereiche (z. B. Sicherheitsmanagement, vgl. Abschnitt 5.5.4) zur Ermittlung von Zuständigkeiten.
Kurzbeschreibung	Eine möglichst kurze Beschreibung zum Zweck und dem Informationsgehalt.
Status	Der Status einer Netzinformation: *aktuell*, *veraltet*, *abgelöst*
Nachfolger	Referenziert auf die Netzinformation, durch die diese abgelöst wurde (Status ist dann *abgelöst*).
Persistenz	Ein Indikator für die Managementplattform zum Umgang mit dem Objekt: Entweder *persistent* (Datum ist persistent) bzw. *transient* (Datum wird entfernt, sobald es veraltet ist).

Tabelle 5.2: Notwendige Attribute in Netzereignissen und -Zuständen.

Attribut ist eine Liste von für die Netzinformation relevanter **Aufgabenbereiche**. Durch sie wird die Sichtbarkeit und der Zugriff von Nutzern auf Netzinformationen abgeleitet (vgl. Abschnitt 5.5.5). Die Zuordnung des Aufgabenbereichs ist in dieser Arbeit neu eingeführt und in den in Kapitel 4 betrachteten Ansätzen nicht unterstützt.

Außerdem muss für jede Netzinformation durch eine **Kurzbeschreibung** spezifiziert werden, was sie jeweils konkret darstellt. Schließlich muss die Aktualität der jeweiligen Netzinformation über ein entsprechendes Feld **Status** beschrieben werden. Dieses kann eine Netzinformation als *aktuell*, generell *veraltet* oder spezifisch veraltet und *abgelöst* durch eine neuere konkrete Netzinformation **Nachfolger** abgelöst deklarieren. Im letzteren Fall ist die Referenz auf das ablösende Objekt notwendig. Da Netzinformationen nicht selten in einer sehr großen Zahl auftreten, wird zudem ein Attribut zur Speicherverwaltung eingeführt: Ist eine Netzinformation persistent (**Persistenz** ist wahr) wird sie nicht von der Managementplattform automatisch aus dem Speicher gelöscht, andernfalls kann sie bei Veralten oder nach einer festgelegten Zeit entfernt werden. Fast alle Attribute werden ausschließlich bei Erstellung der Netzinformation belegt und lediglich die **Aufgabenbereiche**, der **Status**, **Nachfolger** und die **Persistenz** können nachträglich geändert werden.

5.4.4.3 Erweiterbarkeit

Wie auch in [183] im Kontext des *Network and Security Objects Kit* (NSOK) genannten Modellierungskonzept spielt auch hier ein **vererbungsbasierter** Ansatz eine wesentliche Rolle. Dabei wird, ausgehend von einer gemeinsamen Wurzelklasse genereller Netzinformation (*Information*), zwischen einer Klasse für ein Netzereignis (`Event`) und einem Netzzustand (`State`) unterschieden. Von diesen beiden Klassen können dann entsprechende Detailklassen abgeleitet werden. Die gleiche Grundstruktur wird in diesem Konzept für Netzinformationen vorgesehen (vgl. Abbildung 5.15). Die Klasse `Netzinformation` definiert die gemeinsamen Attribute aus dem letzten Abschnitt und vererbt sie an die Klassen `Netzereignis` und `Netzzustand`.

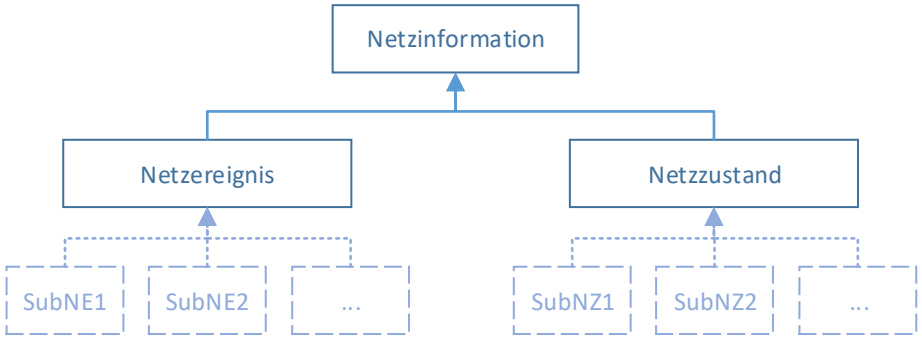


Abbildung 5.15: `Netzereignis` und `Netzzustand` erben Attribute von `Netzinformation` und vererben sie selbst wiederum an benutzerdefinierte Subklassen.

Erstellte benutzerdefinierte Subklassen lassen sich in zweierlei Hinsicht verwenden:

a) Für weitere auf ihnen wiederum basierende **Spezialisierungen**.

b) Als **Attribute** in anderen Netzinformationen.

Bezüglich *a)* können so Oberklassen gebildet werden, auf deren Basis einzelne Spezifizierungen aufgebaut werden: So kann beispielsweise eine von der Klasse `Netzzustand` abgeleitete Klasse `InstanzZähler` mit zusätzlichen Attributen `Zielinstanz` (referenziert auf jew. betrachtete FSNMOC `Instanz`) sowie `Anzahl` (als Element aus der Menge der natürlichen Zahlen) als Basis für viele spezialisierte Netzinformations-Klassen wie `AnzahlAngemeldeterNutzer`, `AnzahlProzesse`, `AnzahlNetzverbindungen` verwendet werden, welche alle einen gleichen Attributsatz haben. Bezüglich *b)* kann nicht nur die Klasse `FSNMOC` und alle von ihr abgeleiteten Klassen genutzt werden, sondern gleichermaßen die Klasse `Netzinformation` mit Unterklassen. Ersteres ist ein offensichtlich notwendiges Element zur Beschreibung von Netzinformationen; Letzteres ist hingegen notwendig, um allgemein **komplexere Netzinformationen** aus Bestehenden zu konstruieren – beispielsweise eine Aggregatorklasse zur Abbildung des Zustands einer Netzkomponente zu einem bestimmten Zeitpunkt.

Zur Modellierung konkreter Netzinformationen können unterschiedliche Standards und bestehende Werke als Vorlage dienen: Für den Bereich NFV beschreibt die ETSI beispielsweise in NFV-IFA 027 (siehe [184]) Größen aus dem Performanzmanagement und in NFV-TST 008 (siehe [185]) Metriken für Rechen-, Speicher- und Netzressourcen. Als Vorlage für Sicherheitsereignisse kann beispielsweise STIX (vgl. Abschnitt 4.2.5.1) genutzt werden.

5.4.5 Framework für Alarme

Alarme stellen gemäß [52] fortbestehende Hinweise auf einen Fehlerzustand dar. Alarme werden in diesem Konzept erst in einer Managementplattform generiert. Vermeintliche Alarme, die von der gemanagten IT-Infrastruktur an die Managementplattform gemeldet werden, stellen in diesem Konzept daher per se keine Alarme in diesem Sinne dar. Diese stellen im Verständnis dieser Arbeit zunächst Netzinformationen dar und müssen erst durch Normalisierungsschritte in Alarme (oder andere Informationselemente) überführt werden.

Alarme beschreiben ähnlich wie Netzinformationen nicht notwendigerweise an MO gebundene Informationen. Ein Kernelement von Alarmen ist eine **Kritikalitätseinstufung**, die, gemeldet an Nutzer und MAPPs, eine Aufgabenpriorisierung erlaubt. Die Kritikalitätseinstufung ist jedoch anwendungsfallabhängig zu modellieren und sollte fein- als auch grobgranulare Stufen erlauben.

5.4.5.1 Umsetzung und Erweiterbarkeit

In Vorarbeiten [183] wurde bereits ein einfaches Frameworkkonzept für Alarme für das föderierte Netz- und Sicherheitsmanagement beschrieben. Der Fokus darin liegt in der flexiblen Modellierbarkeit von Alarmen unter Berücksichtigung der Kritikalitätseinstufung. Beides kann demnach durch einen objektorientierten Ansatz und durch Vererbung unterstützt werden. Das zentrale Element darin ist eine einzige Klasse `AlertType`, die in Abbildung 5.16 das Wurzelelement darstellt.

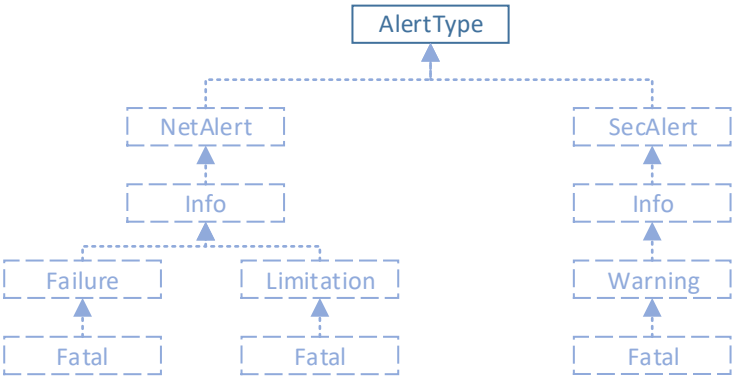


Abbildung 5.16: Alarm-Kernelement `AlertType` mit einem beispielhaft davon abgeleiteten Alarmsystem. Abbildung basierend auf [183].

Wie in [183] beschrieben wurde, kann eine einfache Unterscheidung von Alarmen genauso wie ihre **Kritikalität** durch Vererbung festgelegt werden. Im Sinne des objektorientierten Verständnisses, dass jede Kindklasse eine speziellere Ausprägung einer Elternklasse ist, kann abgebildet werden, dass jeder Alarm höherer Kritikalität ebenfalls Charakteristika aller niedrigeren Kritikalitäten aufweist. In Abbildung 5.16 wird beispielsweise direkt unterhalb der Klasse `AlertType` zwischen solchen betreffend Netzen allgemein (Klasse `NetAlert`) und solchen mit Relevanz für das Securitymanagement (Klasse `SecAlert`) beschrieben. Eine weitere Aufteilung kann ebenfalls durch Ableitung beschrieben werden, wie hier am Beispiel der Elemente `Failure` und `Limitation` gezeigt wird. Über abstrakte Klassen kann durch den Entwickler gesteuert werden, welche Elemente zur Gruppierung dienen und welche konkret instanziierbar sind. Beim Vererbungspfad unterhalb des `SecAlert`-Elements ist die Umsetzung von drei Kritikalitätsstufen *Info*, *Warning* und *Fatal* illustriert. Kritischere Elemente werden stets von weniger kritischen abgeleitet.

5.4.5.2 Attribute und Methoden von Alarmen

Auf Attribute wird in [183] nicht näher eingegangen. Andere vergleichbare Formate wie das Java-Interface `Alarm` im ONOS-Projekt werden u. a. durch eine ID, eine Referenz auf ein betroffenes MO, eine Quelle, verschiedene Zeitstempel und eine Kritikalität beschrieben [186]. In OpenStacks Monasca-Dienst werden Alarme nach [138] ohne benutzerdefinierte Angabe von Zeitstempeln, jedoch mit u. a. ihrer Kritikalität, einer Mandanten-ID und verschiedenen *Actions* beschrieben. Actions beschreiben insbesondere Benachrichtigungsmethoden [150]. Einen vergleichbaren Ansatz mit Actions nutzt auch OpenStacks Aodh (vgl. [141]) und OSM (vgl. [112]). Die Alarmbeschreibung der genannten Ansätze umfassen darüber hinaus Zeitstempel und Bearbeitungszustände. In Aodh wird zusätzlich eine Projekt-ID und ein Feld für die Begründung eines Alarmzustands angegeben [141].

Ein Teil dieser Attribute wird auch in diesem Konzept übernommen. So muss jeder Alarm eine föderationsweit eindeutige **ID**, eine **Beschreibung** und einen **Zeitstempel** haben. Da Alarme in diesem Konzept stets in der Managementplattform generiert werden, entspricht der Zeitstempel dem Zeitpunkt des Generierens. Die **Kritikalität** wird, wie im vorherigen Abschnitt

beschrieben wurde, implizit über die Vererbungshierarchie beschrieben. Eine weitere Dimension der Klassifikation von Alarmen wird optional über eine Menge von benutzerdefinierten **Alarmkategorien** (z. B. nach Inhalt wie *Malware* oder *Systemausfall*) vorgesehen. Sie soll eine Klassifikation auf Basis des Vererbungsbaumes (vgl. vorheriger Abschnitt) ergänzen. Die Zuweisung einer Mandanten-ID oder Projekt-ID, wie es die OpenStack-Dienste Monasca und Aodh umgesetzt haben, ist in diesem Konzept nicht feingranular genug. Verantwortlichkeiten im Management werden durch die Verknüpfung von Aufgabenbereichen mit verschiedenen Alarmkategorien bestimmt, wie in Abschnitt 5.5.4.1 beschrieben wird. Alarme müssen als wesentliches Informationselement zur Benachrichtigung von Nutzern eine **Serialisierungs-** und für den Austausch zwischen Managementplattformen untereinander eine **Deserialisierungsfunktionalität** anbieten.

5.4.6 Unterscheidung von Ist- und Soll-Zustand

Nicht nur die offenbar notwendige Beschreibung des aktuellen Stands der gemanagten Umgebung, sondern ebenfalls die Beschreibung eines angestrebten Zustands ist im Netzmanagement teilweise notwendig. Durch Letzteres wird beispielsweise Netzmanagement auf Basis von modellgetriebenen Richtlinien umgesetzt, wie es beispielsweise in XOS (vgl. Abschnitt 4.5.1) verfolgt wurde. Die Umsetzung kann in diesem Konzept einfach über das Anlegen mehrerer MO-Repräsentationen realisiert werden: So kann eine `FSNMOC`-Instanz den Ist-Zustand darstellen und eine weitere `FSNMOC`-Instanz den Soll-Zustand. Ersteres unterstützt entsprechend reale MOC-Funktionen, die Funktionen des tatsächlichen MOs implementieren. Letzteres würde in diesem Fall ausschließlich virtuelle MOC-Funktionen unterstützen, die den Soll-Zustand auslesen und modifizieren können. Der Ansatz ist insofern angelehnt an den in OpenDaylight verfolgten Ansatz (vgl. Abschnitt 4.3.1.1) mit unterschiedlichen Datenspeichern. Die in den vorherigen Abschnitten beschrieben Frameworks können dafür genutzt werden, wodurch keine dedizierte Frameworklösungen zur Beschreibung des Soll-Zustands notwendig ist.

5.5 Organisationsmodell

Die Beschreibung eines Organisationsmodells für FSNs lässt sich (auch gemäß Hot-Spot-Analyse in Abschnitt 3.6.2.4) durch vier unterschiedliche, aber zusammenhängende Frameworks unterstützen:

- Das allgemeine **Föderationsmodell** unterstützt die Beschreibung grundlegender Partnerstrukturen und -Beziehungen.
- Das Framework für **administrative Domänen** unterstützt die Kapselung zusammenhängender IT-Ressourcen und Nutzern.
- Ein Framework für **Nutzer** und **Nutzerrollen**.
- Ein Framework zur Beschreibung von **Aufgaben** und damit verbundenen **Zuständigkeiten**.

Darüber hinaus benötigt es ein Framework zur Regelung von **Sichtbarkeit und Zugriff** auf Informationselemente in Abhängigkeit mit den anderen Strukturen des Organisationsmodells. Dabei liegt ein Schwerpunkt in der Unterstützung **unterschiedlicher Geltungsbereiche**, da die Organisation von FSNs gleichzeitig zentral (einheitlich innerhalb Föderation) sowie dezentral (ergänzend innerhalb einzelner administrativer Domänen) erfolgen kann.

5.5.1 Framework für Föderationsmodelle und -Beziehungen

Das Framework für Föderationsmodelle dient zur Beschreibung der Elemente einer Föderation, ihrer Parteien und dazwischen bestehenden Beziehungen sowie grundlegender Strukturierungselemente von Ressourcen. Eine Übersicht ist in Abbildung 5.17 gezeigt.

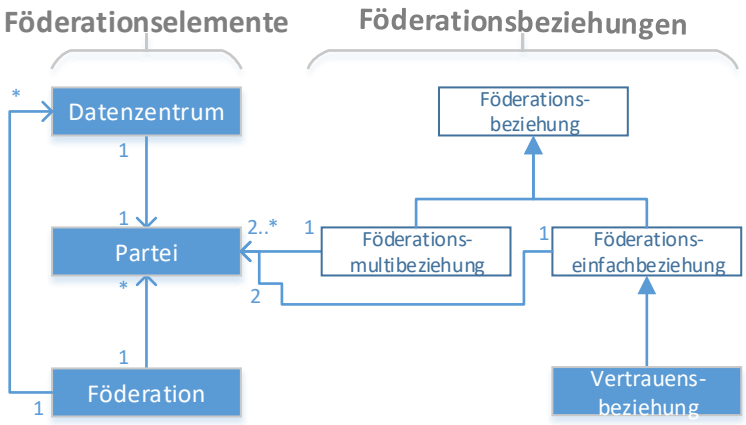


Abbildung 5.17: Übersicht über Elemente zur Beschreibung von Föderationen und darin vorkommenden Beziehungen.

Ähnliche Ansätze zur Beschreibung von Föderationen wurden beispielsweise in der *Federated Infrastructure Discovery and Description Language* (FIDDLE) [187][188] zur Beschreibung föderierter Testbeds vorgenommen. FIDDLE erlaubt die Modellierung einer Föderation mit einzelnen Föderationsmitgliedern sowie die generische IT-Infrastruktur. Die Modelle berücksichtigen jedoch nicht alle im Netzmanagement notwendigen Zusammenhänge in FSNs und beispielsweise auch keine Beziehungen zwischen Föderationspartnern. Auch ist die Granularität der Organisation zu grob und muss für das Netzmanagement feiner untergliedert werden (beispielsweise durch administrative Domänen).

5.5.1.1 Elemente zur Föderationsbeschreibung

Elemente zur Beschreibung einer Föderation umfassen generell die jeweiligen Föderationspartner, die in diesem Konzept als **Partei** bezeichnet werden, sowie **Datenzentren**. Ein eigenes Element zur Beschreibung der **Föderation** ist an sich nicht notwendig im Sinne einer Zusammenfassung mehrerer Parteien als Föderation – diese Beziehung kann auch implizit angenommen werden. Es dient aber als Frameworkelement zur Kapselung aller relevanten Strukturen zur Beschreibung einer Föderation als Gesamtes. Außerdem wird so eine mögliche Erweiterbarkeit des Konzepts auf eine Multi-Föderations-Umgebung erleichtert. Diese wird allerdings in dieser Arbeit nicht explizit betrachtet.

Das Element `Föderation` ist im Rahmen dieser Arbeit daher als einzelnes, zentrales Objekt im Modell instanziierbar. Es wird zunächst durch eine eindeutige **ID** beschrieben. Es verwaltet alle der Föderation zugehörigen `Partei`- sowie `Datenzentrum`-Instanzen. Da weitere Charakteristika von Föderationen in FSNs, wie sie in Abschnitt 2.2.2 ermittelt wurden, für das Management unter Umständen von Bedeutung sind, können diese ebenfalls zur Beschreibung dieses Elements genutzt werden. Insbesondere organisatorische Charakteristika wie die **Dauer** (mit zeitlichem **Anfang**) der Föderation werden hier festgehalten. Bei Bedarf sollten anwendungsfallabhängig weitere Elemente jedoch ebenfalls zur Beschreibung genutzt werden. Ein für die Organisation von **Domänenüberschneidungen** notwendigerweise verknüpftes Attribut ist die **Domänenkreuzorganisationstabelle**, die in Abschnitt 5.5.2.3 beschrieben wird. Das Element `Föderation` dient noch einem weiteren Zweck, wie in Abschnitt 5.5.4.2 beschrieben

ist: Als Punkt zur Festlegung von föderations- bzw. netzweiten Vorgaben, die für alle Domänen gelten. Dies wird durch einen Verweis auf verfügbare **Funktionsbereiche** (vgl. Abschnitt 5.7.1) und **Aufgabenbereiche** sowie darin definierte **Zuständigkeit**-Instanzen ermöglicht. Auch Föderationsbeziehungen (vgl. folgender Abschnitt), sowie Beziehungen zwischen Domänen (vgl. Abschnitt 5.5.2.2) werden an dieser Stelle zentral verwaltet.

Das Element `Partei` beschreibt allgemein einen Föderationspartner wie eine Organisation oder auch eine Einzelperson. Innerhalb der Föderation muss jede Partei mindestens durch eine eindeutige **ID** sowie eine **Kurzbeschreibung** repräsentiert werden. Falls anwendungsfallabhängig notwendig, können durch Ableitung speziellere Parteitypen unterschieden werden (z. B. Anbieter, Kunde, Organisation, Einzelperson, usw.). Innerhalb einer Föderation gibt es mindestens zwei Parteien, wobei in der Theorie keine Maximalbeschränkung existiert. Jeder Partei ist ein Satz an jeweils dazugehörigen **Nutzerinstanzen** (entsprechend der Nutzerbeschreibung basierend auf dem Nutzer-Framework in Abschnitt 5.5.3) zugewiesen.

Das Element `Datenzentrum` beschreibt einen durch eine Partei verwalteten Ort, der in einer Föderation IT-Ressourcen zur Verfügung stellt. Jedes Datenzentrum muss eine innerhalb der Föderation eindeutige **ID** und einen **Namen** haben sowie eine Referenz auf genau eine **Partei**, die das Datenzentrum verwaltet. Auf dieser Basis und in Kombination mit Realisierungsabhängigkeiten (vgl. Abschnitt 5.4.2.2) können **Besitzabhängigkeiten** von MOs bestimmt werden. Darin gestellte IT-Ressourcen werden jedoch nicht notwendigerweise immer von dieser Partei verwaltet, sondern sind im Kontext von Domänen organisiert. Durch den Verweis auf eine Partei an dieser Stelle ist sichergestellt, dass ein Datenzentrum nicht mehreren Parteien zugewiesen ist. Ein Datenzentrum verweist außerdem auf den durch darin **unmittelbar** der Föderation bereitgestellten Satz an **IT-Ressourcen** (Referenz auf Instanzen jeweiliger Unterklassen von `FSNMOC`). Des Weiteren muss für jedes Datenzentrum eine Angabe zum **Standort** gegeben sein, da diese nicht selten in Managemententscheidungen miteinbezogen werden (z. B. hinsichtlich Ort von Datenhaltung oder auch hinsichtlich rechtlicher Aspekte). Der notwendige Detailgrad des Standorts variiert dabei anwendungsfallabhängig relativ stark: Teilweise ist eine Unterscheidung von Ländern ausreichend und gewollt, in anderen Fällen muss ein Raum innerhalb eines Gebäudes unterschieden werden (z. B. in unterschiedlichen Brandabschnitten). Aufgrund dessen werden Implementierungsdetails des Standorts in diesem Konzept nicht näher spezifiziert.

5.5.1.2 Föderationsbeziehungen und Vertrauen

Als **Föderationsbeziehungen** werden in diesem Konzept Beziehungen zwischen Parteien innerhalb einer Föderation bezeichnet. `Vertrauensbeziehungen` werden als Ausprägung davon explizit betrachtet. Weitere anwendungsfallspezifische Beziehungen (z. B. die Dokumentation einer Anbieter-Kunden-Beziehung zwischen Parteien) sind denkbar und stellen teilweise eine notwendige Managementinformation dar. Eine strukturierte Erweiterung soll durch dieses Framework unterstützt werden. Föderationsbeziehungen können beispielsweise auch als SLAs interpretiert werden, wie es im Kontext von föderiertem Cloud-Computing zwischen Dienstanbietern untereinander, aber auch zwischen Dienstanbietern und Kunden bereits untersucht wurde (vgl. z. B. [189]). In diesem Kontext ist gerade auch die Modellierung von Vertrauensbeziehungen ein aktives Forschungsthema. So wird beispielsweise in [190] ein Vertrauenswert zwischen Parteien im Cloud-Computing auf Basis von SLA-Vorgaben sowie Feedback der Parteien berechnet. Derartige Ansätze können beispielsweise genutzt werden, um Vertrauensaspekte für alle Parteien in FSNs automatisch zu generieren.

Wie in Abbildung 5.17 gezeigt wird, erfolgt generell eine Unterscheidung zwischen zwei Arten von Föderationsbeziehungen, welche von einer gemeinsamen Oberklasse `Föderationsbeziehung` abgeleitet sind: Zum einen eine *einfache* Föderationsbeziehung (`Föderationseinfachbeziehung`) die stets einen Zusammenhang zwischen genau zwei Parteien herstellt, zum anderen eine `Föderationsmultibeziehung`, die mehrere Parteien miteinander in Ver-

bindung bringt. Föderationsbeziehungen, die durch keine dieser zwei Arten abgebildet werden können, können entsprechend von der Klasse `Föderationsbeziehung` abgeleitet werden.

Die Klasse **`Föderationsbeziehung`** vererbt als essenzielles Attribut eine innerhalb der Föderation eindeutige **ID** an alle davon abgeleiteten Elemente. Ein Feld für eine **Kurzbeschreibung** dient zudem zur Erläuterung der jeweiligen Beziehung.

Die Klasse **`Föderationseinfachbeziehung`** beschreibt zusätzlich die Attribute **Subjekt** und **Objekt** (welche jeweils auf `Partei`-Instanzen referenzieren). Diese geben auch eine Richtung der Föderationsbeziehung vor; analog wie in dem in Abschnitt 5.4.2 beschriebenen Framework für Managementbeziehungen.

Die Klasse **`Föderationsmultibeziehung`** hingegen stellt eine ungerichtete Beziehung zwischen zwei oder gar mehreren Parteien dar. Sie wird durch eine Menge von **Parteien** beschrieben. So können einerseits ungerichtete Beziehungen beschrieben werden oder beispielsweise auch Parteien gruppiert werden (z. B. als IT-Ressourcen-Anbieter und komplementäre IT-Ressourcen-Nutzer).

Die Klasse **`Vertrauensbeziehung`** ist eine Ableitung einer `Föderationseinfachbeziehung`, da die Einschätzung des Vertrauens zwischen zwei Parteien zunächst meist uneinvernehmlich und teilweise nicht übereinstimmend ist. Sie wird vielmehr von Parteien unabhängig voneinander getroffen. Die jeweilige Abstufung des Vertrauensniveaus von Föderationspartnern untereinander ist anwendungsfallspezifisch zu definieren und durch das Framework in besonders diesem Fall zu unterstützen. Hier bietet sich ein Hierarchiesystem mit mehreren Abstufungen des Vertrauens an.

Auswirkungen von Föderationsbeziehungen spielen vor allem in der Bestimmung der Sichtbarkeit und des Zugriffs auf Informationselemente eine wichtige Rolle, wie in Abschnitt 5.5.5 beschrieben wird.

5.5.2 Framework für administrative Domänen

Eine administrative Domäne bildet im Management von FSNs eine feingranularere Struktur zur Beschreibung eines Kontexts innerhalb der Föderation. Administrative Domänen setzen Elemente aus anderen Frameworks, vor allem MOs, Nutzer, Rollen und Aufgaben, in einen gemeinsamen Kontext.

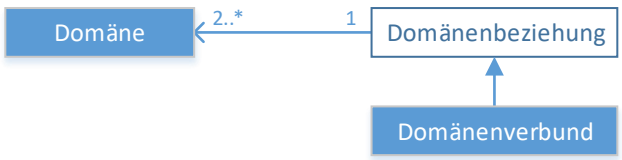


Abbildung 5.18: Elemente zur Beschreibung einer administrativen Domäne (`Domäne`) sowie von Beziehungen zwischen Domänen (`Domänenbeziehung`). Ein Verbund von administrativen Domänen (`Domänenverbund`) wird als spezielle `Domänenbeziehung` davon abgeleitet.

Im Vergleich zur vielfältigen Ausprägung von beispielsweise MOCs oder Managementbeziehungen in FSNs ist die von Domänen selbst eher uniform. Mögliche Erweiterungen sind daher über das Hinzufügen weiterer anwendungsfallspezifischer Attribute als über eine Ableitung spezialisierter Domänen vorgesehen. Anders verhält es sich jedoch bei der Berücksichtigung von Beziehungen zwischen Domänen. Hier sind vielerlei hilfreiche Erweiterungen wie beispielsweise die Beschreibung eines (temporären) Domänenverbunds denkbar. Wie in Abbildung 5.18 gezeigt ist, wird neben einem Element `Domäne` zur Beschreibung einer Domäne ein separates Ele-

ment Domänenbeziehung zur Beschreibung von Beziehungen zwischen Domänen vorgesehen. Ein Domänenverbund Domänenverbund wird konkret von Domänenbeziehung abgeleitet.

5.5.2.1 Administrative Domänen

Eine **administrative Domäne** wird über das Element Domäne (vgl. Abbildung 5.18) modelliert. Um eindeutig identifizierbar zu sein, wird jedes Element Domäne über eine eindeutige **ID** sowie eine **Kurzbeschreibung** beschrieben. Darüber hinaus hält es eine Menge an Referenzen auf Instanzen von FSNMOC, sowie eine Menge an Referenzen auf **Entität**-Instanzen (d. h. Nutzer oder Managementjobs), die durch die jeweilige Domäne umfasst werden. Optional kann der Geltungsbereich für eine bestimmte Domäne durch Referenzen auf Zuständigkeit-Instanzen (die in Abschnitt 5.5.4.2 erklärt werden) festgelegt werden.

Die Erweiterung des Elements Domäne erfolgt in erster Linie attributbasiert, da alle in dieser Arbeit im Rahmen der Szenarien betrachteten administrative Domänen generell von ihrem Einsatzzweck homogen sind. Gleichzeitig wird eine Ableitung unterschiedlicher Domänentypen nicht ausgeschlossen.

5.5.2.2 Domänenbeziehungen

Da eine Domänenbeziehung mit einer konkreten Bedeutung verknüpft ist, ist die entsprechende Klasse Domänenbeziehung abstrakt und kann nicht direkt instanziiert werden. Wie in Abbildung 5.18 gezeigt wird, kann eine beispielhafte Ausprägung eines Domänenverbunds über die Klasse Domänenverbund davon abgeleitet und als nicht-abstraktes Element instanziiert werden. Wie in Abschnitt 5.5.5 erläutert wird, kann die Sichtbarkeit und der Zugriff auf Informationselemente auch unter Einfluss von Domänenbeziehungen flexibel festgelegt werden. Die im nächsten Abschnitt beschriebene Behandlung von Domänenüberschneidungen ist hingegen unabhängig von Domänenbeziehungen.

Wie alle anderen Elemente müssen auch Domänenbeziehungen innerhalb einer Föderation eindeutig adressierbar sein und werden daher durch eine föderationsweit eindeutige **ID** beschrieben. Ebenso erhalten sie eine **Beschreibung**, durch die beispielsweise der Zweck der Domänenbeziehung geschildert werden kann. Das Kernelement einer Domänenbeziehung bildet jedoch eine Menge von mindestens zwei **Domänen**, die über die Beziehung in Verbindung gebracht werden.

5.5.2.3 Domänenbildung und Behandlung von Überschneidungen

Neben der Modellierung von Domänen muss durch eine Managementplattform auch ein **Regelsatz zur Anwendung** von administrativen Domänen forciert werden, damit eine im Management notwendige klare Strukturierung von Verantwortlichkeiten unterstützt wird. Dazu zählt, dass *a)* jedes MO **einen Managementkontext besitzt**, d. h. mindestens einer Domäne zugeordnet ist, genauso wie, dass *b)* Verantwortlichkeiten auch bei sich **überschneidenden administrativen Domänen** klar geregelt sind. Überschneidungen treten auf, wenn MOs mehr als einer administrativen Domäne zugeordnet sind. Das in diesem Abschnitt vorgestellte Konzept behandelt derartige Überschneidungen. Überschneidungen durch Domänenbeziehungen werden in Abschnitt 5.5.5 im Gesamtkontext zu anderen Kriterien behandelt.

Die Behandlung von Aspekt *a)* erfolgt in diesem Konzept durch Prüfung der Zugehörigkeit von in Instanzen der Klasse Datenzentrum (vgl. Abschnitt 5.5.1.1) referenzierten Instanzen der Klasse FSNMOC (d. h. direkt in einem Datenzentrum bereitgestellte Ebene-0-IT-Ressource) zu mindestens einer administrativen Domäne. Des Weiteren dadurch, dass jede neue Instanz zunächst der administrativen Domäne ihres jeweiligen Hosts zugewiesen wird. Diese Beziehung kann einfach auf Basis der Modellierung der Abhängigkeit zwischen Host- und Gast-Systemen (vgl. Klasse AVRealisierung aus Abschnitt 5.4.2.2) abgeleitet werden. Das Gleiche gilt auch

beispielsweise beim Aufsetzen einer **Anwendung** (beschrieben durch das gleichnamige von `FSNMOC` abgeleitete Element) auf einem `SystemInstanz`-Element. Eine manuelle Änderung der Domänenzuweisung ist nachträglich möglich.

Die Behandlung von *b)* Überschneidungen in administrativen Domänen wird durch Aufteilung der Aufgaben und Zuständigkeiten auf die sich überschneidenden Domänen gelöst. So sollen redundante, unklare Zuweisungen verhindert werden und vielmehr ein Ansatz mit Hauptverantwortlichen und optional eingeteilten Vertretern aus den Domänen verfolgt werden.

Für die Behandlung von Überschneidungen wird dazu in diesem Framework die Datenstruktur **Domänenkreuzorganisationstabelle** (DKOT) eingeführt, die für ein jeweiliges MO eine wiederverwendbare Aufgabenverteilung über mehrere Domänen ermöglicht. Sie wird durch eine Tabelle dargestellt, in der jedes MO, das in mehreren Domänen liegt, jeweils als Schlüssel fungiert und auf einen entsprechenden **Domänenkreuzorganisationseintrag** (DKOE) referenziert. Ein DKOE beschreibt sich **überschneidende Domänen** sowie eine neue explizite **Aufgabenzuweisung** für sich explizit **überschneidende MOs** darin. Durch eine Zuweisung mehrerer Rollen zu einer Aufgabe ist die Unterscheidung zwischen einer hauptverantwortlichen Rolle und Vertreterrollen möglich. Zur besseren Verwaltung wird ein DKOE zudem über eine **Kurzbeschreibung** sowie eine eindeutige **ID** beschrieben. Bei einer Domänenüberschneidung werden etwaige zuvor über das `Zuständigkeit`-Element (vgl. Abschnitt 5.5.4.2) zugewiesene Zuständigkeiten für die Dauer der Überschneidung überschrieben.

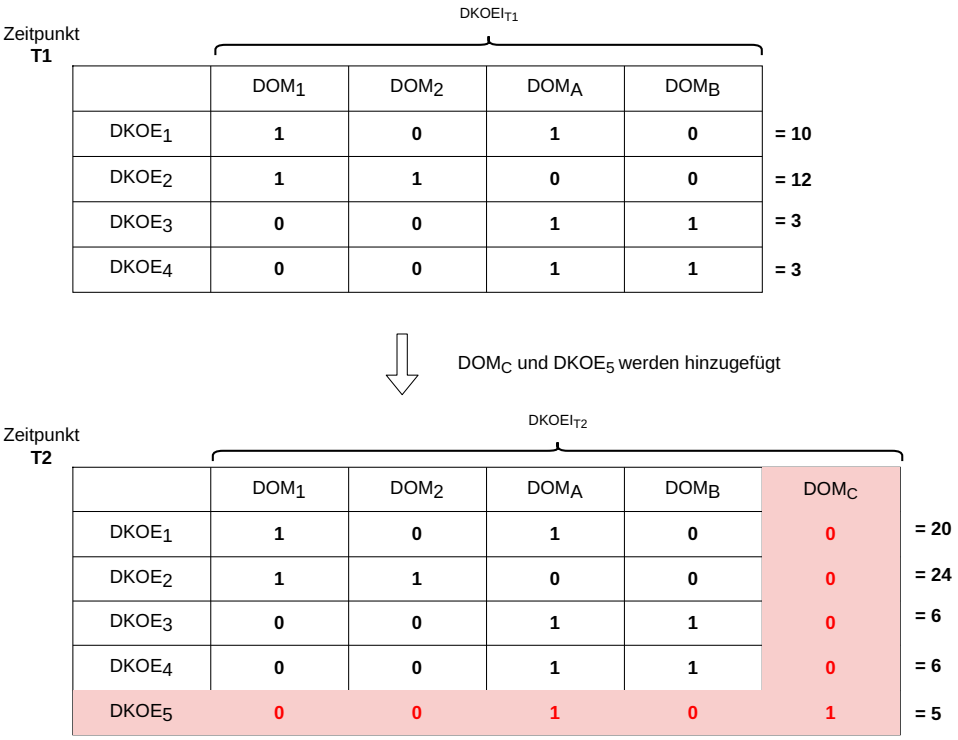


Abbildung 5.19: Funktionsprinzip der Domänenkreuzorganisationstabelle zur effizienten Verwaltung von Domänenüberschneidungen. Für jedes DKOE wird ein Index-Wert DKOEI berechnet, der eine effiziente Suche erlaubt.

Durch die Trennung von DKOT und einzelnen Überschneidungen als DKOE wird die Wiederverwendbarkeit bereits definierter Strukturen ermöglicht. Dafür muss die DKOT auch als Verzeichnis für verwendete DKOE-Elemente mit passenden Suchfunktionen fungieren, damit bereits abgebildete Überschneidungen nicht redundant angelegt werden. Die **Suchfunktion** muss entsprechend mindestens gemäß einer Auswahl übergebener Domänen variabler Anzahl dazu ex-

akt passende DKOEs effizient zurückliefern. Die DKOT überführt dafür eine Menge an Domänen in eine fixe, erweiterbare Ordnung und verweist für jeden DKOE auf den zutreffenden Eintrag. Eine einfache Erweiterbarkeit ergibt sich durch eine Ordnung bezüglich des Zeitpunktes des Hinzufügens oder der Definition einer Domäne. Darin ergibt sich eine chronologische Ordnung der administrativen Domänen gemäß ihrer Erstellung. Der Verweis eines DKOE kann dann speichereffizient über einen Index-Wert **DKOEI** dargestellt werden, welcher sich in konstanter Zeit mit anderen DKOEI-Werten vergleichen lässt. Nachträgliches Hinzufügen kann durch Padding mit weiteren nachfolgenden Nullen ausgeglichen werden (hier lediglich durch einen bitweisen Shift-Left-Befehl für zuvor bestehende DKOEI-Werte). Alternativ ist auch eine Neuberechnung des DKOEI für alle DKOEs vertretbar, da sie vergleichsweise unaufwendig ist.

```
1  // Beschreibt eine Domänenüberschneidung
2  class DKOE {
3    String ID;
4    String Kurzbeschreibung;
5
6    // Menge sich überschneidender Domänen
7    Set<Domäne> Domänen;
8    // Zuweisung von Aufgaben zu Rollen sich überschneidender Domänen
9    Tabelle<Aufgabe, Set<Rolle>> Aufgabenverteilung;
10 }
11
12 class DKOT {
13   // Eigentliche Zuweisung von der Überschneidung betroffenen MOs zu
14   // Domänen-Kreuzorganisationeintrag-Elementen
15   Tabelle<FSNMOC, DKOE> KDTabelle;
16
17   // DKOE Register
18   DKOERegister DKOEReg;
19 }
```

Listing 5.7: Pseudocode-Beschreibung der Domänenkreuzorganisationstabelle `DKOT`, in der `FSNMOC`-Instanzen auf Domänen-Kreuzorganisationseinträge `DKOE` referenzieren.

Ein Beispiel der DKOT ist in Abbildung 5.19 gezeigt. Die in der obersten Zeile gezeigten Domänen spiegeln die chronologische Ordnung (links ist das älteste, rechts das neueste) gemäß ihrer Berücksichtigung wider. Die Zeilen (ab inkl. Zeile 2) repräsentieren die DKOEIs. Bei einer Modifikation der DKOT durch Hinzufügen des $\texttt{DKOE}_5$ mit den Domänen $\texttt{DOM}_\texttt{A}$ und $\texttt{DOM}_\texttt{C}$ wird letztere Domäne neu eingefügt und die neue DKOEI gebildet. Die DKOEIs der anderen DKOEs werden mit einer entsprechenden nachfolgenden Null ergänzt. Eine beispielhafte **effiziente Suchanfrage** ab Zeitpunkt *T2* zu DKOE-Elementen, die eine Aufgabenverteilung über die Domänen $\texttt{DOM}_\texttt{A}$ und $\texttt{DOM}_\texttt{B}$ vornehmen, hätte entsprechend dem Beispiel in der gezeigten Illustration den DKOEI *6* (binär gemäß DKOE-Register *00110*). Entsprechende Treffer würden dann $\texttt{DKOE}_3$ sowie $\texttt{DKOE}_4$ ergeben. Das DKOE-Register bietet einen weiteren Vorteil: Bei **Entfernen einer Domäne** innerhalb der Managementplattform lassen sich Abhängigkeiten zu DKOEs (bzw. Domänenüberschneidungen) sehr effizient abfragen. So kann über bitweise Und-Operationen aus den generierten DKOEI-Werten der einzelnen DKOEs die Abhängigkeit zu einer beliebigen Anzahl an Domänen durch die binär kodierte Position der Domäne in der Spalte des DKOT berechnet werden.

Beim Erstellen einer Domäne sollte das DKOE-Register direkt um das entsprechend generierte Domänenelement in der Managementplattform-Implementierung erweitert werden. Bevor eine Domäne entfernt wird, muss hingegen zuvor in dem DKOE-Register auf etwaige Abhängigkeiten zu DKOEs geprüft werden. Falls keine bestehen, kann das entsprechende Domänenelement aus dem DKOE-Register entfernt werden. Die DKOT sowie die Definition eines DKOE kann wie in Listing 5.7 beschrieben werden. Eine Pseudocode-Implementierung des DKOE-Registers mit Funktionen zur Suche von DKOE-Elementen und DKOE-zu-Domänen-Abhängigkeiten ist in Listing 5.8 gezeigt.

```
class DKOERegister {

  // Liste zur dynamischen chronologischen
  // Verwaltung von Domänenelementen
  Liste<Domäne> DomEinträge;

  // Tabelle zur Verwaltung von DKOE-Einträgen
  // und Zuweisung des DKOEI-Wertes
  Tabelle<DKOE, Integer> DKOEs;

  // Generiere DKOEI für chronologisch sortierte Domänen
  private Integer generiereDKOEI (Set<Domäne> domänen) {
    Integer wert = 0;
    for (i = DomEinträge.size()-1 downto 0) {
      if (DomEinträge.get(i) in domänen) {
        // Bei Überschneidung
        wert += 2**(DomEinträge.size()-1-i);
      }
    }
  }

  public Set<DKOE> findeÜberschneidungen (Set<Domäne> doms) {
    Integer i = generiereDKOEI(doms);
    Set<DKOE> ergebnis;
    for key: DKOEs {
      if (DKOEs[key] == i) {
        ergebnis.add(key);
      }
    }
    return ergebnis;
  }

  // Weitere Methoden zum Hinzufügen und Entfernen von
  // Domänen und DKOEs
  // ...
}
```

Listing 5.8: Pseudocode-Beschreibung des DKOE-Registers mit Suchfunktionen.

5.5.3 Nutzer- und Rollen-Framework

Nutzer und Rollen stellen neben Aufgaben in diesem Konzept die Komponenten für die Verteilung von Zuständigkeiten im Management von FSNs dar. Dabei wird im Zuge der Programmierbarkeit von FSNs nicht nur davon ausgegangen, dass Entitäten, die eine Managementplattform nutzen, ausschließlich menschliche Nutzer umfasst, sondern auch diese unterstützende Dienste bzw. *Jobs*, d. h. Anwendungen, die auf die Managementplattform zugreifen und mit einer Aufgabe verbundene Tätigkeiten automatisiert ausführen (z. B. Auswerten von durch die Managementplattform gesammelte Log-Dateien).

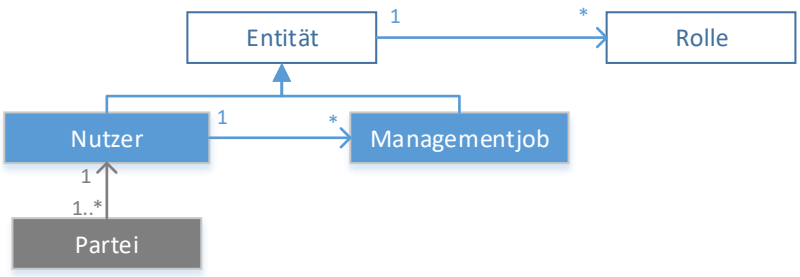


Abbildung 5.20: Elemente zur Beschreibung von Nutzer-Entitäten und -Rollen in einer Managementplattform.

Nutzer und Rollen sind in beinahe allen im letzten Kapitel betrachteten bestehenden Lösungen bereits etablierte Strukturen. Entsprechend wird in diesem Abschnitt nicht im Detail auf deren Umsetzung eingegangen und mehr ein Fokus auf Besonderheiten und ihre Einordnung in die in diesem Konzept definierten Frameworks gelegt.

5.5.3.1 Nutzer-Entitäten

In diesem Konzept werden, wie im linken Teil der Abbildung 5.20 gezeigt wird, nicht nur `Nutzer`, sondern auch Funktionskennungen durch die Klasse `Managementjob` vorgesehen. Beide stellen eine Entität mit über Aufgaben zugewiesenen Zuständigkeiten im Netzmanagement dar (vgl. Abschnitt 5.5.4). Entsprechend sind sie von diesem gemeinsamen Elternelement abgeleitet. Jede Entität wird durch eine föderationsweit eindeutige **ID** einer jeweiligen Instanz und eine Menge an zugewiesenen **Rollen** beschrieben. Ein Nutzer kann beliebig viele Rollen zugewiesen bekommen. Darüber hinaus wird im Rahmen dieses Konzepts die Festlegung der **Privilegienstufe** der Entität zum Zugriff auf das NBI der Managementplattform notwendig (siehe Abschnitt 5.6.2.7). Entsprechend wird auf eine `PrivilegienBaustein`-Instanz referenziert. Wenn keine Privilegienstufe explizit angegeben ist, kann die geringste Privilegienstufe angenommen werden.

Im Konzept ist vorgesehen, dass Funktionskennungen, die durch `Managementjob`-Instanzen beschrieben werden, nicht für sich stehend genutzt werden können. Damit die mit den Nutzerkennungen erfüllte Aufgabe stets in der Verantwortlichkeit eines tatsächlichen Nutzers liegt, werden Funktionskennungen entsprechend Nutzerkennungen zugewiesen. Nutzerkennungen, die durch Instanzen der `Nutzer`-Klasse repräsentiert werden, werden daher zusätzlich zu den vorher genannten und von der Klasse `Entität` geerbten Attributen durch eine Menge an ihnen zugewiesenen **Funktionskennungen** beschrieben, wie auch in Abbildung 5.20 durch die entsprechende Assoziation verdeutlicht wird.

5.5.3.2 Rollen

Rollen müssen im Netzmanagement anwendungsfallspezifisch flexibel beschreibbar sein. In Abschnitt 3.1.2.1 wurden beispielsweise zwei grundlegende Rollen des Betriebs – Netzmanager und Managementplattformadministratoren – identifiziert. Diese Einteilung ist jedoch kaum ausreichend für das Netzmanagement an sich. Wie im nächsten Abschnitt beschrieben wird, teilt sich das Netzmanagement in diesem Konzept in mehrere Aufgabenbereiche. Rollen stellen hier die Schnittstelle zwischen Aufgaben und Nutzern dar und dienen ihrer flexiblen Modellierung. Die Modellierung einer neuen Rolle kann durch Ableitung von Kindklassen von dem Element `Rolle`, das in Abbildung 5.20 gezeigt ist, umgesetzt werden. Eine Rolle wird lediglich durch eine föderationsweit eindeutige **ID** beschrieben.

Rollen sind insbesondere an zwei Stellen in diesem Konzept relevant: Zum einen bei der *üblichen* Zuweisung einer oder mehrerer Aufgaben zu Rollen, wie in Abschnitt 5.5.4.2 beschrieben wird. Diese sind gültig, soweit keine Domänenüberschneidungen auftreten. Zum anderen sind Rollen daher bei der Auflösung von Domänenüberschneidungen relevant und werden zur fallspezifischen Neueinteilung der Aufgaben verwendet. Letzterer Fall wurde zuvor in Abschnitt 5.5.2.3 beschrieben.

5.5.4 Framework für Managementaufgaben

Managementaufgaben stellen in diesem Konzept zentrale Strukturierungselemente zur Organisation im föderierten Netz und in administrativen Domänen (vgl. Abschnitt 5.5.2) dar. Sie steuern Zuständigkeiten und legen in Kombination mit anderen Strukturierungselementen damit verbundene Zugriffsmöglichkeiten auf Elemente und Funktionen einer Managementplattform fest, wie in Abschnitt 5.5.5 beschrieben wird. In diesem Abschnitt wird die Beschreibung von Aufgaben und ihre Einteilung von Aufgaben zu Rollen innerhalb von Domänen adressiert.

5.5.4.1 Elemente zur Beschreibung von Aufgaben

Um eine feingranulare Untergliederung von *Aufgaben* abzubilden, werden in diesem Konzept nicht nur ebendiese, sondern ebenfalls *Aufgabenbereiche* zur Gruppierung von Aufgaben vorgesehen. Aufgabenbereiche selbst untergliedern unmittelbar funktionale Bereiche des Netzmanagements (siehe Abschnitt 5.7.1). Wie in Abbildung 5.21 illustriert ist, referenziert das zentrale Element `Föderation` auf ein oder mehrere Elemente `Aufgabenbereich`, welche selbst jeweils mindestens auf ein Element `Aufgabe` verweisen. Es muss mindestens ein Aufgabenbereich mit einer Aufgabe definiert sein, mit dem Rollen im Management mittels des `Zuständigkeit`-Elements (siehe nächster Abschnitt) verknüpft sind. Eine Konkretisierung von Aufgabenbereichen und Aufgaben erfolgt daher flexibel über die Instanziierung der Elemente. Ein Aufgabenbereich wird entsprechend durch einen innerhalb der Föderation eindeutigen **Namen** (z. B. „Securitymanagement“), eine **Kurzbeschreibung** des jeweiligen Aufgabenbereichs, genauso wie eine Menge an mit dem Aufgabenbereich verknüpften **Aufgabe**-Instanzen beschrieben. Außerdem referenziert ein Aufgabenbereich auf eine Menge an **Alarmkategorien**. Alarme, die mit diesen Alarmkategorien klassifiziert sind, fallen dann in die Zuständigkeit des jeweils ebenfalls damit verknüpften Aufgabenbereichs. Eine Alarmkategorie kann jeweils durch mehrere Aufgabenbereiche referenziert werden.

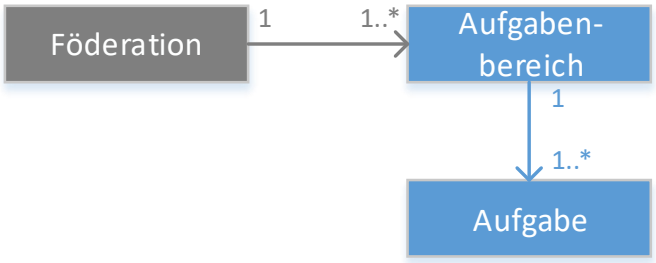


Abbildung 5.21: Das Element `Aufgabenbereich` fasst mehrere `Aufgabe`-Elemente innerhalb einer Föderation zusammen. Das `Aufgabenbereich`-Element ist an das `Föderation`-Element (grau gekennzeichnet) aus dem Framework für Föderationsmodelle geknüpft.

Jede Aufgabe wird durch eine innerhalb der Föderation eindeutige **ID**, einen **Namen** (z. B. „Überwachung“) und eine **Kurzbeschreibung** beschrieben. Anders als bei einem Aufgabenbereich muss das Attribut **Name** in der Föderation nicht notwendigerweise eindeutig sein, sondern gleichartige Aufgaben können im Kontext unterschiedlicher Aufgabenbereiche vorkommen. Namen von Aufgaben müssen aber im Kontext desselben Aufgabenbereichs eindeutig sein. Beispielsweise ist eine Aufgabe mit dem Namen „Überwachung“ nicht nur im Kontext des Aufgabenbereichs *Sicherheitsmanagement*, sondern auch im Aufgabenbereich *Performanzmanagement* denkbar. Beide müssen sich jedoch dann über ihre eindeutige **ID** unterscheiden. Außerdem kann jeder Aufgabe eine Menge an `FSNMOCK`- bzw. davon abgeleiteten Klassen zugewiesen werden. Durch diese können für Aufgaben dedizierte Typen von Netzkomponenten (z. B. ein Network-Intrusion-Detection-System im Sicherheitsmanagement) zugewiesen werden, die zur Erfüllung der Aufgabe notwendig sind (vgl. auch Abschnitt 5.5.5.2).

Wie im folgenden Abschnitt beschrieben wird, werden durch dieses Framework Aufgabenbereiche sowie ein Vorschlag einer Zuständigkeitsverteilung immer zentral für eine Föderation vorgegeben und ist zunächst in allen Domänen darin gültig. Gleichzeitig erlaubt das Framework die Anpassung der Zuständigkeitsverteilung an die jeweiligen Umstände innerhalb einer Domäne.

5.5.4.2 Gültigkeitsbereiche und Zuständigkeiten

Durch die Aufteilung des Netzmanagements in FSNs auf unterschiedliche unabhängige Parteien treffen nicht selten mehrere Organisationskulturen aufeinander. Vor allem sind Rollen und

Aufgaben im Netzmanagement nicht einheitlich definiert. Sie sind beispielsweise von Strukturen wie der Zusammensetzung und Größe des Administrationsteams einer Partei abhängig. Daher kann in diesem Konzept die Einteilung von **Zuständigkeiten**, d. h. die Zuweisung von Aufgaben zu einer Nutzerrolle, zusätzlich für **jede Domäne individuell** durch das Element `Zuständigkeit` vorgenommen werden. Für die gesamte Föderation hingegen muss durch `Zuständigkeit` eine Zuweisung vorgenommen werden, welche als **Voreinstellung** für alle Domänen gültig ist, wenn für eine Domäne keine spezifischere Einteilung festgelegt ist. Zuständigkeiten werden jedoch selektiv *überschrieben*: Sobald eine Zuweisung für eine Domäne existiert, ist diese gültig, andernfalls die Zuweisung auf Föderationsebene.

Wie in Abbildung 5.22 gezeigt ist, wird eine `Zuständigkeit`-Instanz entsprechend von einer zentralen `Föderation`-Instanz sowie keine oder aber auch mehrere weitere `Zuständigkeit`-Instanzen jeweils durch die in einer Föderation definierten `Domäne`-Instanzen referenziert. Dabei muss eine Zuweisung auf Föderations- und auch auf Domänenebene eindeutig sein: Eine Aufgabe darf in einem Kontext nicht mehreren Rollen zugewiesen werden, damit Aufgaben nicht missverständlich definiert sind. Jede `Zuständigkeit`-Instanz verweist auf genau eine `Aufgabe`-Instanz als Teil eines Aufgabenbereichs sowie auf genau eine `Rolle`-Subklasse. Eine Zuweisung erfolgt entsprechend auf dem Klassentyp selbst.

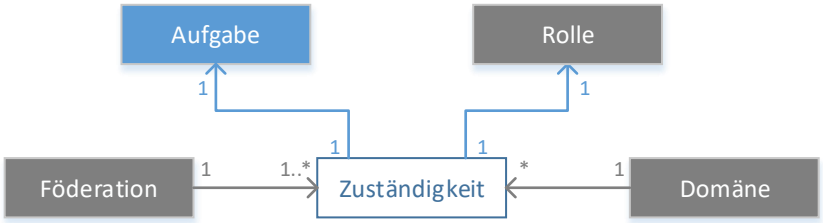


Abbildung 5.22: `Zuständigkeit` stellt eine Richtlinie zur Ausprägung organisatorischer Strukturen dar. Sie wird entweder auf die gesamte Föderation oder auf einzelne Domänen angewendet.

Eine beispielhafte Nutzung und Zuweisung von Zuständigkeiten über das Element `Zuständigkeit` ist in Abbildung 5.23 gezeigt. In diesem Fall gibt die zentrale `Föderation`-Instanz über die `Zuständigkeit`-Instanzen `z1` bis `z4` eine Zuständigkeitsverteilung insofern vor, dass die jeweiligen Aufgaben der Aufgabenbereiche für Security- und Performance-Management (`s-p-man`) und Lifecycle-Management (`lifecycle-man`) über jeweils eine Rollenklasse `SPManager` und Letzteres über `LifecycleManager` durchgeführt wird. Eine feingranularere Unterteilung der einzelnen Aufgaben gibt es daher auf dieser Ebene nicht. Im Beispiel gibt es für Domäne `d1` eine Alternativzuweisung: Die Aufgaben der Überwachung (`monitor`) und Steuerung (`control`) im Aufgabenbereich Security- und Performancemanagement werden über die `Zuständigkeit`-Instanzen `z5` und `z6` auf die Rollenklassen `SPMonitorManager` und `SPControlManager` feingranularer unterteilen. Die generelle Zuweisung des Aufgabenbereichs Lifecycle-Management würde hier nicht überschrieben werden und von der Voreinstellung auf Föderationsebene übernommen werden.

5.5.5 Mandantenfähiger Zugriff auf Informationselemente

Mandantenfähigkeit bezeichnet ein Softwarearchitekturmuster, in dem eine Anwendung den Zugriff mehrerer Mandanten handhaben kann [191]. Im Netzmanagement bezieht sich diese Fähigkeit vor allem auf die Zugriffssteuerung auf Managementfunktionen und -Informationen für unterschiedliche Nutzer. Diese Daten und Funktionen sind insbesondere Elemente des Informationsmodells und werden im Rahmen dieser Beschreibung als *Informationselemente* bezeichnet. Dieses Konzept unterscheidet zwischen der **Sichtbarkeit** und dem **Zugriff** auf Informationselemente.

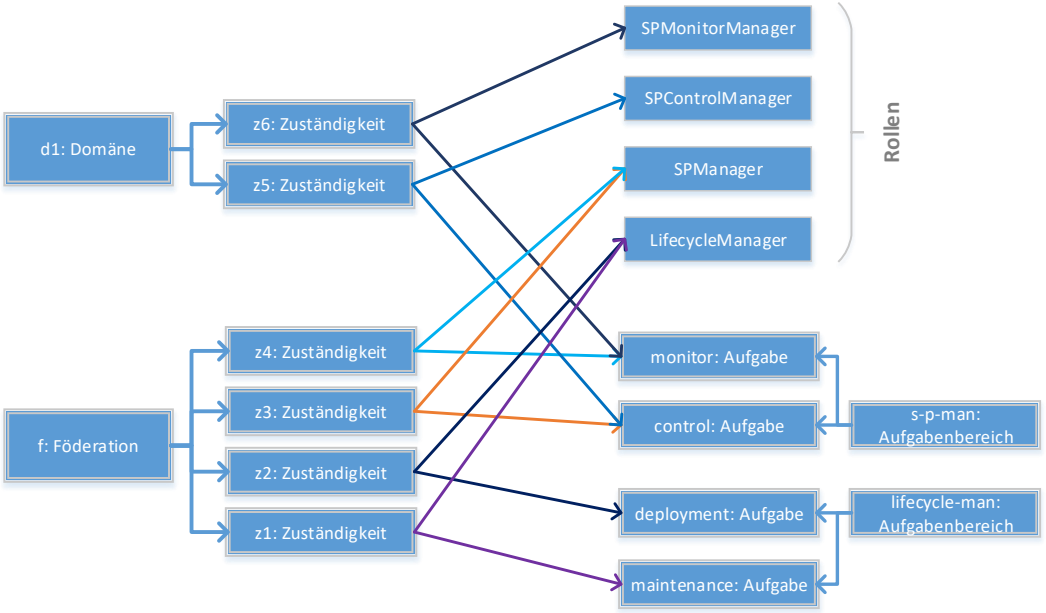


Abbildung 5.23: Beispielhafte Definition von Aufgaben und Rollen, mit einer föderationsweiten Zuweisung von Zuständigkeiten und einer separaten Zuweisung für Domäne `d1`. Die farbliche Kodierung der Assoziationen dient der Übersichtlichkeit.

Ein anderer Frameworkansatz zur Zugriffssteuerung auf MOC-Funktionen wurde in Vorarbeiten dieser Dissertation in [181] vorgeschlagen. Im Folgenden wird jedoch ein für die in dieser Arbeit entwickelten Frameworks des Informationsmodells integrierter ganzheitlicher Ansatz vorgestellt.

5.5.5.1 Gesteuerte Informationselemente

Elemente, deren Zugriff und Sichtbarkeit in einer Managementperspektive zu steuern sind, umfassen solche, die zum einen den Zustand der gemanagten Infrastruktur wiedergeben und zum anderen ihre Steuerung erlauben. Elemente, die den Zustand der gemanagten Infrastruktur **wiedergeben** und daher in ihrer **Sichtbarkeit** steuerbar sein müssen, sind die Folgenden:

- `FSNMOC`-**Instanzen** zur allgemeinen Darstellung von MOs und zur Kapselung von Attributen und MOC-Funktionen (vgl. folgende Elemente). Dazu zählen auch **Managementanwendungen**.
- `Attribut`-**Instanzen**, die Detailinformationen eines MO in Form einer `FSNMOC`-Instanz oder ihrer Klassifikationen (`FSNMOCK`-Instanz) beschreiben.
- **MOC-Funktionen** eines MOs. **Plattformfunktionen** (vgl. Abschnitt 5.7.2.1) gehören dagegen **nicht** zu gesteuerten Informationselementen, da sie sich nicht in Abhängigkeit zu Organisationsstrukturen wie Domänen, Domänenbeziehungen oder Föderationsbeziehungen stellen lassen.
- `Abhängigkeit`-**Instanzen** bzw. Managementbeziehungen zwischen `FSNMOC`-Instanzen.
- `Netzinformation`-**Instanzen**, die in Zusammenhang mit einer bestimmten `FSNMOC`-Instanz stehen können.

Elemente, die den Zustand der gemanagten Infrastruktur **beeinflussen** und deren **Zugriff** gesteuert werden soll, sind im Informationsmodell insbesondere die Folgenden:

- **MOC-Funktionen**, die von entsprechenden FSNMOCK unterstützt werden.
- **Managementanwendungen**, welche in diesem Fall auf MOC- oder Plattformfunktionen zugreifen.

Bei **MOC-Funktion-Instanzen** macht der Aspekt der Sichtbarkeit als alleinstehendes Merkmal üblicherweise wenig Sinn, da sie an sich keine Informationen darstellen, sondern rein eine Abbildung einer Funktionalität eines MO beschreiben. Entsprechend sind der Zugriff und die Sichtbarkeit gleichbedeutend.

Auch **Managementanwendungen** stellen eine Besonderheit dar: Die tatsächliche Nutzbarkeit einer Managementanwendung hängt von der Möglichkeit eines bestimmten Nutzers ab, darin verwendete Daten einzusehen und auf Funktionen der Managementplattform zugreifen zu können. Eine feingranulare Trennung wird daher notwendig. Dennoch ist die Steuerbarkeit der Sichtbarkeit und des Zugriffs von Managementanwendungen sinnvoll: Einerseits kann einem Nutzer so eine Übersicht über jeweils nutzbare Managementanwendungen generiert werden, andererseits kann so die Sichtbarkeit auf durch Managementanwendung weiterverarbeitete Daten eingeschränkt werden. Eine Managementplattform muss folglich ausgehend vom Nutzer Informationselemente, auf die über eine Managementanwendung zugegriffen wird, einschränken.

Als weitere Informationselemente können darüber hinaus Elemente des **Organisationsmodells** selbst betrachtet werden: Föderationen und Föderationsbeziehungen, Aufgabenbereiche und Aufgaben, Domänen und Domänenbeziehungen, Nutzer und Rollen sowie Zuständigkeiten. Sie werden hier jedoch nicht als unmittelbare Informationselemente des operativen Betriebs angesehen, sind andernfalls jedoch analog wie die zuvor beschriebenen Informationselemente zu behandeln.

5.5.5.2 Kriterien für Sichtbarkeit und Zugriff

Die Möglichkeit eines Nutzers, Informationselemente einsehen oder darauf zugreifen zu können, ist in FSNs durch den Föderationsaspekt von einer Mehrzahl an Kriterien abhängig. Primär dienen administrative Domänen zur Herstellung eines administrativen Bezugs zwischen einem Nutzer und einem MO. Entsprechend ist die **geteilte Domänenzugehörigkeit** ein grundlegendes Kriterium. Die Domänenzugehörigkeit erlaubt zunächst immer den Zugriff auf alle Elemente in einer Domäne, soweit er nicht von anderen Kriterien eingeschränkt wird. Damit verbunden ist das Kriterium der **Aufgabenzuweisung** für ein MO. Mit der Verantwortlichkeit eines Nutzers für eine konkrete Aufgabe ist die Sichtbarkeit und der Zugriff auf `Attribut`-Instanzen, (z. B. Security-relevante Attribute), MOC-Funktionen, Managementbeziehungen und Netzinformationen (z. B. sind IDS-Notifikationen auch für den Security-Aufgabenbereich relevant) verbunden. Ein wiederum mit dem Kriterium der Aufgabenzuweisung verknüpftes Kriterium ist das der MO-Art, d. h. der Klassifikation einer `FSNMOC`-Instanz über `FSNMOCK`-Elemente: Aufgabenrelevante MOs wie Security-Anwendungen (z. B. Firewall und IDS) oder Anwendungen zur Ermittlung der Ressourcennutzung (Accounting) sollen ihren jeweiligen Aufgaben zuordenbar sein. Dieses Kriterium regelt daher generell die Sichtbarkeit von `FSNMOCK`-Instanzen. Da die Zuweisung von `FSNMOCK`-Elementen zu zentral definierten Aufgaben erfolgt, ist dieses Kriterium föderationsweit einheitlich. Ein weiteres Kriterium sind **Domänenbeziehungen**. Durch Domänenbeziehungen sollen beispielsweise (aber nicht nur) temporäre Zugriffe und Sichtbarkeiten über mehrere Domänen hinweg ermöglicht werden. Durch dieses Kriterium wird generell die Sichtbarkeit und der Zugriff auf alle Informationselemente geregelt. Da Domänenbeziehungen über alle administrativen Domänen in einer Föderation definiert werden, sind damit verbundene Sichtbarkeits- und Zugriffsberechtigungen ebenso zentral festzulegen.

Ein Kriterium, das ausschließlich in Föderationen vorkommt, ist das der **Föderationsbeziehungen**. Durch diese wird der Zugriff auf Informationselemente auf Basis von Beziehungen zwischen Föderationsparteien geregelt. Beispielsweise in Abhängigkeit von Vertrauensbeziehungen, wodurch Parteien aus ihrer Sicht nicht vertrauenswürdigen Parteien in der Föderation Sichtbarkeit und Zugriff auf die von ihnen bereitgestellten IT-Ressourcen und darauf realisierten virtuellen IT-Ressourcen einschränken oder aber für vertrauenswürdige Parteien explizit freigeben. Auch dieses Kriterium ist generell auf jede Art von Informationselement anzuwenden. Die Besonderheit hier ist jedoch, dass Informationselemente mit Bezug zu einer bestimmten Partei betroffen sind und insbesondere zu einer Partei gehören (d. h. von dieser ursprünglich in die Föderation eingebracht oder davon abgeleitet wurden). Die Festlegung dieses Kriteriums kann auf Basis einer zentral vorgegebenen Auswahl an Vorlagen geschehen (z. B. die Definition von drei Vertrauensstufen, abgeleitet von `Vertrauensbeziehung` mit jeweils festgelegten Berechtigungen), die jedoch teils von einzelnen Partnern (insbesondere `Föderationseinfachbeziehung`-Instanzen), teilweise zentral für die ganze Föderation (insb. `Föderationsmultibeziehung`) nutzbar sind.

Neben diesen, weitestgehend für sich stehenden Kriterien, sind darüber hinaus die Folgen von Sichtbarkeits- und Zugriffskonfigurationen zu beachten:

- **MO-Sichtbarkeit**: Sobald eine `FSNMOC`-Instanz für einen Nutzer **nicht** sichtbar ist, so sind auch diese beschreibende `Attribute`-Instanzen sowie MOC-Funktion-Instanzen nicht sichtbar bzw. zugreifbar. Das Gleiche gilt für verknüpfte `Netzinformation`-Instanzen und auf das MO-bezogene Managementbeziehungen.

- **Managementanwendungen** sind einerseits eigenständige Informationselemente und andererseits Konsumenten anderer Informationselemente. Insofern hängt auch der nutzbare Funktionsumfang einer Managementanwendung von ihrem jeweiligen Nutzer ab.

5.5.5.3 Kriterienverwaltung und Kollisionsbehandlung

Durch die Berücksichtigung mehrerer Kriterien wird eine geeignete **Kollisionsbehandlung** notwendig. Kollisionen können an zwei unterschiedlichen Stellen auftreten:

- Eine **Inter-Kriterien-Kollision**, bei der sich Regeln mindestens zweier Kriterien widersprechen: Beispielsweise wird der Zugriff auf eine Funktion eines MO einem Nutzer einerseits auf Basis einer entsprechenden Aufgabenverteilung erlaubt, jedoch aufgrund einer Vertrauensbeziehung verboten.

- Eine **Intra-Kriterium-Kollision**, bei der sich zwei Elemente desselben Kriteriums widersprechen: Beispielsweise wenn ein Nutzer aufgrund seiner Rollen in einer Domäne mehr als eine Aufgabe im Management erfüllen muss und sich die jeweiligen Sichtbarkeits- und Zugriffsregeln widersprechen.

Kollisionsbehandlung muss szenarioabhängig getroffen werden. In diesem Konzept wird die Kollisionsbehandlung über die hier **Zugriffsverwaltungsbaum** (ZVB) genannte Struktur gelöst. Der ZVB dient als im Framework unterstützte Form zur Festlegung einer **Richtlinie zur multikriteriellen Zugriffsbestimmung** für Managementplattformbetreiber. Die Knoten des ZVB (im Folgenden als *ZVB-Knoten* bezeichnet) stellen jeweils einen Kriterienblock dar. In der Organisation von FSNs wurden die folgenden Kriterien als Kriterienblöcke identifiziert:

- Domänenzugehörigkeit und Aufgabenzuweisung

- Föderationsbeziehungen

- Domänenbeziehungen

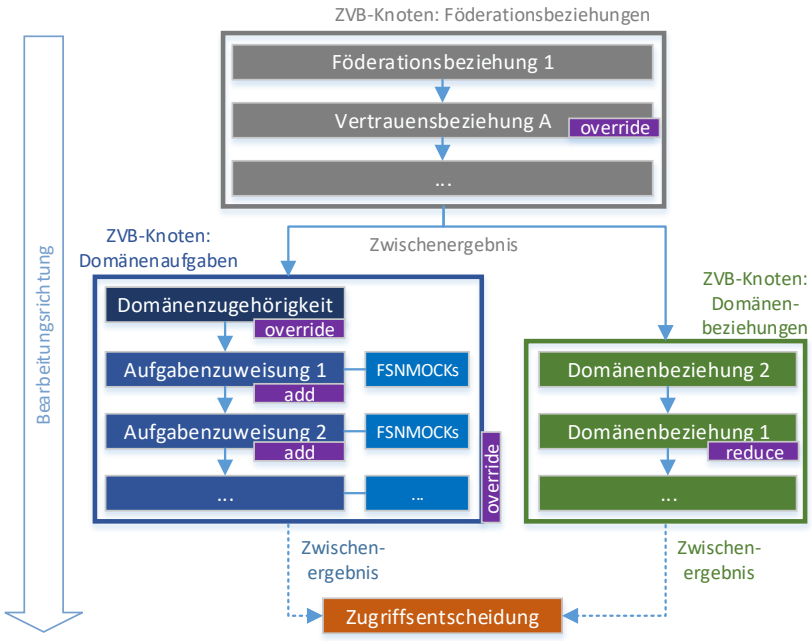


Abbildung 5.24: Beispielhafter Zugriffsverwaltungsbaum als Überprüfungspfad mit Kriterien als ZVB-Knoten und jeweiligen Folgen von ZVB-Elementen. Die Farben verdeutlichen die Kriteriengruppen.

ZVB-Knoten können im ZVB in eine für einen spezifischen Anwendungsfall geeignete Reihenfolge zur Abarbeitung angeordnet werden (z. B. direkt seriell oder auch mit Fallunterscheidung). Ein ZVB-Knoten beschreibt (mit Ausnahme der Domänenzugehörigkeit) eine Folge von Elementen desselben Kriteriums, den *ZVB-Elementen*. Innerhalb jedes ZVB-Knotens erfolgt eine Teilentscheidung der Sichtbarkeit und des Zugriffs auf ein Informationselement für einen Nutzer. Generell ist die Modellierung des konkreten ZVBs szenarienspezifisch durch ein Framework zu unterstützen. Das betrifft einerseits die Konstellation der ZVB-Knoten (z. B. Reihenfolge und Fallunterscheidungen) der aufeinander aufbauenden Kriterien, andererseits die Reihenfolge der ZVB-Elemente innerhalb der Knoten. Eine Entscheidung vorhergehender ZVB-Knoten kann durch Entscheidungen nachfolgender ZVB-Knoten modifiziert werden. Falls keine Modifikation angegeben ist, bleibt das letzte Zwischenergebnis erhalten. Innerhalb eines ZVB-Knotens gilt jeweils das gleiche Prinzip für die Folge der ZVB-Elemente. Die Reihenfolge der ZVB-Knoten und ZVB-Elemente ist daher essenziell.

Die **Modifikation** der Entscheidung wird mit unterschiedlichen Flags pro ZVB-Knoten und Element beschrieben:

- Ein *override*-Flag kennzeichnet eine **Überschreibung** einer Teilentscheidung. Hier zu beachten ist die Ausnutzung von Optimierungen. Kann beispielsweise eine Entscheidung eines vorhergehenden ZVB-Knotens nicht mehr überschrieben werden, können entsprechende nachfolgende Pfade ignoriert werden.

- Eine **Kombination** der Zugriffsberechtigungen, die durch ein *add*-Flag gekennzeichnet wird. Dies ist beispielsweise notwendig, um Nutzern mit mehreren Aufgabenzuweisungen eine kombinierte Sicht auf Informationselemente zu erlauben.

- Eine **Einschränkung** der Berechtigungen über ein *reduce*-Flag. Identifizierte Berechtigungen eines Blocks werden daher von dem Eingangsergebnis abgezogen.

Abbildung 5.24 zeigt eine beispielhafte sinnvolle Modellierung eines ZVB: Darin gibt der ZVB-Knoten *Föderationsbeziehungen* die Entscheidungsgrundlage vor, welche entweder durch den ZVB-Knoten der *Domänenbeziehungen* oder der *Domänenaufgaben* beeinflusst werden kann. Naheliegenderweise umfasst letzterer ZVB-Knoten das Kriterium der Domänenzugehörigkeit eines Nutzers und das der in Domänen verbundenen Aufgaben gemeinsam. Kriterien der Domänenzugehörigkeit und Domänenbeziehungen sind meist disjunkt. In Ausnahmefällen gibt es auch Konstellationen, in denen beides zutrifft: Ein Nutzer teilt sich eine Domäne mit dem jeweils angefragten Informationselement und ist in einer anderen Domäne, für die eine Beziehung zu der Domäne des jeweiligen Informationselements besteht. Die Domänenzugehörigkeit wird wiederum durch den Aspekt der Aufgabenzuweisung und damit verknüpfter FSNMOCK-Vorgaben eingeschränkt. Die Kriterieninstanzen sind darin szenarienabhängig in einer sinnvollen Bearbeitungsreihenfolge aufgereiht. Die Bearbeitungsrichtung ist in diesem Fall von föderationsspezifischen hin zu domänenspezifischen Aspekten festgelegt.

Der hier vorgestellte Ansatz definiert Informationselemente und Kriterien des Zugriffs im Netzmanagement von FSNs. Darüber hinaus stellt der Zugriffsverwaltungsbaum ein Konzept zur Entscheidungsfindung über alle Kriterien unter Berücksichtigung einer Kollisionsbehandlung dar. Mögliche Umsetzungen dieser Konzepte sind jedoch durch andere bestehende und etablierte Frameworks wie **XACML** denkbar. Die in diesem Konzept beschriebene Lösung geht jedoch über eine einfache Zugriffsentscheidung hinaus und erlaubt auch die effiziente Ermittlung aller zugreifbarer Elemente eines Nutzers in einem Kontext (siehe folgender Abschnitt). Eine administrative Domäne stellt stets diesen Kontext dar, ohne den zugreifbare Elemente nicht ermittelbar sind. Derartige Funktionen sind über reine XACML-basierte Zugriffsentscheidungen nicht effizient möglich.

5.5.5.4 Kriterienausprägung und Zugriff auf Informationselemente

Zugriffsentscheidungen für die in Abschnitt 5.5.5.1 beschriebenen Informationselemente auf Basis der Kriterien aus Abschnitt 5.5.5.2 werden in erster Linie über den im vorherigen Abschnitt beschriebenen Zugriffsverwaltungsbaums (ZVB) verwaltet. Für jedes ZVB-Element in einem ZVB-Knoten wird unter Verwendung von **Tags** der Zugriff über eine sogenannte **Zugriffsregel** auf bestimmte Informationselemente explizit erlaubt bzw. verboten. Tags können Klasseninstanzen (vgl. Abschnitt 5.3.2.2) oder auch Klassen von Elementen des Informationsmodells (vgl. Abschnitt 5.3.2.1) kennzeichnen. Eine Zugriffsregel unterstützt wahlweise ein **White-** genauso wie ein **Blacklisting** von Tags. Alle Zuweisungen von Kriterienklassen zu jeweils einer Zugriffsregel werden als **Zugriffregelsatz** bezeichnet. Über den Zugriffsregelsatz können ZVB-Elemente hinsichtlich ihrer Kriterienausprägung festgelegt werden. Die Beschreibung eines Zugriffregelsatzes wird in Listing 5.9 zusammengefasst sowie eine Beispielzuweisung für drei stringbasierte Tags *TagA* bis *TagC* gezeigt.

```
1  class Zugriffsregel {
2    // Default ist Whitelisting;
3    boolean BlackListing = false;
4
5    // Menge berücksichtigter Tags
6    Set<Tags> Tags;
7  }
8
9  // Zugriffregelsatz
10 Tabelle<Klasse<Kriterium>, Zugriffsregel> Zugriffsregelsatz;
11
12
13 // Beispielzuweisung mit allgemeiner
14 //  Vertrauensbeziehung und Blacklisting
15 Zugriffregelsatz.add(Vertrauensbeziehung,
16   new Zugriffsregel(true, ["TagA", "TagB", "TagC"]));
```

Listing 5.9: Pseudocodedarstellung eines Regelsatzes mit Beispielinstanziierung.

Zugriffsregeln gelten für die ZVB-Elemente der Kriterien für Föderationsbeziehungen, Domänenbeziehungen, sowie Aufgabenzuweisungen. Kriterien wie die Domänenzugehörigkeit oder MO-Sichtbarkeit werden dagegen explizit durch die Domänenzuweisung oder implizit durch die anderen Kriterien gesteuert.

Die Entscheidung bzgl. der Sichtbarkeit und des Zugriffs durch den zuvor in Abbildung 5.24 vorgegebenen ZVB wird über die den jeweiligen Nutzer betreffende Kriterienelemente bestimmt. In Abbildung 5.25 wird dazu ein konkretes Beispiel für einen Nutzer *Alice* gezeigt. Angenommen, Alice betrifft *Vertrauensbeziehung A*, d. h. das angefragte Informationselement beschreibt eine IT-Ressource, die einer anderen Partei zugewiesen ist als die, der Alice angehört. Das Informationselement steht jedoch in Bezug zu einer administrativen Domäne, der Alice angehört (Kriterienelement der *Domänenzugehörigkeit* ist erfüllt) und Alice hat durch *Aufgabenzuweisung 2* eine Aufgabe darin inne. Die Bedeutung von *Vertrauensbeziehung A* und *Aufgabenzuweisung 2* sind im Zugriffsregelsatz definiert. In diesem Beispiel würde das Zwischenergebnis aus dem ZVB-Knoten *Föderationsbeziehungen* durch das Ergebnis des ZVB-Knotens *Domänenaufgaben* überschrieben werden.

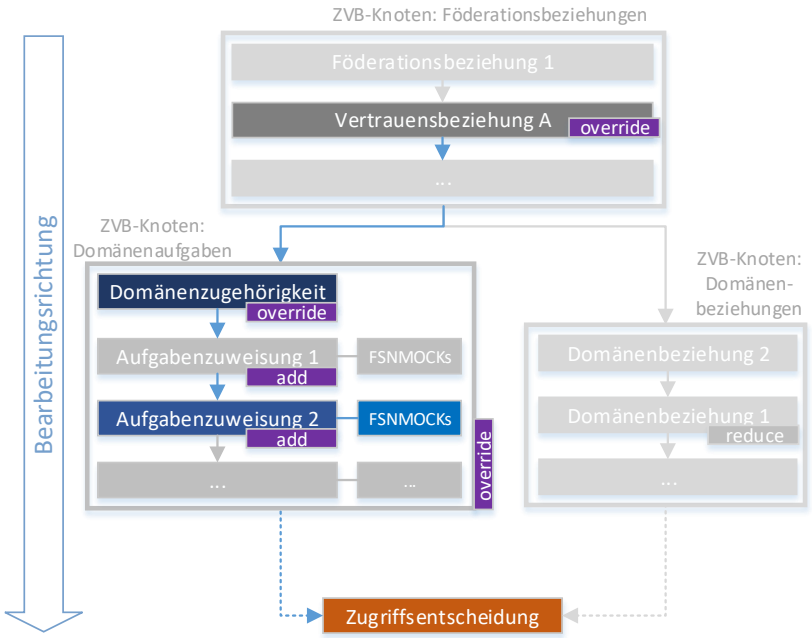


Abbildung 5.25: Beispiel relevanter Kriterien (visuell hervorgehoben) der Sichtbarkeits- und Zugriffsermittlung im Zugriffsverwaltungsbaum auf ein Informationselement *X* für einen fiktiven Nutzer *Alice*.

In gemanagten Umgebungen mit vielen Informationselementen muss jedoch auch eine **Übersicht über für Nutzer relevante Informationselemente** in einer Domäne effizient generiert werden können. Zu diesem Zweck sind aus dem ZVB für einen Nutzer im Kontext einer **spezifizierten administrativen Domäne** alle Tags bestimmbar, welche die Sichtbarkeit und den Zugriff von Informationselementen **erlauben**. Durch eine Verknüpfung von Tags mit durch sie markierten Informationselementen (vgl. Abschnitt 5.3.2) können Letztere effizient bestimmt werden.

Für die Bestimmung von Tags, die für einen **Nutzer** in einer Domäne sichtbare Informationselemente kennzeichnen, wird der ZVB unter Berücksichtigung ebendieser **Domäne** und der Bearbeitungsrichtung betrachtet. Dazu wird der Bearbeitungspfad durch den ZVB nachvollzogen und die entsprechende Menge an Tags, die einem Nutzer den Zugriff bzw. die Sichtbarkeit

erlauben, nachvollzogen: Das Flag **override** ersetzt die vorherige Menge an Tags, **add** fügt Tags aus Whitelisting-Mengen bzw. die invertierte Blacklisting-Menge hinzu, und **reduce** entfernt Tags aus Blacklisting-Mengen bzw. invertierte Whitelisting-Mengen.

5.6 Kommunikationsmodell

Die in diesem Konzept beschriebenen Frameworks des Kommunikationsmodells dienen der Anbindung der gemanagten Infrastruktur an die Managementplattform über das SBI sowie zur Erweiterung der API der Managementplattform über das NBI zur Anbindung von Nutzern, Managementanwendungen und anderen Managementplattform-Instanzen in einem verteilten System. Außerdem wird ein Framework für Schnittstellen zu Datenbanken zur Speicherung von Informationselementen des Netzmanagements beschrieben. Die drei Schnittstellen teilen sich zentrale Komponenten, die ebenfalls beschrieben werden. Eine Übersicht aller Komponenten ist in Abbildung 5.26 gezeigt. Sie werden in den folgenden Abschnitten erläutert.

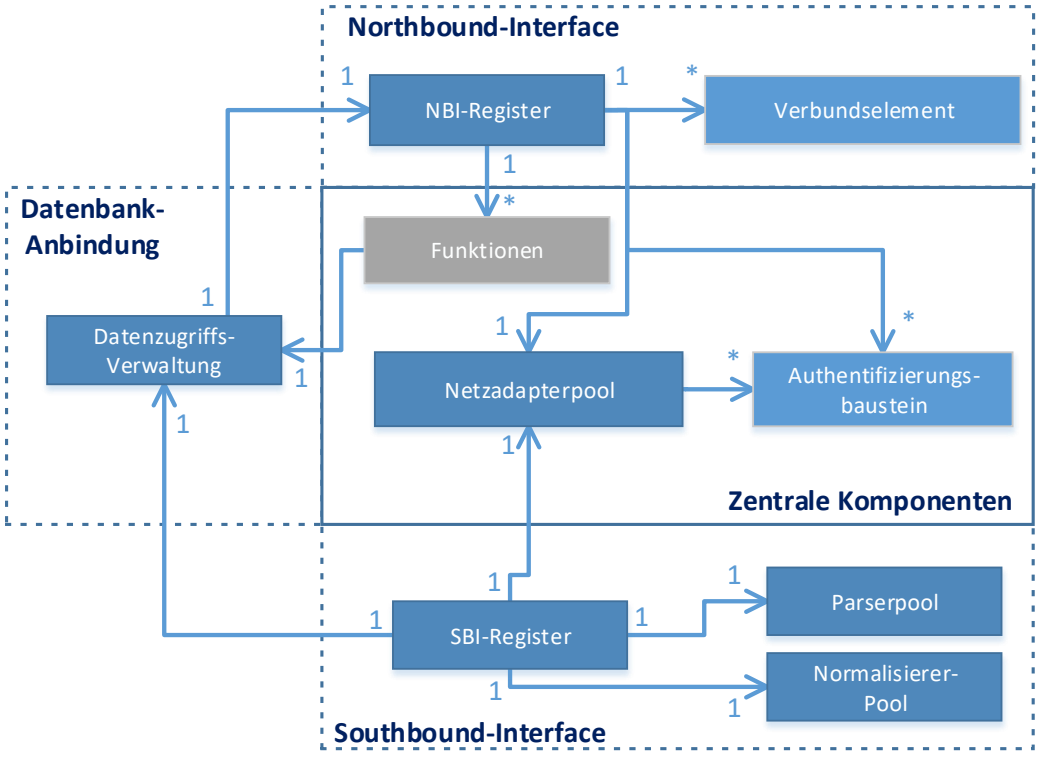


Abbildung 5.26: Übersicht über zentrale und schnittstellenspezifische Komponenten des Kommunikationsmodells. Die Komponente *Funktionen* ist Teil des Funktionsmodells und beschreibt Funktionen der Managementplattform.

5.6.1 Southbound-Interface

Gemäß dem operativen Prozess in Abschnitt 5.1.1.1 muss das SBI *a)* Daten der gemanagten IT-Infrastruktur empfangen und abrufen, *b)* unterschiedliche Datenformate parsen und *c)* die Daten in ein normalisiertes Format gemäß dem Informationsmodell bringen können.

Abbildung 5.27 zeigt einerseits die **Architektur** des SBI. Andererseits zeigt die Abbildung den grundlegenden **automatisierten Prozess** vom Empfang eines Datums von der gemanagten IT-Infrastruktur bis zur Integration in das Informationsmodell der Plattform. Für eine bessere Übersichtlichkeit ist der Prozess an dieser Stelle beschrieben und nicht im Funktionsmo-

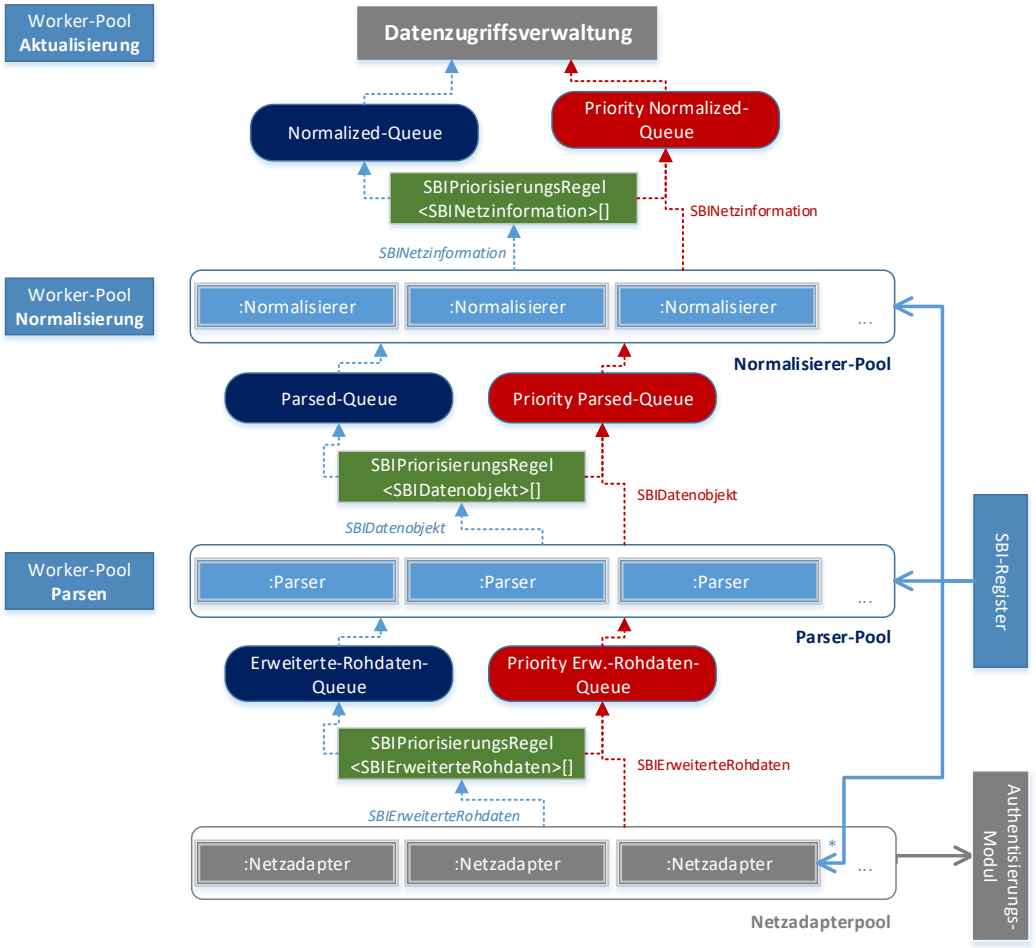


Abbildung 5.27: Architektursicht auf Komponenten, Schnittstellen und Datencontainer (`SBIErweiterteRohdaten`, `SBIDatenobjekt` und `SBINetzinformation`) im SBI. Gestrichelte Linien deuten einen Datenfluss mit jeweiligem Datencontainer an.

dell. Demnach sind implementierte **Komponenten** wie Netzadapter, Parser, und Komponenten zur Informationsnormalisierung nicht einzeln, sondern jeweils in einem **Pool** organisiert. Eine Information aus der gemanagten Infrastruktur wird durch einen Konnektor abgerufen oder durch eine Schnittstelle empfangen, muss durch einen passenden *Parser* in ein Objekt konvertiert werden und danach einem passenden *Normalisierer* zugewiesen werden. Letztendlich folgt auf jede Information eine Operation zur Aktualisierung des bestehenden Datenmodells: Beispielsweise das Anlegen eines neuen Zustands oder die Aktualisierung eines bestehenden Zustands. Das *SBI-Register* koordiniert den SBI-spezifischen Verarbeitungsprozess. Im SBI ist aufgrund potenziell in sehr großen Netzen sehr vieler quasi gleichzeitig empfangener Informationen eine Parallelisierung der Verarbeitungsprozesse wichtig. Ein entsprechender Ansatz ist in Abschnitt 5.6.1.4 beschrieben.

5.6.1.1 Parsen empfangener Daten

Dem Prozess des Parsens vorgelagert ist die *Erweiterte-Rohdaten-Queue*: Die eigentlichen über einen Netzadapter empfangenen Daten liegen noch im ursprünglichen serialisierten Datenformat (z. B. *JSON*, *XML* oder *CSV*) vor. Sie sind in einem Vorverarbeitungsschritt in einem Datencontainer mit Identifikator für das jeweilige verwendete **Datenformat** gekapselt. Das Hinzufügen dieses und weiterer notwendiger Metadaten wird durch den jeweiligen bearbeiten-

den Netzadapter realisiert. Das hier als **Erweiterte Rohdaten** bezeichnete Datencontainerformat ist in Listing 5.10 gezeigt. Es dient als Eingabe für einen Parser des Parser-Pools, welcher anhand des Attributs `fDatenformat` ausgewählt wird. Weitere Metadaten, um die die Rohdaten erweitert werden, sind der Empfangszeitpunkt, die ID des Quell-Datenzentrums und die IP-Adresse der Quell-Komponente. Die zwei letzteren Daten dienen der Identifikation der MO-ID der Quell-Komponente. Bereits dieser Datencontainer enthält potenziell Informationen zur Normalisierung der Rohdaten: Eine Menge von Normalisierungsmarken, die in Abschnitt 5.6.1.2 erläutert und an nachfolgende Datencontainerformate weitergegeben werden.

```
1 class SBIErweiterteRohdaten {
2   String fRohdaten;
3   Datenformat fDatenformat;
4   Zeitstempel fEmpfangsZeit;
5   Datenzentrum fQuellDatenzentrum;
6   IPAdresse fQuelle;
7   Set<Normalisierungsmarke> fNormMarken;
8 }
```

Listing 5.10: Pseudocodedarstellung des Datencontainerformats für Erweitere Rohdaten.

Der Typ `Datenformat` entspricht einem durch den jeweiligen Managementplattform-Entwickler und mit einem entsprechenden Schlüssel im Parser-Pool übereinstimmenden Wert, der geeignete Parser im Parser-Pool identifiziert. Der Parser-Pool verhält sich wie eine Tabelle, die von einem Datenformat-Identifikator auf den jeweils passenden Parser verweist, wie in Abbildung 5.28 gezeigt ist.

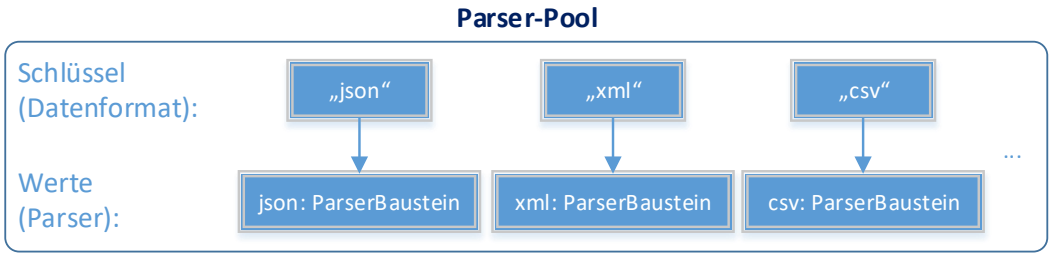


Abbildung 5.28: Funktionsweise des Parser-Pools als Tabelle: Parser (hier beispielhaft) werden über `Datenformat`-Instanzen ausgewählt.

```
1 interface ParserBaustein {
2   public SBIDatenobjekt parse(SBIErweiterteRohdaten r) throws ParserFehler;
3 }
```

Listing 5.11: Pseudocode-Darstellung der Klasse `ParserBaustein`.

```
1 class SBIDatenobjekt<Datenobjekt> {
2   Datenobjekt fDatenobjekt;
3   Zeitstempel fEmpfangsZeit;
4   FSNMOC fQuellMO;
5   Set<Normalisierungsmarke> fNormMarken;
6 }
```

Listing 5.12: Pseudocodedarstellung des Datencontainerformats für ein geparstes Datenobjekt.

Ein Parser ist eine Implementierung der Klasse `ParserBaustein` (siehe Listing 5.11). Die Klasse gibt die Eingabe- und Ausgabe-Schnittstellen vor, um ein Element aus der *Erweiterten-Rohdaten-Queue* in ein für die *Parser-Queue* geeignetes Element zu überführen. Bei einer `ParserBaustein`-Instanz dient der zuvor beschriebene Datencontainer **`SBIErweiterte-Rohdaten`** als **Eingabe**. Die **Ausgabe** eines Parserbausteins ist der Datencontainer **`SBIDaten-objekt`** (vgl. Listing 5.12). Er beschreibt ein, je nach verarbeitetem Datenformat, strukturiert

zugreifbares Datenobjekt (z. B. JSON-Objekt, XML-Objekt oder CSV-Objekt), das aus dem vorherigen serialisierten Objekt generiert wurde. Der Empfangs-Zeitstempel und die Menge an Normalisierungsmarken werden aus dem `SBIErweiterteRohdaten`-Objekt übernommen. Ein implementierter `ParserBaustein` übernimmt ebenfalls die Aufgabe, sofern möglich, aus den zuvor rudimentären Quellenangaben des eingegebenen `SBIErweiterteRohdaten`-Objekts (IP-Adresse und Datenzentrum-ID der Quellkomponente) das eindeutige Quell-MO zu ermitteln und es im `SBIDatenobjekt`-Container jeweils zu referenzieren.

5.6.1.2 Normalisierung von Datenobjekten

Am SBI empfangene Daten dienen letztendlich der Aktualisierung des über das Informationsmodell beschriebenen Zustands der gemanagten IT-Infrastruktur und seiner Komponenten: Beispielsweise FSNMOC-Instanzen, Managementbeziehungen und Netzinformationen. **Netzinformationen** aus Abschnitt 5.4.4 geben empfangene Daten aus der gemanagten Infrastruktur am besten wieder. Entsprechend werden sie in diesem Konzept als Format für ihre Normalisierung genutzt. Die Ableitung mehrerer Netzinformationen aus einer `SBIDatenobjekt`-Instanz, wie es vom Parser zu einem *Normalisierer* gegeben wird, ist ebenfalls denkbar. Ein Normalisierungsbaustein gibt daher eine Liste an `Netzinformation`-Instanzen inkl. Metadaten zurück, wie Listing 5.13 zeigt. Die für das Framework wichtige Methode ist `normalize`. Sie wird bei der Bearbeitung durch einen Worker mit entsprechender Eingabe eines Datenobjekts aus der *Parsed-Queue* aufgerufen. In diesem Schritt ist eine Fehlerbehandlung (Exception `NormalisierungsFehler`) unbedingt notwendig, da für eine Eingabe nicht notwendigerweise ein passender Normalisierer implementiert ist. Der als Ausgabe verwendete Datencontainer `SBINetzinformation` ist in Listing 5.14 zusammengefasst. Er umfasst eine Menge generierter Netzinformationen, weiterhin den Empfangszeitstempel sowie das Quell-MO.

```
1 interface NormalisierungsBaustein {
2   public SBINetzinformation normalize(SBIDatenobjekt r) throws
        NormalisierungsFehler;
3 }
```

Listing 5.13: Pseudocode-Darstellung des Interface `NormalisierungsBaustein` als Basis für Normalisiererkomponenten im Normalisierer-Pool.

```
1 class SBINetzinformation {
2   Set<Netzinformation> fNetzinformationen;
3   Zeitstempel fEmpfangsZeit;
4   FSNMOC fQuellMO;
5 }
```

Listing 5.14: Pseudocodedarstellung des Datencontainerformats für normalisierte Netzinformationen.

Analog zum Parser-Pool werden alle Normalisierer in einem Normalisierer-Pool organisiert. Die **Auswahl** geeigneter Normalisierer erfolgt zunächst durch das im Eingabeobjekt vorhandene Quell-MOs. Für damit verknüpfte `FSNMOCK`-Instanzen (d. h. den Klassifikationen des jeweiligen Quell-MO, z. B. OpenDaylight) sollen geeignete Normalisierer registriert sein. Da das Quell-MO potenziell auch mehrfach klassifiziert sein kann (z. B. als OpenDaylight- und als LXC-Instanz), ist dieses Kriterium nicht einschränkend genug (wenn auch bei geeigneter Trennung der Systeme üblicherweise ausreichend). Insofern wird darüber hinaus die Art des Datenobjekts (z. B. ein JSON-Objekt oder ein XML-Objekt) grundsätzlich zur weiteren Unterscheidung herangezogen. Ein drittes Kriterium zur Auswahl geeigneter Normalisierer stellen benutzerdefinierte, aber strukturiert genutzte Schlüssel dar, die hier *Normalisierungsmarken* genannt werden. Sie können die Auswahl von Normalisierern insbesondere für Pull-Zugriffe auf die gemanagte Infrastruktur einschränken, wie im weiteren Verlauf dieses Abschnitts beschrieben wird. Jeder Normalisierer **muss** hingegen hinsichtlich geeigneter FSNMOCK und Datenformate klassifiziert sein. Normalisierungsmarken sind dagegen optional.

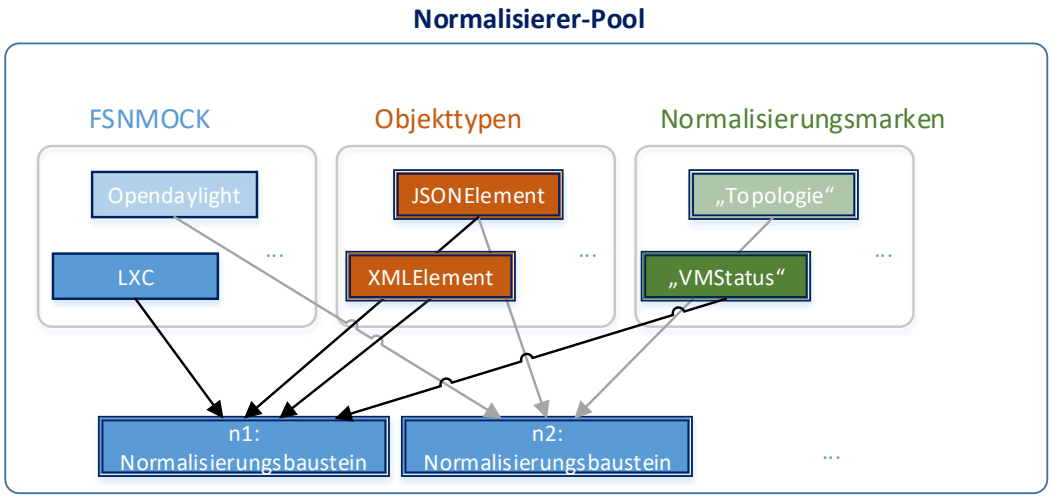


Abbildung 5.29: Funktionsweise des Normalisierer-Pools, illustriert an einem Beispiel. Die Verwaltung der Normalisierer wird durch die drei Kriterien der `FSNMOCK`, Objekttypen und Normalisierungsmarken begründet. Die Farben verdeutlichen die unterschiedlichen Auswahlkriterien.

Die Auswahl von Normalisierern wird durch den Normalisierer-Pool selbst umgesetzt, wie an einem Beispiel in Abbildung 5.29 gezeigt wird. Der Fokus in diesem Beispiel liegt dabei auf Normalisierer `n1`, welcher insbesondere für von LXC-Instanzen abgerufene XML- oder JSON-Elemente normalisieren kann. Eine Mehrfachzuweisung von Kriterien derselben Gruppe ist möglich und entsprechend der jeweiligen Implementierung eines Normalisierers zu vergeben: Beispielsweise können Normalisierer von Entwicklern für nur genau ein FSNMOCK und pro Datenformat erstellt werden, jedoch auch solche, die Daten unterschiedlicher FSNMOCK (z. B. verschiedene SDN-Controller) und für mehrere Datenformate normalisieren können. Auch ist im Beispiel die Normalisierungsmarke „VMStatus“ (hier z. B. als String) mit Normalisierer `n1` verknüpft.

Die benutzerdefinierten **Normalisierungsmarken** eignen sich insbesondere für die Markierung im Kontext von MO-Funktionsaufrufen (vgl. Abschnitt 5.4.3) und der Verknüpfung für die jeweilige Rückgabe geeigneter Normalisierer. Ein Beispiel wird in Abbildung 5.30 gezeigt: Bei Pull-Abruf von Topologiedaten von einem SDN-Controller wird genau ein geeigneter Normalisierer über eine entsprechende Normalisierungsmarke identifiziert. Aufseiten der von einem MO abgerufenen Informationen erfolgt das durch Einbringen bei Funktionsaufruf und Durchreichen der entsprechenden Marke vom Konnektorbaustein in das generierte `SBIErweiterteRohdaten`-Element.

Eine Zuweisung von Normalisierungsmarken ist jedoch auch anders einsetzbar: Die Managementplattform *probiert* selbst zu Funktionsaufrufen an einem MO mehrere Normalisierer aus, merkt sich diejenigen, die anwendbar sind (d. h. keinen Fehler generiert haben) und generiert entsprechende Normalisierungsmarken zur Verknüpfung. Die Managementplattform *lernt* in diesem Fall selbst (zumindest bei Pull-Aufrufen), welche Normalisierer zu welchen Rückgaben von Funktionsaufrufen anwendbar sind. Neuerstellte Normalisierer müssten dann jedoch ebenfalls ausprobiert und nachträglich markiert werden.

Für das in den letzten Absätzen beschriebene Auswahlverfahren auf Basis der Kriterien der Quell-MO-Klassifikation, des Datenformats und von Normalisierungsmarken, wird insofern das folgende **Vorgehen** festgelegt:

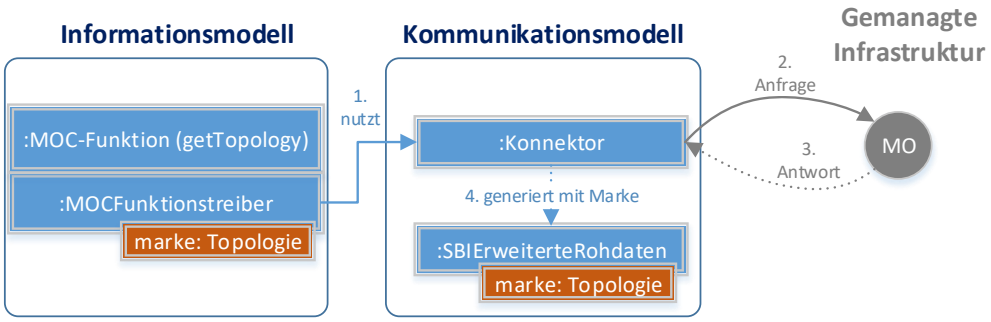


Abbildung 5.30: Vereinfache beispielhafte Darstellung der Weiterreichung einer Normalisierungsmarke (orange) von einer MOC-Funktionsdefinition an einen Konnektor (Pull-Zugriff) zur gemanagten Infrastruktur. Dieser generiert aus den empfangenen Daten eine `SBIErweiterteRohdaten`-Instanz mit der jeweiligen Normalisierungsmarke.

- Fall 1: Die Eingabe in Form einer `SBIDatenobjekt`-Instanz ist **nur durch MOCKs des Quell-MO** und das Datenformat klassifizierbar. Dieser Fall ist für ein gültiges `SBIDatenobjekt` immer gegeben. Hier wird mit allen Normalisierern, die für entsprechende für das Quell-MO klassifizierte FSNMOCKs sowie das entsprechende Datenformat geeignet sind, eine Normalisierung der Daten versucht: Fehlerhafte Anwendungen von Normalisierern lösen `NormalisierungsFehler` aus und werden verworfen. Das Ergebnis erfolgreicher Normalisierungen wird in Form eines `SBINetzinformation`-Objekts in die *Normalized-Queue* eingehängt.

- Fall 2: Das `SBIDatenobjekt` ist **zusätzlich durch Normalisierungsmarken** markiert. In diesem Fall wird die Grundmenge an passenden Normalisierern auf Basis zutreffender Normalisierungsmarken weiter eingeschränkt. Liefern alle eingeschränkten Normalisierer eine fehlerhafte Anwendung, kann auf die Grundmenge (wie Fall 1) beim Versuch der Normalisierung zurückgegriffen werden.

Dieses differenzierte Vorgehen erlaubt die Wiederverwendung von Normalisierungsmarken für unterschiedliche FSNMOCK: Beispielsweise kann die Normalisierungsmarke *Topologie* für mehrere SDN-Controller verwendet werden, die unterschiedliche Normalisierer benötigen, da zunächst die Grundmenge an passenden Normalisierern aus Informationen zur Quell-MO-Klassifikation (FSNMOCKs) gebildet und durch Normalisierungsmarken weiter verfeinert wird. Auf diese Weise kann die Menge an Normalisierungsmarken übersichtlicher gehalten werden. Die Aktualisierung des Speichermodells über die Komponente **Datenzugriffsverwaltung** wird in Abschnitt 5.6.3.2 erläutert.

5.6.1.3 SBI-Register

Das SBI-Register dient als zentrales Verwaltungselement im SBI. Es referenziert auf den Normalisierer-Pool, den Parser-Pool, und insbesondere auf einzelne Konnektor- und Schnittstellenbausteine (vgl. Abschnitt 5.6.4.2) innerhalb des zentralen Netzadapterpools, die im SBI verfügbar sind (vgl. Abbildung 5.27). Das SBI-Register dient mit dieser Struktur insbesondere zwei Zwecken:

- Es koordiniert den in Abbildung 5.27 gezeigten Prozess, verwaltet die drei gezeigten Queues und die Benachrichtigung der Worker. Die Benachrichtigung erfolgt ereignisgesteuert asynchron gemäß einem Producer-Consumer-Muster.

- Es koordiniert außerdem proaktive Abfragen (Pull) von Informationen aus der gemanagten Infrastruktur. In diesem Konzept sind derartige Abfragen und Zugriffe auf die ge-

managte IT-Infrastruktur über MOC-Funktionen (siehe Abschnitt 5.4.3) abstrahiert. Entsprechend stellt das SBI-Register einen geeigneten Konnektor für eine MOC-Funktion bereit.

Bei der Umsetzung des ersten Zwecks kann auf in der Informatik und Softwareentwicklung etablierte Konzepte zurückgegriffen werden. Der zweite Zweck bedarf hingegen einer Definition anwendungsspezifischer Funktionen des SBI-Registers:

- Die **Auswahl eines Konnektors** zur Umsetzung einer MOC-Funktion in der gemanagten IT-Infrastruktur. Dazu gehört, dass Netzadapter anhand ihres unterstützten Protokolls (z. B. HTTP, SSH, NETCONF, usw.) ausgewählt und an diese dann jeweils, im geeigneten Datenformat, Anfragen an gemanagte Netzkomponenten gesendet werden können. Dieser Aspekt wird in Abschnitt 5.6.4.2 geklärt.

- Die **Rückgabe** eines Ergebnisses einer Abfrage über einen Konnektor an die nutzende MOC-Funktion. Hier soll auswählbar sein, in welchem Grad der Vorverarbeitung die Daten zurückgegeben werden sollen. Eine MOC-Funktion kann daher auswählen, welcher in den vorherigen Abschnitten beschriebenen Datencontainer zurückgegeben werden.

- Eine **Bypass-Möglichkeit** bezüglich der Speicherung abgerufener Daten in der Datenbank: Der in Abbildung 5.27 gezeigte Fall, dass jedes von der gemanagten IT-Infrastruktur abgerufene/erhaltene Datum ebenfalls in der Datenbank der Managementplattform gespeichert wird, stellt hier den Normalfall dar. Die Speicherung ist jedoch nicht immer sinnvoll oder gewünscht, weshalb bei derartigen Pull-Zugriffen optional die Möglichkeit zur direkten und ausschließlichen Verarbeitung durch die aufrufende Komponente (z. B. eine Managementanwendung) und ohne Speicherung gegeben ist.

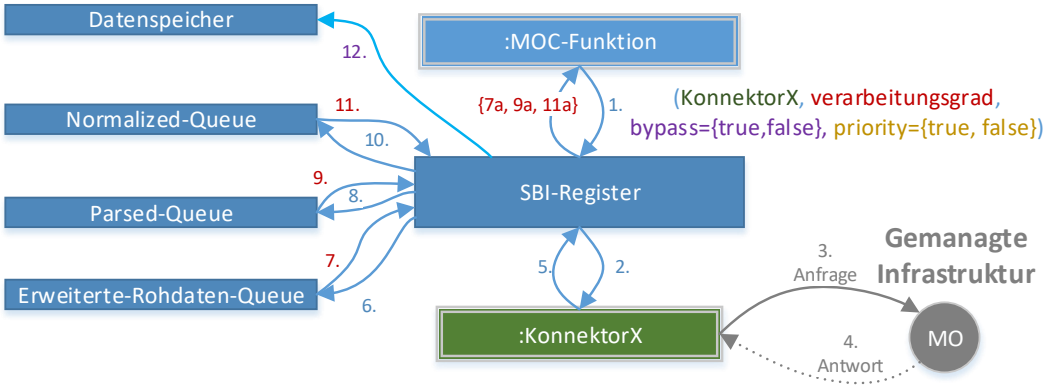


Abbildung 5.31: Illustration der Funktionen des SBI-Registers: Konnektor-Auswahl (grün), Auswahl des Verarbeitungsgrads der Rückgabe (rot), Bypass-Möglichkeit bei Speicherung (violett). Die Nummerierung der Übergänge zeigt die Bearbeitungsreihenfolge auf.

Abbildung 5.31 beschreibt die vom SBI-Register bereitgestellten Funktionen für Pull-Aufrufe. In Schritt 1 wird ein passender Konnektor nach Typ ausgewählt. Außerdem wird der Verarbeitungsgrad der zurückgegebenen Informationen des Aufrufs festgelegt und bestimmt, ob die bei dem Aufruf gesammelten Informationen nicht gespeichert, d. h. die Speicherung umgangen wird. Der Verarbeitungsgrad beschreibt, ob dem Aufrufer der MOC-Funktion das entsprechende `SBIErweiterteRohdaten`-, `SBIDatenobjekt`- oder `SBINetzinformation`-Element zurückgegeben werden, wodurch die Rückmeldung von SBI-Register zu MOC-Funktion im Schritt *7a* (erweiterte Rohdaten), *9a* (geparstes Datenobjekt) oder *11a* (normalisierte Netzinformation) stattfindet. Schritt 12 (Speicherung der gesammelten Daten) wird nur durchgeführt, sofern *bypass* nicht gesetzt ist. In diesem Fall kann die Verarbeitung der Anfrage nach Rückgabe des

jeweiligen Objekts gestoppt werden. Der in Abbildung 5.31 gezeigte Parameter *priority* beeinflusst, ob die dadurch abgerufenen Informationen (ab Schritt 6) priorisiert bearbeitet werden. Die Bevorzugung von Verarbeitungsschritten wird im folgenden Abschnitt näher behandelt.

5.6.1.4 Parallelisierung und priorisierte Bearbeitung

Zentrale Komponenten zur Parallelisierung des Prozesses der Aufbereitung von über das SBI abgerufenen Informationen aus der gemanagten IT-Infrastruktur sind Queues und Worker-Pools. Ein mehrstufiger Worker-Pool und ein Producer-Consumer-basierter Ansatz erlauben eine kontrollierte nebenläufige Überführung der Daten aus den beschriebenen Datencontainern. Wie in [192] beschrieben wird, kann ein *naiver* Ansatz durch die Erzeugung eines neuen eigenen Threads oder Prozesses pro am SBI eintreffenden Datenpakets erhebliche Performanzeinbußen mit sich bringen. Über Pools ist die **Anzahl der parallelen Prozesse** (im Konzept als *Worker* bezeichnet) immer statisch und eine konfigurierbare Stellschraube, die an die verfügbaren Betriebsressourcen (insb. CPU-Ressourcen) der Managementplattform für jeden der drei Worker-Pools für das Parsen, die Normalisierung und Aktualisierung angepasst werden kann.

Ein weiterer wichtiger Aspekt, der in diesem Konzept zu einer performanten Verarbeitung im SBI beiträgt, ist die **Priorisierung von Datencontainern** pro Verarbeitungsschritt. So soll besonders eine **Quasi-Echtzeitverarbeitung** wichtiger Informationen aus der gemanagten IT-Infrastruktur erreicht werden: Dedizierte Prioritäten-Queues – *Priority {Erweiterte-Rohdaten, Parsed, Normalized}-Queue* in Abbildung 5.27 – werden vorrangig abgearbeitet. Elemente der entsprechenden *unpriorisierten* Queues werden insofern erst abgearbeitet, wenn keine priorisierten Elemente der jeweils gleichen Ebene vorhanden sind.

Die **Priorisierung von Elementen** ist szenarioabhängig zu treffen. Um einerseits diese Flexibilität zu ermöglichen, andererseits einen einfach nutzbaren Ansatz zu bieten, sieht dieses Konzept dafür folgenden Ansatz vor:

- Aktive Pull-Abfragen über eine MOC-Funktion können individuell explizit im Rahmen des Parameters *priority* (vgl. Abbildung 5.31) angeben, ob abgerufene Informationen – gekapselt in einer `SBIErweiterteRohdaten`-Instanz – aus der gemanagten IT-Infrastruktur in die *Priority-Erweiterte-Rohdaten-Queue* eingereiht werden.

- Elemente, die in einem vorherigen Verarbeitungsschritt in einer Priority-Queue waren, werden auch im darauffolgenden Schritt in die entsprechende Priority-Queue eingereiht.

- Daneben ist ein frameworkbasierter Ansatz notwendig, sodass Entwickler bzw. Betreiber von Managementplattformen selbst **Regelsätze** zur Priorisierung von Datencontainern festlegen können. Dieser Ansatz wird im Folgenden beschrieben.

Eine Priorisierungsregel für Datencontainer ist über die Klasse `SBIPriorisierungsRegel` vorgegeben. Priorisierungsregeln können jedem Verarbeitungsschritt zugeordnet werden. Eine `SBIPriorisierungsRegel` ist eine abstrakte parametrisierbare Templateklasse, die, wie in Listing 5.15 gezeigt ist, eine Methode `priorisiere` besitzt.

```
1 abstract class SBIPriorisierungsRegel<A> {
2   public abstract boolean priorisiere(A paket) throws SBIPriorisierungsFehler;
3 }
```

Listing 5.15: Pseudocodedarstellung der Templateklasse `SBIPriorisierungsRegel`.

Die Methode `priorisiere` erwartet als Parameter ein Objekt der generischen Klasse `A` und wird je nach Verarbeitungsschritt parametrisiert, wie auch zuvor in Abbildung 5.27 gezeigt ist: Die grünen Elemente – der jeweils nächsten Queue vorgelagert – stellen pro Verarbeitungsschritt eine Menge an parametrisierten Priorisierungsregeln dar. Für jeden nicht bereits priori-

sierten Datencontainer wird durch Anwendung aller zutreffenden `SBIPriorisierungsRegel`-Instanzen geprüft, ob es in die folgende Priority-Queue eingereiht wird (Rückgabewert ist `true`), oder in der nicht-priorisierten Queue bleibt (Rückgabewert ist `false` oder bei fehlerhafter Verarbeitung). Eine zutreffende Priorisierungsregel reicht zur Priorisierung eines Datencontainers aus.

Die Implementierung der jeweiligen Priorisierungsregel durch Methode `priorisiere` obliegt dabei einem Entwickler, der die Entscheidung des Rückgabewertes der Funktion auf Basis der Auswertung der verfügbaren Felder des jeweils als Parameter übergebenen Datencontainers (z. B. Auf Basis von Quell-Informationen einer `SBIDatenobjekt`-Instanz) umsetzen kann. Eine Behandlung fehlerhafter Implementierungen von Parsern, Normalisierern und der Aktualisierung ist an dieser Stelle notwendig: Es muss verhindert werden, dass Worker in Dauerschleifen gefangen werden oder durch Methoden zur Blockierung des Threads diesen unbrauchbar machen. Daher sind für die Verarbeitung der Datencontainer sowie die Priorisierung **Time-Outs** sinnvoll, nach deren Ablaufen die jeweilige Bearbeitung als fehlerhaft betrachtet wird. Auch individuelle Fehlerbehandlung muss berücksichtigt werden.

5.6.2 Northbound-Interface und Inter-Plattform-Kommunikation

Das NBI ist für die Kommunikation mit Nutzern, Managementanwendungen oder auch anderen Managementplattform-Instanzen zuständig. Während die ersteren zwei Kommunikationszwecke bereits in den meisten bestehenden Arbeiten etabliert sind (z. B. auch im Software-Defined-Networking [193]), wird die Kommunikation mit anderen Managementplattform-Instanzen oft über sogenannte *West-* oder *Eastbound-Interfaces* umgesetzt, um den gleichwertigen Status der kommunizierenden Managementplattform-Instanzen anzudeuten. In diesem Konzept erfüllt diese Funktionalität jedoch auch das NBI, insbesondere da notwendige Komponenten und Abläufe wiederverwendbar sind und somit redundante Bausteine einer Managementplattform vermieden werden können.

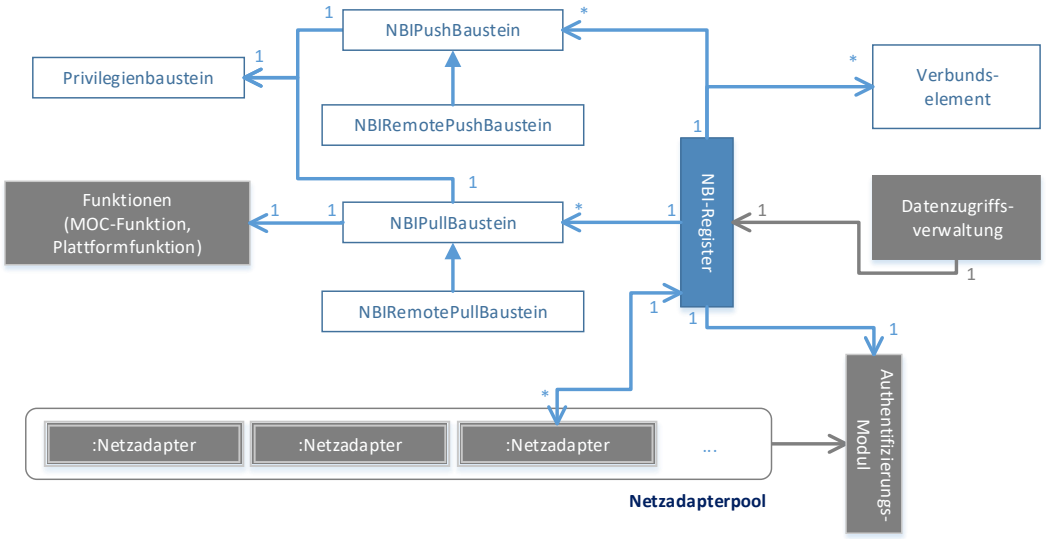


Abbildung 5.32: Ausschnitt wichtigster Komponenten des NBI und ihrer Zusammenhänge.

Abbildung 5.32 zeigt die Komponenten des NBI. Die Klassen `NBIPullBaustein`- und `NBIPushBaustein` implementieren Push- und Pull-Kommunikation und der damit verbundene `Privilegienbaustein` dient zur Unterscheidung von NBI-Elemente für die Kommunikation mit einerseits Managementanwendungen und andererseits anderen Managementplattformen in einem Verbund. Eine Managementplattform in einem Verbund wird über das Element

`Verbundselement` beschrieben. Das NBI-Register koordiniert schließlich die Kommunikation über das NBI.

In diesem Konzept werden zwei Dimensionen zur Erweiterung des NBI vorgesehen. Die erste wird durch den **Kommunikationsmechanismus** (Push oder Pull) beschrieben: Ein Pull-Zugriff entspricht der Ausführung einer Plattformfunktion der Managementplattform. Eine Push-Kommunikation entspricht hingegen einer Benachrichtigung. Die zweite wird durch die **Lokalität** (lokal oder remote) der Anbindung beschrieben: Der lokale Zugriff auf das NBI entspricht einer Erweiterung auf der Managementplattform über eine objektorientierte Programmierschnittstelle. Der remote Zugriff findet hingegen über eine Netzschnittstelle statt.

```
1  abstract class NBIPullBaustein {
2
3    Identifikator       fID;
4    String              fBeschreibung;
5    PrivilegienBaustein fPrivileg;
6    PlattformFunktion   fPlatFunktion;
7
8    // Behandelt lokalen Funktionsaufruf.
9    // Ergebnis wird an NBI-Register zurückgegeben.
10   abstract Tabelle<String, FunktionsWert> lokalerAufruf(Tabelle<String,
         FunktionsWert> params);
11
12   // Nur intern genutzt bei Aufruf, Beispielimplementierung:
13   // 1. Abgleich Zugreifbarkeit über Privilegienbaustein
14   // 2. Abgleich Nutzerzugriff über Authentifizierungsbaustein
15   // 3. Autorisierungs-Überprüfung via Zugriffsverwaltungsbaum aus dem
         Organisationsmodell
16   private abstract boolean kannZugreifen(Credentials c);
17
18   // Überladene Funktion ohne MO-ID
19   Tabelle<String, FunktionsWert> ausführen(Credentials c, Tabelle<String,
         FunktionsWert> params)
20   {
21     return ausführen(c, params, "");
22   }
23
24   // Überladene Funktion mit MO-ID
25   Tabelle<String, FunktionsWert> ausführen(Credentials c, Tabelle<String,
         FunktionsWert> params, Identifikator moid)
26   {
27     if kannZugreifen(c)
28     {
29       if (isValid(moid) and exists(moid))
30         params.add("__mo", getMOInstanceByID(moid))
31       return lokalerAufruf(params);
32     }
33   }
34 }
```

Listing 5.16: Pseudocodedarstellung der Klasse `NBIPullBaustein` zur Beschreibung lokaler Pull-Schnittstellen am NBI.

5.6.2.1 Lokale Plattformfunktionsaufrufe (Pull)

Der lokale Pull-Zugriff stellt in diesem Konzept gleichzeitig die Basis für den remoten Pull-Zugriff dar und wird über die Klasse `NBIPullBaustein` realisiert (vgl. Listing 5.16). Der Zweck eines `NBIPullBaustein` ist die Bereitstellung einer definierten **Plattform**- oder **MOC-Funktion** über das NBI, auf die er referenziert. Eine verknüpfte Instanz eines **Privilegienbausteins** erlaubt die Unterscheidung des Zugriffs für *a)* Nutzer und Managementanwendungen oder für *b)* Managementplattform-Instanzen (vgl. Abschnitt 5.6.2.7). Die **ID** des Bausteins muss eindeutig sein und erlaubt den späteren Zugriff auf diesen über das NBI-Register. Das Element `fBeschreibung` ist eine **Kurzbeschreibung** der Schnittstellenerweiterung. Die ab-

strakte Methode `lokalerAufruf` der Klasse `NBIPullBaustein` implementiert die Typ-Prüfung der Parameter und des Rückgabewertes sowie den eigentlichen Funktionsaufruf über den MOC-Funktionstreiber (vgl. Abschnitte 5.4.3.2 und 5.4.3.3).

Um eine neue Funktion über das NBI zugreifbar zu machen, wird die implementierte `NBIPull-Baustein`-Instanz am NBI-Register registriert, wie in Abschnitt 5.6.2.8 beschrieben ist. Die Methode `ausführen` wird durch das NBI-Register aufgerufen. Dabei handelt es sich um eine überladene Funktion, die einerseits mit, andererseits ohne MO-ID aufgerufen werden kann. Diese ID wird vom Aufrufenden (Nutzer oder Managementanwendung) übergeben und gibt an, auf welchem MO eine MOC-Funktion aufgerufen wird. Sie wird auf Validität überprüft und das entsprechende MO-Objekt anhand dessen zugreifbar gemacht. Das MO-Objekt wird als zusätzlicher benannter Parameter in die Parameterliste zum Funktionsaufruf der zu implementierenden Methode `lokalerAufruf` übergeben. Bei Plattformfunktionen wird entsprechend keine MO-ID verlangt, da sie nicht im Kontext eines MO ausgeführt werden.

Der lokale Aufruf einer mit einem NBI-Baustein verknüpften Plattformfunktion wird durch das NBI-Register wie folgt behandelt:

1. Die aufrufende lokale Managementanwendung wird in die Laufzeitumgebung der Managementplattform eingebunden und fragt den Zugriff auf die zentrale `NBIRegister`-Instanz an.

2. Die entsprechend auszuführende Funktion wird durch die Managementanwendung über die `fID` des NBI-Bausteins am NBI-Register identifiziert und angefragt.

3. Die Managementanwendung ruft die Methode `ausführen` der erhaltenen `NBIPullBaustein`-Instanz auf, durch welche zunächst die Möglichkeit zum Zugriff über die Methode `kannZugreifen` abgefragt wird. Die Methode `kannZugreifen` ist abstrakt, da ihre Implementierung von der Implementierung eines abstrakten `Credentials`-Objekts und dem Nutzermanagement abhängig ist. Das Element `Credentials` wird in Abschnitt 5.6.4.1 beschrieben.

4. Im positiven Fall wird die eigentliche MOC-Funktion über die Methode `lokalerAufruf` ausgeführt und ihr Ergebnis (je nach Ergebnistyp, vgl. Abschnitt 5.6.1.3) zurückgegeben.

5.6.2.2 Remote Plattformfunktionsaufrufe (Pull)

Das Zugänglichmachen einer Plattform- oder MOC-Funktion über eine Netzschnittstelle erfolgt ähnlich wie für eine lokale Schnittstelle. Die dafür vorgesehene und in Listing 5.17 beschriebene Klasse `NBIRemotePullBaustein` stellt eine Spezialisierung der Klasse `NBIPullBaustein` dar und erweitert sie um notwendige Funktionen für Netzschnittstellen:

- **Zuordnung** eines `NBIRemotePullBaustein`-Elements zu einem **Aufruf-Kontext** einer Netzschnittstelle. Der Kontext beschreibt ein spezifisches Element zur Unterscheidung von Ressourcen an einer Netzschnittstelle. Im für das NBI gebräuchliche HTTP-Protokoll wird dieser Kontext beispielsweise durch den jeweils angefragten *Uniform Resource Locator* (URL) gemeinsam mit der HTTP-Methode (z. B. *GET oder POST*) gebildet und identifiziert eine angefragte Funktion. Dagegen bei *Remote-Procedure-Call*-realisierenden Protokollen wie SOAP wird die gewünschte Funktion direkt durch einen eindeutigen Namen referenziert [194]. In diesem Konzept wird dieser Aspekt zum einen durch eine Zuweisung des Kontexts des Aufrufs zu einem `NBIRemotePullBaustein` im NBI-Register umgesetzt.

- **Deserialisierung** von über eine Netzschnittstelle bereitgestellten **Parametern** in das benannte Parameterformat `Tabelle<String, FunktionsWert>` durch die Methode

`deserialisiereParameter`. Bei der Deserialisierung von MOC-Funktionen muss darauf geachtet werden, dass der Wert der MO-ID zur Identifikation des MOs, auf dem die MOC-Funktion ausgeführt wird, ebenfalls deserialisiert und in die überladene Methode `ausführen` für Parameter *moid* übergeben wird (vgl. letzter Abschnitt).

- **Serialisierung** des **Rückgabewertes** in eine Zeichenkette aus dem gegebenen Format `Tabelle<String, FunktionsWert>`, die als Antwort an die Netzschnittstelle zurückgegeben und an den anfragenden externen Dienst geschickt werden kann.

```
1  abstract class NBIRemotePullBaustein<T> extends NBIPullBaustein {
2
3    Konnektor fWeiterleitungKonnektor;
4
5    // Funktion zur Deserialisierung von per remotem Funktionsaufruf
6    // übergebenen Parametern.
7    // Params enthält ebenfalls die MO-ID der FSNMOC-Instanz, falls notwendig
8    abstract Tabelle<String, FunktionsWert> deserialisiereParameter(
9                   String params);
10
11   // Funktion zur Serialisierung der Rückgabe von Funktion remoteCall.
12   // Ergebnis dient als Rückgabe des jew. remoten Funktionsaufrufs.
13   abstract String serialisiereRückgabe(Tabelle<String, FunktionsWert> res);
14
15   // Dient der Inter-Plattform-Kommunikation
16   abstract String weiterleiten(Credentials c, String params);
17 }
```

Listing 5.17: Pseudocodedarstellung der Klasse `NBIRemotePullBaustein` als Erweiterung der Klasse `NBIPullBaustein`.

Die Aufrufbehandlung einer über die remote NBI-Schnittstelle bereitgestellte Plattform- oder MOC-Funktion läuft durch die folgende Schritte ab:

1. Eine externe Managementanwendung oder eine andere Managementplattform schickt eine Anfrage über einen Funktionsaufruf an eine mit dem NBI verknüpfte Schnittstelle.

2. Die serialisierten Daten (insb. Parameter) der Anfrage sowie der Kontext werden an das NBI-Register weitergeleitet, welches die auf den Anfragekontext registrierte `NBIRemotePullBaustein`-Instanz identifiziert. Datenstrukturen zum Filtern geeigneter Bausteine werden in Abschnitt 5.6.2.8 beschrieben.

3. Das NBI-Register deserialisiert die von der Netzschnittstelle erhaltenen Parameter über die Methode `deserialisiereParameter` des entsprechenden Bausteins und speichert die deserialisierten Plattformfunktionsparameter zwischen.

4. Analog zum lokalen Aufruf wird die geerbte Methode `ausführen` mit den deserialisierten Parametern aufgerufen. Sie führt eine Zugriffsüberprüfung durch und führt im positiven Fall die entsprechende Plattformfunktion aus.

5. Die Rückgabe wird über die Methode `serialisiereRückgabe` serialisiert und über die zur Anfrage genutzte Schnittstelle wieder zurückgegeben.

Da eine `NBIRemotePullBaustein`-Instanz jedoch ebenfalls gleichzeitig alle Funktionen der Elternklasse `NBIPullBaustein` implementiert, kann sie ebenso durch einen lokalen Zugriff ausgeführt werden. Die in Listing 5.17 ebenfalls gezeigte Feldvariable `fWeiterleitungKonnektor` beschreibt einen Konnektor, der zur Weiterleitung des Aufrufs im Managementplattform-Verbund genutzt werden soll und beim Anlegen der `NBIRemotePushBaustein`-Instanz gemeinsam mit der Methode `weiterleiten` implementiert werden muss. Die Methode gibt die dann durch die entsprechend ausführende Managementplattform zurückgegebene serialisier-

te Rückgabe der Funktion zurück. Die Weiterleitung im Managementplattform-Verbund wird in Abschnitt 5.6.2.6 näher beschrieben.

5.6.2.3 Benachrichtigungen (Push) an lokale Anwendungen

Ein Publish-Subscribe-Ansatz (vgl. [195]) für Push-Benachrichtigungen ist in bereits bestehenden Arbeiten und Plattformen im Kontext softwarebasierter Netze weit verbreitet. Verschiedene Arbeiten beschreiben ihn als effiziente Möglichkeit zur Überwachung von NFV-Umgebungen (z. B. [196]), verteilten SDN-Umgebungen (siehe [197]) und auch Bestandteil vieler bestehender Plattformen (vgl. Übersicht in Abschnitt 4.1).

In diesem Konzept beschreibt ein `NBIPushBaustein` die Erweiterung des NBI in Form einer **lokalen Push**-Funktionalität. Die wichtigsten Elemente der Klasse `NBIPushBaustein` sind in Listing 5.18 beschrieben. Sie hat eine **ID** zwecks einer eindeutigen Adressierbarkeit sowie einen damit verbundenen **Privilegienbaustein** zur Festlegung der Nutzbarkeit für Nutzer, Managementanwendungen oder Managementplattform-Instanzen. Push-spezifisch ist ein **Kriterium zur Filterung von Inhalten** sowie die Priorität der Bearbeitung. Das **Kriterium** wird durch eine oder mehrere abstrakte zu implementierende `NBIPushKriterium`-Instanzen beschrieben. Darin wird als Spezifikator der Elementtyp von Informationselementen (hier als generischer Typ `T`) beschrieben, z. B. für eine `FSNMOC`-Instanz oder eine `Abhängigkeit`- und `Netzinformation`-Instanz. Über die abstrakte Methode `istRelevant` können zudem weitere Kriterien für den durch `T` beschriebenen Elementtyp implementiert werden. Die Funktionalität dafür liegt in der Datenzugriffsverwaltung: Sie registriert Änderungen im Speicher und wendet auf Elementen des Typs `T` die Methode `istRelevant` für den Typ jeweils passender am NBI-Register vorhandener `NBIPushBaustein`-Instanzen an. Die **Priorität** einer `NBIPushBaustein`-Instanz erlaubt ihre bevorzugte Bearbeitung und somit eine Quasi-Echtzeitverarbeitung kritischer Informationen auf Kosten einer Verzögerung niedrigpriorisierter Push-Benachrichtigungen. Die Prioritätenstufen kann anwendungsfallspezifisch festgelegt werden: Beispielsweise durch drei Stufen *normal*, *hoch*, *höchst* wie in Listing 5.18 beispielhaft umgesetzt ist.

```
1  abstract class NBIPushBaustein {
2
3    Identifikator          fID;
4    String                 fBeschreibung;
5    PrivilegienBaustein    fPrivileg;
6    Set<NBIPushKriterium>  fKriterium;
7    Priorität              fPriorität;
8  }
9
10 abstract class NBIPushKriterium<T> {
11   Class<T>  fTyp;
12
13   // Wenn Resultat true: Das Element erfüllt Kriterium,
14   // andernfalls false.
15   abstract boolean istRelevant(T element);
16 }
17
18 // Beispielhafte Belegung der Prioritäten
19 enum Priorität {
20   normal, hoch, höchst;
21 }
```

Listing 5.18: Pseudocodedarstellung der Klassen `NBIPushBaustein`, `NBIPushKriterium` und `Priorität` zur Beschreibung lokaler Push-Schnittstellen am NBI.

Eine Push-Benachrichtigung erfordert die Angabe eines Endpunkts und im lokalen Kontext einer geteilten Laufzeitumgebung entsprechend einer **Callback-Funktion**. Diese wird in diesem Konzept dynamisch über das NBI-Register verwaltet, wie in Abschnitt 5.6.2.8 beschrieben wird.

Eine Callback-Funktion für Push-Nachrichten wird durch die in Listing 5.19 gezeigte Klasse `NBICallbackFunktion` beschrieben. Darin sind **Zugangsdaten** (Klasse `Credentials`, aus Abschnitt 5.6.4.1) eines Nutzers der Managementplattform zur Authentisierung des Zugriffs am NBI hinterlegt. Die `Credentials` müssen mindestens den Nutzer der Managementplattform identifizieren, über den die Push-Benachrichtigungen angefordert werden, sowie Informationen zur Authentifizierung angeben. Informationen in Benachrichtigungen werden über den Zugriffsverwaltungsbaum autorisiert (vgl. Abschnitt 5.5.5.2).

```
1 abstract class NBICallbackFunktion<T> {
2
3   // Zur Authentisierung eines Nutzers an der Managementplattform
4   Credentials   fPlattformCredentials;
5
6   // T entspricht Typ des Speicherelements
7   abstract void callback(T element);
8 }
```

Listing 5.19: Pseudocodedarstellung der Klasse `NBICallbackFunktion` als Endpunkt für Push-Benachrichtigungen vom NBI.

Der Ablauf zur **Registrierung und dem Abonnement** einer neuen lokalen Push-Benachrichtigung am NBI ist wie folgt:

1. **Registrierung** einer NBI-Push-Benachrichtigung durch Implementierung einer `NBIPushBaustein`-Instanz und ihrer Registrierung am NBI-Register. Dabei muss vor allem auf die Wahl der maximal notwendigen Priorität geachtet werden, sowie auf entsprechende Privilegieneinschränkung. Außerdem muss darauf geachtet werden, dass keine äquivalente `NBIPushBaustein`-Instanz existiert, an der ein Abonnement bereits möglich und welche dann zu bevorzugen ist.

2. Das **Abonnement** einer NBI-Push-Benachrichtigung wird lokal durch Registrierung einer entsprechenden `NBICallbackFunktion`-Instanz am NBI-Register durchgeführt.

Eine **Abmeldung** von einem Abonnement erfolgt durch das Entfernen der entsprechenden `NBICallbackFunktion`-Instanz. Die Abmeldung einer NBI-Push-Benachrichtigung erfolgt dagegen durch Entfernen der entsprechenden `NBIPushBaustein`-Instanz.

Der Ablauf der **Ausführung** einer lokalen Push-Benachrichtigung unter Annahme des Vorhandenseins einer passenden `NBIPushBaustein`-Instanz läuft wie folgt ab:

1. Die Änderung eines Speicherelements vom (hier generischen) Typ `T` wird an der Datenzugriffsverwaltung erkannt (vgl. Abschnitt 5.6.3.3).

2. Im NBI-Register werden dann schrittweise alle höchst- dann hoch- und dann normalpriorisierten `NBIPushBaustein`-Instanzen traversiert und zutreffenden Typen – die Klasse von `fTyp` einer `NBIPushBaustein` passt zu Typ `T` – die Methode `istRelevant` ausgeführt.

3. Bei allen `NBICallbackFunktion`-Instanzen die mit `NBIPushBaustein`-Instanzen verknüpft sind, deren `istRelevant`-Methode *true* zurückgeben, wird überprüft, ob das entsprechende Speicherelement und alle damit direkt verknüpften Elemente (z. B. MOC-Attribute, Netzinformationen, usw.) vom im `Credentials`-Objekt angegebenen Nutzer zugegriffen werden kann. Die Überprüfung wird mithilfe des Zugriffsverwaltungsbaums, wie in Abschnitt 5.5.5.2 beschrieben, durchgeführt und die so ermittelte zum Zugriff erlaubte Teilmenge der Speicherelemente über die entsprechende `callback`-Funktion zurückgegeben.

5.6.2.4 Benachrichtigungen (Push) an remote Anwendungen

Der Schritt von lokalen Push-Benachrichtigungen zu remoten Push-Benachrichtigungen ist weitestgehend analog zum Schritt von lokalen zu remoten Pull-Zugriffen am NBI. Unterschiede liegen jedoch darin, dass für eine remote Push-Benachrichtigung keine Parameter verarbeitet werden und daher auch deren Deserialisierung wegfällt. Des Weiteren hängt, wie auch bei lokalen Push-Benachrichtigungen, die Authentifizierung und Autorisierung am jeweiligen Abonnement selbst. Eine remote Push-Benachrichtigung zu Änderungen von Speicherelementen wird durch die Klasse `NBIRemotePushBaustein` (siehe Listing 5.20) registriert. Die Klasse erbt alle Methoden und Feldvariablen der Klasse `NBIPushBaustein` und wird gleichartig behandelt. Eine abstrakte Methode `serialisiereNachricht` generiert eine serialisierte Form des Benachrichtigungsinhalts `element`. Auch eine zentrale Serialisierung von Elementen wird durch die Methode nicht ausgeschlossen.

```
1  abstract class NBIRemotePushBaustein extends NBIPushBaustein {
2
3    // Methode zur Serialisierung eines Elements
4    // der generischen Klasse T
5    abstract <T> String serialisiereNachricht(T element);
6  }
```

Listing 5.20: Pseudocodedarstellung der Klasse `NBIRemotePushBaustein` zur Beschreibung remoter Push-Schnittstellen am NBI.

Da im remoten Fall die Push-Benachrichtigung über Netzschnittstellen erfolgt, wird der Empfängerendpunkt statt der Klasse `NBICallbackFunktion` durch eine Klasse `NBIRemotePush-Endpunkt` beschrieben. Wie in Listing 5.21 gezeigt ist, besteht der wesentliche Unterschied in der Ersetzung der eigentlichen Callback-Funktion `callback` durch eine für Netzschnittstellen angepasste Methode `senden`. Diese erwartet als Parameter nicht nur das eigentliche Element in bereits serialisierter Form (durch die zuvor erwähnte Methode `serialisiereNachricht`), sondern ebenfalls eine generische Konnektorinstanz zum Versand der Informationen. Da remote Benachrichtigungen durch das NBI-Register koordiniert werden, wird eine passende Konnektorinstanz durch ebendieses wie in Abschnitt 5.6.4.2 ausgewählt. Dedizierte Zugangsdaten zur Authentisierung an einem zugriffsgeschützten remoten Empfangsendpunkt für Benachrichtigungen (z. B. einem HTTPS-Dienst mit Basic-Authentication) sind nicht fix vorgesehen, sondern müssen je nach Anwendungsfall implementiert werden. Die Freiheit der Implementierung an dieser Stelle erlaubt daher auch nicht nur das Senden an einen Endpunkt, sondern an potenziell beliebig viele.

```
1  // K ist die jeweilige Konnektorklasse
2  abstract class NBIRemotePushEndpunkt<K> {
3
4    // Nutzerauthentisierung an Managementplattform
5    Credentials fPlattformCredentials;
6
7    // Ein passender Konnektor vom Typ K wird
8    // durch das NBI-Register übergeben.
9    abstract void senden(String element, K konnektor);
10 }
```

Listing 5.21: Pseudocodedarstellung der Klasse `NBIRemotePushEndpunkt` zur Beschreibung einer remoten Datensenke für Push-Benachrichtigungen.

Die Organisation der `NBIRemotePushBaustein`-Instanzen ist analog zu lokalen Pendants gemäß ihrer Priorität (vgl. Abschnitt 5.6.2.8). Anders als ein lokaler Push-Baustein referenziert jede `NBIRemotePushBaustein`-Instanz auf eine Liste von `NBIRemotePushEndpunkt`-Instanzen, die die Endpunkte der Senken (z. B. Managementanwendungen oder Managementplattform-Instanzen) darstellen.

Die **Registrierung** einer remoten Push-Benachrichtigung erfolgt analog zu einer lokalen Variante durch Instanziierung der Klasse `NBIRemotePushBaustein` und ihrer Registrierung am NBI-Register. Auch die Ablaufbeschreibung für ein **Abonnement** remoter Push-Benachrichtigungen ist ähnlich wie bei lokalen Push-Benachrichtigungen. Statt der direkten Übergabe des jeweiligen Elements an eine Callback-Funktion kommt hier jedoch zunächst der Schritt der Serialisierung des Elements hinzu, sowie das darauffolgende Senden der serialisierten Form über den in der `NBIRemotePushBaustein`-Instanz angefragten Konnektor.

5.6.2.5 Beschreibung eines Managementplattformverbunds

In diesem Konzept wird ein Managementplattform-Einsatz im dezentralisierten Verbund und ohne übergeordnete zentrale Instanz verfolgt, um in geographisch verteilten Netzen mit mehreren Datenzentren ausreichend Skalierbarkeit zu bieten und einen *Single-Point-of-Failure* zu vermeiden. Die Architektur wurde bereits im FEDCON-Framework in [181] beschrieben und ist in Abbildung 5.33 in den Gesamtkontext dieser Arbeit gesetzt.

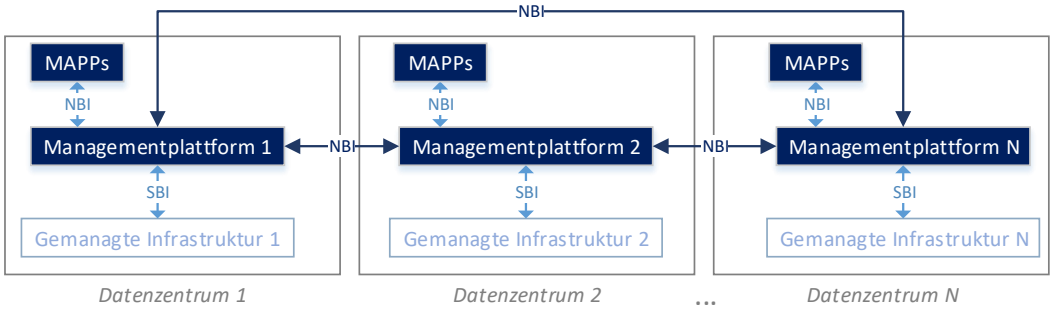


Abbildung 5.33: Architektur eines Managementplattform-Verbunds, basierend auf [181]. Managementplattformen kommunizieren darin über ihr jeweiliges NBI, über das auch Managementanwendungen (MAPPs) innerhalb der Datenzentren an Managementplattformen angebunden sind.

Da IT-Ressourcen datenzentrumsübergreifend gemanagt werden, sind Zugriffe von Nutzern oder Managementanwendungen über das NBI einer für ein *Datenzentrum 1* zuständigen *Managementplattform 1* auf Speicherelemente und Plattformfunktionen einer *Managementplattform 2* in *Datenzentrum 2* notwendig. Diese Nutzeroperationen im Plattformverbund unterscheiden sich jedoch von der ebenfalls betrachteten Inter-Managementplattform-Kommunikation, welche in Abschnitt 5.6.2.7 behandelt wird. Letztere kann jedoch die Basisfunktionalität für die hier betrachteten Nutzeroperationen stellen, wie im Folgenden erläutert wird. In [181] wird die Modellierung des Verbunds als Gesamtes beschrieben: Jede Managementplattform-Instanz im Verbund besitzt ein zentrales `ClusterEnvironmentModel`, das eine föderationsweit eindeutige ID eines Datenzentrums auf die jeweiligen Zugriffsendpunkte in Form je einer IP-Adresse und eines Ports abbildet. Die Architektur der Publikation erwartet entsprechend, dass ein Datenzentrum in einem FSN von genau einer Managementplattform bedient wird. Regeln zur Unterteilung eines FSN in Datenzentren sind nicht vorgegeben, sodass Ressourcen mit demselben Standort nach Bedarf in mehrere **logische Datenzentren** unterteilt werden können.

In diesem Konzept wird diese Idee weiter generalisiert, sodass weiterhin die Einteilung des FSN durch Datenzentren nicht vorgegeben ist, jedoch darüber hinaus mehrere Datenzentren auch von einer Managementplattform verwaltet werden können. Die Architektur ist somit flexibler gestaltet. In einem Managementplattformverbund muss jede Managementplattform die jeweils anderen Managementplattformen als Modell halten. In diesem Konzept wird eine Managementplattform im Verbund durch die Klasse `Verbundselement` beschrieben. Sie wird durch eine föderationsweit eindeutige **Managementplattform-ID**, einen **Dienstzugriffsendpunkt** sowie einer Menge an durch die Plattform gemanagte **Datenzentren** beschrieben. In

diesem Konzept wird die Umsetzung der Managementplattform-ID sowie des Dienstzugriffsendpunkts nicht konkret vorgegeben. Die Entscheidung für Letzteres begründet sich darin, dass die Umsetzung der Kommunikationstechnologie so nicht eingeschränkt wird: Beispielsweise ist sie so unmittelbar und ohne Zwischendienste, jedoch auch wie in ONOS über einen dedizierten Dienst [198] realisierbar.

5.6.2.6 Kommunikation im Managementplattform-Verbund

Für Push- und Pull-Zugriffe im Plattform-Verbund werden in diesem Framework drei jeweils valide Varianten der Unterstützung vorgesehen, die auch vom Speichermodell abhängen:

1. **Variante A: Zustands-asynchrone Plattformen**: In diesem Fall verwaltet jede Managementplattform ausschließlich Speicherelemente des von ihr gemanagten Datenzentrums – der Speicherzustand wird nicht über Managementplattformen synchronisiert. In diesem Fall werden alle Anfragen, die über den Verwaltungsbereich einer Managementplattform hinausgehen, im Rahmen des Plattformverbunds an die jeweils zuständige Managementplattform weitergeleitet. Im Verbund müssen auch Push-Benachrichtigungen und Pull-Zugriffe weitergeleitet werden.

2. **Variante B: Zustands-synchrone Plattformen**: Im Verbund wird der Speicherzustand von Managementplattformen untereinander synchron gehalten. In diesem Fall muss die Synchronisation des Speicherzustands über das NBI durch Plattformfunktionen implementiert werden (hier bieten sich insbesondere remote Push-Benachrichtigungen an) sowie die Weiterleitung von MOC-Funktionen (direkter Zugriff auf die gemanagte IT-Infrastruktur, Pull). Push-Benachrichtigungen können durch jede Plattform gleichermaßen Datenzentrums-lokal behandelt werden – durch den Synchronisations-Overhead jedoch u. U. mit geringerer Performanz. Dieser Ansatz wird beispielsweise auch von OpenDaylight zentral durch einen *Leader* im Verbund verfolgt [199].

3. **Variante C: Gemeinsame Speicherbasis**: Die Speicherbasis und der Zugriff darauf ist für den Plattform-Verbund zentral organisiert – der Zugriff ist einheitlich. In diesem Fall muss lediglich die Weiterleitung des direkten Zugriffs auf die IT-Infrastruktur (MOC-Funktionen) implementiert werden. Push-Benachrichtigungen können wie bei Variante B gehandhabt werden.

Die Entscheidung für die jeweilige Variante ist anwendungsfallspezifisch zu treffen. Generell kann jedoch angenommen werden, dass die Effizienz der Ressourcennutzung hinsichtlich Netzverkehr, Speicherzugriffszeit und Rechenzeit, gleichzeitig aber auch der Aufwand einer Implementierung bei Variante A am höchsten ist und beides hin zu Variante C sinkt. Unter Verwendung bestehender dezentraler Datenbanksysteme kann beispielsweise Variante B einfacher implementiert werden als Variante A. Unter Verwendung eines zentralen Datenbanksystems in Variante C fällt die Speichersynchronisation weg. Bei Varianten besteht aber die Notwendigkeit zur Implementierung von **Pull-Zugriffen** im Verbund. Ein Konzept dazu wurde bereits im FEDCON-Framework [181] entwickelt. Die verfolgte Lösung funktioniert unter der Annahme, dass jede **Managementplattform-Instanz denselben Funktionsumfang** im Verbund hat, d. h. die gleichen MOC- und Plattformfunktionen über ihr jeweiliges NBI bereitstellt. Ein heterogener Funktionsumfang im Verbund ist zum einen für die in dieser Arbeit zugrundeliegenden Szenarien (und vielen anderen) nicht sinnvoll, zum anderen erheblich komplizierter, wodurch die Verwendbarkeit der Frameworks in der Praxis eingeschränkt wird.

Variante A bietet sich in FSNs an, deren Koordinierung weitestgehend dezentral ist. Ein Beispiel ist das Szenario 1 aus Abschnitt 3.3 dieser Arbeit. In dieser Variante werden die Managementinformationen zudem vorwiegend im jeweiligen Datenzentrum gespeichert, sodass Datenschutzaspekte einfacher umsetzbar sein. Variante C bietet sich dagegen in FSNs mit überwiegend zentraler Koordination wie dem Szenario 2 (siehe Abschnitt 3.4) dieser Arbeit an. Die

Daten werden dann zentral gespeichert und sind für ihren Zweck am einfachsten zugreifbar. Variante B bietet sich wiederum in Szenarien an, in denen bestimmte Managementinformationen im FSN überall zugreifbar sein sollen, jedoch auch geregelt ausgetauscht werden können. Datenschutzaspekte können auch hier einfacher berücksichtigt werden.

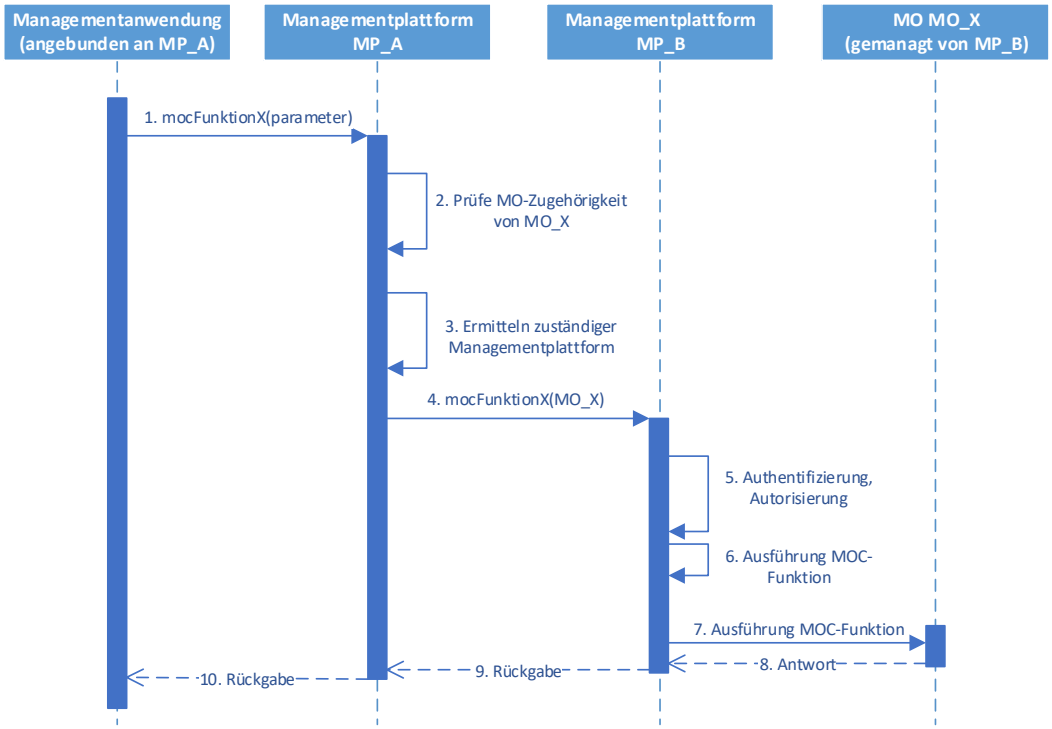
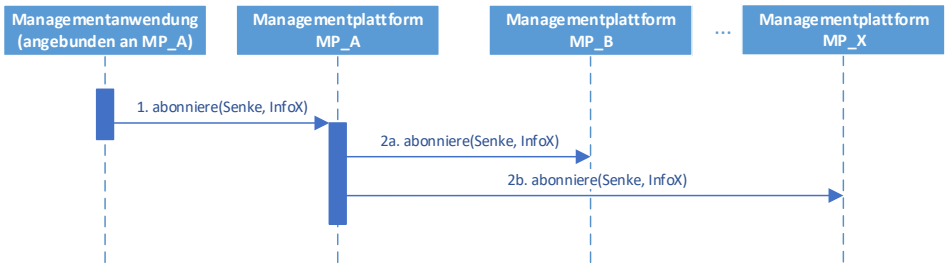


Abbildung 5.34: Sequenzdiagramm zur Illustration eines remoten Pull-Zugriffs im Managementplattform-Verbund. Managementplattform MP_A leitet die Anfrage an die zuständige Managementplattform MP_B weiter und gibt die Rückgabe an die anfragende Managementanwendung zurück.

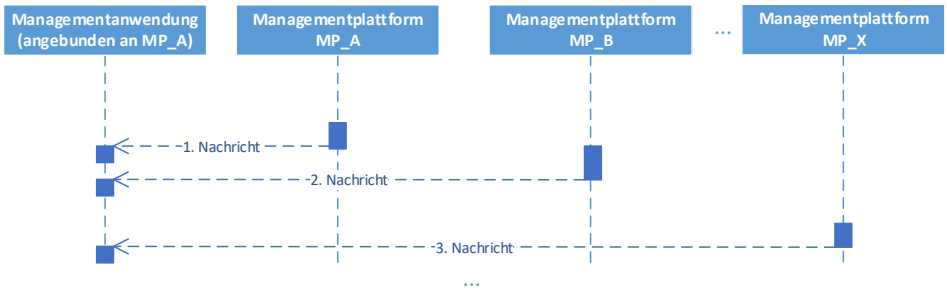
Das Ablaufdiagramm für remote Pull-Zugriffe im Managementplattform-Verbund ist beispielhaft in Abbildung 5.34 gezeigt. Führt darin eine an Managementplattform *MP_A* angebundene Managementanwendung die Methode `mocFunktionX` *(1)* mit Angabe der MO-ID *MO_X* aus, wird auf MP_A über Elemente des Frameworks zur Föderationsbeschreibung geprüft, auf welchem Datenzentrum MO_X betrieben wird *(2)*. Falls MP_A nicht selbst für das Datenzentrum, in dem MO_X betrieben wird, zuständig ist, ermittelt sie die zuständige Managementplattform – in diesem Fall MP_B *(3)*. Managementplattform MP_A ruft daraufhin die Methode `mocFunktionX` über das NBI von `MP_B` *(4)* auf, welche dieselbe Funktion wie eine gewöhnliche Pull-Anfrage am NBI ausführt *(5-7)* und das Ergebnis an MP_A zurückgibt. Diese gibt das Ergebnis wiederum an die ursprünglich anfragende Managementanwendung zurück *(8-10)*. Die Implementierung der Weiterleitung remoter Pull-Zugriffe im Verbund wird direkt durch die Methode `weiterleiten` im entsprechenden `NBIRemotePullBaustein` (vgl. Abschnitt 5.6.2.2) realisiert. Zu beachten ist, dass der hier beschriebene Ablauf nur für von **remote zugängliche Funktionen** des NBI durchgeführt werden kann, da diese die notwendige Funktionalität zur Deserialisierung von Parametern und Serialisierung von Rückgabewerten bieten. Lokale Pull-Zugriffe am NBI werden daher gewollt nicht unterstützt, um die ausschließlich lokal vorgesehene Nutzung der NBI-Funktionen nicht zu verletzen.

Die Umsetzung von **Push-Benachrichtigungen ohne gemeinsame Speicherbasis** (insb. für Variante A) kann im Plattform-Verbund über Managementplattformen als Relaisstati-

on realisiert werden. Dabei wird eine Push-Benachrichtigung bei der eigentlichen Quell-Managementplattform installiert, dann jedoch nicht von dem anfragenden Nutzer bzw. der anfragenden Managementanwendungen direkt abonniert, sondern von der Relais-Managementplattform.



(a) Abonnement von Push-Benachrichtigungen im Verbund und ohne gemeinsamen oder synchronisierten Datenspeicher.



(b) Push-Benachrichtigungen im Verbund und ohne gemeinsamen oder synchronisierten Datenspeicher.

Abbildung 5.35: Sequenzdiagramm zur Verdeutlichung des Ablaufs des Abonnements von Push-Benachrichtigungen und die Weiterleitung von Push-Benachrichtigungen im Verbund.

Bei vollständig verteilter Speicherbasis im Managementplattform-Verbund erfasst jede Managementplattform darin ausschließlich den Zustand der von ihr jeweils gemanagten IT-Infrastruktur, wodurch allgemein keine Duplikate entstehen. Unter der Voraussetzung, dass alle Managementplattformen denselben Satz an Push-Benachrichtigungen unterstützen, kann das Abonnement remoter Push-Benachrichtigungen durch eine Managementanwendung auf allen Managementplattformen homogen verteilt werden. Abbildung 5.35 zeigt diesen Vorgang zum einen für das Abonnement, zum anderen für den Versand remoter Push-Benachrichtigungen.

Das **Abonnement im Managementplattform-Verbund** (vgl. Abbildung 5.35a) kann vom Nutzer dabei zentral an einer Managementplattform vorgenommen werden *(1)*. Auf diese erste Eintragung an der Managementplattform MP_A folgt die Propagierung des Abonnements auf alle anderen Managementplattformen MP_B bis MP_X *(2)* auf Basis der in Abschnitt 5.6.2.5 beschriebenen Verbundselemente. Die `Senke` beschreibt dabei einen Dienst-Endpunkt der jeweiligen Managementanwendungen, an die die Push-Benachrichtigung versendet wird. Die Kündigung des Abonnements wird in gleicher Weise durchgeführt.

Der **Versand von Push-Benachrichtigungen** ist in Abbildung 5.35b demonstriert. Darin schicken die jeweiligen Managementplattformen im Verbund asynchron unabhängig voneinander und sobald ein Speicherelement die Kriterien in der jeweiligen gemanagten Umgebung erfüllt, an die zuvor für die Benachrichtigung registrierte Senke bzw. Managementplattform.

5.6.2.7 Privilegienebenen des NBI-Zugriffs

Die Koordination im Managementplattform-Verbund wird über das NBI durchgeführt. Jedoch werden auch Nutzeroperationen unter Umständen über den Managementplattform-Verbund koordiniert, wie beispielsweise in Abschnitt 5.6.2.6 beschrieben ist. Ein anderer Anwendungsfall ist die Anbindung von externen Diensten für das Netzmanagement, beispielsweise **Verzeichnisdienste** zur Nutzerorganisation oder **Ticketsysteme** zur Unterstützung der Organisation.

Operationen im Managementplattform-Verbund werden in diesem Konzept zunächst wie Plattformfunktionen (vgl. Abschnitt 5.7.2.1) oder auch MOC-Funktionen (definiert in Abschnitt 5.4.3) als privilegiert nutzbare Operationen auf der gemanagten IT-Infrastruktur realisiert. Der Zweck einer Funktion des NBI wird rein logisch durch **Privilegienbausteine** getrennt. Die dadurch beschriebenen Privilegien sind als Hierarchie konzipiert und der herkömmlichen Autorisierung bzw. Sichtbarkeitsprüfung (durch den Zugriffsverwaltungsbaum, vgl. Abschnitt 5.5.5) weiter einschränkend vorgelagert.

Ein Beispiel einer Ableitung von Privilegienstufen auf Basis objektorientierter Prinzipien ist in Abbildung 5.36 gezeigt. In diesem Beispiel stellt die Klasse `PrivilegNormal` eine Klassifikation für *übliche* am NBI bereitgestellte Operationen für Nutzer und Managementplattformen bereit. Die Klasse `PrivilegHoch` umfasst dabei aufgrund der Ableitung von `PrivilegNormal` alle Funktionen dieses Privilegs, sowie weitere, die ausschließlich ihr selbst zugewiesen sind. Auch ist eine feingranularere Abstufung möglich, beispielsweise mit drei oder mehr Privilegienebenen. Dies ist beispielsweise zur Unterscheidung von Funktionen für den Managementplattform-Verbund (z. B. mit höchsten Privilegien), Funktionen für Managementanwendungen (z. B. mittlere Privilegien) und für externe Dienste wie einem Verzeichnisdienst (z. B. niedrige Privilegien) mit sehr spezifischem Nutzungsumfang von Managementfunktionen denkbar.

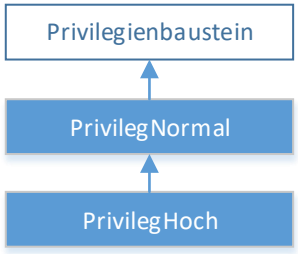


Abbildung 5.36: Beispielumsetzung von Privilegien durch die abstrakte Klasse `PrivilegienBaustein`.

Die Zuweisung von Push- und Pull-Operationen des NBI wurde in den vorherigen Abschnitten beschrieben. Demnach wird jedem `NBIPullBaustein` und jedem `NBIPushBaustein` – für lokale Schnittstellen sowie implizit gleichzeitig remoten Schnittstellen des NBI – eine `PrivilegienBaustein`-Instanz zugewiesen. Das Gegenstück dazu bildet eine gleichartige Zuweisung einer `PrivilegienBaustein`-Instanz zu einem (realen oder funktionalen) Nutzer bzw. Nutzereintrag im Authentifizierungsbaustein (beschrieben in Abschnitt 5.6.4.1). Entsprechend erfolgt bei Zugriff auf eine am NBI verfügbare Schnittstelle zunächst der Abgleich zwischen dem Privilegienlevel der jeweiligen Schnittstelle und dem des Nutzers.

5.6.2.8 Verwaltung von Schnittstellen und NBI-Register

Die Verwaltung der Schnittstellen des NBI wird durch das NBI-Register zentral umgesetzt. Dieses ist ebenfalls der Anlaufpunkt für die De- / Registrierung von Schnittstellen sowie dem lokalen Aufruf von Pull-Zugriffen, wodurch das NBI-Register als **zentrale Komponente** realisiert ist. Es erfüllt somit drei Aufgaben:

- Bereitstellung von **Verwaltungsfunktionen** für Schnittstellen.
- Organisation der Schnittstellen über jeweils geeignete **Datenstrukturen**.
- **Zugriffskoordination**.

Verwaltungsfunktionen von Schnittstellen am NBI teilen sich insbesondere wieder entsprechend in Push- und Pull-Bausteine auf. Für **Pull-Elemente** ist eine Funktionalität zur **Registrierung** einer neuen `NBIPullBaustein`- bzw. `NBIRemotePullBaustein`-Instanz notwendig. Für lokale Zugriffe sind keine weiteren Angaben außer den eigentlich notwendigen Baustein-Instanzen notwendig. Für remote Zugriffe ist außerdem die Angabe eines Aufrufkontexts notwendig, für den eine vorgesehene Instanz des `NBIRemotePullBaustein` aktiviert wird. Für HTTP-basierte Zugriffe ist dies beispielsweise eine URL, für deren Behandlung die durch den Baustein durchgereichte Funktion der Managementplattform ausgeführt wird. Der Kontext dient ebenfalls zur effizienten Verwaltung der Bausteine, wie im nächsten Absatz beschrieben wird. Eine weitere Funktion für Pull-Elemente ist die **Auflistung unterstützter Schnittstellen**, welche automatisch aus den jeweiligen Elementen der Pull-Bausteine generiert werden kann und insbesondere die ID, Beschreibung, notwendige Parameter (Parameter-Namen und Typen) und unterstützte Rückgabewerte (ebenfalls Variablen-Namen und Typen) ausgibt. Schließlich bietet das NBI-Register Funktionalität zum **Ausführen von Pull-Zugriffen des NBI** und übernimmt die bereits in den Abschnitten 5.6.2.1 und 5.6.2.2 beschriebene Aufrufkoordination. Verwaltungsfunktionen für **Push-Elemente** umfassen die **Registrierung** neuer `NBIPushBaustein`- und `NBIRemotePushBaustein`-Instanzen mit allen notwendigen Angaben, wie in den vorherigen Abschnitten beschrieben wurde. Die Funktion zur **Auflistung von Push-Elementen des NBI** umfassen ebenfalls eine Beschreibung sowie Informationen zum Abonnement (z. B. eine remote Schnittstelle) und zur Kündigung eines Abonnements der jeweiligen Benachrichtigung. Funktionen zum Abonnement und -Kündigung werden durch das NBI-Register realisiert.

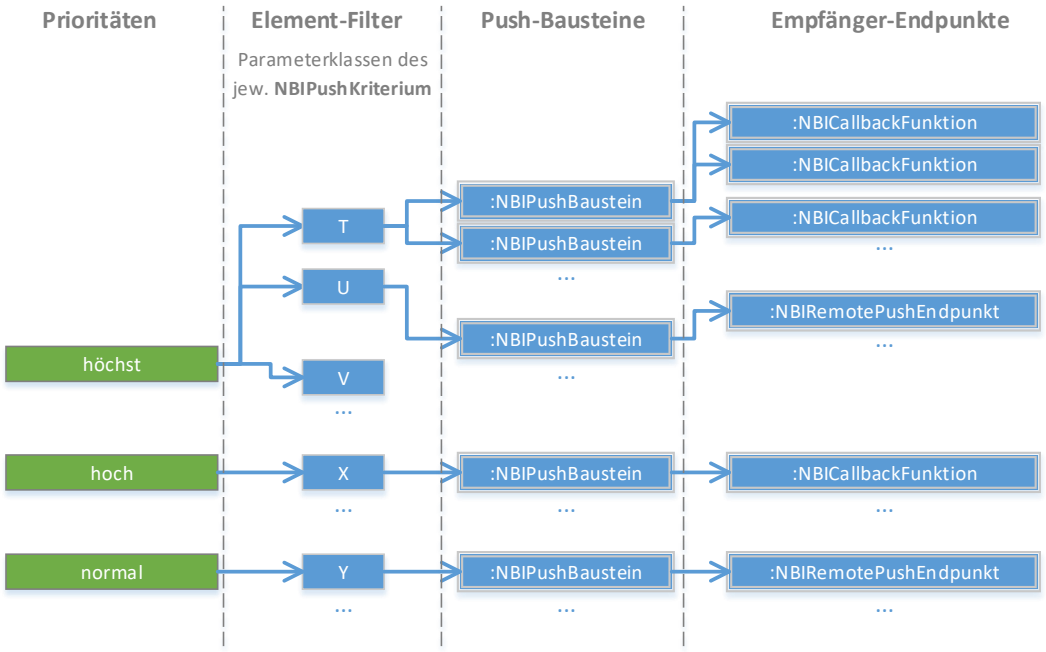


Abbildung 5.37: Effiziente Verwaltung zentraler Komponenten von Push-Benachrichtigungen am NBI. Prioritätenklassen (grün) verweisen auf Klassen von Informationselementen aus den jeweiligen `NBIPushKriterium`-Instanzen, diese auf NBI-Bausteine und diese wiederum auf Callback-Funktionen bzw. remote Endpunkte.

Datenstrukturen zur effizienten Speicherung von Push- und Pull-Bausteinen richten sich nach der Art des Zugriffs auf die Baustein-Instanzen und wurden teilweise bereits in den entsprechenden Abschnitten zuvor erläutert. **Lokal aufrufbare Pull-Bausteine** werden auf Basis ihrer ID angefragt und ausgeführt. Ein effizienter Zugriff findet daher auch über eine Tabelle statt, welche ausgehend von der jeweiligen eindeutigen ID des Bausteins auf die damit verknüpfte `NBIPullBaustein`-Instanz verweist. **Remote ausführbare Pull-Bausteine** werden über den Aufrufkontext identifiziert: `NBIRemotePullBaustein`-Instanzen sind an einen eindeutigen Kontext gebunden.

Die Verwaltung von **Push-Bausteinen** ist für die lokale und remote Variante hingegen homogen. Alle `NBIPushBaustein`-Instanzen werden gemäß ihrer Priorität verwaltet. Je nachdem, ob es sich um einen lokal oder remote unterstützten Zugriff handelt, werden mit der entsprechenden Baustein-Instanz dann entweder `NBICallbackFunktion`-Instanzen (lokal) oder `NBIRemotePushEndpunkt`-Instanzen (remote) verknüpft. Die Verwaltung kann über eine mehrstufige Tabelle umgesetzt werden: Wie in Abbildung 5.37 gezeigt ist, werden `NBIPush-Baustein`-Instanzen auf eine oder mehrere Callback-Funktionen (Endpunkte für Nachrichtenempfänger) und Erstere wiederum über überwachte Parameterklassen (z. B. `Netzinformation`-Elemente) abgebildet. Die Wurzelelemente bilden die einzelnen `Priorität`-Klassen, wodurch zum einen mehrere `NBIPushBaustein`-Instanzen denselben Filterklassen und zum anderen derselbe `NBIPushBaustein` für mehrere Callback-Funktionen verwendet werden. Auf diese Weise können mehrere Empfänger angegeben werden. Der Ansatz erlaubt eine hohe Flexibilität in der Verwaltung von Benachrichtigungen und eine flexible Definition der Prioritäten. Die Abarbeitung erfolgt schließlich von Elementen in der höchsten hin zu denen der niedrigsten Prioritätenklasse.

5.6.3 Datenbankanbindung

Managementplattformen benötigen eine Anbindung zu einer Datenbank, die den konkreten aktuellen (und teilweise auch vergangenen) Zustand der gemanagten IT-Ressourcen sowie Verwaltungsinformationen aus den vier Teilmodellen einer Managementarchitektur persistent oder transient speichert. Dieses Konzept fokussiert sich auf die drei folgenden Kernaufgaben:

1. Eine **flexible Beschreibung der Datenbankanbindung** unter Berücksichtigung der Integration der anderen Frameworks dieses Konzepts.

2. Das **Einpflegen von am SBI empfangenen Daten** und Informationen der gemanagten IT-Infrastruktur in das Speichermodell (d. h. die föderationsweite Datenlage, repräsentiert durch eine zentrale oder verteilte Datenbank).

3. Unterstützung eines **Benachrichtigungsdienstes** zur Realisierung von Push-Benachrichtigungen.

Die zentrale Komponente der Datenbankanbindung ist die **Datenzugriffsverwaltung**. Sie übernimmt die Organisation der Frameworkkomponenten zur Lösung der drei zuvor genannten Aufgaben. Eine Übersicht über Komponenten der Datenbankanbindung ist in Abbildung 5.38 gezeigt.

5.6.3.1 Beschreibung von Speicheroperationen

Für die Nutzung notwendiger Operationen gängiger Datenbanksysteme lassen sich dabei ganz allgemein auf CRUD-(englisch für *Create*, *Read*, *Update* und *Delete*)-Operationen zurückführen: In [199] beispielsweise für das dokumentenorientierte Datenbanksystem MongoDB oder in [200] im Kontext der Erstellung von Tests für NoSQL-Datenbanken, oder auch in [201] zum Zweck eines Performanzvergleichs zwischen MongoDB und des relationalen Datenbanksystems MySQL. Angelehnt an diese Verallgemeinerung werden auch in diesem Framework

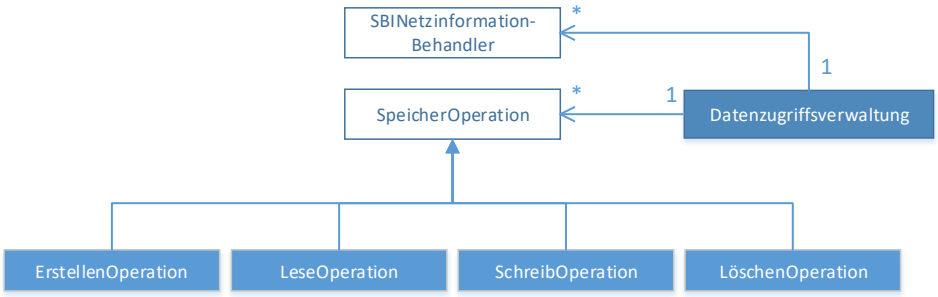


Abbildung 5.38: Übersicht über Komponenten der Datenbankanbindung.

Teiloperationen als abstrakte Klasse `Speicheroperation` einzeln definierbar, jedoch gruppiert in das CRUD-Operationsmodell, gestaltet. Wie in Abbildung 5.38 gezeigt, werden die die Teiloperationen repräsentierenden Klassen `ErstellenOperation`, `LeseOperation`, `SchreibOperation`, `LöschenOperation` davon abgeleitet. Falls notwendig, ist ebenfalls eine Ableitung weiterer Operationen möglich. Anders als die abstrakte Klasse `SpeicherOperation` können die davon abgeleiteten Klassen instanziiert werden.

Jede Instanz der Klasse `SpeicherOperation` muss bei Erstellung durch ein Feld **Beschreibung** insofern nutzerverständlich beschrieben werden. Für die Identifikation einer Speicheroperation im Betrieb werden zwei Elemente vorgesehen:

- Ein Feld vom Typ `SpeicherOperationGruppe` als eine Liste der Informationstypen, die durch die Operation geändert werden. Dazu zählen Informationen der in diesem Konzept definierten Frameworks wie *MOCs*, *Netzinformationen*, *Nutzer* oder *Domänen*. Die Granularität wird dem Entwickler je nach Anwendungsfall überlassen.

- Zum anderen hat jede Speicheroperation einen innerhalb der zuvor beschriebenen Gruppe eindeutigen `Identifikator`.

Die beschriebenen Elemente werden an jede Kind-Klasse von `SpeicherOperation` vererbt. Jede Speicheroperation und somit Subklasse hat zudem eine abstrakte und zu implementierende Methode `ausführen`, wie in Listing 5.22 gezeigt ist.

```
1 abstract class SpeicherOperation {
2   SpeicherOperationGruppe fGruppe;
3   Identifikator fIdentifikator;
4
5   abstract <A, B> B ausführen(A input) throws SpeicherOpFehler;
6 }
```

Listing 5.22: Pseudocodedarstellung der Klasse `SpeicherOperation`.

Alle Speicheroperationen arbeiten im Kontext der Methode `ausführen` auf einem Parameter des generischen Typs `A` und müssen als Konvention **das geänderte Datenobjekt** der hier generischen Klasse `B` zurückgeben. Insofern gilt implizit die Konvention, dass eine definierte Speicheroperation so gestaltet sein muss, dass sich **maximal ein Element** im Rahmen ihrer Ausführung ändern kann.

Falls keine Änderung erfolgt (d. h. ein neuer Wert entspricht dem alten Wert) soll der jeweilige Methodenaufruf von `ausführen` ein dies kennzeichnendes Dummy-Element (z. B. einer Klasse `Leer`) zurückgeben. Die Rückgabe des geänderten Elements ist insbesondere der Ausgangspunkt für den Benachrichtigungsdienst für das NBI, wie in Abschnitt 5.6.3.3 beschrieben wird. Im Fehlerfall bei Ausführung einer Speicheroperation wird eine Exception des Typs

SpeicherOpFehler generiert. Der hohe Freiheitsgrad in der Implementierung einer Speicheroperation schränkt die Darstellung der in diesem Konzept beschriebenen objektorientierten Formate über ein datenbankspezifisches Format nicht ein. Entsprechend sind unterschiedliche Arten von Datenbanksystemen (z. B. relational oder dokumentenorientiert) möglich.

5.6.3.2 Integration von Informationen aus dem SBI

Abschnitt 5.6.1 beschreibt die Komponenten des SBI und die Prozesse angefangen beim Abruf von Daten aus der gemanagten IT-Umgebung bis zu einer ersten Normalisierung der Daten und ihrer Darstellung als SBINetzinformation-Elemente. Dieser Abschnitt knüpft daran an und beschreibt ein Teilframework der Datenbankanbindung zur Aktualisierung des Speichermodells einer Managementplattform auf Basis dieser Informationen.

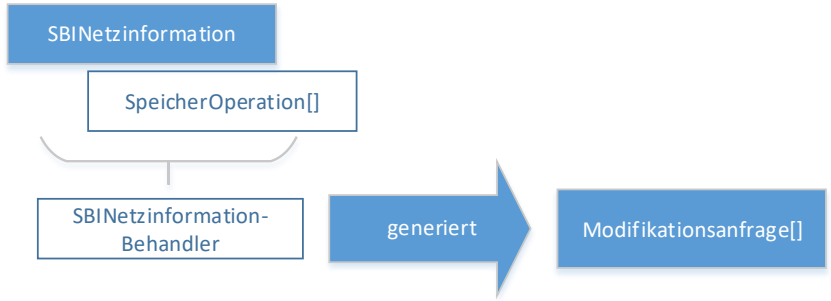


Abbildung 5.39: Komponenten zur Überführung eines SBINetzinformation-Elements zu Modifikationsanfrage-Elementen.

Die Ausgangsbasis dazu bieten die SBINetzinformation-Elemente aus Abschnitt 5.6.1.2 und die im vorherigen Abschnitt beschriebenen SpeicherOperation-Elemente. Wie in Abbildung 5.39 gezeigt wird, dienen sie auch als Eingabe für das für die Überführung in das Speichermodell verantwortliche Element SBINetzinformationBehandler. Dieses generiert für genau eine SBINetzinformation und alle implementierten SpeicherOperation-Instanzen eine chronologisch geordnete Liste an Modifikationsanfrage-Elementen – insbesondere da eine SBINetzinformation beliebig viele Netzinformation-Elemente kapselt. Durch Berücksichtigung der zeitlichen Reihenfolge der Ausführung der Speicheroperationen bleiben voneinander abhängige Elemente konsistent. Werden beispielsweise durch dieselbe Modifikationsanfrage einerseits eine FSNMOC-Instanz angelegt sowie andererseits Managementbeziehungen zu anderen FSNMOC-Instanzen gespeichert, benötigen die Managementbeziehungen mindestens den Identifikator der jeweiligen FSNMOC-Instanzen. Folglich ist die Ausführung nur in dieser Chronologie sinnvoll.

```
1  abstract class SBINetzinformationBehandler {
2    abstract Set<Modifikationsanfrage> behandle(SBINetzinformation input) throws
         SBINetzinformationBehandlungsFehler;
3    abstract boolean kannBehandeln(SBINetzinformation input);
4  }
5
6  class Modifikationsanfrage <A> {
7    SpeicherOperation fOperation;
8    A fParameter;
9
10   boolean checkTyp() throws SpeicherOpFehler {
11     // Prüfung, ob Typ von fParameter auf notwendigen Parameter der
12     // Speicheroperation fOperation passt.
13   }
14 }
```

Listing 5.23: Pseudocodedarstellung der Klassen SBINetzinformationBehandler und Modifikationsanfrage.

Zur Überführung von SBINetzinformation- in Modifikationsanfrage-Elemente bietet jede SBINetzinformationBehandler-Instanz eine abstrakte Methode behandle, wie Listing 5.23 zeigt. Diese Methode wird und die Methode kannBehandeln wird durch einen Entwickler implementieren. Letztere überprüft das übergebene SBINetzinformation-Element auf Behandelbarkeit. Listing 5.23 zeigt zudem eine Pseudocode-Implementierung der Klasse Modifikationsanfrage, welche als Feldvariable zum einen eine konkrete **Speicheroperation** sowie einen dazugehörigen generischen **Parameter** beschreibt. Jede Instanz hat zudem eine Methode checkTyp zur Überprüfung der Kompatibilität des Parametertyps mit dem angegebenen Typ der Speicheroperation. Diese dienen vonseiten des SBI als Änderungsanfragen und werden **asynchron** von der Datenzugriffsverwaltung abgearbeitet.

Die Zahl eingesetzter SBINetzinformationBehandler-Instanzen in einer Managementplattform ist von der Heterogenität der Daten und Komponenten der gemanagten IT-Infrastruktur abhängig. Die effiziente Auswahl zu SBINetzinformation-Instanzen passender Elemente ist daher ein wichtiger Aspekt, welcher durch einen zweistufigen Filter-Ansatz gelöst wird. Dafür werden *1)* SBINetzinformationBehandler-Instanzen zunächst **FSNMOCK-Klassen** zugewiesen, deren abgefragte Informationen sie einarbeiten können. Da die Quell-FSNMOC-Instanz auch in SBINetzinformation-Instanzen hinterlegt ist (vgl. Abschnitt 5.6.1.2), kann daher für FSNMOCK-Klassen, die der Quell-FSNMOC-Instanz zugewiesenen sind, eine Auswahl potenziell passender Behandler getroffen werden. Diese werden *2)* weiter eingeschränkt, indem die Methode kannBehandeln der jeweiligen vorausgewählten SBINetzinformationBehandler-Instanz ausgeführt wird.

Die eigentliche Ausführung dieses Prozesses wird durch die Datenzugriffsverwaltung koordiniert und wie folgt durchgeführt:

1. Ein SBINetzinformation-Element wird an der Datenzugriffsverwaltung (z. B. durch das SBI) registriert.

2. Die Datenzugriffsverwaltung wählt auf Basis der im SBINetzinformation-Element übergebenen FSNMOC-Instanz potenziell geeignete SBINetzinformationBehandler-Instanzen aus und schränkt die Auswahl weiter durch Aufruf der Methode kannBehandeln in der jeweiligen Instanz ein.

3. Bei geeigneten SBINetzinformationBehandler-Instanzen wird die Methode behandle unter Parametrisierung mit der zu bearbeitenden SBINetzinformation-Instanz sowie allen verfügbaren SpeicherOperation-Instanzen ausgeführt und das Ergebnis – eine chronologisch geordnete Liste an Modifikationsanfrage-Instanzen – zwischengespeichert.

4. Die Liste an Modifikationsanfrage-Instanzen wird abgearbeitet und für gültige Elemente – deren Methodenaufruf checkTyp *true* zurückgibt – wird die damit verknüpfte Speicheroperation mit jeweiligem Parameter aufgerufen.

5.6.3.3 Benachrichtigungsdienst für das NBI

Die Datenbankanbindung und insbesondere ihre zentrale Komponente, die Datenzugriffsverwaltung sind ein essenzieller Baustein bei der Unterstützung der **Quasi-Echtzeitbenachrichtigung** von Managementanwendungen. Dafür erkennt die Datenzugriffsverwaltung Änderungen im Speicherzustand der Datenbank und meldet sie an die Push-Funktionalität des NBI.

Das Kernelement zur Überwachung von Änderungen des Speicherzustands der Managementplattform ist die in Abschnitt 5.6.3.1 definierte Klasse SpeicherOperation – insbesondere die schreibenden Kind-Klassen ErstellenOperation, SchreibOperation und LöschenOperation, da ihre Ausführung zu Änderungen im Speicherzustand führen. Auf diese Weise werden

einerseits durch das NBI als auch andererseits durch das SBI – hier implizit über `Modifikationsanfrage`-Elemente – vorgenommene Änderungen am Speicherzustand berücksichtigt.

Aufseiten des NBI bildet die in der Klasse `NBIPushBaustein` (und implizit in `NBIRemotePushBaustein`) verwendete Klasse `NBIPushKriterium` (vgl. Abschnitte 5.6.2.3) das Gegenstück zur Bestimmung der Relevanz einer Speicherzustandsänderung. Die Klasse ist über eine generische Klasse `T` parametrisierbar und speichert diese ebenfalls. Der Anwendungsbereich einer `NBIPushBaustein`-Instanz auf Speicherelemente dieses Typs `T` – bzw. hier im Rahmen einer SpeicherOperation als Rückgabe-Typ `B` bezeichnet – wird so grob eingegrenzt. Die Basisfunktion für den Benachrichtigungsdienst der Datenzugriffsverwaltung ist daher die Erstellung einer Übersicht über zu überwachende Elemente dieses generischen Typs `B`. Eine schnell zugreifbare Übersicht bildet in diesem Fall eine Tabelle, die von einer Klasse `B` auf einen booleschen Wert verweist: Referenziert `B` auf *true*, ist die Klasse überwacht.

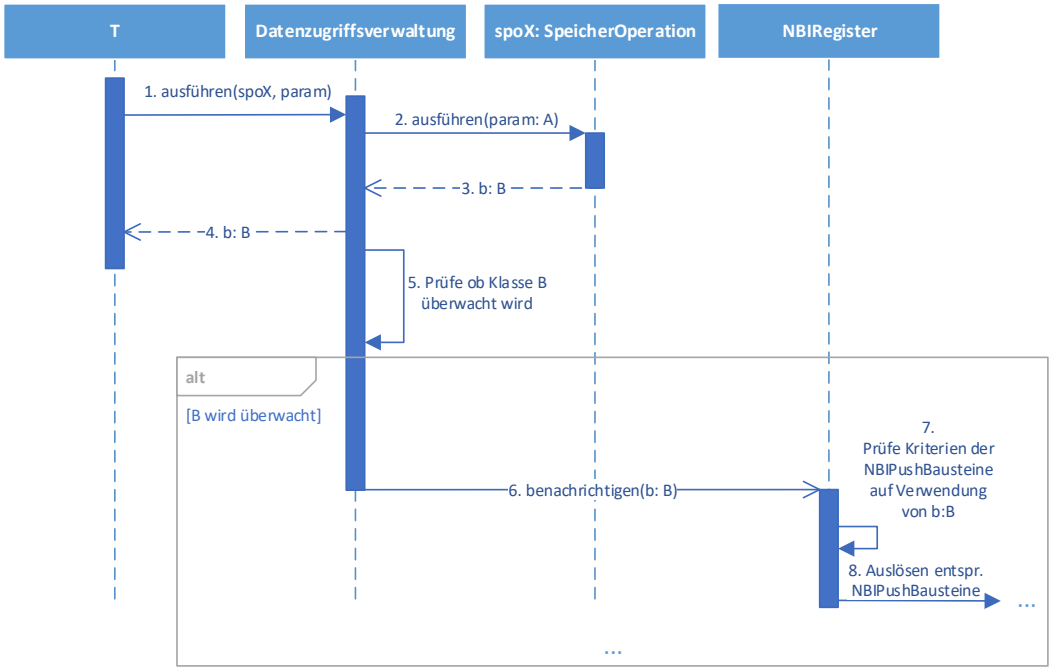


Abbildung 5.40: Ablauf der Überwachung von Speicherelementen durch die Datenzugriffsverwaltung und Benachrichtigung des NBI.

Abbildung 5.40 zeigt den Ablauf der Überwachung des Speicherzustands durch die Datenzugriffsverwaltung: Die Datenzugriffsverwaltung als verwaltende Komponente von `SpeicherOperation`-Instanzen stellt ebenfalls den Zugriffspunkt ihrer Ausführung dar. Führt beispielsweise eine Instanz einer hier generischen Klasse `T` (z. B. eine MOC-Funktion) eine Speicheroperation `spoX` über die Datenzugriffsverwaltung aus *(1)*, identifiziert letztere ebendiese Speicheroperation und ruft ihre Methode `ausführen` mit dem ebenfalls übergebenen Parameter `param` auf *(2)*. Die Rückgabe – das geänderte Speicherelement `b` *(3)* – wird dann einerseits an die ursprünglich anfragende Komponente *(4)* zurückgegeben, andererseits ihr Typ `B` von der Datenzugriffsverwaltung hinsichtlich Überwachung überprüft *(5)*. Wird Klasse `B` überwacht, ist das Element `b` ein Kandidat für Push-Benachrichtigungen über das NBI; die Datenzugriffsverwaltung benachrichtigt entsprechend das NBI-Register *(6)*, welche für alle mit dieser Klasse verknüpften Elemente (vgl. Abschnitt 5.6.2.3 bzw. darin Abbildung 5.37) durch Überprüfung der `NBIPushKriterium`-Instanzen identifiziert *(7)*. Wenn ein Kriterium zutrifft – die jeweilige Methode `istRelevant` gibt *true* zurück – wird die entsprechende `NBIPushBaustein`-Instanz ausgelöst *(8)*.

5.6.4 Zentrale Komponenten des Kommunikationsmodells

Zentrale Komponenten des Kommunikationsmodells werden am NBI genauso wie am SBI benötigt. Dazu zählen insbesondere Netzadapter, die die Kommunikation mit der gemanagten Infrastruktur am SBI sowie mit Managementanwendungen am NBI realisieren. Darüber hinaus zählt dazu der im folgenden Abschnitt beschriebene Authentifizierungsbaustein, über den der Zugriff auf die gemanagte IT-Infrastruktur am SBI sowie auf Funktionalität der Managementplattform gesteuert wird. Für die Datenbankanbindung werden diese zentralen Komponenten nicht genutzt, da an dieser Stelle keine Flexibilität der Größenordnung wie am SBI oder NBI notwendig ist, sondern eine Anbindung und Authentifizierung eher statischen Charakter hat.

5.6.4.1 Authentifizierungsbaustein

Eine flexible Authentisierung und Authentifizierung findet potenziell an zwei Schnittstellen einer Managementplattform statt. Auf der einen Seite in zwei Fällen am **NBI**:

1. Eine **Authentifizierung** von Nutzern bei Zugriff von Managementanwendungen auf Plattformfunktionen, die eine Managementplattform über Schnittstellen bereitstellt (Pull).

2. Eine **Authentisierung** beim Zugriff auf Schnittstellen von Managementanwendungen bei Benachrichtigungen (Push): Die Managementplattform stellt für eine Schnittstelle einer Managementanwendung entsprechend Anmeldedaten bereit.

Analog ist eine Authentisierung bzw. Authentifizierung am **SBI** notwendig:

4. Eine **Authentifizierung** beim Zugriff auf Schnittstellen von MOs auf der Managementplattform (Push). Die Managementplattform authentifiziert entsprechend das Quell-MO beim Empfang von Daten.

5. Eine **Authentisierung** beim Zugriff auf von MOs bereitgestellte Schnittstellen (Pull).

In FSNs sind Informationen zur **Authentisierung (Fälle 2 und 4)** an MOs (SBI) in der Regel sehr heterogen, beispielsweise über (z. B. SSH-) Public-Keys, Benutzername-Passwort-Kombinationen oder Tokens. Meistens sind mehrere im Netz notwendig. Dazu sollen `FSNMOCK`-Klassen (d. h. Komponenten-Klassifikationen) mit einer weiteren Klasse `Credentials` (vgl. Abbildung 5.41) verknüpft werden können, welche optional derartige Informationen bereitstellt. Von dieser Klasse können dann die verschiedenen Authentisierungs-Informationstypen (z. B. HTTP-Token, Nutzername-Passwort-Kombination, usw.) strukturiert abgeleitet werden. Da netzgebundene Managementanwendungen in diesem Konzept durch die `NMAPP`-Klasse beschrieben werden, ist hier ein einheitlicher Ansatz möglich.

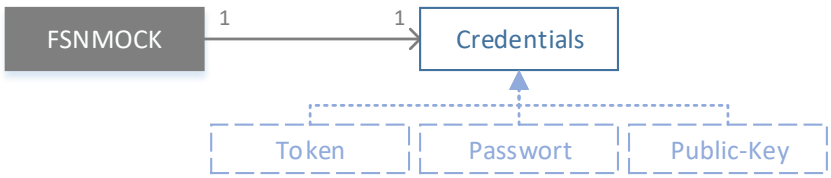


Abbildung 5.41: Mit der Klasse `FSNMOCK` verknüpftes `Credentials`-Element und davon beispielhaft abgeleitete spezifische Authentifizierung-Informationstypen.

Die **Authentifizierung für Pull-Zugriff am NBI (Fall 1)** wird über Nutzer der Managementplattform realisiert. In diesem Konzept werden Plattformnutzer über die in Abschnitt 5.5.3 beschriebenen `Entität`-Elemente beschrieben. Jede Entität muss dahingehend um Informationen zur Authentifizierung erweitert werden, beispielsweise eine Liste gültiger Tokens, öf-

fentlicher Schüssel oder Informationen zur Kontrolle von Passwörtern (z. B. kryptographische Hashfunktionen). Die Durchführung der Authentifizierung wird durch das NBI-Register übernommen.

Eine **Authentifizierung für Push-Benachrichtigungen am SBI (Fall 3)** ist dagegen nicht direkt durch Authentifizierungsinformationen der Nutzer einer Managementplattform verknüpft. Der Grund dafür ist, dass eine Verwendung von Nutzerinformationen der Managementplattform in gemanagten Netzkomponenten zu deren Kompromittierung führen könnte und sie ebenfalls zur Authentifizierung am NBI benutzt werden könnten. Daher sind analog zum Fall 1 und 3 in diesem Fall Authentifizierungsinformationen – ebenfalls über das `Credentials`-Element – direkt mit einer Schnittstelle des SBI verknüpft (d. h. ein `Schnittstelle`-Element verweist auf ein `Credentials`-Element). Die Durchführung der Authentifizierung wird automatisch durch die jeweilige Schnittstelle durchgeführt.

5.6.4.2 Netzadapter und -Bausteine

Aktuelle Controller- und Managementplattformen haben sich bereits mit der Umsetzung von Netzadapterpool am SBI als auch NBI beschäftigt. Eine Normalisierung von MO-Funktionen, wie sie in Vorarbeiten dieser Dissertation als sinnvolles Kriterium gefordert und in FED-CON [181] zusammen mit einem Lösungsvorschlag beschrieben wurde, fehlt jedoch meist.

Der **Netzadapterpool** ist eine zentrale Komponente zur Verwaltung aller in der Managementplattform für das SBI oder NBI genutzter Netzadapter. Als **Konnektor** wird ein Netzadapter bezeichnet, der im Kontext des SBI eine Pull-Operation (d. h. MOC- und Plattformfunktionen zugänglich macht) erlaubt und im Kontext des NBI eine Push-Operation durchführt. Als **Schnittstelle** wird ein Netzadapter bezeichnet, der im Kontext des SBI eine Gegenstelle für Push-Operationen vonseiten der MOs und im Kontext des NBI eine Gegenstelle für Pull-Operationen vonseiten Managementanwendungen oder -Plattformen darstellt.

Die **Auswahl eines Konnektors** am NBI bzw. am SBI wird durch das jeweils zuständige NBI- oder SBI-Register bearbeitet. Abbildung 5.42 illustriert den Ablauf beispielhaft: Die Grundlage der Auswahl und Weitergabe des passenden Konnektors ist die Referenzierung von durch das SBI verwendeten Konnektoren durch das SBI-Register, sowie analog die Referenzierung von durch das NBI verwendete Konnektoren durch das NBI-Register. Es können auch mehrere Konnektorinstanzen derselben Klasse (z. B. mehrere HTTP-Konnektoren) durch eines der Register gehalten werden, um mehrere gleichzeitige Anfragen parallelisieren zu können, wenn eine Instanz in Nutzung ist. Ebenso ist auch denkbar, dass Instanzen einer bestimmten Konnektorklasse auf Anfrage erzeugt werden. Die Konnektor-Auswahl am SBI unterscheidet sich von der am NBI. Am SBI ist die initiierende Komponente der `MOCFRTreiber`, der einen Konnektor eines entsprechenden Typs am SBI-Register anfragt und durch dieses zurückgegeben wird. Am NBI werden Konnektoren in der `senden`-Methode jeder `NBIRemotePushEndpunkt`-Instanz benötigt. Die initiierende Komponente ist jedoch das NBI-Register selbst, wodurch es direkt bei Ausführung der Funktion einen gemäß Typ passenden Konnektor auswählt und nutzt.

Für die **Integration einer Schnittstelle** als Push-Gegenstelle am SBI bzw. als Pull-Gegenstelle am NBI ist ein Registrierungsmechanismus notwendig: Auf der einen Seite referenzieren das NBI- und das SBI-Register auf vom NBI respektive SBI genutzte Schnittstellen. Auf der anderen Seite muss das NBI- bzw. SBI-Register einer Schnittstelle bei Registrierung eine Callback-Funktion übergeben. Sie erlaubt es der jeweiligen Schnittstelle, die durch einen Verbindungseingang ausgelöste Verarbeitung der Daten zu initiieren. Im Falle des NBI-Registers führt ein valider Aufruf einer remoten API-Funktion der Managementplattform zur Auswahl der entsprechend behandelnden `NBIRemotePullBaustein`-Instanz. Im Falle des SBI-Registers führt die valide Push-Übertragung vonseiten eines MO zu einer Generierung eines `SBIErweiterteRohdaten`-Objekts sowie dessen Anhängen an die (möglicherweise priorisierte) *Erweiterte-Rohdaten-Queue*.

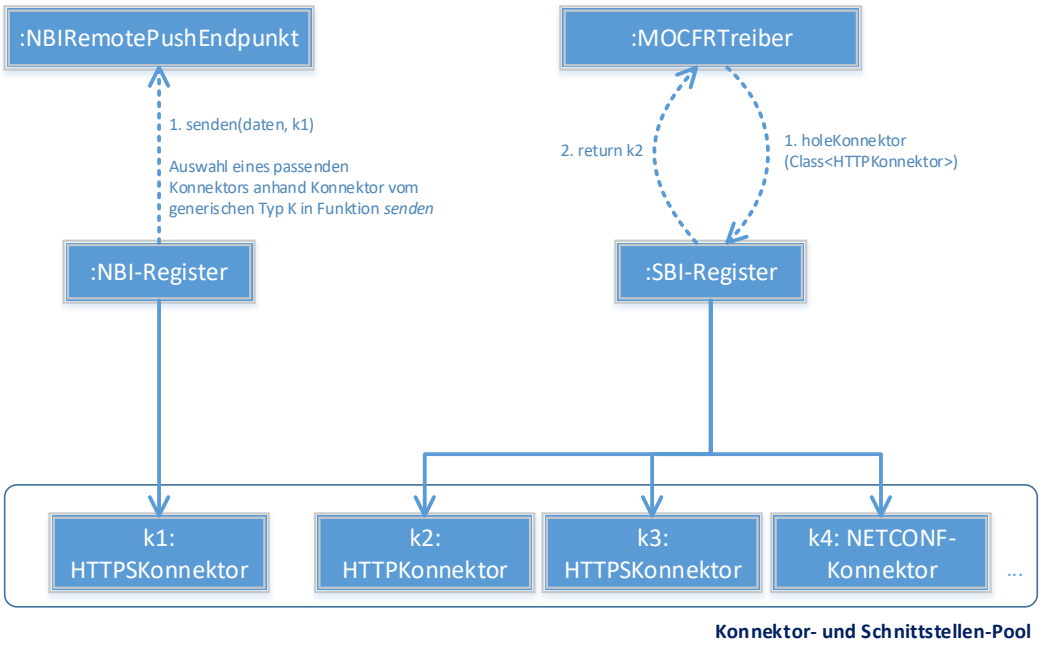


Abbildung 5.42: Anfrage eines geeigneten Konnektors für NBI-Push- und SBI-Pull-Funktionalität über das NBI- respektive SBI-Register und damit je verknüpfte Konnektoren.

Abbildung 5.43 zeigt eine beispielhafte Durchführung eines Funktionsaufrufs am NBI (links) sowie einer Bearbeitung einer Push-Kommunikation am SBI (rechts). In ersterem Fall findet der Funktionsaufruf durch eine Managementanwendung statt (Schritt 0). In Schritt 1 ruft die empfangene Schnittstelle die entsprechende Callback-Funktion mit Aufrufkontext, den Daten sowie dem Verbindungsobjekt am NBI-Register auf, welches den zuständigen `NBIRemotePullBaustein` auswählt und seine Funktionalität ausführt (Schritte 2 bis 5). Das NBI-Register nutzt schließlich das entsprechende Verbindungsobjekt, um eine Antwort mit Rückgabe an die aufrufende MAPP zu schicken.

Das Beispiel auf der rechten Seite in Abbildung 5.43 zeigt hingegen die Durchführung einer Push-Datenübertragung durch ein MO am SBI der Managementplattform. In diesem Fall löst ebenfalls die empfangene Schnittstelle die Bearbeitung durch Aufruf einer Callback-Funktion am SBI-Register aus. Dem Callback wird ein `SBIErweiterteRohdaten`-Objekt sowie ein Objekt zur Behandlung der Verbindung übergeben. Das SBI-Register führt im Fall, dass das erhaltene `SBIErweiterteRohdaten`-Objekt valide ist, zum einen die Einreihung des Objekts in die (u. U. priorisierte) *Erweiterte-Rohdaten-Queue* sowie die Bestätigung an das Quell-MO der Daten über das erhaltene Verbindungsobjekt durch.

5.7 Funktionsmodell

Managementfunktionen sollen dem Netzmanagement in diesem Konzept als Strukturierungsgrundlage dienen. Das Funktionsmodell unterstützt daher eine szenarienspezifische Adaptierbarkeit von Managementfunktionen. Sie werden darüber hinaus als Generalisierung von Aufgabenbereichen gesehen. Die Funktionalität der Managementplattform über das in diesem Konzept beschriebene Maß hinaus wird über lokal oder remote angebundene Managementanwendungen erweitert. Funktionen, die die Elemente und Strukturen aus diesem Konzept betreffen, werden dagegen über sogenannte *Plattformfunktionen* erweitert. **Plattformfunktionen** können von Nutzern aufgerufen werden und erlauben beispielsweise die Adaptierung des Organisationsmodells, wie das Anlegen einer administrativen Domäne sowie eines Nutzer-

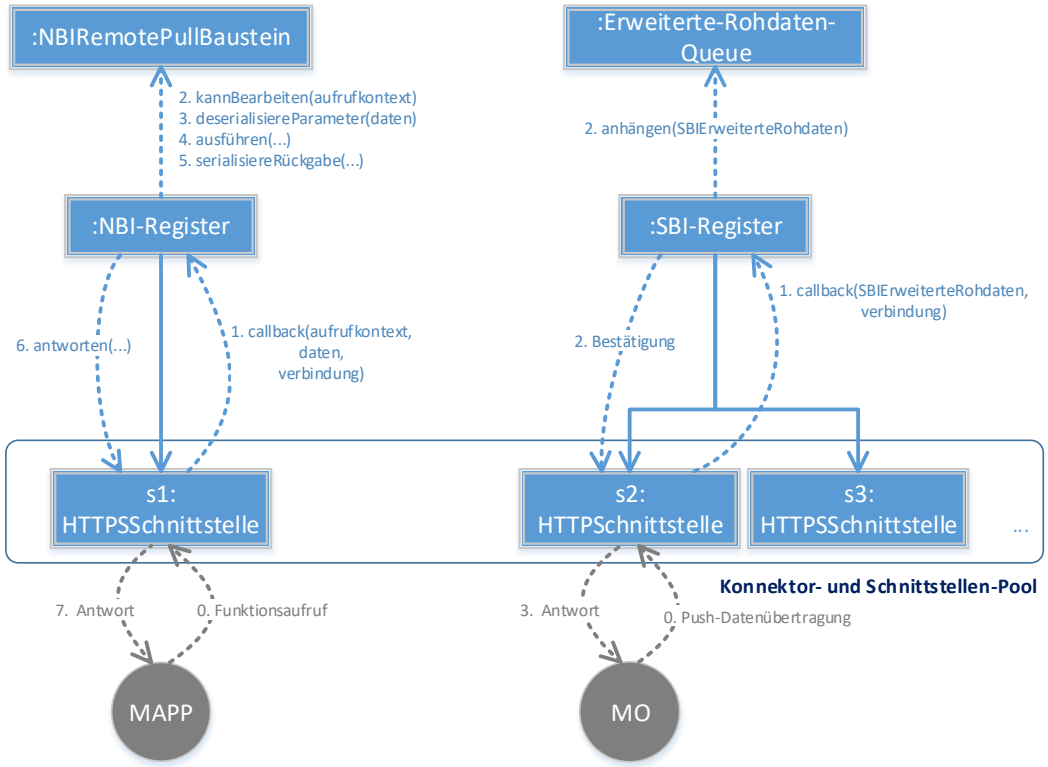


Abbildung 5.43: Integration von Schnittstellen in das SBI und NBI. Das SBI- und NBI-Register verweisen auf genutzte Schnittstellen und installieren eine Callback-Funktion.

eintrags oder auch die Zuweisung von Zuständigkeiten. In diesem Konzept wird ein Rahmen für Plattformfunktionen vorgegeben. Der letzte im Funktionsmodell berücksichtigte Aspekt ist die **Automatisierung der Kernprozesse**.

5.7.1 Funktionale Managementbereiche

Ein funktionaler Managementbereich, wie ein Element der FCAPS-Gruppe, wird über eine zentrale Elternklasse `Funktionsbereich` abgeleitet.

5.7.1.1 Modellierung eines Funktionsbereichs

Die Klasse `Funktionsbereich` ist abstrakt und kann daher nicht direkt instanziiert werden, sondern dient als Elternklasse zur Beschreibung spezifischer funktionaler Bereiche. Eine `Funktionsbereich`-Instanz dient als Ankerpunkt für damit verknüpfte Elemente. Ein Funktionsbereich wird über eine föderationsweit eindeutige **ID** sowie eine **Beschreibung** des Zwecks sowie spezifischer **Aufgaben** des Funktionsbereichs (siehe nächster Abschnitt) beschrieben. `Funktionsbereich`-Elemente werden zentral in der `Föderation`-Instanz der Managementplattform referenziert. Ein Funktionsbereich wird pro Ausprägung lediglich einmal zentral instanziiert.

5.7.1.2 Aufgabenbasierte Untergliederung von Funktionsbereichen

Eine konkretisierte Ausprägung einer `Funktionsbereich`-Instanz steht in Verbindung mit den bereits im Organisationsmodell beschriebenen Aufgabenbereichen und Aufgaben aus Abschnitt 5.5.4. Wie in Abbildung 5.44 gezeigt ist, referenziert die Klasse `Funktionsbereich` und folglich davon abgeleitete Klassen auf keine, eine oder mehrere Instanzen der Klasse

`Aufgabenbereich`. Diese kann wiederum in mehrere `Aufgaben` unterteilt werden. Auf diese Weise ist eine Konkretisierung von Managementfunktionen je nach Anwendungsfall in vier Formen möglich und wird dem Managementplattform-Entwickler überlassen:

- Ein Funktionsbereich verweist auf genau einen Aufgabenbereich, durch den der jeweilige Funktionsbereich in mehrere Teilaufgaben unterteilt wird. In diesem Fall ist der Aufgabenbereich gleichbedeutend mit dem Funktionsbereich.

- Ein Funktionsbereich verweist auf mehrere Aufgabenbereiche. Aufgabenbereiche unterteilen den Funktionsbereich dann feingranular in mehrere Teilbereiche. So könnte beispielsweise der Funktionsbereich *Sicherheitsmanagement* in die Aufgabenbereiche *Härtung* und *Penetration-Testing* aufgeteilt werden, welche wiederum durch konkrete Aufgaben beschrieben werden.

- Mehrere Funktionsbereiche verweisen auf denselben Aufgabenbereich, wodurch beispielsweise in ihren Aufgaben teilweise überschneidende Funktionsbereiche wie das `Fehlermanagement` und das `Sicherheitsmanagement` zusammengeführt werden können.

- Auch sind Mischvarianten der drei zuvor genannten Formen für mehrere Funktionsbereiche möglich und Funktionsbereiche können sich in Aufgabenbereichen überschneiden.

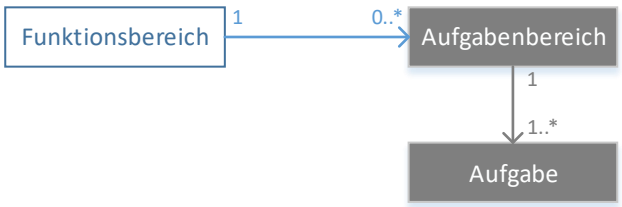


Abbildung 5.44: Die Konkretisierung und Organisation von Funktionsbereichen wird durch die bereits definierten Aufgabenbereiche aus dem Organisationsmodell vorgenommen. Eine `Funktionsbereich`-Instanz kann auf beliebig viele `Aufgabenbereich`-Instanzen verweisen.

5.7.1.3 Verknüpfung relevanter Managementanwendungen und MOCKs

Der im vorherigen Abschnitt beschriebene organisatorisch-strukturierende Aspekt der Unterteilung eines Funktionsbereichs in Aufgabenbereiche und Aufgaben ist nur einer ihrer Nutzen. Funktionsbereiche können darüber hinaus durch eine Menge an MOCKs beschrieben werden, die zur Implementierung des jeweiligen Funktionsbereichs notwendig sind und seine Umsetzungsplanung beschreiben: Beispielsweise benötigt es im Funktionsbereich des Securitymanagements spezifische Systeme wie Firewalls oder auch Systeme zur *Intrusion Detection* oder zum *Security Information and Event Management*. Eine unmittelbare Beeinflussung des Zugriffs über diese Zuweisung ist hingegen nicht sinnvoll, da sie zu grobgranular ist. In diesem Konzept wird daher vorgesehen, dass wie in Abschnitt 5.5.5.2 beschrieben wurde, Zugriffsentscheidungen für MOCKs durch ihre Bindung an spezifische Aufgaben beeinflusst werden können.

5.7.2 Erweiterung der Managementplattformfunktionalität

Die Funktionalität der Managementplattform wird über sogenannte *Plattformfunktionen* erweitert. Sie erfüllen vor allem Funktionen zum Anlegen, Löschen, Lesen oder zur Modifikation von Elementen aus den in diesem Konzept beschriebenen Frameworks des Organisations-, Kommunikations- und Funktionsmodells und mit Ausnahme von MOs auch aus dem Informationsmodell. Die Modifikation von MOs aus dem MOC-Modell wird dagegen über MOC-Funktionen (vgl. Abschnitt 5.4.3) umgesetzt, die jeweils an ein bestimmtes MO gebunden sind.

Darüber hinausgehende Funktionalität sollte in Managementanwendungen definiert und implementiert werden.

5.7.2.1 Plattformfunktionen

Im Informationsmodell wurden MOC-Funktionen in Abschnitt 5.4.3 beschrieben. Diese erweitern die Managementplattform um Funktionen, die einerseits durch MOs der gemanagten IT-Infrastruktur selbst bereitgestellt werden oder die zur Erfassung oder Änderung der Repräsentation des Zustands eines MOs in der Managementplattform führen. **Plattformfunktionen** unterscheiden sich daher in den folgenden Punkten von MOC-Funktionen:

- Plattformfunktionen sind im Gegensatz zu MOC-Funktionen nicht an `FSNMOCK`-Elemente gebunden und stellen entsprechend keine Funktionalität von MOs, sondern von der Managementplattform selbst dar. Folglich beschreiben Plattformfunktionen eine Funktionalität, die entkoppelt von spezifischen MOs und ihrer Modellrepräsentation sind.

- Im Gegensatz zu *realen MOC-Funktionen* (siehe Abschnitt 5.4.3.2) sind Plattformfunktionen nicht in den in Abschnitt 5.6.1.3 beschriebenen Prozess der Verarbeitung von über das SBI abgerufene Daten angebunden. Der Verarbeitungsprozess des SBI steuert von MOs abgerufene Daten und führt zur strukturierten Integration der Daten in das Informationsmodell. Plattformfunktionen können jedoch davon losgelöst Netzadapter nutzen, greifen jedoch nicht auf klassische MOs zu.

Plattformfunktionen sind genau wie MOC-Funktionen über das NBI, entweder lokaler oder von remote, über das Pull-Muster zugreifbar, wie in den Abschnitten 5.6.2.1 und 5.6.2.2 beschrieben wird. Plattformfunktionen werden über eine Klasse `PlattformFunktion` beschrieben, die eine eindeutige **ID** sowie benannte **Parameter** und **Rückgabewerte** als Attribute haben (vgl. Listing 5.24). Plattformfunktionen werden jeweils durch einen eigenen **Treiber**, analog zu virtuellen MOC-Funktionstreibern in Abschnitt 5.4.3.3 und **ohne Referenz** auf ein konkretes MOC implementiert. Die Beschreibung durch einen sinnvollen **Namen** erlaubt aus Nutzersicht eine effizientere Suche nach Plattformfunktionen. Namen müssen nicht eindeutig sein, um beispielsweise überladene Funktionen ähnlich wie im Paradigma der objektorientierten Programmierung zu unterstützen. Eine informelle **Kurzbeschreibung** hilft bei der Beschreibung des Zwecks der Funktion. Plattformfunktionen stellen, wie in Abbildung 5.45 gezeigt wird, die Elternklasse von MOC-Funktionen dar. Auf diese Weise können reale als auch virtuelle MOC-Funktionen im Rahmen der objektorientierten Frameworks genauso gehandhabt werden wie Plattformfunktionen selbst. Wie ebenfalls in der Abbildung gezeigt wird, stellt die Klasse `PlattformFunktionTreiber` ebenso eine Elternklasse der Treiberklassen für MOC-Funktionen dar.

```
1 class PlattformFunktion {
2   Identifikator  fID;
3   String fName;
4   String fBeschreibung;
5   Tabelle<String, FunktionsTyp> fParameterTypen;
6   Tabelle<String, FunktionsTyp> fRückgabeTypen;
7   PlattformFunktionTreiber fTreiber;
8 }
```

Listing 5.24: Pseudocode-Implementierung der Klasse `PlattformFunktion`.

Anders als MOC-Funktionen, die zur Repräsentation eines MO gehören, zählen Plattformfunktionen nicht zu Informationselementen, deren Sichtbarkeit und Zugriff auf Basis des in Abschnitt 5.5.5 entworfenen Zugriffsverwaltungsbaums gesteuert werden. Vielmehr erfolgt die Zugriffssteuerung auf Basis einer einfachen Rollenzuweisung, wie Abbildung 5.45 zeigt. Demnach ist die Nutzbarkeit einer Plattformfunktion direkt mit einer oder mehreren Rollen (aus dem Nutzer- und Rollenframework in Abschnitt 5.5.3) verknüpft. Nutzer, die eine entspre-

chende Rollenzuweisung haben, können damit verknüpfte `PlattformFunktion`-Instanzen wie MOC-Funktionen ausführen.

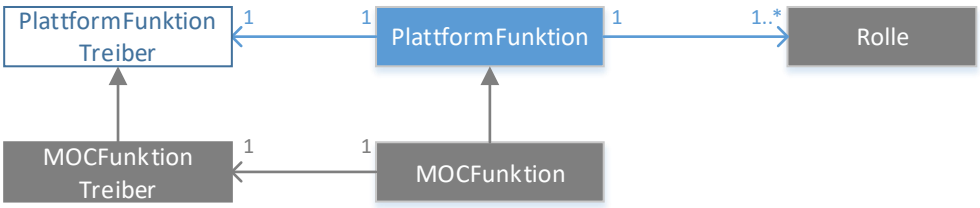


Abbildung 5.45: Beschreibung von Plattformfunktionen. Die Zugreifbarkeit auf eine Plattformfunktion wird an eine oder mehrere Rollen gekoppelt. Plattformfunktionen werden durch Treiber implementiert, die als Elternklasse der Treiber für MOC-Funktionen dienen.

5.7.2.2 Zusammengesetzte Funktionen

Eine zusammengesetzte Funktion beschreibt eine Reihe an von mehreren aufeinander aufbauenden Plattform- oder MOC-Funktionen, die durch einen Aufruf realisiert werden. Beispielsweise wird im Kontext einer SDN-Firewall-Anwendung eine Funktion zum Sperren eines Systems aus dem Netz in OpenFlow meist nicht durch die Installation eines isolierten Befehls, sondern vielmehr durch die Installationen mehrerer Flow-Regeln auf unterschiedlichen OpenFlow-Switches realisiert. In diesem Frameworkkonzept werden dazu zwei Möglichkeiten der Umsetzung vorgesehen:

- Zusammengesetzte Funktionen werden über lokal oder remote angebundene Managementanwendungen implementiert. Da Managementanwendungen in diesem Konzept wie klassische MOs behandelt werden, kann die Funktion über eine reale MOC-Funktion, wie in Abschnitt 5.4.3.2 beschrieben, anderen Managementanwendungen wieder zugänglich gemacht werden.

- Zusammengesetzte Funktionen werden über herkömmliche Plattform- und MOC-Funktionen implementiert, die andere Plattform- und MOC-Funktionen verwenden. Die Verwendung bestehender Plattform- und MOC-Funktionen wird in Abschnitt 5.6.2.1 beschrieben. Sie werden durch das zentrale NBI-Register verwaltet und können dort über ihre zugewiesene eindeutige ID angefragt werden.

5.7.3 Automatisierung der Kernprozesse

Die Automatisierung der Kernprozesse des Netzmanagements, die Überwachung und Steuerung, wird hauptsächlich durch die Funktionalität des Kommunikationsmodells, aber auch des Informationsmodells gestützt. Elemente und die Funktionalität des Organisationsmodells tragen nicht unmittelbar, sondern vielmehr unterstützend als koordinierender Part bei: Beispielsweise bei der Festlegung von Verantwortlichkeiten und damit verknüpften Zuständigkeiten und Zugriffsentscheidungen. In Abbildung 5.46 werden die unterschiedlichen frameworkübergreifenden und aufeinander aufbauenden Teilschritte gezeigt:

- Daten, die von MOs der gemanagten IT-Infrastruktur empfangen werden, werden über den priorisierbaren Verarbeitungsprozess am SBI verarbeitet und normalisiert (Schritte 1-4, vgl. Abschnitt 5.6.1).

- Normalisierte Elemente werden über die Datenzugriffsverwaltung in die bestehenden Modelle eingepflegt. Durch Abonnements überwachte Elemente werden an das NBI-Register gemeldet (siehe Abschnitt 5.6.3.3). Der Übergang wird durch Schritt 5 illustriert.

- Am NBI-Register werden Benachrichtigungen gemäß dem in Abschnitt 5.6.2.3 beschriebenen Ansatz priorisiert und an lokale oder auch netzgebundene (vgl. Abschnitt 5.6.2.4) Managementanwendungen gesendet (Schritte 6 und 7).

- Die eigentliche Auswertung der Überwachungsdaten wird genauso wie Steuerungsentscheidungen auf Managementanwendungen durchgeführt.

- Managementanwendungen führen über am NBI bereitgestellte MOC-Funktionen Steuerungsoperationen auf der gemanagten IT-Infrastruktur aus (Schritte 9 und 10, vgl. Abschnitte 5.6.2.1 und 5.6.2.2). MOC-Funktionen führen potenziell wieder zu einer Zustandsänderung oder einer Rückgabe des Funktionsaufrufs durch das MO, sodass der Automatisierungsprozess weitergeführt werden kann (Schritt 1, siehe Abschnitt 5.6.1.3).

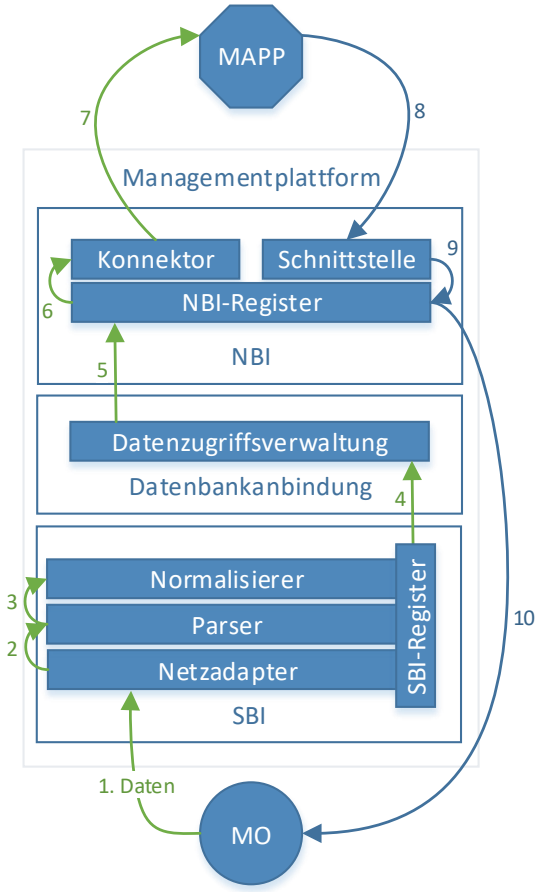


Abbildung 5.46: Vereinfachter Ablauf einer automatisierten Überwachung (grüne Kanten) und darauf aufbauender automatisierter Steuerungsoperationen (blaue Kanten).

Auf die einzelnen Teilprozesse wird mit Ausnahme der Evaluierung in Managementanwendungen in den jeweils referenzierten Abschnitten im Detail eingegangen. Auf eine erneute Beschreibung wird in diesem Abschnitt daher verzichtet.

Der beschriebene Prozess basiert auf Erkenntnissen, die in Vorarbeiten zu dieser Dissertation in [202] veröffentlicht wurden. Der Prozess in der Veröffentlichung beschränkt sich im Gegensatz zu dem hier gezeigten jedoch lediglich auf den Anwendungsfall von 5G-Netzen. Auch darin beschriebene Kriterien zur mandantenfähigen Verarbeitung beziehen sich nur auf die Zugehörigkeit eines Nutzers zu einer Organisation und wurden in dieser Dissertation um domänenspe-

zifische und aufgabenbasierte Kriterien erweitert (vgl. Abschnitt 5.5.5). Auch wurden in dieser Dissertation Informationselemente sehr viel umfangreicher betrachtet.

5.7.4 Verwaltung von Managementanwendungen

In gängigen Lösungen des Netzmanagements für FSNs oder Teilbereiche darin, wie sie in Kapitel 4 betrachtet wurden, werden Managementanwendungen nicht als MOs angesehen. Die Betrachtung von Managementanwendungen als Teil des Informationsmodells, wie es in diesem Konzept in Abschnitt 5.4.1.1 beschrieben wird, bringt jedoch Vorteile mit sich:

- Managementanwendungen als zentrale Elemente zur Bereitstellung von Managementfunktionen müssen selbst zuverlässig funktionieren und selbst managebar werden.
- Durch MOC-Funktionen können Funktionen von Managementanwendungen anderen Managementanwendungen über die Managementplattform bereitgestellt werden (vgl. Abschnitt 5.4.3).
- Der Zugriff auf Funktionen von Managementanwendungen kann einheitlich zu Funktionen der gemanagten IT-Infrastruktur geregelt werden (vgl. Abschnitt 5.5.5).

Die Bereitstellung von Funktionen von Managementanwendungen untereinander über die Managementplattform erfordert, dass Managementanwendungen selbst eine API definieren, auf die über die Managementplattform zugegriffen werden kann. Auf diese Weise können einerseits Managementanwendungen im Sinne einer Erweiterung der Managementplattform selbst genutzt und Funktionen sowohl lokaler als auch remoter Managementanwendungen geregelt untereinander zugänglich gemacht werden.

Wie in den Szenarien im Kontext der Anforderungsanalyse dieser Arbeit beschrieben wurde, ist teilweise eine unterschiedliche Nutzung von Managementanwendungen notwendig. Managementanwendungen werden üblicherweise einerseits einheitlich für die gesamte föderierte IT-Infrastruktur angewendet, andererseits ist jedoch auch die Nutzungsbeschränkung auf einzelne Domänen denkbar. Durch die Betrachtung von Managementanwendungen als MOs kann die Nutzung einer Managementanwendung für einen bestimmten Nutzer eingeschränkt werden: Sobald ein Nutzer mit seiner nutzerspezifischen Kennung auf die Funktionen einer Managementanwendung zugreift, kann die Managementanwendung beispielsweise durch die Managementplattform überprüfen, ob der Nutzer eine Zuständigkeit im jeweiligen Funktions- oder Aufgabenbereich in einer Domäne innehat und dem Nutzer den Zugriff verweigern oder erlauben. Die Managementplattform muss dafür Funktionen zur Authentifizierung auch an Managementanwendungen bereitstellen.

5.8 Zusammenfassung der Kernkonzepte für FSNs

In diesem Abschnitt wird eine Zusammenfassung der in diesem Kapitel beschriebenen Lösungen unter Berücksichtigung der Herausforderungen des Betriebs in FSNs nach Abschnitt 3.1 beschrieben. Ein detaillierter Abgleich mit den in dieser Arbeit gestellten Anforderungen an Managementplattformen in FSNs wird in der Evaluierung dieser Arbeit in Abschnitt 8.3 durchgeführt.

- **Ressourcenschichtung.** Die Herausforderung der Ressourcenschichtung wird hauptsächlich durch ein geeignetes Modell für Managementbeziehungen in Abschnitt 5.4.2 behandelt. Darin sind inhärent anpassbare Realisierungsabhängigkeiten vorgesehen, durch die beliebige Virtualisierungstiefen abgebildet werden können. Durch je nach Typ beschriebene Funktionen der Managementbeziehungen können sie traversiert werden.

- **Remote Konfigurierbarkeit.** Eine grundlegend remote Konfigurierbarkeit von MOs in FSNs wird durch ein Framework zur Modellierung von MOC-Funktionen beschrieben (vgl. Abschnitt 5.4.3). In Verbindung mit einem Framework zur Modellierung von MOCs, das die Ähnlichkeit von Attributen und auch Funktionen von Netzkomponenten berücksichtigt (vgl. Abschnitt 5.4.1) und gleichfalls die Wiederverwendbarkeit von MOC-Funktionen erlaubt, können Funktionen von Netzkomponenten über die Managementplattform flexibel zugänglich gemacht werden.

- **Geographische Verteilung.** Die geographische Verteilung wird im Framework für Föderationsmodelle (vgl. Abschnitt 5.5.1) in der Beschreibung eines Datenzentrums in der Föderation vorgesehen. Durch das Framework für Managementbeziehungen kann ein beliebiges MO stets auf genau eine Ebene-0-Ressource in einem Datenzentrum und folglich den Betriebsort zurückgeführt werden. Der Betriebsort kann daher in das Netzmanagement einfließen. Durch Konzepte zum Betrieb eines Managementplattformclusters über alle Datenzentren (siehe Abschnitt 5.6.2.6) wird auch der Betrieb einer verteilten Managementplattform über alle Standorte berücksichtigt.

- **Unmittelbare Steuerung.** Wie in Abschnitt 5.7.3 zusammengefasst wurde, wird eine unmittelbare automatisierte Steuerung auf Basis von Überwachungsdaten durch mehrere zusammenwirkende Frameworks des Konzepts unterstützt. Durch ein flexibles SBI (vgl. Abschnitt 5.6.1) können beliebige gemanagte Netzkomponenten an die Managementplattform ohne den zusätzlichen Einsatz von Agentensystemen angebunden werden.

- **Ressourcen-Pooling.** Ressourcenpools werden in FSNs in der Regel über dedizierte Komponenten wie VIMs aus dem NFV-Paradigma verwaltet. VIMs werden im Framework zur Modellierung von MOCs in Abschnitt 5.4.1 inhärent berücksichtigt.

- **Skalierbarkeit.** Die Skalierbarkeit von FSNs wird im Konzept auf verschiedenen *Ebenen* unterstützt. Die Skalierung der Größe der gemanagten IT-Infrastruktur wird durch eine mögliche verteilte Managementplattformarchitektur in Abschnitt 5.6.2.6 berücksichtigt. Die Skalierung bzw. Flexibilität organisatorischer Aspekte wie die Anzahl der Partner, Datenzentren und Nutzer in einem FSN wird durch einen allgemein flexiblen Ansatz ihrer Modellierung unterstützt.

- **Ausmaß des Netzes.** Sehr große gemanagte Netze können im Konzept durch eine geeignete Unterteilung unterstützt werden: In dem in diesem Konzept verfolgten Ansatz des Betriebs einer Managementplattform pro Datenzentrum und Nutzung aller Managementplattformen in einem Verbund (vgl. Abschnitt 5.6.2.5) ist die Interpretation eines Datenzentrums zentral. Ein physisches Datenzentrum kann beispielsweise in mehrere logische Datenzentren mit jeweils einer Managementplattform unterteilt werden.

- **Heterogenität.** Die hohe Heterogenität von Netzkomponenten in FSNs wird durch das Framework für MOCs in Abschnitt 5.4.1 unterstützt. Es gibt eine Vererbungsstruktur zur strukturierten und einfachen Ableitung neuer Komponenten von jeweils geeigneten Elternelementen der Paradigmen des SDN und NFV vor. Auf diese Weise können MOCs, ihre Attribute und Funktionen wiederverwendet werden ohne neue Komponenten von Grund auf neu zu modellieren. Durch einen klassifikationsbasierten Ansatz der Modellierung von MOCs bleiben bestehende Modelle gültig, sodass auch die Funktion eines MOs im laufenden Betrieb flexibel modifizierbar ist.

- **Weitgehend agentenlose Ansätze.** Das Framework für MOC-Funktionen erlaubt den direkten Zugriff auf die API von Netzkomponenten. MOC-Funktionen können wiederverwendet und für unterschiedliche Implementierungen der APIs der Netzkomponenten durch Treiber an das jeweilige MO angepasst werden. Sie sind direkt an MOs gebunden. Dennoch ist die Integration von Agenten weiterhin analog möglich.

- **Vertrauen.** Der in bestehenden Managementplattformen vernachlässigte Aspekt der Föderation und von Föderationsbeziehungen zwischen Parteien wird in diesem Konzept allgemein in Abschnitt 5.5.1 adressiert. Vertrauensbeziehungen stellen eine Spezialisierung einer Föderationsbeziehung dar. Durch den in Abschnitt 5.5.5.1 beschriebenen Ansatz für Mandantenfähigkeit kann die Sichtbarkeit und der Zugriff auf Elemente der gemanagten IT-Infrastruktur inhärent durch Föderationsbeziehungen und folglich auch Vertrauensbeziehungen (sowie unter Berücksichtigung anderen Kriterien) gesteuert werden.

- **Domänenbildung.** Der ebenfalls in bisher bestehenden Managementplattformen weniger beachtete Aspekt administrativer Domänen wird in diesem Konzept ausführlich betrachtet. Über das entsprechende Framework in Abschnitt 5.5.2 können Domänen und Beziehungen zwischen Domänen modelliert werden. Sie stellen ebenfalls ein zentrales Kriterium zur Zugriffssteuerung auf Informationselemente dar (vgl. Abschnitt 5.5.5.1). Darüber hinaus wird in diesem Konzept ein Lösungsansatz zur Auflösung von Domänenüberschneidungen in Abschnitt 5.5.2.3 und der effizienten Bestimmung von Domänenüberschneidungen vorgestellt. Bisherige Managementplattformen vernachlässigen auch diesen grundlegenden Aspekt bisher.

- **Betriebs- und Managementkonzepte.** Bisher bestehende Managementplattformen erlauben wenig Flexibilität in der Organisation des Netzmanagements. In diesem Konzept wird ein Ansatz zur Modellierung einer föderationsweiten Organisationsstruktur sowie einer darin feingranulareren domänenweiten Organisationsstruktur beschrieben (vgl. Abschnitt 5.5.4.2). Auf diese Weise können Gegebenheiten wie die Größe des Administratorenteams in einer Domäne oder der von einer Partei verfolgte Managementprinzip besser im Management der föderierten IT-Infrastruktur unterstützt werden.

- **Entscheidungsauswirkungen.** Durch das in Abschnitt 5.4.2 beschriebene Framework für Managementbeziehungen können beliebige Abhängigkeiten zwischen MOs modelliert werden. In Verbindung mit dem Framework für Föderationen aus Abschnitt 5.5.1 können so Auswirkungen lokaler Managemententscheidungen, wie beispielsweise die Abschaltung eines VM-Hosts, bereits vorab bewertet oder beispielsweise auch verhindert werden, sofern sie über die Managementplattform direkt umgesetzt werden. Beispielsweise kann eine Überprüfung auf noch bestehende Abhängigkeiten von einer IT-Ressource auf der Managementplattform die Ausführung der Abschaltung eines VM-Hosts verhindern.

- **Aufteilung von Ressourcen.** Die Aufteilung von Ressourcen wird generell über administrative Domänen unterstützt. Eine administrative Domäne umfasst einen Satz an IT-Ressourcen bzw. MOs, Nutzern und kann gar eine eigene Organisationsstruktur haben, wie im Aspekt der *Betriebs- und Managementkonzepte* beschrieben wurde.

Kapitel 6

Prototypische Implementierung

Inhalt

6.1 Festlegung des Implementierungsrahmens 220
6.2 Grundlegendes zur Implementierung 221
6.2.1 Ausgangsbasis der Implementierung 221
6.2.2 Allgemeine Umsetzung und Struktur 221
6.3 Frameworks des Informationsmodells 221
6.3.1 MOC-Framework 222
6.3.2 Framework für Managementbeziehungen 223
6.3.3 Framework für MOC-Funktionen 224
6.3.4 Framework für Netzereignisse und -Zustände 225
6.3.5 Framework für Alarme 225
6.4 Frameworks des Organisationsmodells 226
6.4.1 Framework für Föderationsmodelle und -Beziehungen 226
6.4.2 Framework für administrative Domänen 226
6.4.3 Nutzer- und Rollen-Framework 227
6.4.4 Framework für Managementaufgaben 227
6.4.5 Gültigkeitsbereiche und Zuständigkeiten 228
6.4.6 Mandantenfähiger Zugriff auf Informationselemente 228
6.5 Frameworks des Kommunikationsmodells 230
6.5.1 Southbound-Interface 230
6.5.2 Northbound-Interface 233
6.6 Frameworks des Funktionsmodells 236
6.6.1 Funktionserweiterung und Automatisierung der Kernprozesse 236
6.6.2 Funktionale Managementbereiche 237

Im vorherigen Kapitel 5 wurden Frameworkkonzepte zur Beschreibung notwendiger Informationen und zur Unterstützung zentraler Prozesse für Managementplattformen in FSNs vorgestellt und in einen zusammenhängenden Gesamtkontext gesetzt. In diesem Kapitel wird eine prototypische Implementierung ausgewählter Teile der Frameworks beschrieben. Eine begründete Auswahl des Implementierungsrahmens findet im nächsten Abschnitt statt. Allgemeine Designentscheidungen der Implementierung werden in Abschnitt 6.2 dargelegt. Im Anschluss werden Besonderheiten der Implementierung der Frameworkkonzepte beschrieben.

6.1 Festlegung des Implementierungsrahmens

Eine vollumfassende Implementierung aller Frameworks überschreitet den Rahmen dieser Forschungsarbeit. Insbesondere szenarienabhängig austauschbare, aber gleichwertige Teile sind ohne signifikante Aussagekraft für die Evaluierung des Gesamtkonzepts und werden daher in der Implementierung nicht berücksichtigt. Der Fokus der Implementierung liegt vielmehr auf Frameworkkomponenten, die eine besondere Bedeutung für die beispielhafte Anwendung in Kapitel 7 und Evaluation in Kapitel 8 haben.

Die Frameworks des **Informationsmodells** werden weitestgehend vollständig prototypisch implementiert. Lediglich die Trennung von Ist- und Soll-Zustand wird bei der Implementierung nicht berücksichtigt, da die Abbildbarkeit des tatsächlichen Zustands der gemanagten Infrastruktur ebenfalls die Abbildbarkeit eines gewünschten Zustands impliziert. Dieser Aspekt wird darüber hinaus bereits in bestehenden Konzepten wie YANG und NETCONF (vgl. Abschnitt 4.2.2) oder auch OpenDaylight (siehe Abschnitt 4.3.1) in der Praxis eingesetzt.

Die Frameworks des **Organisationsmodells** sind ein wesentlicher Bestandteil des Konzepts. Dadurch, dass sie unterschiedliche organisatorische Ausprägungen von Föderationen in SNs beschreiben müssen, sind sie für die Evaluation besonders wichtig und werden vollständig implementiert. Die Abbildung organisatorischer Aspekte von Föderationen wird in bestehenden Managementplattformen zudem nur unzureichend betrachtet, wodurch die Notwendigkeit der Implementierung und Evaluierung an dieser Stelle bekräftigt wird.

Die Frameworks aus dem **Kommunikationsmodell** beschreiben in dieser Arbeit insbesondere einen mit den Frameworks der anderen Modelle vollintegrierten Ansatz, sind für sich genommen jedoch nicht spezifisch für das Management von FSNs. Dem Kommunikationsmodell kommt jedoch eine besondere Bedeutung für eine quasi-echtzeitfähige Automatisierung der Überwachung und Steuerung zu. Der Fokus der Implementierung im Kommunikationsmodell liegt daher auf

- einer prototypischen Implementierung der Strukturen und des Prozesses zur Verarbeitung von am SBI empfangenen Daten aus der gemanagten Infrastruktur,
- der Umsetzung von Push-Benachrichtigungen und Pull-Zugriffen (hier jedoch ohne Feedback-Prozess zum SBI) am NBI, welche auch die Basis für eine Kommunikation im Managementplattform-Verbund bildet,
- sowie der Beschreibung von föderationsspezifischen Komponenten wie beispielsweise des Verbundselements oder den in diesem Kontext eingeführten Privilegienebenen.

Die Datenbankanbindung ist nicht Teil der Implementierung des Kommunikationsmodells.

Die Implementierung der Frameworks des **Funktionsmodells** fokussiert sich auf die Erweiterbarkeit der Managementplattformfunktionalität durch Plattformfunktionen und funktionale Managementbereiche als organisatorische Strukturen. Komponenten der automatisierten Überwachung und Steuerung werden bereits im Rahmen der anderen Modelle wie dem Kommunikationsmodell implementiert. Modelle zur Beschreibung und Verwaltung von Managementanwendungen werden als Teil des Informationsmodells implementiert.

Die im Konzept beschriebenen **modellübergreifenden Komponenten** werden in der Prototypen-Implementierung in den meisten Frameworks benötigt. Sie werden ebenfalls implementiert, jedoch in diesem Kapitel nicht explizit beschrieben, da ihr Beitrag zum Konzept nicht ausschlaggebend ist.

6.2 Grundlegendes zur Implementierung

Durch den hohen Detailgrad der konzeptionellen Beschreibung der einzelnen Frameworks in Kapitel 5 kann die Implementierung der Komponenten darin an vielen Stellen direkt wie beschrieben umgesetzt werden. Der Schwerpunkt der Beschreibung der Implementierung in den folgenden Abschnitten liegt entsprechend auf Aspekten, die im Konzept nur abstrakt dargestellt werden.

6.2.1 Ausgangsbasis der Implementierung

Um auch die praktische Nutzbarkeit der Prototypen-Implementierung der in dieser Arbeit beschriebenen Frameworks für Managementplattformen in FSNs hervorzuheben, werden sie nicht als von bestehenden Ansätzen entkoppelte Lösung implementiert. Vielmehr werden sie auf Basis der Entwicklungsumgebung einer bestehenden Managementplattform umgesetzt: **ONOS** stellt beispielsweise eine geeignete Trägeranwendung dar. Wie in Abschnitt 4.3.2 im Detail erläutert wird, basiert ONOS auf Apache Karaf und zeichnet sich durch eine gute Erweiterbarkeit aus. Des Weiteren bietet es eine genaue Dokumentation zur Einrichtung der Entwicklungsumgebung mit dem zum Zeitpunkt der Implementierung unterstützen Release 11 des **Java Development Kits** (JDK) [203]. Der Quelltext des ONOS-Projekts ist auf der Plattform Github[XVII] frei zugänglich und wird als Implementierungsbasis in Version 2.6 verwendet. ONOS selbst ist unter der Apache-Lizenz 2.0 veröffentlicht und daher ohne spezielle Einschränkungen als Trägeranwendung nutzbar.

6.2.2 Allgemeine Umsetzung und Struktur

Da ONOS als Basisplattform genutzt wird, werden die in dieser Arbeit beschriebenen Frameworks in der dafür verwendeten Werkzeugumgebung implementiert. Das gemäß [203] genutzte Werkzeug zur Verwaltung von abhängigen Softwarebibliotheken und zum automatisierten Bauen des Projekts ist **Bazel**. Die Prototypen der einzelnen Frameworks werden daher in einem eigenen Verzeichnis `onos/federationframeworks` neben den unterschiedlichen ONOS-Teilprojekten nach zugeordnetem Modell analog zur Strukturierung des Konzepts erstellt. Eine Übersicht der Verzeichnisstruktur ist in Abbildung 6.1 dargestellt. Im Build-Tool wird die Trennung der einzelnen Frameworks insofern berücksichtigt, dass jedes als zunächst eigenständiges Bazel-Projekt mit einer jeweils eigenen „BUILD“-Spezifikationsdatei im Frameworkverzeichnis versehen wird. Die BUILD-Datei spezifiziert, wie ein Softwareprojekt gebaut werden soll und welche Abhängigkeiten es hat. Auch wenn die Namen der Frameworkkomponenten im Konzeptkapitel zum besseren Verständnis des Zwecks des jeweiligen Elements weitestgehend deutschsprachig sind, werden die Frameworks ausschließlich englischsprachig implementiert und dokumentiert. Auf diese Weise wird die spätere Verwendbarkeit der prototypischen Implementierung – beispielsweise in anschließenden Forschungsarbeiten – weiter erhöht. Generell wird bei der Implementierung auf eine geeignete Fehlerbehandlung und eine weitestgehend vollständige Dokumentation der Klassen geachtet, auf welche in den folgenden Abschnitten jedoch nicht im Detail eingegangen wird.

6.3 Frameworks des Informationsmodells

Die Frameworks des Informationsmodells stehen jeweils für sich und sind mit Ausnahme von Abhängigkeiten untereinander auch unabhängig voneinander verwendbar. Auf diese Weise können die Frameworks auch selektiv in bestehenden Managementplattformen oder weiterführenden Konzepten genutzt werden. Frameworks binden jedoch teilweise Komponenten anderer Frameworks dieser Arbeit ein, um das im vorherigen Kapitel beschriebene Konzept als Gesamtes darzustellen.

[XVII] `https://github.com/opennetworkinglab/onos`

```
onos ........................... ONOS und die ONOS-Entwicklungsumgebung
                                 dienen als übergeordnetes Projekt.

  federationframeworks ...... Die eigentliche Framework-Implementierung
                              ist nach den vier Teilmodellen
                              strukturiert.
    infomodel
      alarms
      dependencies
      mocfunctions
      mocs
      netinfo
    commmodel
      crossinterface
      database
      northboundinterface
      southboundinterface
    orgamodel
      access
      domains
      federations
      responsibilities
      tasks
      users
    funcmodel
      managementfunction
      platformfunction
    inter
  …............................. Andere Unterverzeichnisse des ONOS-
                                 Projekts.
```

Abbildung 6.1: Verzeichnisstruktur der Implementierung und Einbettung in ONOS.

6.3.1 MOC-Framework

Die Umsetzung des MOC-Frameworks ist sehr nah an der vorgeschlagenen konzeptionellen Komponentenarchitektur aus Abschnitt 5.4.1.1 gehalten. Das gilt sowohl für die eigentlichen MOCs (`FSNMOC` und davon abgeleitete Klassen) als auch für die Klassifikatorklassen, welche von der Klasse `FSNMOCK` abgeleitet werden. Lediglich der Dienstzugriffspunkt kann in der Implementierung nicht direkt von einer `FSNMOC`-Klasse referenziert werden, da sonst eine zyklische Abhängigkeit zwischen dem MOC-Framework und dem Framework für MOC-Funktionen entsteht. Zur Lösung wird eine zusätzliche Klasse `MOCToServicePointAssociation` im Framework für MOC-Funktionen implementiert, die einen Dienstzugriffspunkt mit einem MOC verknüpft. Die Attribut-Klassen `Attribute` und `Complex` wurden ebenfalls dem Konzept entsprechend umgesetzt.

Eine Besonderheit im MOC-Framework ist die Notwendigkeit zur Implementierung einer zusätzlichen, nicht im Konzept spezifizierten Komponente zur effizienten Verwaltung von Klassifikationen von MOs. Die hier `ClassificationRegister` genannte Klasse ist im Singleton-Muster implementiert und erlaubt die Abfrage der Klassifikationen eines bestimmten MOs. Zu diesem Zweck werden, wie in Listing 6.1 beschrieben ist, alle MO-repräsentierenden `FSNMOC`-Instanzen anhand ihrer eindeutigen ID im `ClassificationRegister` registriert (Feld `classificationRegister`). Für jede `FSNMOC`-Instanz wird wiederum der Satz angewendeter Klassifikator-Elemente (vgl. Abbildung 5.7 im Konzeptkapitel) referenziert (Feld `idToMOMapping`). Auf diese Weise kann in der Managementplattform über eine gültige ID auf

die damit attribuierte FSNMOC-Instanz und damit auf deren Klassifikator-Elemente zugegriffen werden. Diese Funktionalität ist beispielsweise am SBI notwendig, um ein MO im Modell zu identifizieren, oder auch für Pull-Zugriffe am NBI, um eine MOC-Funktion eines bestimmten MOs anhand dessen ID zu identifizieren.

```
1  /**
2  * Helper structure for search operations mo <--> classifications.
3  * It also allows getting an MO instance by its ID.
4  */
5  public class ClassificationRegister {
6      /**
7      * For singleton
8      */
9      private static ClassificationRegister instanceOfThis;
10
11     /**
12     * Get classifications of an MO in an efficient way.
13     */
14     private Map<FSNMOC, Set<Classifier>> classificationRegister;
15
16     /**
17     * A mapping of an MO's ID to the actual object.
18     */
19     private Map<Identificator, FSNMOC> idToMOMapping;
20
21     /**
22     * Functions for retrieving the central instance of the
23     * classification register.
24     * @return Instance of the {@link ClassificationRegister}
25     */
26     public static ClassificationRegister getInstance() {
27         if (instanceOfThis == null) {
28             instanceOfThis = new ClassificationRegister();
29         }
30         return instanceOfThis;
31     }
32
33     // Public functions for MO registration and lookups...
```

Listing 6.1: Ausschnitte der Klasse ClassificationRegister mit Fokus auf der Singleton-Implementierung und Verwaltung von FSNMOC-Instanzen mit jeweiligen Klassifikation-Elementen.

6.3.2 Framework für Managementbeziehungen

Ähnlich wie das MOC-Framework lassen sich auch die Klassen des Frameworks für Managementbeziehungen durch einen objektorientierten Ansatz nah am Konzept implementieren. Lediglich für die Verwaltung der Abhängigkeiten und der damit verbundenen Funktionalität zum Traversieren der durch das Framework erstellbaren Abhängigkeitenstrukturen (vgl. Abschnitt 5.4.2.2 im Konzept) wurde zusätzlich eine übergeordnete zentrale Managementstruktur, die Klasse DependencyManager, implementiert. Jede erstellte Abhängigkeit zwischen zwei FSNMOC-Instanzen muss entsprechend in einer in der Managementplattform zentralen Instanz dieser Klasse registriert und bei Entfernen einer Abhängigkeit analog ausgetragen werden. Die DependencyManager-Klasse ist im Singleton-Muster implementiert.

Die Kernelemente in der DependencyManager-Klasse sind, wie in Listing 6.2 gezeigt ist, zwei Map-Elemente, die von FSNMOC-Instanzen auf jeweils eine weitere Map verweisen. Letztere bildet den Klassentyp der Ableitung einer Dependency (die Implementierung der Abhängigkeit-Klasse im Konzept in Abschnitt 5.4.2.2) auf die entsprechenden Instanzen desselben Klassentyps ab. Auf diese Weise dient die jeweils zweite referenzierte Map als Filter für einen laufzeiteffizienten Zugriff.

```java
/**
 * Manages and registers all dependencies ({@link Dependency}) and
 * especially supports the functions of the different types of dependencies.
 */
public class DependencyManager {
    // MOC --> Class<? extends Abhängigkeit> --> Abhängigkeit
    /**
     * Central structure for holding dependencies where an FSNMOC is
     * the subject.
     */
    private Map<FSNMOC, Map<Class<? extends Dependency>, ArrayList<Dependency>>>
        subjektDeps;

    // MOC --> Class<? extends Abhängigkeit> --> Abhängigkeit
    /**
     * Central structure for holding all dependencies
     * for the object FSNMOCs
     */
    private Map<FSNMOC, Map<Class<? extends Dependency>, ArrayList<Dependency>>>
        objektDeps;

    // Functions follow below...
```

Listing 6.2: Speicherstrukturen der Klasse `DependencyManager` zur Verwaltung von Abhängigkeiten-Instanzen zwischen MOs.

6.3.3 Framework für MOC-Funktionen

Die Implementierung des Frameworks für MOC-Funktionen basiert auf dem Framework zur Erweiterung der Managementplattformfunktionalität. Dieses wird zum Zweck der Übersichtlichkeit ebenfalls in diesem Abschnitt beschrieben. Dabei handelt es sich vor allem um die Klasse `PlatformFunction` und Klassen zur Beschreibung von Parameter- und Rückgabe-Typen und -Werten `FunctionType` und `FunctionValue`, da diese bereits in der Klasse `PlatformFunction` genutzt werden müssen.

Die Umsetzung der Klassen `MOCFunction`, `MOCFunctionReal` und `MOCFunctionVirtual` können in einem objektorientierten Ansatz weitestgehend wie im Konzept beschrieben implementiert werden. Die **Treiberfunktionen** `MOCFRDriver` und `MOCFVDriver` für reale und virtuelle MOC-Funktionen werden analog zu MOC-Funktionen in einer Vererbungshierarchie strukturiert umgesetzt und von dem Treiber für Plattformfunktionen `PlatformFunctionDriver` abgeleitet. Alle Treiberfunktionen sind nicht wie im Konzept beschrieben als Klassen, sondern als Java-Interfaces implementiert, da sie so mehr ihrer Funktion entsprechen. Die allgemeinste Implementierung eines Treibers ist die von Managementplattformfunktionen (d. h. `PlatformFunctionDriver`). Sie beschreibt nur die abstrakte Methode `execute` mit Parametern und Rückgabewerten, die gemäß dem Konzept über benannte `Map`-Elemente umgesetzt werden. In der Implementierung erweitert der Treiber für virtuelle MOC-Funktionen `MOCFVDriver` den für Plattformfunktionen um die Möglichkeit, als Kontext des Funktionsaufrufs ein MO der gemanagten Infrastruktur (bzw. eine `FSNMOC`-Instanz) zu setzen und abzufragen. Auf die MO-Modellrepräsentation wird in der Managementplattform selbst die jeweilige MOC-Funktion angewendet (z. B. das Setzen der Beschreibung eines MOs). Die Treiberklasse für reale MOC-Funktionen `MOCFRDriver` erweitert die der virtuellen MOC-Funktionstreiber um Funktionen zum Setzen und Holen eines konkreten Dienstzugriffspunkts (Klasse `ServicePoint`), da die jeweilige Funktion im Gegensatz zu virtuellen MOC-Funktionen auf dem MO selbst ausgeführt werden und nicht nur auf dessen Modellrepräsentation. Darüber hinaus beschreibt `MOCFRDriver` Funktionen zum Umgang mit Parametern, um den Verarbeitungsprozess am SBI zu steuern (vgl. Abschnitt 5.6.1.3 des Konzepts): Beispielsweise den gewünschten Verarbeitungsgrad, die Möglichkeit zum Bypass am Speicher vorbei oder die Priorisierung einer Anfrage. Im Konzept sind diese Elemente als Parameter der `ausführen`-Methode (siehe Lis-

ting 5.5) beschrieben. Durch die eingeführte Vererbungshierarchie bieten sich in der eigentlichen Implementierung jedoch dafür eigenständige Methoden neben der `execute`-Methode an.

Anders als bei Plattformfunktionen werden Treiber bei **MOC-Funktionen** nicht mit der eigentlichen Funktionsdefinition, sondern mit einem konkreten MOCK verknüpft. Daher wird, wie im Konzept in Abschnitt 5.4.3.2 beschrieben, eine Klasse `DriverMap` zur Treiberzuweisung zu einer MOCK-Klasse und der implementierten MOC-Funktion implementiert. Sie unterstützt Methoden zur dynamischen (De-) Registrierung und der effizienten Suche einer passenden Treiberklasse. Da `MOCFunction` von `PlatformFunction` abgeleitet ist, wird die Treiberinstanz im Konstruktor bei MOC-Funktionen durch eine Treiberattrappe `DummyFunctionDriver` versteckt. Eine Treiberfunktion kann dann nicht mehr direkt zugewiesen werden. Die `execute`-Methode der Treiberattrappe generiert bei Aufruf stets eine `UnsupportedOperationException` und ist daher nicht verwendbar.

Eine weitere Besonderheit ist die Umsetzung der Zuweisung einer MOC-Funktion zu einer bestimmten MOC-Klassifikation `FSNMOCK`-Instanz. Im Konzept, vor allem wie in Abbildung 5.12 gezeigt, wird direkt von einem `FSNMOCK`-Element auf die entsprechend diesen MOC-Klassifikator beschreibenden Elemente von `MOCFunktion`-Elementen verwiesen. In der hier umgesetzten prototypischen Implementierung ist das hingegen nicht möglich, da es ansonsten zu einer gegenseitigen **zyklischen Abhängigkeit** zwischen dem MOC-Framework und dem Framework für MOC-Funktionen kommt. Das MOC-Framework wird an dieser Stelle bereits zur Beschreibung der MOC-Funktionstreiber verwendet. In dieser Implementierung wurde dieses Problem mit der Einführung einer weiteren Klasse `MOCKToFunctionAssociation` gelöst, welche sich ebenfalls im Framework für MOC-Funktionen befindet und die Zuweisung von `FSNMOCK` zu einer Menge (`Set`-Interface in Java) von `MOCFunction` vornimmt. Zyklische Abhängigkeiten zwischen den Frameworks für MOCs und MOC-Funktionen werden so vermieden.

6.3.4 Framework für Netzereignisse und -Zustände

Das Framework für Netzereignisse und -Zustände weicht in der Implementierung nicht wesentlich von dem im Konzept vorgeschlagenen objektorientierten Ansatz ab. Es ist insbesondere abhängig von modellübergreifenden Komponenten wie Zeitstempeln und Identifikatoren. Neben inhaltlichen Attributen referenziert eine Netzinformation auf eine Menge (`Set`) an Aufgabenbereichen `TaskDomain` aus dem Framework für Managementaufgaben (in Abschnitt 6.4.4).

6.3.5 Framework für Alarme

Die Implementierung des Frameworks für Alarme besteht aus lediglich zwei Kernklassen: Zum einen die Klasse `AlertType`, die gemäß Konzept (siehe Abschnitt 5.4.5) einen Alarm implementiert, sowie eine benutzerdefinierbare Aufzählung `AlertCategory`, durch die Alarmkategorien implementiert werden. Die Klasse `AlertType` ist als abstrakte Klasse implementiert und konkrete Alarme müssen, wie im Konzept beschrieben wurde, als Hierarchie davon abgeleitet werden. Die Klasse `AlertType` definiert darüber hinaus eine abstrakte Methode `String serialize()` zur Serialisierung und eine abstrakte Methode `void deserialize(String serialized)` zur Deserialisierung eines Alarms. Beide Methoden müssen abhängig von der jeweiligen Spezialisierung eines Alarms durch Entwickler implementiert werden. Die Methode `deserialize` ist auf ein bereits instanziiertes Objekt anwendbar: Sie belegt die Felder des Objekts mit den über den Parameter `serialized` beschriebenen Zustand des Alarms. Dafür müssen die Setter-Methoden der Attribute eines Alarms mindestens die Zugriffsklasse *protected* aufweisen, damit auf sie durch die Deserialisierungsmethode zugegriffen werden kann.

6.4 Frameworks des Organisationsmodells

Große Teile der Frameworks im Organisationsmodell beschreiben Komponenten zur Modellierung der Umgebung und organisatorischer Gegebenheiten in FSNs. Diese lassen sich weitestgehend, wie sie im Konzept beschrieben wurden, implementieren. Dieser Abschnitt zur Beschreibung der Implementierung fokussiert sich daher auf Aspekte, die auf abstrakterer Ebene im Konzept beschrieben wurden oder die im praktischen Konzept für ein Funktionieren in der Praxis erweitert werden mussten.

6.4.1 Framework für Föderationsmodelle und -Beziehungen

Für die Organisation der Föderation und aller Modellelemente darin dient im Konzept wie in der Implementierung das Framework für **Föderationsmodelle** und **Föderationsbeziehungen** als zentrale Stelle der Modellierung der Gesamtumgebung. Die Klasse `Federation` erfüllt im Allgemeinen diesen zentralen Zweck, mit Ausnahme der für die Prototypen-Implementierung teilweise eingesetzten, über das Singleton-Muster realisierten Hilfsstrukturen wie dem zuvor im Informationsmodell beschriebenen `DependencyManager` oder `ClassificationRegister`. Da es in einer Managementplattform-Implementierung immer nur ein `Federation`-Element gibt, ist auch dieses im Singleton-Muster implementiert und referenziert neben den im Konzept bereits genannten Elementen (insb. Datenzentren, Parteien, Aufgabenbereiche, die Kreuzorganisationstabelle usw., vgl. Abschnitt 5.5.1.1) auch das zentrale `ClassificationRegister`- und `DependencyManager`-Element in der Föderation. Auch wenn die zentrale Referenz auf diese in einem ganzheitlichen Ansatz (unter Verwendung aller Frameworks) in der ebenfalls im Singleton-Muster implementierten `Federation`-Klasse die Notwendigkeit der Implementierung der beiden genannten Hilfsstrukturen im Singleton-Muster obsolet macht, so ist sie in einem selektiven Ansatz (z. B. werden nur das MOC-Framework oder Managementbeziehungsframework genutzt) notwendig, da die Frameworks zunächst alle für sich stehen und einzeln einsetzbar sein sollen.

6.4.2 Framework für administrative Domänen

Das Framework für administrative Domänen besteht im Wesentlichen aus drei miteinander verknüpften Teilen:

- Die Komponenten zur Beschreibung von Domänen selbst.
- Elemente zur Beschreibung von Domänenbeziehungen.
- Die Domänen-Kreuzorganisationstabelle (DKOT) zur Behandlung von Domänenüberschneidungen.

Die ersten zwei Teilframeworks lassen sich, gemäß dem Konzept in den Abschnitt 5.5.2.1 und 5.5.2.2 umsetzen. Daher fokussiert sich die Beschreibung der Implementierung in diesem Abschnitt insbesondere auf die Umsetzung der **Domänen-Kreuzorganisationstabelle**. Die Kernkomponente der Domänen-Kreuzorganisationstabelle – implementiert durch die Klasse `DomainCrossOrganizationTable` – ist das DKOE-Register `DCOERegister` zur Verwaltung von Domänenüberschneidungen. Dieses wurde auch weitestgehend gemäß der Beschreibung in Abschnitt 5.5.2.3 implementiert. Eine in der Implementierung neu eingeführte Methode dieser Klasse ist

```
public void updateDCOE(DomainCrossOrganizationEntry e){ ... },
```

die einen Eintrag zur Beschreibung einer Überlappung, einen `DomainCrossOrganizationEntry`, einen neuen DKOE-Index `DCOEIndex` auf Basis aller registrierter Domänen generiert.

Die Funktion wird verwendet, wenn sich ein `DomainCrossOrganizationEntry` ändert, beispielsweise eine weitere Domäne in die Überschneidung aufgenommen oder eine Domäne daraus entfernt wird. Darüber hinaus wurde eine weitere Methode

```
private void updateAllDCOEs(){ ... }
```

in die prototypische Implementierung aufgenommen, durch die die DKOE-Indizes aller Domänenüberschneidungen neu berechnet wird. Die Methode wird aufgerufen, wenn sich der Satz an in der Föderation grundsätzlich registrierter Domänen ändert. Auf diese Weise werden mögliche Inkonsistenzen bei der Verwaltung der Domänenüberschneidungen, bei gleichzeitig absehbar geringem Rechenaufwand, sichergestellt.

Im Konzept der DKOT wurde zudem nicht detailliert auf notwendige Funktionen eingegangen. Die DKOT weist grundsätzlich jedem MO, das zu mehr als einer Domäne gehört, den jeweiligen DKOE zur Modellierung der Domänenüberschneidung zu. Wesentliche Methoden dazu umfassen das Registrieren eines neuen MO mit Überschneidung, das Entfernen eines MO aus der Liste (falls eine Überschneidung aufgelöst wurde) und die Aktualisierung einer Überschneidung. Jede dieser Operationen muss die Konsistenz der Daten sicherstellen, d. h. prüfen, ob referenzierte Domänen, DKOEs und das MO existieren. Auch wird beim Entfernen von Elementen wie `Domain`-Instanzen geprüft, ob diese nicht noch in einem DKOE in Verwendung sind. Letzteres wird insbesondere durch die Methode `DCOERegister.rmDomain(Domain d)` abgedeckt, welche auch über die DKOT zugreifbar gemacht wird. Als Hilfsfunktion dazu dient die ebenfalls in der `DCOERegister`-Klasse definierte Methode

```
public Set<DomainCrossOrganizationEntry> findDomainOverlaps(final
                        Set<Domain> ds)
```

zur Ermittlung von Überschneidungen der Domänen `ds`. Diese wird mit der zu löschenden Methode aufgerufen. Fall die Methode keine leere Menge zurückgibt, kann die entsprechende zu löschende Domäne nicht entfernt werden, da sie in den ermittelten DKOEs verwendet wird. Die Implementierung von Domänenbeziehungen kann direkt, wie es in Abschnitt 5.5.2.2 beschrieben wurde, umgesetzt werden.

6.4.3 Nutzer- und Rollen-Framework

Das Framework für Nutzer und Rollen lässt sich wie im Konzeptkapitel beschrieben implementieren. Neben modellübergreifenden Komponenten hängt das Framework insbesondere vom NBI ab. Aus diesem wird der in Abschnitt 5.6.2 eingeführte Privilegienbaustein – welcher über die Klasse `PrivilegeComponent` implementiert wird – genutzt, um die Privilegienebene der jeweiligen Nutzerentität zu beschreiben. Wird bei der Instanziierung einer Entität (`Entity`-Klasse) keine Privilegienebene explizit angegeben, so wird ihr als Default-Wert die niedrigste Ebene *NORMAL* zugewiesen.

6.4.4 Framework für Managementaufgaben

Besonderheiten bei der Umsetzung des `Federation`-Elements in diesem Framework ist die Eindeutigkeit von Aufgabenbereichen, d. h. `TaskDomain`-Elementen, und Aufgaben entsprechenden `Task`-Elementen in der Modellierung einer Föderation. Dazu werden `TaskDomain`-Instanzen in der Föderation über eine `Map` ausgehend von ihrem Namen abgebildet. Gleiches gilt innerhalb eines Aufgabenbereichs für darin registrierte `Task`-Instanzen. Auf diese Weise gibt es in der Föderation keine Aufgabenbereiche, genauso wie innerhalb eines Aufgabenbereichs keine Aufgaben mit demselben Namen.

6.4.5 Gültigkeitsbereiche und Zuständigkeiten

Die zwei unterschiedlichen in dieser Arbeit betrachteten Gültigkeitsbereiche sind einerseits *a)* die Föderation als Gesamtes sowie andererseits *b)* domänenlokale Bereiche. Die Implementierung von **Zuständigkeiten** in einem Gültigkeitsbereich wird über das wie in Listing 6.3 gezeigt implementierte `Responsibility`-Element realisiert. Wie im Konzept beschrieben wird, verknüpft es eine zu parametrisierende Rollenklasse vom Typ `Class<T>` mit einem Aufgabenelement des Typs `Task` und impliziert somit die Zuständigkeit dieses Rollentyps für die spezifizierte Aufgabe.

```
1  public class Responsibility<T extends Role> {
2
3      /**
4      * The role, the task of this responsibility
5      * is assigned to.
6      */
7      private Class<T> role;
8
9      /**
10     * The task, this' responsibility's role
11     * is assigned to.
12     */
13     private Task task;
14
15     // Constructor, Getter, Setter...
16 }
```

Listing 6.3: Attribute der Klasse `Responsibility`.

Auf eine unmittelbare Referenzierung auf `Responsibility`-Instanzen von den Klassen `Federation` und `Domain` wurde in der Implementierung zugunsten der Vermeidung von **zyklischen Abhängigkeiten** verzichtet. Vielmehr wurde eine zusätzliche Struktur `ResponsibilitiesMap` implementiert, welche zentral die Verknüpfung von Domänen und dem Föderationselement mit den jeweiligen entsprechend lokalen bzw. globalen Zuständigkeiten organisiert. `Responsibility`-Instanzen für beide Fälle müssten daher im entsprechenden Kontext in einer Instanz dieser Klasse registriert werden. Auch übernimmt die `ResponsibilitiesMap` die Sicherstellung der Eindeutigkeit der Zuständigkeiten – d. h. eine Aufgabe darf im selben Gültigkeitsbereich nicht durch unterschiedliche Rollen abgedeckt werden. Beim Hinzufügen einer neuen Instanz prüft die Klasse daher, ob eine Zuständigkeit für die entsprechende Aufgabe bereits existiert und generiert in diesem Fall eine `ResponsibilityAlreadyExistsException`-Instanz.

6.4.6 Mandantenfähiger Zugriff auf Informationselemente

Das zentrale Element zur Steuerung des Zugriffs und der Sichtbarkeit von Informationselementen ist in diesem Konzept der Zugriffsverwaltungsbaum (ZVB). Dieser wird über die abstrakte Klasse `AccessManagementTree` implementiert.

6.4.6.1 Zugriffsverwaltungsbaum

Die Umsetzung der **Kernmethoden** der Klasse `AccessManagementTree` ist wesentlich vom jeweiligen Einsatzszenario abhängig. Die durch sie beschriebenen abstrakten Methoden umfassen die Funktionalität zur Ermittlung der Sichtbarkeit und Zugreifbarkeit auf die jeweiligen Informationselemente-Klassen:

- `getAccessibleMOs(Entity e, Domain context)` gibt für eine Nutzerentität zugreifbare MOs zurück.

- `getAccessibleAttributes(Entity e, FSNMOC mo, Domain context)` ermittelt zugreifbare Attribute eines MOs für eine Nutzerentität.
- `getAccessibleFunctions(Entity e, FSNMOC mo, Domain context)` ermittelt die zugreifbare MOC-Funktionen eines MOs für einen Nutzer.
- `getAccessibleNetworkInformation(Entity e, Domain context)` ermittelt für einen Nutzer zugreifbare Netzinformationen.
- `getAccessibleMODependencies(Entity e, FSNMOC mo, Domain context)` ermittelt die für einen Nutzer zugreifbaren Managementbeziehungen für ein MO.
- `getAllowedInformationTags(Entity e, Domain context)` ermittelt alle `InformationTag`-Instanzen, die für eine Nutzerentität im Kontext einer Domäne `context` *erlaubt* sind. Informationselemente, die durch diese Tags markiert sind, sind für die Nutzerentität sichtbar. Die Methode kann als Basisimplementierung für die zuvor genannten Methoden für spezifische Informationselemente genutzt werden.

Bei Anwendung dieses Konzepts und der szenarienabhängigen Implementierung der genannten abstrakten Methoden ist zu beachten, dass Einsicht in Kriterien der jeweils angefragten `Entity`- bzw. Nutzer-Instanz (z. B. für einen Nutzer relevante Föderationsbeziehungen) notwendig ist. Dazu wird die zentrale `Federation`-Instanz als Inventar aller Informationselemente genutzt. Die Ermittlung der zugreifbaren Elemente erfolgt stets in einem administrativen Kontext, d. h. einer übergebenen administrativen Domäne. Durch die Unterscheidung von Berechtigungen innerhalb einer Domäne und Berechtigungen, die durch Domänenbeziehungen entstehen, ist die Angabe der betrachteten Domäne als Kontext unerlässlich.

6.4.6.2 ZVB-Knoten, -Elemente und -Kriterien

Ein ZVB-Knoten ist in Form der ebenfalls abstrakten und generischen Klasse `AMTNode` implementiert, auf die in der `AccessManagementTree`-Klasse als Wurzelelement verwiesen wird. Verschiedenen **Arten der Verzweigung** des ZVB werden über entsprechende nicht-abstrakte Klassen implementiert: Einfach **seriell** durch `AMTNodeSerial` mit genau einem nachfolgenden Knoten, **verzweigt** durch `AMTNodeCases` mit einer Liste von nachfolgenden ZVB-Knoten oder als **Blattknoten** durch `AMTNodeLeaf` ohne Nachfolgerknoten.

Die **ZVB-Elemente** in einem ZVB-Knoten werden über die Klasse `AMTElement` realisiert. Da in jedem ZVB-Knoten ein homogener Satz an ZVB-Elementen enthalten ist, die in einer vorgegebenen Reihenfolge durchgegangen werden, wird jede `AMTNode`-Instanz über eine bestimmte ZVB-Elementklasse parametrisiert:

```
public abstract class AMTNode<T extends AMTElement> { ... }
```

Jede `AMTElement`-Instanz in einem `AMTNode` beschreibt ein bestimmtes **Kriterium** bei der Sichtbarkeits- und Zugriffsentscheidung – z. B. eine konkrete Klasse der Föderations- und Vertrauensbeziehung oder eine konkrete Klasse einer Aufgabenzuweisung. Da die in FSNs identifizierten Kriterien zur Sichtbarkeits- und Zugriffsentscheidung (siehe Abschnitt 5.5.5.2) aufgrund ihrer Unterschiedlichkeit keine gemeinsame Elternklasse haben, wird in der Implementierung die einheitliche Nutzung durch eine Interface-Klasse `Criterion` realisiert (vgl. Listing 6.4), um die vom jeweils genutzten Kriterium implementiert werden muss. Es definiert die Methode `getCriterionClass`, die den jeweiligen Typ des Kriteriums zurückgibt.

Die jeweilige **Operation**, die durch einen ZVB-Knoten, aber auch ein ZVB-Element angewendet wird – im Basissatz des Konzepts (jedoch nicht darauf beschränkt) *override*, *reduce*, *add* – ist

über die Enumeration `AMTOperation` realisiert. Zusätzlich eine *None*-Operation, die das vorherige Ergebnis ohne weitere Operation übernimmt. Die Klassen `AMTNode` und `AMTElement` referenzieren auf die jeweilige `AMTOperation`-Instanz.

```
1  /**
2   * This class allows a uniform handling of
3   * heterogeneous visibility and access criteria.
4   * @param <T> Type of criterion element
5   */
6  public interface Criterion<T> {
7      /**
8       * This method retrieves the type of an criterion element.
9       * @return Class of type of criterion element
10      */
11     Class<T> getCriterionClass();
12 }
```

Listing 6.4: Interface `Criterion` zur einheitlichen Anwendung von Sichtbarkeits- und Zugriffskriterien.

6.4.6.3 Zugriffsregelsatz

Den regelbestimmenden, den `AccessManagementTree` ergänzenden Teil, bildet der **Zugriffsregelsatz** `AccessRuleSet`. Er definiert die mit einem Kriterientyp jeweils verknüpften Tags. Dies wird für die prototypische Implementierung durch eine einfache Abbildung der Kriterien-Klasse auf eine einzelne Zugriffsregel `AccessRule` realisiert, welche ein Black- oder Whitelisting von Tags definiert (vgl. Abschnitt 5.5.5.4). Tags, wie sie in Abschnitt 5.3.2 beschrieben werden, werden über die generische Klasse `InformationTag` implementiert, sodass je nach Anwendungsfall geeignete Tags verwendet werden können.

6.5 Frameworks des Kommunikationsmodells

Wie zuvor in Abschnitt 6.1 beschrieben, liegt der Fokus der Implementierung im Kommunikationsmodell auf der Umsetzung des Prozesses zur Verarbeitung von am SBI empfangenen oder abgerufenen Daten der gemanagten IT-Infrastruktur und den Push- und Pull-Strukturen des NBI. Weitere Komponenten aus anderen Frameworks des Kommunikationsmodells wie frameworkübergreifende Komponenten werden gemäß ihrer Notwendigkeit für die Evaluierung umgesetzt. Die Datenbankanbindung ist nicht Fokus der Implementierung.

6.5.1 Southbound-Interface

Die Struktur des SBI-Frameworks orientiert sich an den unterschiedlichen Schritten in der Verarbeitung von Daten und Informationen im Gesamtprozess. Wie in Abbildung 6.2 gezeigt ist, erfolgt die Einteilung der Packages in der Implementierung gemäß dem jeweiligen Verarbeitungsschritt. Eine Besonderheit stellt die Implementierung des SBI-Registers dar, welche als Klasse `SBIRegister` direkt im Package „southboundinterface“ liegt, da es alle Komponenten zusammenführt und den Verarbeitungsprozess koordiniert.

6.5.1.1 Komponenten und Parallelisierung

Die Klasse `SBIRegister` stellt den zentralen Punkt der Koordinierung der Verarbeitung von Daten am SBI dar. Wie im Konzept in Abschnitt 5.6.1 beschrieben wird, referenziert es auch in der Implementierung auf den Netzadapter-, Parser- und Normalisierer-Pool, welche jeweils wie im Konzept beschrieben umgesetzt werden. Auch ist eine Referenzierung auf die drei Queue-Paare (drei nicht-priorisierte Queues sowie drei priorisierte Queues) notwendig. Die **Queues** werden einheitlich von durch in Java bereitgestellte `LinkedBlockingQueue`-Klassen implementiert, welche so lange den zugreifenden Thread blockieren, bis Elemente hinzugefügt – oh-

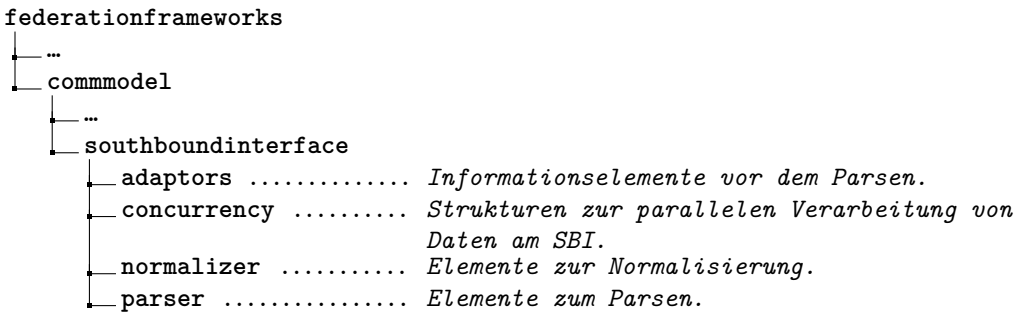


Abbildung 6.2: Paketstruktur der Southbound-Interface-Implementierung.

ne Beschränkung wie hier bis zu $2^{31} - 1$ Elemente – bzw. entfernt (d. h. Queue ist nicht leer) werden können [204]. Als Implementierung des Interface `BlockingQueue` sind diese zudem mit Threads sicher verwendbar [205]. Die vor den Queues angewendeten Priorisierungsregeln werden für jedes Queue-Paar separat in Form von Mengen (Interface `Set` in Java) hinterlegt.

Das SBI-Register referenziert auf alle drei **Worker-Pools** (für Parsen, die Normalisierung und Aktualisierung), welche in Java durch `ExecutorService`-Objekte und hier im Speziellen jeweils durch einen *Cached-Threadpool* implementiert sind. Diese sind praktisch selbstverwaltend: Sie erstellen nach Bedarf neue Threads, verwenden ebenfalls bereits bestehende automatisch wieder, und entfernen ungenutzte Threads nach 60 Sekunden aus dem Pool [206]. Im Kontext der später in dieser Arbeit vorgenommenen Evaluierung stellt diese Implementierung eine Baseline dar und erlaubt beispielsweise Variationen hinsichtlich der Einrichtung einer fixen Threadpool-Größe im Zusammenspiel mit Größen der Priorisierung der Bearbeitung.

6.5.1.2 Verarbeitungsprozess und Priorisierung

Der gesamte, wenn auch vereinfachte (da rein sequenziell dargestellte) **Prozess** der Verarbeitung von Daten am SBI wird in Abbildung 6.3 anhand eines zu verarbeitenden Elements gezeigt. Der initiale Auslöser des Prozesses ist das Anhängen eines `SBIExtRawData`-Objekts in die `ExtRawDataQueue`-Instanz, in diesem Fall durch ein Objekt der generischen Klasse `T` (in der Praxis üblicherweise ein Netzadapter). Die Methode `put(A e)` (`A` ist die Klasse eines Elements) der Elternklasse `DataQueue<T>` jeder der drei Queues löst in der jeweiligen Queue die zu implementierende Methode `notifyWorker()` aus, welche je nach Queue eine unterschiedliche notify-Methode aufruft. Aus der `ExtRawDataQueue`-Klasse wird die Methode `notifyParse()`, aus der `ParsedQueue` die Methode `notifyNormalize()` und aus der `NormalizedQueue`-Klasse die Methode `notifyUpdate()` im SBI-Register aufgerufen. Diese drei notify-Methoden des SBI-Registers können potenziell von mehreren Threads gleichzeitig aufgerufen werden und sind daher jeweils als `synchronized`-Block realisiert. Infolgedessen kann zu einem Zeitpunkt stets nur ein Thread diesen Block bearbeiten, wodurch Race-Conditions vermieden werden.

Die Methoden `notifyParse()`, `notifyNormalize()` und `notifyUpdate()` des SBI-Registers verwenden in dieser Implementierung alle denselben **Priorisierungsansatz**: Durch die Bereitstellung von Threads durch jeweils Javas Cached-Threadpool wird ohne Priorisierung stets ein Thread zur Bearbeitung eines Elements alloziert bzw. dafür generiert. Die Bearbeitung nicht-priorisierter Datencontainer können potenziell CPU-Ressourcen stark binden und daher die Bearbeitung priorisierter Datencontainer indirekt negativ beeinflussen. Der in der Implementierung verfolgte Ansatz der Priorisierung ist daher, bei jedem Aufruf der jeweiligen notify-Methode am SBI-Register zu überprüfen, ob Elemente in der jeweiligen priorisierten Queue liegen. Ist dies der Fall, werden alle diese Elemente abgearbeitet. Ist die jeweilige priorisierte Queue jedoch zum Zeitpunkt der Überprüfung leer, so wird in der nicht-priorisierten Queue ein definierter Anteil darin befindlicher Elemente abgearbeitet (z. B. 50%), bevor die dazugehöri-

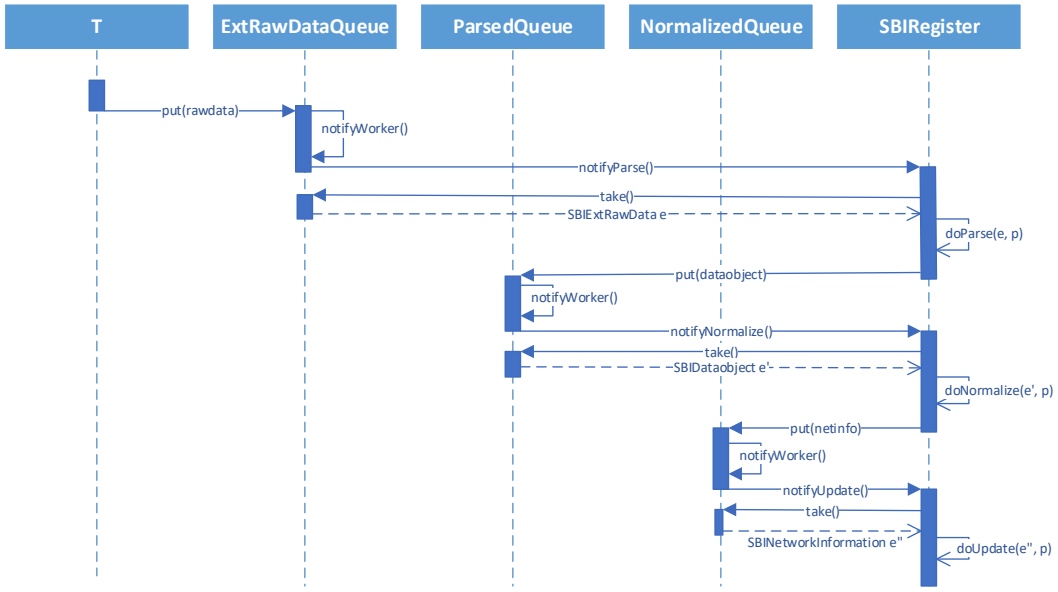


Abbildung 6.3: Vereinfachter grundlegender Ablauf des Prozesses zur Verarbeitung von Daten des SBI.

ge priorisierte Queue erneut auf Elemente überprüft wird. Listing 6.5 zeigt die Umsetzung der Priorisierung am Beispiel der Methode `notifyParse()`. Die Implementierung der Normalisierung und Aktualisierungen ist analog umgesetzt. Ein Austausch des Priorisierungsansatzes ist grundsätzlich anwendungsfallabhängig möglich.

```
1  public synchronized void notifyParse() {
2      // First, check if priority elements are still waiting and handle them
3      int prioCnt = extRawDataQueuePrio.size();
4      while (prioCnt > 0) {
5          try {
6              SBIExtRawData element = extRawDataQueuePrio.take();
7              doParse(element, true);
8          } catch (InterruptedException e) {
9              e.printStackTrace();
10         }
11         prioCnt--;
12     }
13     // Handle a defined amount (default = 50%) of non-priority elements.
14     int nonPrioCnt = (extRawDataQueue.size()*PRIORITY_RATIO);
15     while (nonPrioCnt > 0) {
16         try {
17             SBIExtRawData element = extRawDataQueue.take();
18             doParse(element, false);
19         } catch (InterruptedException e) {
20             e.printStackTrace();
21         }
22         nonPrioCnt--;
23     }
24 }
```

Listing 6.5: Implementierung der Priorisierung am Beispiel der Methode `notifyParse`.

6.5.1.3 Verarbeitung von Eingabe-Containern

Die **Verarbeitung** eines Eingabe-Datencontainers wird für den jeweiligen Schritt über die *do*-Methoden im SBI-Register umgesetzt, welche durch die zuvor beschriebenen notify-Methoden

des SBI-Registers aufgerufen werden (vgl. Abbildung 6.3). Die Funktionsköpfe sind in Listing 6.6 gezeigt. Da alle Methoden die aus dem Eingabe-Datencontainer erfolgreich generierten Ausgabe-Datencontainer direkt in die nächste Queue legen, haben sie keine Rückgabewerte. Der Parameter `isPrioritized` wird immer auf `true` gesetzt, sobald ein Element aus der priorisierten Queue entnommen wird. Dies erlaubt die automatische Einreihung der daraus abgeleiteten Ausgabe-Datencontainer in die nachfolgende priorisierte Queue ohne die Notwendigkeit der Anwendung der entsprechenden Priorisierungsregeln.

```
1 private void doParse(SBIExtRawData dataobj, boolean isPrioritized)
2 private void doNormalize(SBIDataobject dataobj, boolean isPrioritized)
3 private void doUpdate(SBINetworkInformation dataobj, boolean isPrio)
```

Listing 6.6: Im SBI-Register definierte Kernfunktionen zur Verarbeitung von jeweiligen Datencontainern.

Wie in Abbildung 6.3 auf der rechten Seite durch die asynchron durchgeführten Aufrufe der Funktionen `doParse(e, p)`, `doNormalize(e', p)` und `doUpdate(e'', p)` verdeutlicht wird, lösen diese eine **parallelisierte Bearbeitung** durch die Nutzung der drei Cached-Threadpools aus. Da jede Bearbeitung eines Eingabe-Datencontainers im positiven Fall in der Erstellung und dem Hinzufügen eines Ausgabe-Datencontainers in der jeweils nächsten Queue mündet, ist eine asynchrone Behandlung von Rückgabewerten nicht notwendig und die Verarbeitung kann einheitlich aus jeder Queue durchgeführt werden.

Da, wie zuvor in Abschnitt 6.1 beschrieben wurde, die Implementierung der Datenzugriffsverwaltung nicht im Fokus der Evaluation steht und daher nicht berücksichtigt wird, wird anstelle dieser die **Aktualisierung des Speichermodells** durch einen weiteren Pool von Aktualisierungsbausteinen ersetzt. Auf diese Weise kann in den in Kapitel 8 durchgeführten Performanzmessungen ein kompletter Durchlauf am SBI durchgeführt werden, auch wenn die Datenzugriffsverwaltung nicht implementiert wurde. Für jedes im Teilprozess der Normalisierung generierte `NetworkInformation`-Element in einer `SBINetworkInformation`-Instanz kann eine hier `UpdateComponent` genannte Klasse implementiert und angewendet werden. Jede `UpdateComponent`-Instanz beschreibt dafür eine Methode

```
void update(NetworkInformation netinfo, SBINetworkInformation
                context)throws UpdatingException;
```

Über die Methode `update` wird insbesondere auf die zentralen `ClassificationRegister`-, `DependencyManager`- und `Federation`-Elemente zugegriffen, um das Speichermodell anzupassen. Der Zugriff auf die implementierten `UpdateComponent`-Elemente erfolgt analog zur Verwaltung von Parsern und Normalisierern über einen eigens dafür aufgesetzten Pool und einen dazu implementierten Worker-Pool. Die Implementierung ist direkt in die `SBIRegister`-Klasse integriert.

6.5.2 Northbound-Interface

Das Framework für das NBI ist in seiner Package-Struktur ähnlich wie das SBI nach seinen einzelnen Teilkomponenten gegliedert: Komponenten für Push, Pull, den Managementplattform-Verbund und Privilegien. Das NBI-Register als zentrale Komponente liegt auf einer dieser übergeordneten Ebenen. Der Schwerpunkt der Beschreibung der Implementierung liegt auf der Umsetzung der Verwaltungs- und Verarbeitungsprozesse im NBI-Register, da das Modell bereits auf hoher Detailebene im Konzept in Kapitel 5.6.2 beschrieben wurde. Lediglich die Implementierung des **Managementplattform-Verbunds** bringt insbesondere die Notwendigkeit zur Auflösung einer zyklischen Abhängigkeit zwischen den Frameworks mit sich: Die Klasse `ManagementplattformClusterElement` implementiert `Verbundselement` aus dem Konzept

und referenziert auf DataCenter-Instanzen. Dadurch, dass das Element des Privilegienbausteins aus dem NBI-Framework jedoch ebenfalls im Nutzerframework in der Klasse Entity verwendet wird, und die Klasse User wiederum im Föderationsframework in der Klasse Party, wird die besagte zyklische Abhängigkeit geschaffen. Die Problematik dieser wird in der Implementierung durch die Ersetzung der Klasse DataCenter durch einen generischen Typ D gelöst. Die Evaluierung wird hierdurch nicht negativ beeinflusst.

6.5.2.1 Adapterverwaltung am NBI

Die wichtigsten Funktionen des NBI sind die Koordinierung des Zugriffs auf ihm zugewiesene **Netzadapter**, von Pull-Zugriffen und Push-Benachrichtigungen. Für die Anfrage einer konkreten Adapter-Instanz (eine Schnittstelle oder einen Konnektor) stellt das NBI-Register dafür eine Methode

```
public Adapter getAdapterOfClass(Class<? extends Adapter> acl)throws
                    AdapterDoesNotExistException {...}
```

bereit. Als einziger Parameter wird die Klasse des notwendigen Adapters übergeben, welcher eine Kindklasse von Adapter sein muss. Das NBI-Register hinterlegt alle ihm zugewiesenen Adapter-Instanzen über die folgende Zuweisung

```
Map<Class<? extends Adapter>, List<Adapter>> adapters;
```

Die Methode getAdapterOfClass wählt dann eine zufällige, zum angefragten Typ passende Instanz der entsprechenden Liste aus und gibt diese zurück. Falls kein Adapter des angefragten Typs mit dem NBI verknüpft ist, dann wird ein Fehler AdapterDoesNotExistException generiert.

6.5.2.2 Pull-Benachrichtigungen

Für den Aufruf von **lokalen Pull-Zugriffen** stellt das NBI-Register zwei überladene Funktionen bereit: Eine ohne kontextuelle Angabe einer MO-ID für Plattformfunktionen und eine mit der Möglichkeit zur Angabe einer MO-ID für MOC-Funktionen. Listing 6.7 zeigt Letztere beispielhaft. Die Identifikation des auszuführenden Pull-Bausteins erfolgt anhand seines Identifikators. Der dazu passende Baustein wird zu diesem Zweck aus der Menge der registrierten NBIPullComponent-Instanzen gesucht. Die Ausführung geschieht dabei asynchron, d. h., wie im gezeigten Listing zu sehen ist, wird die Ausführung durch eine ExecutorService-Instanz (realisiert als Cached-Threadpool) behandelt und nicht das eigentliche Ergebnis, sondern eine Future-Instanz des Ergebnisses zurückgegeben. Die eigentliche Managementanwendung ist für die Auswertung der Future-Instanz zuständig.

```
1  public Future<Map<String, FunctionValue>>
       executeNBIPullComponentByID(Identificator id, Credentials cred, Map<String,
       FunctionValue> params, Identificator moid) throws
       NBIPullComponentDoesNotExistException {
2      for (NBIPullComponent pc : pullComponents) {
3          if (pc.getId().equals(id)) {
4              return pullHandlingExecutor.submit(() -> {
5                 return pc.execute(cred, params, moid);
6              });
7          }
8      }
9      throw new NBIPullComponentDoesNotExistException("The requested
           NBIPullComponent cannot be found by ID");
10 }
```

Listing 6.7: Funktion zum asynchronen Aufruf einer MOC-Funktion mit Angabe einer MO-ID.

Um ein MO bei einem Pull-Aufruf verwenden zu können, ist die Implementierung einer speziellen `FunctionType`-Subklasse `MOValueType` notwendig: Diese kann als `FSNMOC`-Instanz als Parameter der durch den Pull-Aufruf ausgeführten MOC-Funktion genutzt werden. Die Überführung der übergebenen MO-ID erfolgt in der `execute`-Methode der jeweiligen MOC-Funktion, wie in Listing 6.8 als Ausschnitt gezeigt ist. In der Methode wird zunächst überprüft, ob das übergebene `Credentials`-Element zum Aufruf der MOC-Funktion berechtigt. Im positiven Fall wird die Singleton-Instanz des `ClassificationRegister` (vgl. Abschnitt 6.3.1) angefragt und verwendet, um die MO-Modellinstanz mit der übergebenen MO-ID zu ermitteln. Schließlich wird nach positiver Ermittlung die jeweilige `FSNMOC`-Instanz im Parameter-Typ `MOValueType` gekapselt und die bisherigen Parameter damit erweitert.

```
1  public Map<String, FunctionValue> execute(Credentials c, Map<String,
       FunctionValue> params, Identificator moid) throws MODoesNotExistException,
       CannotExecuteException {
2      // Access check
3      if (canAccess(c)) {
4          ClassificationRegister creg = ClassificationRegister.getInstance();
5          if (creg.idOfMoExists(moid)) {
6              FSNMOC mo = creg.getMOByID(moid);
7
8              Map<String, FunctionValue> newParams = Map.copyOf(params);
9              try {
10                 newParams.put("__mo", new FunctionValue(mo, new MOValueType()));
11             } catch (NonConformValueError e) {
12                 e.printStackTrace();
13             }
14             return executeLocal(newParams);
15         } else {
16             throw new MODoesNotExistException("The MO with given MOID does not
                   exist!");
17         }
18     } else {
19         throw new CannotExecuteException("Access denied for the given
               credentials!");
20     }
21 }
```

Listing 6.8: `execute`-Methode einer `NBIPullComponent`-Instanz. Die Methode prüft zunächst, ob der Zugriff berechtigt ist, ermittelt dann ein MO anhand der übergebenen MO-ID und legt das MO in die Tabelle der Parameter zum Funktionsaufruf.

```
1  public Future<Map<String, FunctionValue>> triggerRemotePull(PullContext c,
       Credentials cred, Map<String, FunctionValue> params, Identificator moid)
       throws NBIPullComponentDoesNotExistException, NBIPullExecutionFailedException
       {
2      if (!pullContexts.containsKey(c)) {
3          throw new NBIPullComponentDoesNotExistException("There is no such Pull
               Context and pull component association.");
4      }
5      NBIRemotePullComponent rpc = pullContexts.get(c);
6      if (!rpc.canProcess(c)) {
7          throw new NBIPullExecutionFailedException("The associated remote pull
               component cannot handle the associated calling context.");
8      }
9      return pullHandlingExecutor.submit(() -> {
10         return rpc.execute(cred, params, moid);
11     });
12 }
```

Listing 6.9: Funktion zum remoten asynchronen Aufruf einer MOC-Funktion unter Angabe einer MO-ID.

Der Aufruf eines **remoten Pull-Zugriffs** wird über eine mit dem NBI verknüpfte Netzschnittstelle initiiert und der Aufrufkontext an das NBI-Register zur Identifikation des jeweiligen NBI-

RemotePullBaustein übergeben. Der Aufrufkontext ist über die abstrakte Klasse `PullContext` realisiert. Darin muss vor der Instanziierung die ebenfalls abstrakte Methode `boolean equals(Object p)` überschrieben werden, über die die Gleichheit zweier unterschiedlicher Kontexte (z. B. unterschiedliche HTTP-Kontexte) festgelegt wird. Das NBI-Register identifiziert eine für einen Aufrufkontext geeignete `NBIRemotePullComponent` anhand einer Abbildung von Ersterer darauf. Die Methode zum Ausführen eines remoten Pull-Zugriffs für eine MOC-Funktion (d. h. mit MO-ID) ist beispielhaft in Listing 6.9 gezeigt. Genau wie beim Aufruf des lokalen Pull-Zugriffs wird eine `Future`-Instanz zurückgegeben, die jedoch dann durch die behandelnde Netzschnittstelle behandelt und das Ergebnis an die anfragende Managementanwendung zurückgegeben wird.

6.5.2.3 Push-Benachrichtigungen

Die wichtigste **Datenstruktur** bei der Bearbeitung von lokalen und remoten **Push-Benachrichtigungen** ist das im Konzept in Abschnitt 5.6.2.8 beschriebene geschachtelte Mapping. Dieses lässt sich über die in Java bereitgestellten `Map`-Elemente weitestgehend abbilden. Lediglich für die Fallunterscheidung zwischen Callback-Funktionen für lokale und remote Endpunkte erfordert die Erstellung einer einfachen Hilfsstruktur `PushEndpointSets`. Diese verwaltet sowohl ein `Set` an `NBICallbackFunction`- als auch `NBIRemotePushEndpoint`-Instanzen, die beide durch eine `NBIPushComponent` (betrifft lokale sowie remote Bausteine) referenziert werden. Das NBI-Register stellt zwei überladene Methoden zum kontrollierten Aufbau dieses Mappings bereit. Durch eine Hilfsmethode wird schrittweise – von der Priorität hin zur Callback-Funktion bzw. einem remote Endpunkt – überprüft, ob ein entsprechendes Element existiert. Im positiven Fall werden vorhandene Strukturen genutzt, andernfalls werden diese entsprechend angelegt.

```
1 public void pushNotification(Object o) {
2     notificationHandlingExecutor.execute(() -> {
3       pushNotificationHelper(o);
4     });
5 }
```

Listing 6.10: Durch das NBI-Register bereitgestellte Methode zur Auslösung von Push-Benachrichtigungen.

Die für andere Komponenten zur **Generierung von Push-Benachrichtigungen** und durch das NBI-Register bereitgestellte Methode ist `pushNotification(Object o)`, die in Listing 6.10 gezeigt wird. Das Push-Konzept beschränkt hinsichtlich einfacher Erweiterbarkeit grundsätzlich nicht die Art der Benachrichtigungselemente, resultierend darin, dass der Eingabeparameter der Methode vom generellsten Typ `Object` ist. Auch hier wird ein Cached-Threadpool `notificationHandlingExecutor` zur Parallelisierung der Überprüfung und eigentlichen Benachrichtigung genutzt, welche durch die private Methode `pushNotificationHelper(Object o)` implementiert wird.

6.6 Frameworks des Funktionsmodells

Die Implementierung der **Teile des Funktionsmodells** umfasst das Framework zur *a)* Erweiterung der Funktionalität von Managementplattformen, das Framework für *b)* Managementfunktionen zur funktionsorientierten Strukturierung der Organisation, sowie den *c)* gestaltbaren Prozess zur automatisierten Überwachung und der Steuerung der gemanagten IT-Infrastruktur.

6.6.1 Funktionserweiterung und Automatisierung der Kernprozesse

Die Implementierung des Frameworks *a)* zur **Erweiterung von der Funktionalität** von Managementplattformen wurde zur besseren Übersichtlichkeit bereits in Abschnitt 6.3.3 im Zuge

von davon abgeleiteten MOC-Funktionen beschrieben. Die Implementierung zentraler Komponenten des *c)* **Automatisierungsprozesses** ist im SBI in Abschnitt 6.5.1 und im NBI in Abschnitt 6.5.2 beschrieben.

6.6.2 Funktionale Managementbereiche

Die Implementierung der *b)* funktionalen Managementbereiche ist stark angelehnt an die Beschreibung im Konzeptkapitel in Abschnitt 5.7.1. Ein **funktionaler Managementbereich** wird durch die Klasse `FunctionDomain` implementiert. Listing 6.11 zeigt die Implementierung der Attribute der Klasse `FunctionDomain`. Die föderationsweit eindeutige ID ist über die `Identificator`-Klasse aus dem `inter`-Package der Frameworks beschrieben. Die Zuweisung von Aufgabenbereichen `TaskDomain` wird über ein Mengenattribut implementiert. So wird bereits durch den Datentyp sichergestellt, dass ein Aufgabenbereich einem Funktionsbereich nicht mehrfach zugewiesen werden kann.

```
/**
* A {@link FunctionDomain} describes a management function domain, as they
* are, for instance, described via ISO's FCAPS. Function areas refer to
* multiple task domains, and describe the organisation structure of
* that function area.
*/
public class FunctionDomain {
    /**
    * The federation unique ID of this function area
    */
    private Identificator id;

    /**
    * A description of this function area
    */
    private String description;

    /**
    * Set of Task Domains that subdivide this function area
    */
    private Set<TaskDomain> taskDomains;

    // Constructor, Getter and Setter ...
}
```

Listing 6.11: Implementierung der Attribute der Klasse `FunctionDomain` zur Darstellung eines Funktionsbereichs aus dem Konzept.

Funktionsbereiche werden als zentrale organisatorische Elemente schließlich **föderationsweit definiert** und entsprechend durch das zentrale `Federation`-Element referenziert, wie in Listing 6.12 gezeigt wird. Die **Eindeutigkeit** eines Funktionsbereichs wird durch die Referenz über einen `Map`-Datentyp von einem eindeutigen Namen aus sichergestellt.

```
/**
 * This class describes a federation with all its elements.
 */
public class Federation {
    // Other Attributes

    /**
     * Defined function areas in this federation description
     */
    private Map<String, FunctionDomain> functionAreas;

    // Constructor, Getter and Setter
}
```

Listing 6.12: Zentrale Referenzierung auf in der Föderation gültige Funktionsbereiche im `Federation`-Element über einen `Map`-Datentyp.

Kapitel 7

Beispielhafte Anwendung der Frameworks

Inhalt

7.1 Beschreibung des Anwendungsszenarios . . . 240
7.1.1 Begründung der Ausprägungen des Basisszenarios . . . 240
7.1.2 Auswahl des Basisszenarios und Anpassungen . . . 241
7.2 Informationsmodell . . . 243
7.2.1 Managementobjektklassen . . . 243
7.2.2 Managementbeziehungen . . . 246
7.2.3 MOC-Funktionen . . . 249
7.2.4 Beispielhafte Netzinformationen . . . 252
7.2.5 Beispielhaftes Alarm-Schema . . . 255
7.3 Organisationsmodell . . . 257
7.3.1 Managementaufgaben . . . 258
7.3.2 Administrative Domänen und Überschneidungsbehandlung . . . 258
7.3.3 Nutzer und Rollen . . . 260
7.3.4 Gültigkeitsbereiche und Zuständigkeiten . . . 260
7.3.5 Föderationsmodell . . . 262
7.3.6 Zugriffsverwaltung . . . 263
7.4 Kommunikationsmodell . . . 268
7.4.1 Southbound-Interface . . . 268
7.4.2 Northbound-Interface . . . 272
7.5 Funktionsmodell . . . 275
7.5.1 Funktionale Managementbereiche . . . 275
7.5.2 Framework zur Erweiterung der Managementplattformfunktionalität 276
7.6 Zusammenfassung des Kapitels . . . 278

In diesem Kapitel wird die Modellierbarkeit von FSN über die in dieser Arbeit beschriebenen Frameworks evaluiert. Die Ausgangsbasis dazu bietet ein Anwendungsszenario, welches im nächsten Abschnitt beschrieben wird. Die Anwendung wird als Modultest unter Einbindung des *JUnit*-Moduls für Java umgesetzt, um entsprechende Werkzeuge wie Assertions nutzen zu können.

Charakteristikum	Ausprägung in FSNs		
Räumliche Dimension	regional	überregional	gemischt
Dauer	kurzfristig	langfristig	
Koordination	implizit	explizit	
Gruppenstruktur	flexibel	fest	
Vertrauen	indirekt	direkt	gemischt
Aufgabenzuordnung	statisch	dynamisch	
Dynamik	statisch	dynamisch	
Rollenvergabe	statisch	dynamisch	
Relation zw. administrativen Domänen	überlappend	disjunkt	
Zielsetzung	überlappend	komplementär	gemischt
Domänenstruktur	global	hierarchisch	unabhängig
Art der Infrastrukturnutzung	symmetrisch	asymmetrisch	

Tabelle 7.1: Berücksichtigung von Charakteristika des Evaluationsszenarios (Dunkelblau: Baseline-Charakteristika; Hellblau: berücksichtigte Alternativen; Weiß: nicht-berücksichtigte Ausprägungen).

7.1 Beschreibung des Anwendungsszenarios

Die Auswahl eines geeigneten Baseline-Szenarios erfolgt auf Basis der zuvor aufgestellten Ausprägungen von Föderationen bei SNs aus Abschnitt 2.2.2. Eine auf die wesentlichen Ausprägungen kondensierte und hinsichtlich ihrer Relevanz für das Evaluationsszenario kodierte Übersicht der Charakteristika ist in Tabelle 7.1 gezeigt. Die Ausprägungen mit dunkelblauem Hintergrund gelten für das Baseline-Szenario. Hellblau gekennzeichnete Ausprägungen sind ebenfalls zu berücksichtigen und werden in der Evaluation als Variation des Baseline-Szenarios diskutiert. Ausprägungen mit weißem Hintergrund werden begründet nicht berücksichtigt, wie im Folgenden erläutert wird.

7.1.1 Begründung der Ausprägungen des Basisszenarios

Die Umsetzbarkeit einiger Ausprägungen von Föderationscharakteristika impliziert die Umsetzbarkeit der für das jeweilige Charakteristikum mögliche alternativen Ausprägungen, wodurch eine Teilauswahl von Ausprägungen (vgl. Tabelle 7.1) begründet ist. Dies ist besonders für die folgenden Charakteristika der Fall:

- Die Abbildbarkeit einer flexiblen und sich häufig ändernden **Gruppenstruktur** impliziert die Möglichkeit der Abbildbarkeit einer festen Gruppenstruktur.

- Eine Handhabbarkeit einer dynamischen Ausprägung der Charakteristika **Aufgabenzuordnung**, **Dynamik** und **Rollenvergabe** impliziert gleichzeitig die Handhabbarkeit dieser Charakteristika in einer statischen Ausprägung.

Weitere Charakteristika für das Anwendungsszenario lassen sich durch ihre jeweilige gemischte Ausprägung einschränken. Dies betrifft vor allem die Charakteristika der **räumlichen Dimension**, des **Vertrauens** und der **Zielsetzung**.

Das Charakteristikum der **Relation zwischen administrativen Domänen** ist für eine Evaluation unter Betrachtung der Ausprägung überlappender Domänen von besonderer Bedeutung. Die Auflösung der dadurch entstehenden Konflikte im Management ist ein wesentlicher Bestandteil dieser Arbeit. Beim Charakteristikum der **Domänenstruktur** wird eine individuell

unabhängige Ausprägung angenommen, da diese allen Projektpartnern am meisten Freiheiten bietet, jedoch für das Management am meisten Koordination erfordert. Eine global einheitliche Ausprägung genauso wie eine individuell unabhängige Ausprägung sind dagegen meist durch eine zentralisierte Verwaltungsinstanz (z. B. einen der Projektpartner) umzusetzen und daher mit geringerem Managementaufwand verbunden.

Eine in der Evaluation zu berücksichtigende Abweichung vom Baseline-Szenario ist einerseits beim Charakteristikum der **Koordination** zu untersuchen. Im Baseline-Szenario wird eine implizite Ausprägung vorgesehen. Sie manifestiert sich insbesondere durch die Absenz einer Koordinierungsinstanz und basiert auf lokal getroffenen Entscheidungen [21]. Auswirkungen einer expliziten Ausprägung auf eine Nutzung der Frameworks werden jedoch ebenfalls betrachtet und in Abschnitt 7.3.3 diskutiert. Andererseits ist eine Abweichung im Charakteristikum der **Art der Infrastrukturnutzung** in beiden Ausprägungen zu untersuchen, wobei eine symmetrische Nutzung als Baseline-Ausprägung genutzt wird. Diese kann, wie in Abschnitt 7.3.5.2 beschrieben wird, insbesondere über Föderationsbeziehungen modelliert und im Netzmanagement behandelt werden. Das Charakteristikum der **Dauer** ist für die Modellierung und damit das Evaluationsszenario unerheblich und wird entsprechend nicht berücksichtigt.

7.1.2 Auswahl des Basisszenarios und Anpassungen

Die Grundlage für das in der Evaluierung genutzte Szenario bildet das in Abschnitt 3.3 beschriebene Szenario 1, einer Kooperation mehrerer Partner im Rahmen eines gemeinsamen Projekts. Dieses erfüllt weitestgehend die begründeten Baseline-Charakteristika aus dem vorherigen Abschnitt und erfordert nur eine geringfügige Anpassung. Lediglich in den Charakteristika der **Räumlichkeit** und der **Zielsetzung** sind Abweichungen vorhanden, welche jedoch durch eine Adaption des Szenarios erreichbar sind:

- **Räumlichkeit**: Durch Annahme der Kooperation regionaler Partner wie Universitäten und Unternehmen aus derselben Stadt sowie weiteren Partnern aus anderen Zeitzonen wird eine gemischte Räumlichkeit simuliert.

- **Zielsetzung**: Die im ursprünglichen Szenario überlappende Zielsetzung der Kooperationspartner wird durch die Miteinbeziehung eines zentralen Hosting-Dienstleisters für das Projekt als gemischte Zielsetzung charakterisiert. Der Hosting-Dienstleister bietet dabei je nach Bedarf einen grundlegenden Satz an IT-Ressourcen für das entstehende FSN, welcher durch IT-Ressourcen der eigentlichen Projektpartner erweitert wird.

Abbildung 7.1 zeigt eine Übersicht über das zur Evaluation genutzte Szenario. Auf Detailumsetzungen des Szenarios wird zur besseren Übersichtlichkeit in den folgenden Abschnitten direkt im Kontext der jeweiligen Umsetzung über die Frameworks eingegangen. Im fiktiven Projekt wird eine Kooperation von zwei regionalen Partnern *P1*, *P2* und einem überregional niedergelassenen Partner *P3* untersucht. Um für das entstehende FSN des Projekts ausreichende IT-Ressourcen gleich ab Projektbeginn bereitzustellen, werden diese von einem Dienstleister *D* eingekauft. Diese Basisinfrastruktur ist in der Abbildung auf **Ebene 0** gezeigt und wird durch eine weitere, durch die Projektpartner P1 bis P3 bereitgestellte Infrastruktur auf derselben Ebene, ergänzt.

Das im Szenario betrachtete Projekt ist hinsichtlich der **Projektorganisation** in zwei Arbeitsgruppen unterteilt. In der ersten Arbeitsgruppe arbeiten die Partner P1 und P2 in einer engen Kooperation zusammen und in der zweiten Arbeitsgruppe die Partner P1 sowie P3. Die Basisressourcen der Ebene 0 werden für die Nutzung innerhalb der Arbeitsgruppen, zusammen mit dedizierten, abgeleiteten IT-Ressourcen zum Betrieb der Managementinfrastruktur – Managementplattforminstanzen *MP1* bis *MP4* im jeweiligen Datenzentrum – und von für innerhalb der Föderation genutzte zentrale Dienste auf **Ebene 1** durch Systemvirtualisierung unterteilt. Anwendungen auf der föderierten Infrastruktur werden auf **Ebene 2** entsprechend ihrer Zugehö-

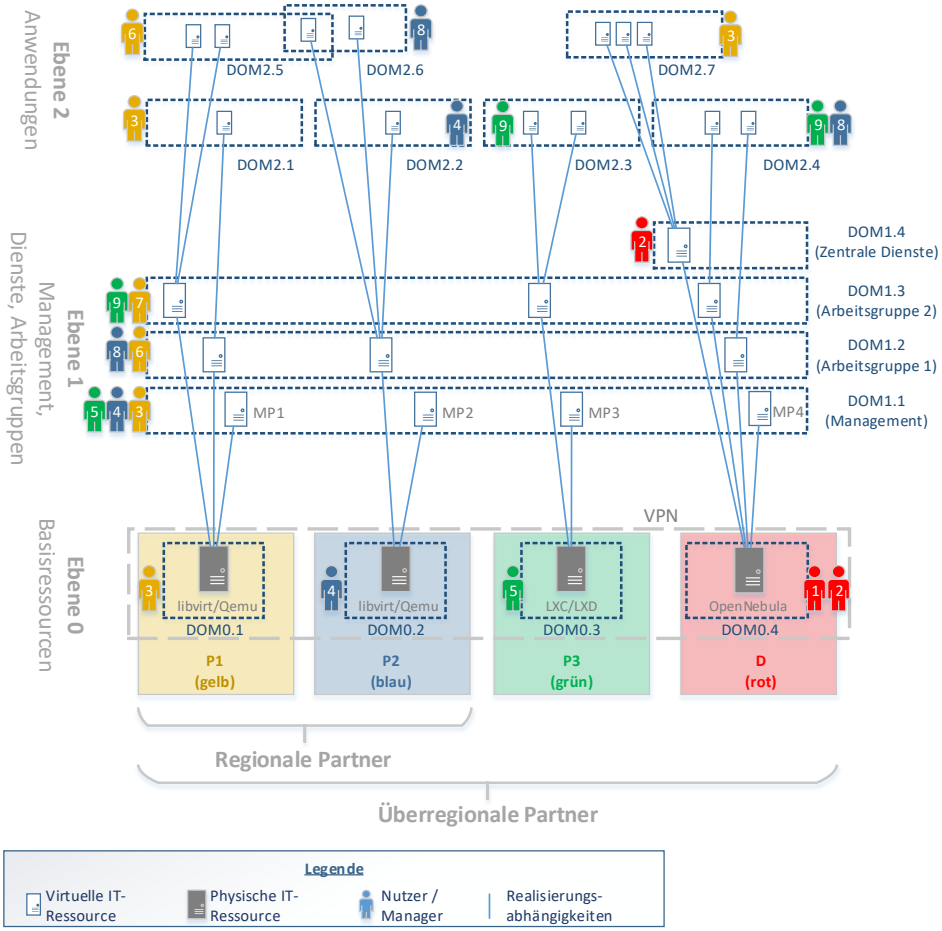


Abbildung 7.1: Übersicht über das Baseline-Evaluationsszenario.

rigkeit auf Basis der IT-Ressourcen der Arbeitsgruppen, dem Management oder der zentralen Dienste realisiert. Realisierungsabhängigkeiten sind zur Verdeutlichung in Abbildung 7.1 als Kanten zwischen IT-Ressourcen der jeweiligen Ebenen gekennzeichnet. IT-Ressourcen auf höheren Schichten werden stets von IT-Ressourcen darunterliegender Schichten realisiert.

Die **Domänenstruktur** ist gemäß der zuvor beschriebenen Einteilung der IT-Ressourcen umgesetzt: Jeder Partner ist zunächst für seine in die Föderation eingebrachten IT-Ressourcen auf Ebene 0 (Domänen *DOM0.1* bis *DOM0.4*) zuständig. Die IT-Ressourcen der Arbeitsgruppen (Domänen *DOM1.2* und *DOM1.3*) auf Ebene 1 werden durch Verantwortliche der jeweils darin agierenden Parteien in einer eigenen Domäne verwaltet. Ressourcen der Managementinfrastruktur (Domäne *DOM1.1*) werden im Sinne einer impliziten Koordination des Szenarios gleichberechtigt durch alle Föderationspartner mit Ausnahme des Hosting-Dienstleisters verwaltet. Der Hosting-Dienstleister hingegen verwaltet die IT-Ressourcen für zentralen Dienste (Domäne *DOM1.4*). IT-Ressourcen auf Ebene 2 (Domänen *DOM2.1* bis *DOM2.7*), die eigentlichen Anwendungen, werden vom jeweiligen Bedarfsträger verwaltet. Für die Evaluierung bedeutsame Domänen stellen besonders die Folgenden dar:

- In Domäne DOM2.4 werden IT-Ressourcen für eine gemeinsame Testumgebung für beide Arbeitsgruppen betrieben. Hier muss insbesondere die Verwaltung zwischen mehreren Verantwortlichen der Parteien koordiniert werden.

- Die Domänen DOM2.5 und DOM2.6 überschneiden sich in einer IT-Ressource. Der entstehende Zuständigkeitenkonflikt muss im Management berücksichtigt werden.

Auf weitere Besonderheiten und Varianten bei der Umsetzung wird in den folgenden Abschnitten direkt eingegangen.

7.2 Informationsmodell

In diesem Abschnitt werden Aspekte des Informationsmodells des Evaluierungsszenarios modelliert. Der Fokus der Beschreibung liegt auf Besonderheiten der Modellierung in FSNs.

7.2.1 Managementobjektklassen

In diesem Abschnitt wird auf Basis des MOC-Frameworks (siehe Abschnitt 5.4.1) die Modellrepräsentation der im Anwendungsszenario relevanten MOs als MOCs realisiert. Dazu wird für jedes MO jeweils eine Klassifizierungsklasse von `FSNMOCK` oder seinen generischen Kindklassen abgeleitet, welche das jeweilige MO beschreibt. Eine Übersicht über die Umsetzung der Klassifikatoren für die jeweiligen MOs des Anwendungsszenarios ist in Abbildung 7.2 gezeigt. Neben den in der Abbildung gezeigten Virtualisierungssystemen (LXC/LXD und libvirt/Qemu) und OpenNebula als VIM werden einige relevante SDN-Komponenten berücksichtigt und eine Besonderheit der Umsetzung von SDN-Controllern in FSNs aufgezeigt und in diesem Abschnitt näher beschrieben.

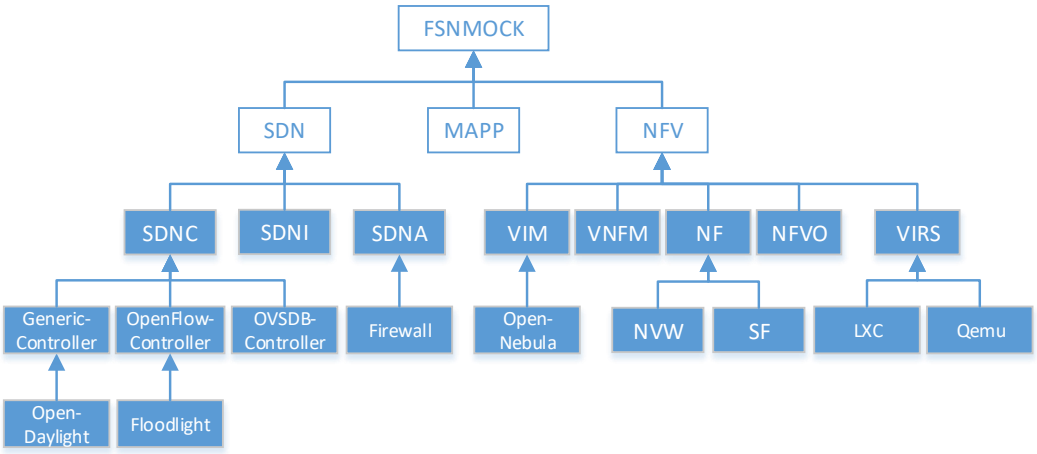


Abbildung 7.2: Übersicht über die resultierenden Klassifikator-Klassen zur Beschreibung der MOs des Anwendungsszenarios.

7.2.1.1 MOC-Klassifikatorklassen und -Attribute

Eine besondere Herausforderung bei der Modellierung von MOCs stellt die Klassifizierung von multifunktionellen FSN-spezifischen MOs wie dem SDN-Controller OpenDaylight dar. Dieser unterstützt mehrere SDN-Implementierungen und ist dynamisch erweiterbar (vgl. Abschnitt 4.3.1). Da in dem in dieser Arbeit verfolgen Ansatz eine Modellierung von MOCs durch Klassifizierung verfolgt wird, können diese Einzelfunktionen jedoch als jeweils eigene Klasse implementiert werden. So wird, wie in Abbildung 7.2 gezeigt ist, für unterschiedliche SDN-Implementierungen jeweils eine abstrakte Controller-Klasse erstellt: Hier exemplarisch ein `OpenFlowController` und `OVSDBController` mit dazugehörigen Attributen. Ein `OpenFlowController` beschreibt beispielhaft die Attribute `OpenFlowPort` und damit den Port, auf dem der OpenFlow-Dienst auf dem Controller erreichbar ist, sowie durch `SupportedOpenFlowVersions` eine Menge an von einem OpenFlow-Controller unterstützten OpenFlow-

Versionen. Die Implementierung eines Mengen-Datentyps im MOC-Framework wird unter Verwendung des `Complex`-Attributtyps über die Abbildung des jeweiligen Mengenelement-Typs auf einen booleschen Wert umgesetzt. `SupportedOpenflowVersionsEnum` beschreibt eine Liste gültiger OpenFlow-Versionen:

```
class SupportedOpenflowVersions extends
Complex<SupportedOpenflowVersionsEnum, Boolean>
```

Der SDN-Controller Floodlight als OpenFlow-basierter Controller wird entsprechend in einer gleichnamigen Klasse und abgeleitet von der Elternklasse `OpenFlowController` umgesetzt.[XVIII] Sie beschreibt Floodlight-spezifische Attribute, die nicht für jeden OpenFlow-Controller zutreffen.

```
1  /**
2    * Set of active features, each identified by a custom ID.
3    */
4  public class ActiveFeatures extends Complex<String, Boolean> {
5
6      /**
7        * Detailed constructor for this attribute
8        *
9        * @param name Name of this attribute
10       * @param description Description of this attribute
11       * @param accessLevel Access level of this attribute
12       */
13     public ActiveFeatures(String name, String description, AccessLevel
           accessLevel) {
14         super(name, description, accessLevel);
15     }
16
17     /**
18       * Default constructor, setting the name, description and access level of
             this
19       * attribute.
20       */
21     public ActiveFeatures() {
22         this("activefeatures", "Set of active features, each identified by a
               custom ID", AccessLevel.ZL_WRITE);
23     }
24
25     /**
26       * Abstract function for checking the values given to this attribute.
27       * E.g. for range checking.
28       *
29       * @param key  Check key
30       * @param value Check value
31       * @return True of value complies, otherwise false
32       */
33     @Override
34     public boolean checkKeyValue(String key, Boolean value) {
35         if (key != null && value != null) {
36             return true;
37         }
38         return false;
39     }
40 }
```

Listing 7.1: Beispielillustration der Implementierung des Attributs `ActiveFeatures` zur Beschreibung von aktivierten Features eines OpenDaylight SDN-Controllers.

Die Klasse `OpenDaylight` wird dagegen entsprechend dem Konzept des OpenDaylight-Controllers auf Basis einer generischen Controller-Subklasse `GenericController` beschrieben. Die

[XVIII] siehe `https://github.com/floodlight/floodlight`

Klasse GenericController erhält dann Attribute, die von allen generischen Controller-Plattformen – zu denen beispielsweise auch ONOS zählt – geteilt werden. Die Klasse OpenDaylight erhält zusätzliche Attribute, die spezifisch für den OpenDaylight-Controller sind, wie hier exemplarisch ActiveFeatures eine Menge an aktiven *Features* der darunterliegenden Karaf-Plattform. Das Attribut ActiveFeatures wird beispielhaft in Listing 7.1 gezeigt. Die Umsetzung auf Basis der MOC-Framework-Klasse Complex erfordert die Implementierung der darin generischen Funktion checkKeyValue sowie eines detaillierten Konstruktors zum Setzen aller Felder dieses Attributs. Der Standardkonstruktor public ActiveFeatures(){...} ohne Parameter wird zur vereinfachten Verwendung im Rahmen der Evaluierung für alle Attribute implementiert.

Die implementierten Attribute werden direkt in die jeweilige FSNMOCK-Subklasse eingebunden, wie das Attribut ActiveFeatures in der Klassifikatorklasse OpenDaylight, wie in Listing 7.2 gezeigt ist. Innerhalb der FSNMOCK-Klassen werden alle Attribute mit Sichtbarkeit public (aus allen anderen Klassen heraus zugreifbar) eingebunden, da die Kontrolle der Belegung direkt durch die Attribut-Klassen erfolgt. Zum Zweck der einfachen Evaluierung wird auch hier und für alle anderen Klassifikatorklassen ein Standardkonstruktor ohne Parameter implementiert.

```
1  /**
2    * FSNMOCK class for describing an OpenDaylight SDN controller instance
3    */
4  public class OpenDaylight extends GenericController {
5
6      /**
7        * Activated features of this Opendaylight instance
8        */
9      public ActiveFeatures activeFeatures;
10
11     // Constructors follow...
12 }
```

Listing 7.2: Nutzung von Attributen am Beispiel der Klasse OpenDaylight.

7.2.1.2 Klassifikation von MOCs

Die eigentliche Beschreibung eines MO wird durch die Klassifizierung durch entsprechende FSNMOCK-Subklassen vorgenommen. Abbildung 7.3 zeigt die Klassifizierung eines System-Instance-Objekts als OpenDaylight-Controller. Dazu wird es grundsätzlich mit einem OpenDaylight-Objekt sowie darüber hinaus mit Objekten klassifiziert, die seine Funktionalität beschreiben: OpenFlowController und OVSDBController. Alle MOs der Ebene 2, die keine Managementplattformen sind, werden als generisches VIRS klassifiziert. Sie sind von Bedeutung für die im nächsten Abschnitt beschriebene Anwendung des Managementbeziehungsframeworks und bilden die Basis der Modellierung einer mehrschichtigen Virtualisierung.

```
1  // List of MOs associated
2  // with Qemu of first partner
3  List<? extends FSNMOC> qemu1Realized = List.of(genericLevel1MOs.get(0),
       genericLevel1MOs.get(1), genericLevel1MOs.get(2));
4
5  // For each MO, create a DVRealisation dependency
6  // from the VIRS to its VM
7  qemu1Realized.stream().forEach(fsnmoc -> {
8      dependencyManager.addDependency(new DVRealisation(qemuInstanceP1, fsnmoc));
9  });
```

Listing 7.3: Beispielhafte Instanziierung und Registrierung von Realisierungsabhängigkeiten.

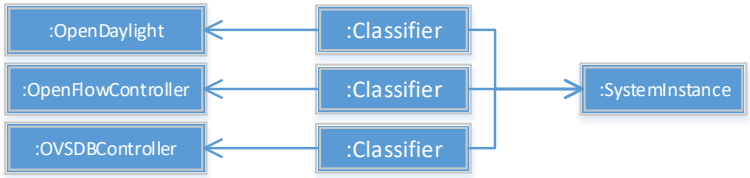


Abbildung 7.3: Beispielhafte Klassifizierung einer MO-Repräsentation als OpenDaylight-Controller und von diesem implementierte Funktionen als OpenFlow- und OVSDB-Controller.

7.2.2 Managementbeziehungen

Das Framework für Managementbeziehungen (vgl. Abschnitt 5.4.2) wird hinsichtlich der Modellierung verschiedenartiger Abhängigkeiten zwischen MOs und einem koordinierten Zugriff auf diese angewendet.

7.2.2.1 Definition von Managementbeziehungen

Es werden zunächst die **Realisierungsabhängigkeiten** betrachtet, wie sie in der Übersicht zum Anwendungsszenario (vgl. Abbildung 7.1) gezeigt werden. Listing 7.3 zeigt die Instanziierung dieser Abhängigkeiten am Beispiel einer Qemu-Instanz `qemuInstanceP1` und durch diese realisierte virtuelle Systeminstanzen, die analog zu den MOCs im vorherigen Abschnitt modelliert wurden.

Andere notwendige Abhängigkeiten sind **Netz-** und **Steuerungsabhängigkeiten**. Sie werden in Abbildung 7.4 gezeigt. Zur Beschreibung der darin grün markierten Kommunikationsverbindungen wird von der Klasse `DHNetwork` die Abhängigkeit `DHNIpNetwork` (im Sinne von *DependencyHorizonalNetworkIpNetwork* zur Beschreibung der Charakteristika der Abhängigkeit) zur Spezifizierung einfacher IP-Netze abgleitet. Die rot gekennzeichnete Gruppe von Ressourcen der Ebene 0 beschreibt ein VPN-Netz. Dieses wird wiederum als Abhängigkeitsklasse `DHNIVpn` als Spezialisierung der zuvor erstellten Abhängigkeitsklasse `DHNIpNetwork` abgeleitet. Die orange gekennzeichneten Verbindungen zwischen den dargestellten Switch-Komponenten und einem Floodlight-SDN-Controller illustrieren jeweils einerseits durch `DHNIpNetwork`-Instanzen beschriebene Kommunikationsbeziehungen. Andererseits beschreiben sie OpenFlow-basierte Steuerungsabhängigkeiten, welche durch eine dafür erstellte und von der Klasse `DVControl` abgeleiteten Klasse `DVCOpenFlow` beschrieben werden. Die Instanziierung der Steuerungsabhängigkeit erfolgt analog zu den zuvor gezeigten Realisierungsabhängigkeiten in Listing 7.3. Die Instanziierung von Netzabhängigkeiten dagegen kann je nach Interpretation ungerichtet – d. h. es benötigt eine Instanz der jeweiligen Netzabhängigkeit zwischen zwei Knoten – oder gerichtet – d. h. eine ungerichtete Verbindung benötigt zwei Instanzen, eine von Knoten A nach B und eine umgekehrt. In dieser Evaluation wurde eine gerichtete Interpretation der Netzabhängigkeiten angenommen und für jede grün und orange gekennzeichnete Abhängigkeit der Klasse `DHNIpNetwork` sowie für jeden Knoten im VPN-Netz (Klasse `DHNIVpn`) wurden zwei Instanzen erstellt. Eine Übersicht über die erweiterte Klassenstruktur ist in Abbildung 7.5 gezeigt.

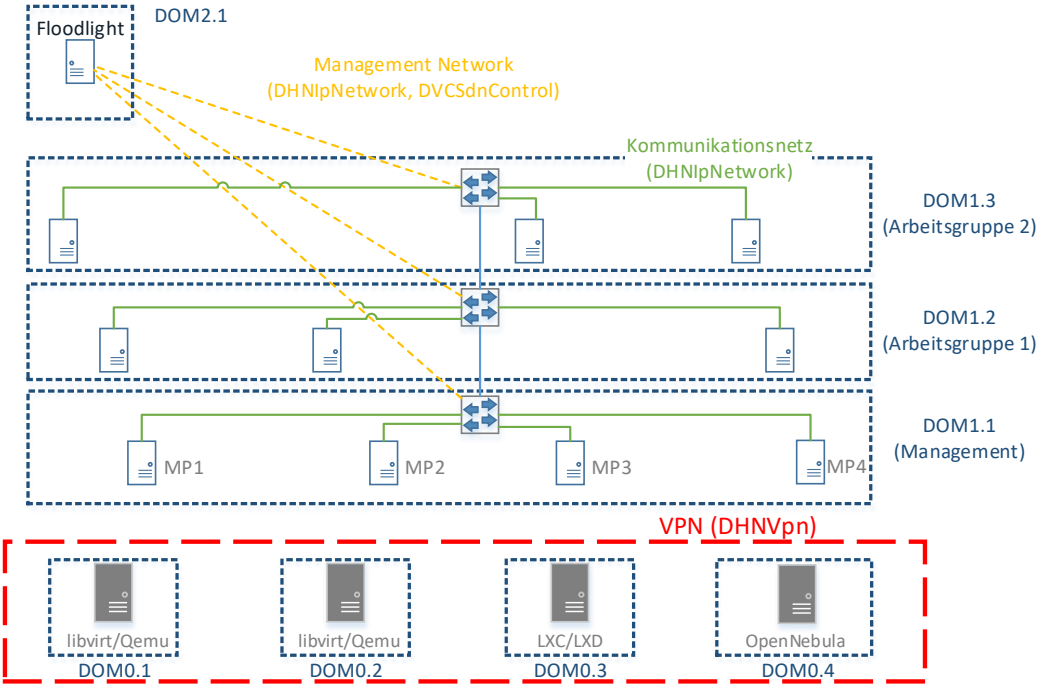


Abbildung 7.4: Beispielhafte Steuerungs- und Netzabhängigkeiten zwischen MOs.

```
1  /**
2  * Class for describing an openflow-controlling dependency, derived
3  * from a generic control dependency.
4  */
5  public class DVCOpenFlow extends DVControl {
6        /**
7        * Constructor to create DVControl dependency
8        *
9        * @param sub subjekt
10       * @param obj objekt
11       */
12       public DVCOpenFlow(FSNMOC sub, FSNMOC obj) {
13           super(sub, obj);
14       }
15 }
```

Listing 7.4: Implementierung einer OpenFlow-basierten Steuerungsabhängigkeit `DVCOpenFlow`.

Die **Erweiterung** des Frameworks für Managementbeziehungen wird stets nach demselben Schema durchgeführt: Zunächst wird die Art der neu zu modellierenden Abhängigkeit als Basis verwendet – d. h. *a)* Steuerungs- *b)* Realisierungs-, *c)* Netz- oder *d)* Systemabhängigkeit. Falls keine dieser Abhängigkeiten die neue Abhängigkeit auf abstrakter Ebene beschreiben kann, werden die darüberliegenden *e)* vertikalen Abhängigkeiten oder *f)* horizontalen Abhängigkeiten als Basis in Betracht gezogen. Falls auch diese nicht geeignet sind, wird eine neue Abhängigkeitsklasse von *g)* der generellsten Klasse – `Dependency` – abgeleitet. Listing 7.4 zeigt die Implementierung der Klasse `DVCOpenFlow`. Diese wird als spezialisierte Steuerungsabhängigkeit von der Klasse `DVControl` abgeleitet und muss entsprechend einen Konstruktor implementieren, der den Konstruktor der Elternklasse aufruft. Die Abhängigkeit beschreibt eine Beziehung eines steuernden zu einem gesteuerten MO. Damit verknüpfte Attribute werden hingegen über die MO-Klassifizierungsklasse `FSNMOCK` und davon abgeleiteter Klassen beschrieben.

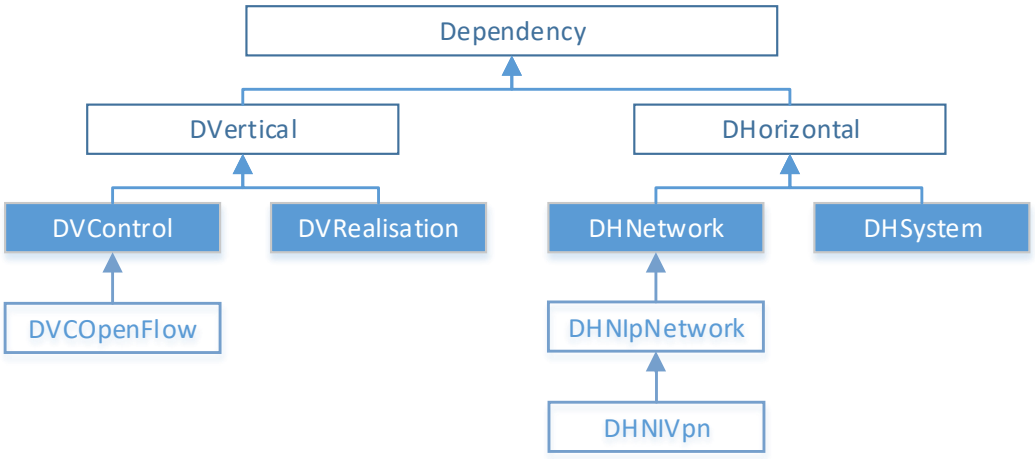


Abbildung 7.5: Klassenstruktur der Erweiterungen von Abhängigkeiten.

```
1  // Test vertical traversing
2  // Traverse paths of all level 2 MOs to their original MO on level 0
3  for (FSNMOC f : genericLevel2MOs) {
4      // Get dependencies where f is the subject
5      ArrayList<DVRealisation> rds = (ArrayList<DVRealisation>)
           dependencyManager.getReverseDepsOfClassOfMOC(f, DVRealisation.class);
6      // Check count
7      assertEquals(1, rds.size());
8      // Check that subject is a level-1-MO
9      assertTrue(genericLevel1MOs.contains(rds.get(0).getSubject()));
10     // Get next dependency towards object and
11     // check that its subject is a level-0-component
12     assertTrue(vpnMOs.contains(((ArrayList<DVRealisation>)
           rds.get(0).down()).get(0).getSubject()));
13 }
14
15 // Test horizontal traversing
16 // Check for switch dom 1.1, whether we can reach each MO once
17 ArrayList<? extends FSNMOC> dom11copy = new ArrayList<>(dom11MOs);
18 DHNIpNetwork d = (DHNIpNetwork)
       dependencyManager.getDependenciesOfClassOfMOC(switchDOM1_1,
       DHNIpNetwork.class).get(0);
19 while (d.nextObject().hasNext()) {
20     DHNIpNetwork next = (DHNIpNetwork) d.nextObject().next();
21     assertEquals(switchDOM1_1, next.getSubject());
22     assertTrue(dom11MOs.contains(next.getObject()));
23     dom11copy.remove(next.getObject());
24     if (next == d) {
25         break;
26     }
27 }
```

Listing 7.5: Beispielanwendung und Test der anwendungsspezifischen Funktionen zum Durchschreiten am Beispiel der Abhängigkeiten `DVControl` und `DHNIpNetwork`.

7.2.2.2 Traversierung der Abhängigkeiten

Die Anwendung der Funktionen zum **Traversieren** der jeweiligen horizontalen Abhängigkeiten `nextObject` und `nextSubject` sowie die der vertikalen Abhängigkeiten `up` und `down` wird als Beispiel an den Netz- (horizontal) und Realisierungsabhängigkeiten (vertikal) in Listing 7.5 gezeigt.

Im ersten Fall wird ein Test zur Rückführung der Realisierung aller Ebene 2 MOs auf jeweils ein MO der Ebene 1 und diese wiederum jeweils auf ein MO der Ebene 0, welche hier in der Liste `vpnMOs` gesammelt sind, implementiert. Der im gleichen Listing gezeigte Test zur Traversierung horizontaler Abhängigkeiten prüft die Erreichbarkeit aller MOs in Domäne DOM1.1, ausgehend von dem in dieser Domäne gezeigten Switch (vgl. Abbildung 7.4). Dafür wird die Liste der MOs in der Domäne kopiert und in `dom11copy` eingefügt. Wird ein MO beim Traversieren erreicht, wird dieses aus der Liste entfernt, wodurch sichergestellt wird, dass alle Netzabhängigkeiten wie gewünscht modelliert wurden.

7.2.3 MOC-Funktionen

Die wesentliche Herausforderung bei der Modellierung von MOC-Funktionen ist die Mehrfachklassifikation von MOs, ähnlich wie im Framework zur Modellierung von MOCs selbst. Insbesondere das Management von MOC-Funktionsdefinitionen und die Treiberimplementierungen stellen wesentliche Aspekte im Konzept dar. Diese Herausforderungen stellen den Fokus in dieser Evaluierung dar und wurden bereits in [181] beispielhaft in einer Lua-basierten Implementierung modelliert.

7.2.3.1 Definition und Zuweisung einer MOC-Funktion

Als Basis zur Beschreibung von MOC-Funktionen dient das MOC-Klassifizierungssystem, das in Abschnitt 7.2.1 beschrieben und in Abbildung 7.2 gezeigt wurde. In diesem Abschnitt wird beispielhaft eine Methode `addSrcBlockingFlow` für alle OpenFlow-basierten SDN-Controller implementiert. Folglich wird die Methode definiert und mittels einer Instanz der Klasse `MOCKToFunctionAssociation` mit der MOCK-Klasse `OpenFlowController` verknüpft, wie in Listing 7.6 gezeigt wird.

```
1  // Define a new MOC function for adding a flow that blocks a source IP
2  // Defined for all OpenFlowController MOCKs
3  MOCFunctionReal mocf = new MOCFunctionReal(SampleID.gen(MOCFunctionReal.class),
4          "addSrcBlockingFlow",
5          "Add a Flow for blocking the traffic from a certain IP",
6          Map.ofEntries(new AbstractMap.SimpleEntry<>("switchID", new DPIDType()),
7                        new AbstractMap.SimpleEntry<>("sourceMAC", new
                               MACAddressType())),
8          Map.ofEntries(new AbstractMap.SimpleEntry<>("success", new
               BooleanType())));
9  MOCKToFunctionAssociation mtfa = new
       MOCKToFunctionAssociation(OpenFlowController.class);
10 try {
11     mtfa.addMOCFunction(mocf);
12 } catch (MOCFunctionAlreadyExistingException e) {
13     e.printStackTrace();
14 }
15
16 // Register driver at MOCKFunctionMap
17 mockFunctionMap.registerRealMOCFunctionDriver(OpenDaylight.class, mocf,
       odldriver);
```

Listing 7.6: Definition einer neuen realen MOC-Funktion.

Die Methode wird als reale MOC-Funktion über ein Objekt der Klasse `MOCFunctionReal` beschrieben, welche als Parameter eine eindeutige ID, den Namen der Methode, ein Mapping von Parameter-Namen und ihren Datentypen, sowie ein Mapping von Rückgabewerten mit ihren jeweiligen Namen und Datentypen fordert. Die Methode erwartet entsprechend eine als Datapath-ID `DPIDType` implementierte ID des Switches, auf dem die Regel installiert werden soll, sowie die zu blockierende MAC-Adresse, implementiert als Typ `MACAddressType`. Als Rückgabe wird ein boolescher Wert `BooleanType` als Indikator des Erfolgs der Ausführung zurückgegeben. Durch die Klasse `MOCKToFunctionAssociation` wird die MOC-Funktion generell der Klasse `OpenFlowController` zugewiesen. Zuletzt wird dann eine Treiberfunktion,

die in den folgenden Absätzen dieses Abschnitts beschrieben wird, vorweggegriffen über eine `MOCKFunctionMap`-Instanz für alle als `OpenDaylight`-MOCK klassifizierten MOs registriert.

7.2.3.2 Treiberimplementierung und -Zuweisung

Die Implementierung einer beispielhaften kompakten Treiberfunktion wird in diesem Fall durch eine Instanz für eine Treiberklasse `MOCFRDriver` für reale MOC-Funktionen umgesetzt. In Listing 7.7 wird als Ausschnitt die dafür zu implementierende Methode `execute` gezeigt. Dabei werden zunächst die zwei Parameter aus der Map übergebener Parameter geholt und entsprechend der erwarteten Typen interpretiert. Die Umsetzung der Methode in OpenDaylight wird durch Implementierung einer Flowbeschreibung umgesetzt. Das Grundgerüst des entsprechenden, auf Basis der OpenDaylight-Beispieldokumentation [207] abgeleiteten Flows, wird als String in der Variable `requestBody` gespeichert. Der Teilstring „#SOURCEMAC#“ beschreibt den Platzhalter der zu sperrenden MAC-Adresse und wird später durch den Wert des übergebenen Parameters ersetzt. Der Dienstzugriffspunkt wird ebenfalls passend zum Aufruf angepasst und beispielsweise um die ID des Switch-Knotens, der passenden Tabelle und der ID des zu implementierenden Flows ergänzt. Schließlich wird die Funktion auf der HTTP-basierten API des jeweiligen Ziel-MOs mit HTTP-Methode *PUT* ausgeführt. Der Erfolg der Methode wird anhand des HTTP-Statuscodes der Antwort der OpenDaylight-Instanz ausgewertet. Statuscode 200 entspricht einer erfolgreichen Ausführung und wird entsprechend als `BooleanType` positiv zurückgegeben. In jedem anderen Fall oder im Fehlerfall wird `false` zurückgegeben. Listing 7.6 zeigt die Zuweisung des Treibers zur MOCK-Klasse `OpenDaylight`.

```
1  @Override
2  public Map<String, FunctionValue> execute(Map<String, FunctionValue> parameter)
       throws ExecutionError {
3          // Parameters
4          FunctionValue<DPIDType, String> switchID = parameter.get("switchID");
5          FunctionValue<MACAddressType, String> sourceMAC =
               parameter.get("sourceMAC");
6          String requestBody = "<?xml version=\"1.0\" encoding=\"UTF-8\"
               standalone=\"no\"?>\n" +
7                  "<flow xmlns=\"urn:opendaylight:flow:inventory\">\n" +
8                  "    <strict>false</strict>\n" +
9                  "    <instructions>\n" +
10                 "        <instruction>\n" +
11                 "            <order>0</order>\n" +
12                 "            <apply-actions>\n" +
13                 "                <action>\n" +
14                 "                    <order>0</order>\n" +
15                 "                    <drop-action/>\n" +
16                 "                </action>\n" +
17                 "            </apply-actions>\n" +
18                 "        </instruction>\n" +
19                 "    </instructions>\n" +
20                 "    <table_id>2</table_id>\n" +
21                 "    <id>126</id>\n" +
22                 "    <cookie_mask>255</cookie_mask>\n" +
23                 "    <installHw>false</installHw>\n" +
24                 "    <match>\n" +
25                 "        <ethernet-match>\n" +
26                 "            <ethernet-destination>\n" +
27                 "               <address>#SOURCEMAC#</address>" +
28                 "            </ethernet-destination>" +
29                 "        </ethernet-match>\n" +
30                 "    </match>\n" +
31                 "    <hard-timeout>0</hard-timeout>\n" +
32                 "    <idle-timeout>0</idle-timeout>\n" +
33                 "    <cookie>1337</cookie>\n" +
34                 "    <flow-name>srcBlock</flow-name>\n" +
35                 "    <priority>99</priority>\n" +
36                 "    <barrier>false</barrier>\n" +
37                 "</flow>";
38
```

```
39          // Prepare return value
40          try {
41              String sp = servp.getServicePoint();
42              sp = sp.replace("<NODEID>", switchID.getValue())
43                      .replace("<TABLEID>", "2")
44                      .replace("<FLOWID>", "126");
45
46              System.out.println(sp);
47
48              if (sourceMAC != null && sourceMAC.getValue() != null) {
49                  requestBody = requestBody.replace("#SOURCEMAC#",
                        sourceMAC.getValue());
50
51                  HttpClient httpc = HttpClient.newHttpClient();
52                  HttpRequest req = HttpRequest.newBuilder()
53                          .uri(URI.create(sp))
54                          .header("Content-Type", "application/xml")
55                          .PUT(HttpRequest.BodyPublishers.ofString(requestBody))
56                          .build();
57
58
59                  // For demonstration purposes as synchronous call
60                  HttpResponse<String> resp = httpc.send(req,
                        HttpResponse.BodyHandlers.ofString());
61                  System.out.println(resp.body().toString());
62                  System.out.println(resp.statusCode());
63                  assertNotNull(resp);
64
65                  // Check for HTTP status code
66                  if (resp.statusCode() == 200) {
67                      return Map.ofEntries(new AbstractMap.SimpleEntry<>("success",
                            new FunctionValue(Boolean.TRUE, new BooleanType())));
68                  }
69              }
70
71          } catch (NonConformValueError nonConformValueError) {
72              nonConformValueError.printStackTrace();
73          } catch (IOException ioe) {
74              ioe.printStackTrace();
75          } catch (InterruptedException e) {
76              e.printStackTrace();
77          }
78
79          FunctionValue<BooleanType, Boolean> result = null;
80          try {
81              result = new FunctionValue(Boolean.FALSE, new BooleanType());
82          } catch (NonConformValueError nonConformValueError) {
83              nonConformValueError.printStackTrace();
84          }
85          return Map.ofEntries(new AbstractMap.SimpleEntry<>("success", result));
86      }
87  };
```

Listing 7.7: Implementierung der `execute`-Methode eines Treibers für reale MOC-Funktionen `MOCFRDriver`.

7.2.3.3 Ausführung der Treiberfunktion

Die beispielhafte Definition eines Dienstzugriffspunktes sowie der Ablauf eines einfachen Aufrufs dieser Treiberfunktion sind in Listing 7.8 gezeigt.

Der Dienstzugriffspunkt wird zum Zweck eines einfachen Beispiels als String-Wert mit passender URL und entsprechend notwendigen Platzhaltern für zu ergänzende Werte der Switch-ID (<*NODEID*>), der Tabelle im Switch (<*TABLE*>) und der ID des dann angelegten Flows (<*FLOW*>) beschrieben. Strukturiertere Ansätze mit zusammengesetzten Feldvariablen sind

üblicherweise zu bevorzugen. Der angelegte Dienstzugriffspunkt des Ziel-MOs wird vor Ausführung der eigentlichen Treiberfunktion der Treiberklasse übergeben und schließlich die zuvor gezeigte `execute`-Methode mit den entsprechenden Parametern aufgerufen. Die Beispielimplementierung wurde mit OpenDaylight 14 und einem über Mininet generierten Open vSwitch getestet. Nach Aufruf der Funktion kann wie erwartet die korrekte Installation des Flow-Eintrags in dem Switch beobachtet werden, wie in Abbildung 7.6 (Eintrag mit *cookie=0x539*) gezeigt wird. Auf eine Auswertung des Rückgabewertes wird in diesem Beispiel verzichtet, muss jedoch in einer produktiven Implementierung durchgeführt werden, genauso wie eine asynchrone Ausführung der Funktion.

```
1  // Define exemplary service point for ODL instance
2  ServicePoint<String> odlServicePoint = new ServicePoint<String>() {
3      @Override
4      public String getServicePoint() {
5        return "http://127.0.0.1:8181/restconf/config/opendaylight-inventory:" +
6               "nodes/node/openflow:<NODEID>/table/<TABLEID>/flow/<FLOWID>";
7      }
8  };
9
10 // Set service point in driver
11 odldriver.setServicePoint(odlServicePoint);
12
13 // Execute driver function
14 try {
15     odldriver.execute(Map.ofEntries(new AbstractMap.SimpleEntry<>("switchID", new
           FunctionValue<DPIDType, String>("1", new DPIDType())),
16                                     new AbstractMap.SimpleEntry<>("sourceMAC",
                                           new FunctionValue<MACAddressType,
                                           String>("00:00:00:00:00:01", new
                                           MACAddressType()))));
17 } catch (ExecutionError executionError) {
18     executionError.printStackTrace();
19 } catch (NonConformValueError nonConformValueError) {
20     nonConformValueError.printStackTrace();
21 }
```

Listing 7.8: Definition eines Dienstzugriffspunktes und Beispielaufruf der Treiberfunktion für die MOC-Funktion `addSrcBlockingFlow`.

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
 cookie=0xa, duration=330.344s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:65535
 cookie=0x539, duration=22.863s, table=2, n_packets=0, n_bytes=0, priority=99,dl_dst=00:00:00:00:00:
01 actions=drop
mininet@mininet-vm:~$ _
```

Abbildung 7.6: Durch MOC-Funktion `addSrcBlockingFlow` generierter Flow in einer Open vSwitch-Instanz in einer Mininet-Testumgebung.

Die Wiederverwendbarkeit von MOC-Funktionen durch unterschiedliche MO-Klassen kann durch das beschriebene Framework für MOC-Funktionen durch *a)* die Wiederverwendung von Treiberimplementierungen und *b)* Implementierung weiterer Treiberfunktionen für MOCKs gewährleistet werden. Fall *a)* gilt bei Ableitung von MOCKs von einem Eltern-MOCK, das für eine MOC-Funktion bereits eine Treiberfunktion implementiert. Fall *b)* bedeutet, dass dieselbe MOC-Funktion für ein anderes MOCK mit einer passenden Treiberfunktion implementiert werden kann. Die hier beispielhaft gezeigte MOC-Funktion `addSrcBlockingFlow` würde beispielsweise durch jeden SDN-Controller unterstützt werden und kann entsprechend angepasst dafür implementiert werden.

7.2.4 Beispielhafte Netzinformationen

In diesem Abschnitt wird die Modellierung und Instanziierung von Informationen im Netzmanagement von FSNs durch das Framework für Netzinformationen gezeigt, die keine At-

tribute eines spezifischen MOs beschreiben. Dazu wird einerseits ein Network-Intrusion-Detection-System-(NIDS)-Ereignis `NIDSEvent` und davon abgeleitet eine NIDS-Anomalieklasse `NIDSAnomaly` jeweils als Netzereignis sowie andererseits eine Klasse zur Beschreibung einer Netztopologie `NetworkTopology` als Beispiel eines Netzzustands gezeigt.

```
/* Eve format sample representation
{
  "timestamp": "2016-01-11T05:10:54.612110-0800",
  "flow_id": 412547343494194,
  "pcap_cnt": 1391293,
  "event_type": "anomaly",
  "src_ip": "192.168.122.149",
  "src_port": 49324,
  "dest_ip": "192.168.100.100",
  "dest_port": 443,
  "proto": "TCP",
  "app_proto": "tls",
  "anomaly": {
    "type": "applayer",
    "event": "APPLAYER_DETECT_PROTOCOL_ONLY_ONE_DIRECTION",
    "layer": "proto_detect"
  }
}*/

// Parser for the used date format
SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZ");

// Sample as NIDSAnomaly object
NIDSAnomaly anomaly = new NIDSAnomaly(
        SampleID.gen(NIDSAnomaly.class),
        Set.of(nidsApp), // source MO
        Set.of(floodlightInstance), // related MOs
        new TimeStamp(dateFormat.parse("2016-01-11T05:10:54.612110-0800")),
        TimeStampSource.SourceMO, // timestamp source
        NetworkInformationType.IntrusionDetectionEvent, // network event type
        Collections.emptySet(), // related task domains
        "Detected anomaly event in the network", // event message
        true, // persistence flag
        InetAddress.getByName("192.168.122.149"), // source ip
        49324, // source port
        InetAddress.getByName("69.195.71.174"), // destination ip
        443, // destination port
        "TLS", // protocol
        AnomalyType.APPLAYER, // anomaly type
        "APPLAYER_DETECT_PROTOCOL_ONLY_ONE_DIRECTION", // anomaly event
        AnomalyLayer.PROTO_LAYER); // anomaly layer
```

Listing 7.9: Beispielhafte Anwendung eines `NIDSAnomaly`-Objekts als Abbildung des gezeigten Beispiels im Eve-Format aus [208].

7.2.4.1 Implementierung von Netzereignissen

Die Implementierung der Klassen `NIDSEvent` und `NIDSAnomaly` orientiert sich an der Dokumentation des JSON-basierten *Eve-Formats* des NIDS Suricata [208]: Demnach wird ein Eve-Eintrag über einen Block mit generellen Daten wie einem Zeitstempel, einem Ereignistyp und Quell- und Ziel-IP-Adressen und dem verwendeten Protokoll beschrieben. Weitere zu spezifische Attribute wie ein Pfad zu einer untersuchten Packet-Capture-(PCAP)-Datei werden hier nicht berücksichtigt, um die definierten Netzinformationen auch zur Beschreibung von Informationen aus anderen Formaten nutzen zu können. Eine Auswahl dieser Attribute wird daher in der Klasse `NIDSEvent` gekapselt. Speziellere Ereignistypen wie Warnungen (*Alert*), Anomalien (*Anomaly*) und viele weitere werden im Eve-Format durch zusätzlich angehängte Attribute ergänzt.

Ein Anomalie-Eintrag, wie er über die Klasse `NIDSAnomaly` implementiert wird, ergänzt die Attribute eines Anomalietyps, einer Anomaliebeschreibung, eines Ereigniscodes und der Schicht, in der die Anomalie erkannt wurde. Die Klasse `NIDSAnomaly` als spezifischere NIDS-Ereignisklasse wird daher von der Klasse `NIDSEvent` abgeleitet. Die Klasse `NIDSEvent` wird hingegen, da es sich um ein Netzereignis handelt, von der Klasse `NetworkEvent` des Frameworks für Netzinformationen abgeleitet und wird daher in der Managementplattform als solches behandelt.

Listing 7.9 zeigt schließlich die Anwendung der `NIDSAnomaly`-Klasse auf einem beispielhaften Eve-Objekt, das an die Umsetzung in [208] angelehnt wurde. Die ersten neun Attribute des Konstruktors beschreiben Elemente des Frameworks für Netzinformationen. So wird hier als Quelle eine modellierte NIDS-Anwendung `nidsApp` angenommen und als beispielhaft damit verknüpftes MO eine Instanz des SDN-Controllers `floodlightInstance`. Schließlich wird der aus dem Originalereignis genommene Zeitstempel geparst und übernommen und der Zeitstempel als vom Quell-MO generiert markiert (`TimeStampSource.SourceMO`). Der Typ des Netzereignisses wird darüber hinaus als ein IDS-Event charakterisiert und schließlich in diesem Fall keine damit verknüpften Aufgabenbereiche (Task-Domains) angegeben. Das Persistenz-Flag kennzeichnet mit dem Wert `true`, dass das generierte Netzereignis automatisch entfernt werden soll. Die darauf folgenden Konstruktorparameter sind typspezifisch für `NIDSEvent` und `NIDSAnomaly` und gemäß dem Ursprungsobjekt übernommen.

```
1  // Generate set of topology links
2  Set<TopologyLink> links =
       dependencyManager.getDependenciesOfClassOfMOC(switchDOM1_1,
       DHNIpNetwork.class).stream().map(dependency -> {
3      DHNIpNetwork ipnetdependency = (DHNIpNetwork) dependency;
4      // Generate list of links
5      return new TopologyLink(new
           TopologyNode(ipnetdependency.getObject().getId().serialize()), new
           TopologyNode(ipnetdependency.getSubject().getId().serialize()));
6  }).collect(Collectors.toSet());
7
8  // Generate set of topology nodes
9  Set<TopologyNode> nodes = links.stream().map(topologyLink -> {
10     return Set.of(topologyLink.getN1(), topologyLink.getN2());
11 }).flatMap(Collection::stream).collect(Collectors.toSet());
12
13 NetworkTopology topo = new NetworkTopology(SampleID.gen(NetworkTopology.class),
14                 Set.of(odlInstance), // source
15                 Set.of(switchDOM1_1), // related MOs
16                 new TimeStamp(System.currentTimeMillis()), // timestamp
17                 TimeStampSource.PlatformGenerated, // timestamp source
18                 NetworkInformationType.NetworkSnapshot, // type of event
19                 Collections.emptySet(), // related task domains
20                 "The topology around the domain-central switch in domain DOM
                       1.1", // description
21                 false, //persistence flag
22                 nodes, // nodes
23                 links); // edges
```

Listing 7.10: Generierung eines Netztopologie-Objekts auf Basis von Informationen aus dem MOC- und Managementbeziehungsframework.

7.2.4.2 Implementierung eines Netzzustands

Der beispielhaft implementierte Netzzustand `NetworkTopology` wird als eine Menge von Knoten und eine Menge von Kanten implementiert. Ein Knoten wird mit einer Hilfsklasse `TopologyNode` implementiert, das hier als im einfachsten Fall einen String-Wert `id` hält. Eine Kante wird durch die Klasse `TopologyLink` implementiert und referenziert auf zwei `TopologyNode`-Instanzen. Diese einfache Form der Implementierung kann beispielsweise zur

Visualisierung einer (Teil-) Netztopologie in einer grafischen Oberfläche genutzt werden. Die eigentliche Netzzustand-Klasse `NetworkTopology` wird selbst von der Klasse `NetworkState` abgeleitet.

Listing 7.10 zeigt, wie beispielhaft aus Informationen des MOC-Frameworks und dem Managementbeziehungsframework die Netztopologie für ein Teilnetz der modellbasierten Anwendungsimplementierung generiert werden kann. Die Netztopologie wird für den zuvor in Abbildung 7.4 gezeigten Switch `switchDOM1_1` in Domäne DOM 1.1 (der Domäne der Managementinfrastruktur) generiert. Dazu wird zunächst die Menge an Kanten aus den `DHNIpNetwork`-Abhängigkeitsobjekten generiert, die für den Switch registriert sind, und diese jeweils abgebildet als `TopologyLink`-Instanz. Für jedes referenzierte Subjekt- und Objekt-MO wird für jede Abhängigkeit ein `TopologyNode`-Objekt mit serialisierter ID des jeweiligen MO generiert. Im nächsten Schritt wird aus dieser Menge an Kanten die Menge an Knoten abgeleitet, indem alle Knoten in einer Menge (`Set`) von Mengen pro Abhängigkeit gesammelt werden. Die Mengeneigenschaft sichert die Eindeutigkeit der `TopologyNode`-Objekte. Die Menge aus Mengen an MOs, die über die Kanten gesammelt wurden, wird über die Methode `flatMap` schließlich zu einer Menge zusammengefasst. Die Instanziierung eines `NetworkTopology`-Objekts erfolgt analog zu der von Netzereignissen.

7.2.5 Beispielhaftes Alarm-Schema

Als beispielhafte Anwendung des Frameworks für Alarme wird das in Abbildung 7.7 gezeigte Alarm-Schema aus [183] implementiert. Von `AlertType` werden zwei unterschiedliche Gruppen von Alarmen abgeleitet: Zum einen generelle Netzalarme über die Klasse `NetAlert` und zum anderen Alarme spezifisch für Sicherheitsvorkommnisse `SecAlert`. Davon jeweils ausgehend wird ein Alarm `Info` mit verhältnismäßig geringster Kritikalität abgeleitet. Im Kontext von Sicherheitsalarmen werden zusätzlich Alarme für höhere Kritikalitäten `Warning` und der höchsten Kritikalität `Fatal` abgeleitet. Im Kontext von Netzalarmen wird hingegen eine weitere Trennung nach dem `Info`-Element eingeführt, einerseits für Ausfälle mit Klasse `Failure` und andererseits für Einschränkungen mit Klasse `Limitation`. Von beiden wird zudem jeweils ein `Fatal`-Element mit höchster Kritikalität abgeleitet. Die Kritikalität einer Klasse richtet sich daher nach der Ebene im Ableitungsbaum des Alarm-Frameworks: Je näher eine Klasse an den Blättern des Baumes ist, desto höher ist die Kritikalität.

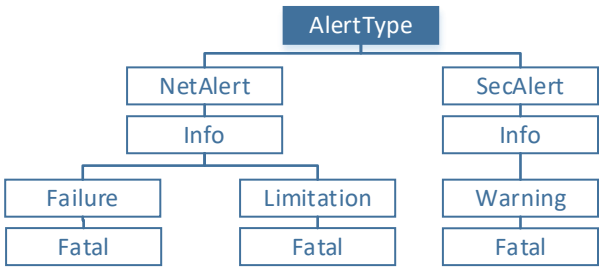


Abbildung 7.7: Beispielhaft implementiertes Alarm-Schema aus [183]. Die blaue Box entspricht dem Frameworkelement, weiße Boxen sind davon abgeleitete Klassen.

Um in der Implementierung Probleme mit der Benennung einer Alarmklasse zu vermeiden – beispielsweise gibt es zwei Klassen mit dem Namen „Info“ und drei Klassen, die „Fatal“ heißen – werden in Java Subpackages für jede Verzweigung genutzt. Eine weitere Besonderheit ist die Unterscheidung zwischen instanziierbaren und nicht instanziierbaren Alarmklassen. So stellen hier die Klassen `NetAlert` und `SecAlert` lediglich Gruppierungselemente dar, sind aber nicht als instanziierbare valide Alarmklassen vorgesehen. Derartige Gruppierungselemente werden in Java als abstrakte Klasse implementiert. Listing 7.11 zeigt stellvertretend für alle Alarmklassen die Implementierung der Klasse `NetAlert`.

```java
public abstract class NetAlert extends AlertType {
    // Constructor
    public NetAlert(Identificator id, Set<AlertCategory> alertCategory, String
        description, TimeStamp timeStamp) {
        super(id, alertCategory, description, timeStamp);
    }

    /**
     * Return a serialized version of this alert
     *
     * @return Alert representation as a string.
     */
    public String serialize() {
        return String.format("%s;%s;%s",
                getAlertCategory().stream().map(alertCategory -> {
                    return alertCategory.toString();
                }).reduce((s, s2) -> {
                    return s + ", " + s2;
                }).get(),
                getDescription(),
                getTimeStamp().getTimeStamp().toString());
    }

    /**
     * Function for deserialization of an alarm. E.g. for exchanging
     * with other management platform instances.
     * Sets the field values according to the serialized form.
     */
    public void deserialize(String serialized) {
        Set<AlertCategory> categories =
            Arrays.stream(serialized.substring(0,
                serialized.indexOf(";")).split(",")).map(s -> {
              return AlertCategory.valueOf(s.strip());
        }).collect(Collectors.toSet());
        String descr = serialized.substring(serialized.indexOf(";") + 1,
            serialized.lastIndexOf(";"));
        TimeStamp timestamp =
            new TimeStamp(Long.getLong(serialized.substring(
              serialized.lastIndexOf(";") + 1)));

        this.setAlertCategory(categories);
        this.setDescription(descr);
        this.setTimeStamp(timestamp);
    }
}
```

Listing 7.11: Implementierung der Klasse `NetAlert`.

```java
Fatal ff = new Fatal(
    Set.of(AlertCategory.MALWARE, AlertCategory.SYSERROR),
    "Exemplary system failure",
    new TimeStamp(System.currentTimeMillis()));
```

Listing 7.12: Beispielhafte Instanziierung der Alarmklasse `Fatal`.

Der Alarmklasse werden demnach keine weiteren Attribute hinzugefügt, sondern sie implementiert insbesondere die Methoden zur Serialisierung und Deserialisierung der Klasse. Zum Zweck der Kürze wird der Alert in eine einfache CSV-Darstellung serialisierbar und von dieser wieder deserialisierbar gemacht. Auf im produktiven Einsatz unbedingt notwendige Fehlerbehandlung und die Beschreibung des Typs des Alerts in der serialisierten Form wird daher in diesem Beispiel bewusst verzichtet. Die Methoden `serialize` und `deserialize` werden genauso in der Klasse `SecAlert` implementiert. Von beiden Klassen abgeleitete Alarmklassen müssen diese daher nicht neu implementieren, da keine neuen Attribute eingebunden werden. Das im Folgenden gezeigte Listing 7.12 zeigt schließlich eine beispielhafte Instanziierung der Klasse

`Fatal` aus dem Alarmklassen-Arm der Sicherheitsalarme. In diesem Fall hat die Alarminstanz zwei Alarm-Kategorien *MALWARE* und *SYSERROR*, eine Beschreibung und einen beispielhaft erstellten Zeitstempel.

7.3 Organisationsmodell

Die Reihenfolge der Anwendung der Frameworks des Organisationsmodells unterscheidet sich von der Reihenfolge ihrer Beschreibung aufgrund von in der Implementierung notwendigerweise zu berücksichtigenden Abhängigkeiten. So wird insbesondere eine Föderation bzw. im Modell ein `Federation`-Element auf Basis grundsätzlich bereits definierter Teilelemente festgelegt: Insbesondere Managementaufgaben, Domänen und Zuständigkeitsbereiche werden in der Implementierung zur initialen Definition einer Föderation benötigt; die gesamte definierte Föderation hingegen im Framework zur Steuerung von Sichtbarkeit und Zugriff von Informationselementen.

Über die Flexibilität aller Teile des Organisationsmodells können die Szenariencharakteristika einer hohen **Dynamik** und flexiblen **Rollenvergabe** behandelt werden. Strukturen auf allen Ebenen (Datenzentren, Parteien, Domänen und darin zu erfüllende Aufgaben durch domänenspezifische Rollen) können flexibel gesteuert und unmittelbar angepasst werden: In diesem Szenario mit impliziter **Koordination** von unterschiedlichen Seiten der Partner über eine entsprechende Zugriffssteuerung auf Plattformfunktionen, wie in Abschnitt 7.3.3 ebenfalls kurz beschrieben wird.

```
1  Task tSPFMonitor = new SPFMonitorTask(
2              SampleID.gen(Task.class),
3              "Monitor",
4              "Monitoring of security, performance " +
5              "and fault metrics in the network");
6  Task tSPFAudit = new SPFAuditTask(
7              SampleID.gen(Task.class),
8              "Audit",
9              "Audit and testing task for security, " +
10             "performance and fault management");
11 Task tSPFConfigure = new SPFConfigureTask(
12             SampleID.gen(Task.class),
13             "Configure",
14             "Configuration and adaption in order " +
15             "to maintain the effectiveness of assets");
16
17 tdSecurityPerformanceFault = new TaskDomain(
18             SampleID.gen(TaskDomain.class),
19             "SecurityPerformanceFaultManagement",
20             "Group of tasks for monitoring, testing" +
21             " and maintaining assets' effectiveness " +
22             "in matters of security, performance and faults",
23             tSPFMonitor,
24             Set.of(AlertCategory.MALWARE, AlertCategory.SYSERROR));
25
26 try {
27     tdSecurityPerformanceFault.addTask(tSPFAudit);
28     tdSecurityPerformanceFault.addTask(tSPFConfigure);
29 } catch (TaskAlreadyExistingException e) {
30     e.printStackTrace();
31 }
```

Listing 7.13: Anlegen eines Aufgabenbereichs für Sicherheit- Performanz- und Fehlermanagement und Beschreibung von Teilaufgaben.

7.3.1 Managementaufgaben

Da das betrachtete Evaluierungsszenario von der Größe eher klein ist, werden die Managementaufgaben nicht möglichst feingranular, sondern vielmehr nach Ähnlichkeit der Teilaufgaben gruppiert. So werden drei der fünf FCAPS-Managementaufgaben [33] Security-, Performance- und Faultmanagement in einem Aufgabenbereich bzw. `TaskDomain`-Element zusammengefasst. Das Abrechnungsmanagement wird in diesem Szenario nicht notwendigerweise benötigt und das Konfigurationsmanagement wird vielmehr durch den zweiten Aufgabenbereich abgedeckt. Der zweite Aufgabenbereich wird schließlich zur Umfassung von VM-Lebenszyklusmanagementaufgaben beschrieben.

Listing 7.13 zeigt beispielhaft die Definition des ersteren `TaskDomain`-Elements `tdSecurityPerformanceFault` für Sicherheits- Performanz- und Fehlermanagement und seiner einzelnen Teilaufgaben. Die Teilaufgaben der Überwachung, Auditierung und Konfiguration werden jeweils als Aufgabe von der Klasse `Task` abgeleitet und mit einer eindeutigen ID, einem Namen und einer Kurzbeschreibung definiert. Die Ableitung einer jeweils neuen Subklasse von `Task` ist in dieser Implementierung insbesondere zur Unterscheidung der verschiedenen Zuständigkeiten im Zugriffsverwaltungsbaum 7.3.6 notwendig. Analoges gilt für die Definition eines Aufgabenbereichs, der jedoch bei Definition unter Angabe eines `Task`-Elements generiert werden muss, um sicherzustellen, dass ein Aufgabenbereich nicht ohne Teilaufgaben festgelegt wird. Darüber hinaus wird einem Aufgabenbereich eine Menge an `AlertCategory`-Elementen übergeben, durch welche für einen Aufgabenbereich relevante Alarme identifiziert werden können. Weitere Teilaufgaben werden schließlich im dann erstellen `TaskDomain`-Objekt durch die Methode `addTask` übergeben.

7.3.2 Administrative Domänen und Überschneidungsbehandlung

Die Anwendung des Frameworks für administrative Domänen umfasst zum einen die Modellierung der in Abbildung 7.1 gezeigten Domänenstruktur und zum anderen der Domänenüberschneidung.

```
1  List<Domain> dom2x = Stream.generate(() -> {
2      return new Domain(SampleID.gen(Domain.class));
3  }).limit(LEVEL2_DOMAIN_COUNT).collect(Collectors.toList());
4
5  // Distribute other MOs on level 2 to all DOMs on that level
6  IntStream.range(1, NUMBER_OF_GENERIC_LEVEL_2_INSTANCES).forEach(i -> {
7    dom2x.get(i % LEVEL2_DOMAIN_COUNT).addManagedObject(genericLevel2MOs.get(i));
8  });
9
10 // Put one MO (index 0) in the domains DOM2.5 and DOM2.6
11 dom2x.get(4).addManagedObject(genericLevel2MOs.get(0));
12 dom2x.get(5).addManagedObject(genericLevel2MOs.get(0));
```

Listing 7.14: Anlegen von Domänen der Ebene 2 und Hinzufügen von MOs.

7.3.2.1 Anlegen administrativer Domänen

Die Instanziierung einer neuen administrativen Domäne erfordert lediglich die Übergabe einer föderationsweit eindeutigen ID. Listing 7.14 zeigt die Modellierung am Beispiel der Domänen und MOs auf Ebene 2 des Szenarios. Die Anzahl der generierten Domänen ist in der Evaluierung festlegbar und wird durch die Konstante `LEVEL2_DOMAIN_COUNT` definiert. Nach dem Anlegen der Domänen werden die MOs der Ebene 2 einzeln auf alle angelegten Domänen verteilt. Ebene 2-MOs sind zum einfacheren Zugriff in der Liste `genericLevel2MOs` organisiert; ihre Anzahl ist durch die Konstante `NUMBER_OF_GENERIC_LEVEL_2_INSTANCES` bestimmt. Eine Überschneidung der Domänen DOM2.5 und DOM2.6 wird über erste darin definierte MO modelliert, wie in Abbildung 7.1 gezeigt wird.

Die **Domänenstruktur** kann durch den in dieser Arbeit konzipierten Ansatz quasi beliebig unterstützt werden. Als Szenariencharakteristika wurden die Ausprägungen *global* einheitlich, *hierarchisch* oder auch *unabhängig* betrachtet, wobei das Anwendungsszenario *unabhängig* ausgeprägt ist (vgl. Abschnitt 7.1.1). Domänenüberschneidungen werden in diesem Konzept feingranular auf Ebene des sich jeweils überschnittenen MOs behandelt und können diese daher unabhängig von der Struktur handhaben.

```
// Instantiate DomainCrossOrganization Table
domainCrossOrganizationTable = new DomainCrossOrganizationTable();

// Example: Register level 2 domains at domain cross organization table
dom2x.stream().forEach(domain -> domainCrossOrganizationTable.addDomain(domain));

// Register domain overlap for domains DOM2.5 and DOM2.6
DomainCrossOrganizationEntry dcoe2526 = new DomainCrossOrganizationEntry(
        SampleID.gen(DomainCrossOrganizationEntry.class),
        "DCOE describing the intersection of the MO in Domain DOM2.5 and DOM2.6");
dcoe2526.addDomain(dom2x.get(4));
dcoe2526.addDomain(dom2x.get(5));
domainCrossOrganizationTable.addIntersectingMO(genericLevel2MOs.get(0), dcoe2526);

// Example: Get overlaps of domains DOM2.5 and DOM2.6
Set<DomainCrossOrganizationEntry> ret =
    domainCrossOrganizationTable.getDomainOverlaps(Set.of(dom2x.get(4),
    dom2x.get(5)));
```

Listing 7.15: Instanziierung der Domänenkreuzorganisationstabelle und Registrierung von Domänen und Domänenüberschneidungen.

```
// Remodel responsibilities for domain intersection
tdSecurityPerformanceFault.getTasks().forEach(task -> {
    try {
        dcoe2526.addTask(task);
        dcoe2526.addTaskRole(task, new SPFManager());
    } catch (TaskAlreadyExistingException e) {
        e.printStackTrace();
    } catch (TaskNotExistingException e) {
        e.printStackTrace();
    } catch (RoleAlreadyExistsException e) {
        e.printStackTrace();
    }
});
```

Listing 7.16: Neuzuweisung von Zuständigkeiten für eine Domänenüberschneidung.

7.3.2.2 Behandlung von Domänenüberschneidungen

Die Definition der Domänen stellt nur den ersten Schritt in der Verwaltung dar. Um **Domänenüberschneidungen** zuverlässig und effizient erfassen zu können, werden die Domänen in der Domänenkreuzorganisationstabelle (DKOT) registriert. Listing 7.15 zeigt zuerst die Instanziierung der DKOT, gefolgt von der beispielhaften Registrierung aller Domänen auf Ebene 2 im Szenario. Domänen der anderen Ebenen werden analog registriert. Im dritten Schritt wird die zuvor bereits beschriebene Domänenüberschneidung eines MO in Domäne DOM2.5 und DOM2.6 über die Instanziierung eines Domänenkreuzorganisationseintrags `dcoe2526` gezeigt. Diesem werden die sich überschneidenden Domänen übergeben und der Eintrag schließlich in der DKOT inklusive dem MO als Element der Überschneidung registriert. Die DKOT stellt schließlich Methoden zur effizienten Erfassung von Domänenüberschneidungen bereit.

Das Frameworkkonzept sieht vor, dass **Zuständigkeiten** für eine Domänenüberschneidung flexibel definiert werden können. Die Festlegung von Zuständigkeiten erfolgt dann für den jeweiligen Domänenkreuzorganisationseintrag wie hier `dcoe2526`. Listing 7.16 zeigt eine beispielhafte Zuweisung der Zuständigkeit der Rolle `SPFManager` (siehe folgender Abschnitt) zu allen

Managementaufgaben für Security-, Performance-, und Faultmanagement. Zunächst muss dafür jede Aufgabe und schließlich für eine vorhandene Aufgabe eine Rolle zugewiesen werden.

In der Anwendung wird darüber hinaus die **Kooperation zweier Projektpartner** in eine Domänenbeziehung `DomainGroup` zwischen den Domänen DOM2.2 und DOM2.3 hergestellt. Durch diese Domänenbeziehung wird eine temporäre Kooperation im Management über diese Domänen hinweg beschrieben. Das Anlegen einer `DomainGroup`-Instanz ist in Listing 7.17 gezeigt. Die Instanz wird demnach mit einer eindeutigen ID und einer Kurzbeschreibung erstellt und danach werden die Domänen DOM2.2 und DOM2.3 (das zweite bzw. dritte Element der Aufzählung `dom2x`) hinzugefügt.

```
1 dom22_23group = new DomainGroup(SampleID.gen(DomainGroup.class),
2   "Temporary management cooperation between DOM2.2 and DOM2.3");
3 dom22_23group.addDomain(dom2x.get(1));
4 dom22_23group.addDomain(dom2x.get(2));
```

Listing 7.17: Anlegen der Domänenbeziehung der `DomainGroup`-Klasse für Domänen DOM2.2 und DOM2.3.

7.3.3 Nutzer und Rollen

Neue Rollen werden von der Klasse `Role` abgeleitet. In diesem Beispiel wird für die zuvor in Abschnitt 7.3.1 definierten Managementaufgaben folgendes einfaches Rollenmodell entwickelt: Für die Aufgaben des Aufgabenbereichs Security-, Performance- und Faultmanagement wird die Rolle `SPFManager` und für die Aufgaben des Aufgabenbereichs die Rolle des `LifecycleManager` angelegt. Auch wenn Rollen potenziell Attribute zugewiesen bekommen können, wird in diesem Fall darauf verzichtet. Die Implementierung wird in Listing 7.18 gezeigt. Eine zusätzliche Rolle `SPFAuditor` wird in diesem Kontext zur beispielhaften Verfeinerung der Verteilung von Zuständigkeiten angelegt. Auf diese wird im nächsten Abschnitt eingegangen.

```
1 public class SPFManager extends Role {}
```

Listing 7.18: Anlegen der Rolle `SPFManager` durch ihre Ableitung von der Klasse `Role`.

Die Beschreibung von Nutzern der Managementplattform wird im einfachsten Fall durch die Instanziierung der Klasse `User` implementiert. Falls eine Erweiterung der einen Nutzer beschreibenden Attribute notwendig ist, wird eine entsprechende Klasse um notwendige Attribute ergänzt.

An dieser Stelle ist das Szenariencharakteristikum der **Koordination** zu berücksichtigen. Für eine implizite Koordination, wie sie in diesem Baseline-Szenario betrachtet wird, kann der Zugriff auf Plattformfunktionen gleichmäßig verteilt werden. Die Nutzer der einzelnen Partner sollen dann gleichartig auf die Funktionalität der Managementplattform zugreifen können. In diesem Fall müssen `NBIPullComponent`- (d. h. der API-Zugriff auf Plattformfunktionen) und *Entity*-Instanzen dieselbe Privilegienebene zugewiesen werden.

7.3.4 Gültigkeitsbereiche und Zuständigkeiten

Die hier definierten Zuständigkeiten knüpfen unmittelbar an die in Abschnitt 7.3.1 definierten Managementaufgaben sowie die im letzten Abschnitt angelegten Rollen an. Im Folgenden wird eine beispielhafte Belegung von Zuständigkeiten auf globaler Ebene mit einer domänenlokalen Verfeinerung gezeigt. Dazu wird als globale Richtlinie eine derartige Belegung implementiert, dass alle Teilaufgaben des Aufgabenbereichs Security-, Performance- und Faultmanagement `tdSecurityPerformanceFault` durch die Rolle des `SPFManager` und alle Teilaufgaben des Aufgabenbereichs des Lifecyclemanagements `tdLifecycle` durch die Rolle des `LifecycleManager` erfüllt werden muss.

7.3.4.1 Zuweisung föderations-globaler Zuständigkeiten

Die Rolle wird der jeweiligen Teilaufgabe durch eine `Responsibility`-Instanz zugewiesen, wie in Listing 7.19 beispielhaft für die Aufgabe *Audit* des Aufgabenbereichs des Security-, Performance- und Faultmanagements gezeigt wird. Die darin definierte Zuständigkeit `globalSPFManagerAudit` wird schließlich an der `ResponsibilitiesMap`-Instanz im globalen Kontext registriert und gilt daher immer, sofern es nicht domänenlokal überschrieben wird.

```
1 // Define global responsibility
2 globalSPFManagerAudit = Responsibility<SPFManager>(
3         SampleID.gen(Responsibility.class),
4         SPFManager.class,
5         tdSecurityPerformanceFault.getTaskByName("Audit"));
6
7 // Register global responsibility
8 responsibilitiesMap.addResponsibility(globalSPFManagerAudit);
```

Listing 7.19: Definition der globalen Zuständigkeit der Rolle `SPFManager` für die Aufgabe *Audit* des Aufgabenbereich `tdSecurityPerformanceFault`.

7.3.4.2 Zuweisung Domänen-lokaler Zuständigkeiten

Das Anlegen einer domänenlokalen Zuständigkeit erfolgt analog wie im globalen Kontext und unterscheidet sich lediglich über die Art der Registrierung an der `ResponsibilitiesMap`-Instanz, wie in Listing 7.20 gezeigt wird. In diesem Fall wird die Zuständigkeit für eine (hier zufällige) Domäne registriert, die der `ResponsibilitiesMap`-Instanz mitgegeben wird. Auf diese Weise kann auch das Szenariencharakteristikum der *dynamischen Aufgabenzuordnung* behandelt werden.

```
1  // Define local responsibility
2  localSPFAuditorResponsibility = new Responsibility<SPFAuditor>(
3          SampleID.gen(Responsibility.class),
4          SPFAuditor.class,
5          tdSecurityPerformanceFault.getTaskByName("Audit"));
6
7  // Retreive arbitrary domain from DomainCrossOrganizationTable
8  Domain someDomain =
       domainCrossOrganizationTable.getDomains().stream().findFirst().get();
9
10 // Register responsibility for a domain
11 responsibilitiesMap.addResponsibility(someDomain, localSPFAuditorResponsibility);
```

Listing 7.20: Beispielhafte Definition einer lokalen Zuständigkeit der Rolle `SPFAuditor` für die Aufgabe *Audit* des Aufgabenbereich `tdSecurityPerformanceFault`.

```
1  // Result: SPFAuditor
2  Class<? extends Role> r1 =
       responsibilitiesMap.getResponsibilityForTask(someDomain,
3          tdSecurityPerformanceFault.getTaskByName("Audit"));
4
5  // Get arbitrary other domain from DomainCrossOrganizationTable
6  Domain otherDomain = domainCrossOrganizationTable.getDomains().stream()
7          .skip(5).findFirst().get();
8
9  // Result: SPFManager
10 Class<? extends Role> r2 =
       responsibilitiesMap.getResponsibilityForTask(otherDomain,
11         tdSecurityPerformanceFault.getTaskByName("Audit"));
```

Listing 7.21: Beispielanfrage der Zuständigkeit für die Aufgabe `Audit` mit unterschiedlichem Ergebnis.

7.3.4.3 Abfrage von Zuständigkeiten

Eine Abfrage der Zuständigkeit für die Aufgabe *Audit* des Security-, Performance- und Faultmanagements gibt für die Domäne mit lokaler Überschreibung eine andere Rolle wie für alle anderen Domänen zurück. Listing 7.21 zeigt das Ergebnis von zwei solchen Anfragen.

7.3.5 Föderationsmodell

Auf Basis der in den vorherigen Abschnitten definierten organisatorischen Strukturen (Managementaufgaben, die Domänenstruktur und -Beziehungen, der Nutzerrollenstruktur sowie den definierten Zuständigkeiten) kann nun das Föderationsmodell vollständig beschrieben werden. Dazu werden im Kontext des Frameworks für Föderationen zunächst die einzelnen Parteien und dazugehörigen Datenzentren des Evaluierungsszenarios angelegt.

7.3.5.1 Anlegen von Parteien und Datenzentren

Listing 7.22 zeigt das Anlegen von Partei P1 und dem dazugehörigen Datenzentrum. Eine Partei `Party` wird mit einem eindeutigem Identifikator und einer Kurzbeschreibung angelegt. Das entsprechende Datenzentrum wird durch dieselben Informationen und darüber hinaus die dafür zuständige Partei (hier die zuvor angelegte Partei P1), den Satz darin unmittelbar organisierter MOs auf Ebene 0 und einer einfachen Ortsbeschreibung angelegt. Die Ortbeschreibung wird zu Demonstrationszwecken als einfacher String realisiert. Die anderen Parteien und Datenzentren werden analog angelegt. Hier wird das Szenariencharakteristikum der **räumlichen Dimension** modelliert, indem der Ort über ein entsprechendes `Location`-Element beschrieben wird und im Netzmanagement als Entscheidungskriterium genutzt werden kann.

```
1  // Describe party P1
2  p1 = new Party(SampleID.gen(Party.class), "P1");
3
4  // Describe the data center of party P1 with simple location
5  dataCenterP1 = new DataCenter(
6          SampleID.gen(DataCenter.class),
7          "Data center of partner 1",
8          p1,
9          Set.of(qemuInstanceP1),
10         new Location("Region1/DC1"));
```

Listing 7.22: Anlegen von Partei P1 und dem dazugehörigen Datenzentrum.

7.3.5.2 Anlegen von Föderationsbeziehungen

Auf dieser Grundlage können Föderationsbeziehungen angelegt werden. In diesem Fall werden beispielhaft zwei verschiedene Vertrauensbeziehungen sowie eine Multibeziehung zur Beschreibung der primären Projektpartner betrachtet. Die ersteren beiden werden direkt von der Klasse zur Beschreibung von Vertrauensbeziehungen `TrustRelationship` als `HighTrustRelationship` und `LowTrustRelationship` für hohes resp. niedriges Vertrauen abgeleitet. Die Multibeziehung zur Beschreibung der primären Projektpartner wird als Klasse `ProjectPartners` von der Klasse `FederationMultiRelationship` abgeleitet. Während die ersteren einfach gerichtete Beziehungen von einer Partei zu einer anderen sind, sind letztere ungerichtete n-zu-n-Beziehungen zwischen allen Parteien in einer Menge. In diesem Beispiel wird angenommen, dass sich die Parteien P1 und P2 als regionale Partner gegenseitig hohes Vertrauen zusprechen, wodurch zwei `HighTrustRelationship`-Instanzen angelegt werden – eine von P1 nach P2 und eine von P2 nach P1. Auch wird angenommen, dass Partei P2 der Partei P3 noch nicht vertraut und daher eine `LowTrustRelationship`-Instanz von P2 nach P3 angelegt wird. Auch werden aber die primären Projektpartner in eine `ProjectPartners` Beziehung zueinander gesetzt, wodurch Dienstleister D eine Sonderrolle bekommt und die unterschiedliche **Zielsetzung** modelliert werden kann. Letztere Beziehung wird beispielhaft in Listing 7.23 gezeigt.

```
ProjectPartners projectPartners = new ProjectPartners(
        SampleID.gen(ProjectPartners.class),
        "Group of primary project partners",
        Set.of(p1, p2, p3))
```

Listing 7.23: Beschreibung der Beziehung der Parteien P1, P2 und P3 als primäre Projektpartner über eine `ProjectPartners`-Instanz.

Durch die Föderationsbeziehung `ProjectPartners` kann auch das Charakteristikum der **Art der Infrastrukturnutzung** modelliert werden. In diesem Fall sind die Nutzer der Infrastruktur die Parteien P1 bis P3. Die Föderationsbeziehung fließt in die Zugriffskontrolle in Abschnitt 7.3.6 ein und bestimmt daher mit, welcher Partner auf welche IT-Ressourcen Zugriff bekommt. Bei einer Alternativausprägung des Charakteristikums, einer ***asymmetrischen Infrastrukturnutzung***, würde auf Basis der Zugehörigkeit eines Nutzers zu einem Föderationspartner der Managementzugriff auf IT-Ressourcen entsprechend eingeschränkt werden.

7.3.5.3 Anlegen des zentralen Föderationsobjekts

Nach Anlegen der Parteien, Datenzentren, Föderationsbeziehungen und Strukturen des Organisations- und Informationsmodells sind alle Informationen für die in Listing 7.24 gezeigte Initialisierung des zentralen Föderationsobjekts vorhanden. Dafür werden das Datum der Initialisierung und die Dauer der Föderation in Tagen, der Satz an Parteien und Datenzentren, der Aufgabenbereiche, der DKOT, der Föderations- und Domänenbeziehungen benötigt. Die hier beispielhaft gezeigte Implementierung mit Mengen- und Map-Generatoren ist für eine produktive Implementierung nicht geeignet, da sie *immutable* (d. h. unveränderbare) Strukturen erzeugt, dient jedoch einer einfachen Demonstration zum Anlegen dieses zentralen Objekts.

```
Federation.initialize(new TimeStamp(System.currentTimeMillis()), // Init-Date
        365, // Duration in days
        Set.of(p1, p2, p3, d), // parties
        Set.of(dataCenterP1, dataCenterP2, dataCenterP3, dataCenterD), // DCs
        Map.of("SecPerfFaultMan", tdSecurityPerformanceFault,
                "LifecycleMan", tdLifecycle), // Task domains
        domainCrossOrganizationTable, // DKOT
        Set.of(highTrustp1p2, highTrustp1p2, lowTrustp2p3), // federation
            relations
        Set.of(dom22_23group)); // Domain relations
```

Listing 7.24: Initialisierung der zentralen `Federation`-Instanz einer Managementplattform zur Beschreibung der Föderation als Ganzes.

7.3.6 Zugriffsverwaltung

Die Implementierung der Zugriffsverwaltung für das Anwendungsszenario besteht aus drei übergeordneten Teilen:

- Die Implementierung des Zugriffsverwaltungsbaums (ZVB) als **Modell** durch die einzelnen Knoten und darin enthaltenen Elemente,
- die Implementierung der **Traversierung** des ZVB, sowie
- die Implementierung des eigentlichen **Zugriffsregelsatzes**.

7.3.6.1 Implementierung des ZVBs als Modell

Das **Modell** des beispielhaft implementierten Zugriffsverwaltungsbaums für das Evaluationsszenario ist in Abbildung 7.8 gezeigt. Demnach wird durch Föderationsbeziehungen (graue

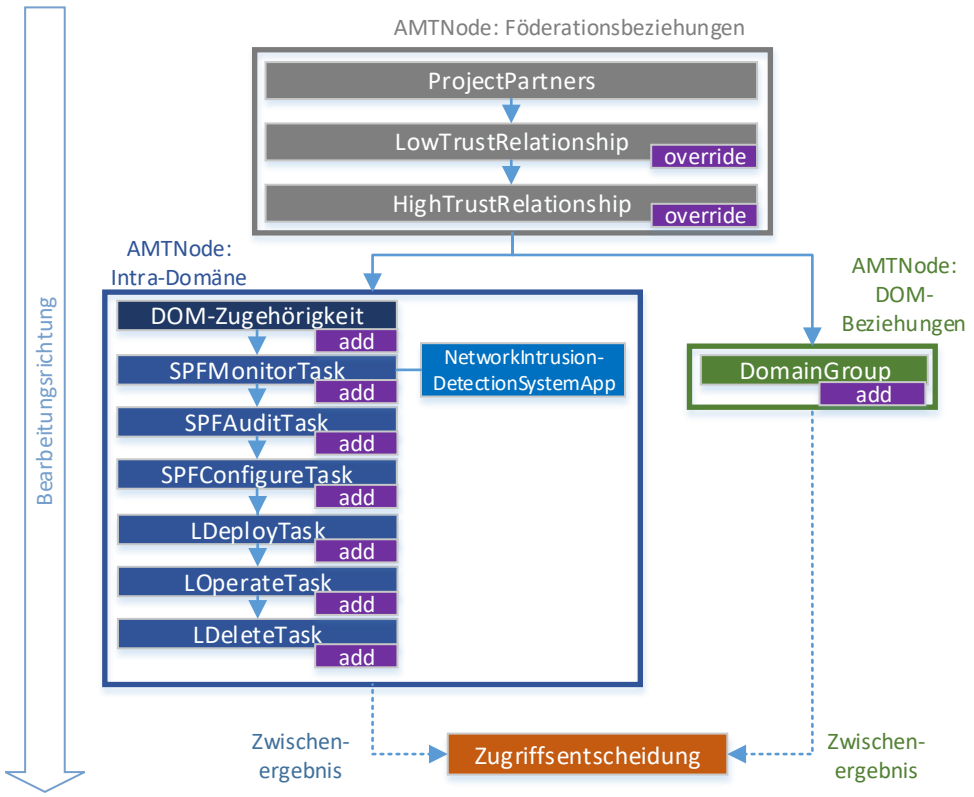


Abbildung 7.8: Umzusetzender Zugriffsentscheidungsbaum für das Evaluationsszenario.

Box) eine Vorauswahl zugreifbarer Informationselemente generiert, welche durch konkrete Domänenzugehörigkeit (blaue Box), Aufgaben oder auch Domänenbeziehungen (grüne Box) erweitert werden.

```
1 // Federation relationship criteria
2 public class FedRelNode extends AMTNodeCases<FedRelElement> { ... }
3 // Domain relationship criteria
4 public class DomRelNode extends AMTNodeLeaf<DomRelElement> { ... }
5 // In-Domain criteria
6 public class InDomainNode extends AMTNodeLeaf<InDomainElement> { ... }
```

Listing 7.25: Ableitung der Knoten im ZVB für Föderationsbeziehungen., Domänenbeziehungen sowie Intra-Domänen-Behandlung.

```
1 public class FedRelElement extends AMTElement<FedRelCriterion> {
2     public FedRelElement(FedRelCriterion criterion, AMTOperation op) {
3         super(criterion, op);
4     }
5 }
6
7 // Restrict criteria classes to deviations from class FederationRelationship
8 public abstract class FedRelCriterion<T extends FederationRelationship>
      implements Criterion<T>
```

Listing 7.26: Ableitung des ZVB-Elements `FedRelElement` für Föderationsbeziehungen sowie des darin zur Parametrisierung enthaltenen Klassenkriteriums `FedRelCriterion`.

Eine Verfeinerung der Struktur wird durch die Implementierung spezifischerer Klassen für Knoten, Elemente und Kriterien, sowie den ZVB selbst erreicht. Die Knoten im ZVB werden

daher von Subklassen von AMTNode abgeleitet, wie in Listing 7.25 gezeigt ist. Die Klasse für Kriterien für Föderationsbeziehungen FedRelNode wird von AMTNodeCases abgeleitet, wodurch die darauffolgende Fallunterscheidung zwischen Kriterien im Kontext von Intra-Domänen und Inter-Domänenbeziehungen im Modell ermöglicht wird. Kriterien bezüglich Domänenbeziehungen (d. h., ein Nutzer fragt Zugriff auf ein Informationselement in einer Domäne an, der er nicht angehört) werden im Knoten DomRelNode beschrieben; Intra-Domänen-Kriterien (d. h. das angefragte Informationselement teilt sich eine Domäne mit dem anfragenden Nutzer) dagegen im Knoten InDomainNode. Beide Kriterien sind von der Klasse AMTNodeLeaf abgeleitet, auf das jeweils kein weiterer Knoten im ZVB folgt. Bei der Ableitung wird zudem die Art der in den Knoten jeweils hinzugefügten ZVB-Elementen spezifiziert, in diesem Fall FedRelElement, DomRelElement und InDomainElement, welche jeweils von der Framework-Klasse AMTElement abgeleitet werden. Beispielhaft wird in Listing 7.26 die Ableitung des Elements für Föderationsbeziehungen FedRelElement gezeigt. Hier von Bedeutung ist insbesondere die Parametrisierung der Klasse AMTElement durch das für Föderationsbeziehungen spezifische Kriterium FedRelCriterion: Wie in demselben Listing gezeigt wird, ist die Art der in Elementen erlaubten Kriterienklassen auf von der Klasse FederationRelationship abgeleitete Klassen festgelegt. Die Modellierung der Elemente für Intra-Domänen-Kriterien und Domänenbeziehungen werden analog umgesetzt. Für Erstere mit Einschränkung auf Aufgaben (Klasse Task) und für letztere mit Einschränkung auf Domänenbeziehungen (Klasse DomainRelationship). Die Besonderheit der Verknüpfbarkeit von Intra-Domänen-Elementen mit MOC-Klassifikationen wird durch eine zusätzlich bei der Klasse InDomainElement eingefügte Klassenvariable

```
Set<Class<? extends FSNMOCK>> associatedFSNMOCKs
```

realisiert, welche FSNMOCK-Klassen enthält. Die Instanziierung der Struktur des gezeigten ZVB wird durch Knoten, Elemente und Kriterien implementiert. Listing 7.27 zeigt die Instanziierung des Kriteriums, des dazugehörigen Elements mit Operation *Override* und des ZVB-Knotens am Beispiel der Föderationsbeziehung für hohes Vertrauen (Klasse HighTrustRelationship). Außerdem zeigt es das Festlegen der Reihenfolge der Elemente innerhalb des Knotens für Vertrauensbeziehungen, wie sie in Abbildung 7.8 gezeigt wurde. Die anderen Knoten werden analog instanziiert.

```
1  // Create AMT criterion
2  FedRelCriterion<HighTrustRelationship> highTrCrit = new
       FedRelCriterion<HighTrustRelationship>() {
3      @Override
4      public Class<HighTrustRelationship> getCriterionClass() {
5          return HighTrustRelationship.class;
6      }
7  };
8  // Create AMT element
9  FedRelElement highTrEl = new FedRelElement(highTrCrit, AMTOperation.OVERRIDE);
10 // Create AMT node
11 FedRelNode fedRelNode = new FedRelNode("Access aspects for federation
       relationships", AMTOperation.NONE, List.of(inDomainNode, domRelNode));
12 // Set order of AMT elements in AMT node for federation relationships:
13 // 1. ProjectPartners
14 fedRelNode.addAMTElement(projPEl, 0);
15 // 2. HighTrustRelationship
16 fedRelNode.addAMTElement(lowTrEl, 1);
17 // 3. LowTrustRelationship
18 fedRelNode.addAMTElement(highTrEl, 2);
19
20 // Instantiate AMT object with root node and rule set
21 baselineAMTree = new BaselineAMTree(fedRelNode, accessRuleSet);
```

Listing 7.27: Erstellung eines Kriterium-Elements für hohes Vertrauen (HighTrustRelationship) und der Operation AMTOperation.OVERRIDE im Knoten für Föderationsbeziehungen, sowie Festlegen der Reihenfolge der ZVB-Elemente und Instanziierung des ZVB-Objekts.

Da die den ZVB beschreibende Klasse `AccessManagementTree` abstrakt implementiert ist und anwendungsspezifisch umgesetzt werden muss, wird für das Anwendungsszenario eine Klasse `BaselineAMTree` abgeleitet. Die Klasse muss je nach Anwendungsfall die beschriebenen Methoden (vgl. Abschnitt 6.4.6) implementieren, was der Traversierungslogik entspricht. Die Instanziierung des `BaselineAMTree` wird ebenfalls in Listing 7.27 gezeigt. Als Parameter erwartet sie den ZVB-Wurzelknoten – in diesem Fall den Knoten für Föderationsbeziehungen – sowie die zu verwendende Instanz des Zugriffsregelsatzes.

7.3.6.2 Anlegen des Zugriffsregelsatzes

Der zweite Schritt dieses Abschnitts befasst sich mit dem Anlegen des **Zugriffsregelsatzes**. Das Anlegen von Regeln erfolgt durch die Erstellung einer neuen `AccessRuleSet`-Instanz und dem Registrieren einzelner Zugriffsregeln daran. Listing 7.28 zeigt das Anlegen eines beispielhaften Zugriffsregelsatzes. Dabei werden für unterschiedliche Kriterienklassen im ZVB – in diesem Fall `SPFMonitorTask`, `HighTrustRelationship`, `LowTrustRelationship` und `DomainGroup` – jeweils Regeln mit verknüpften `InformationTag`-Instanzen angelegt. Der Regel für die Kriterienklasse `DomainGroup` werden hier beispielsweise zwei Tags hinzugefügt – ein kriterienspezifisches („domaingroup“) sowie dasselbe Tag wie für die Klasse `HighTrustRelationship` („highTrust“). Das *false* im Konstruktor der Klasse `AccessRule` legt fest, dass es sich um einen Whitelisting-Ansatz handelt. Die Bedeutung für den Zugriff wird durch das entsprechende `AMTOperation`-Flag für das jeweilige Element bestimmt (vgl. vorheriger Abschnitt).

```
1  // Create rule set
2  AccessRuleSet accessRuleSet = new AccessRuleSet();
3  // Create rules
4  AccessRule monitorRule = new AccessRule(false);
5  monitorRule.addTag(new InformationTag("monitorTask"));
6
7  AccessRule highTrustRule = new AccessRule(false);
8  highTrustRule.addTag(new InformationTag("highTrust"));
9
10 AccessRule lowTrustRule = new AccessRule(false);
11 lowTrustRule.addTag(new InformationTag("lowTrust"));
12
13 AccessRule domainGroupRule = new AccessRule(false);
14 domainGroupRule.addTag(new InformationTag("domaingroup"));
15 domainGroupRule.addTag(new InformationTag("highTrust"));
16
17 // Add rules to access rule set
18 try {
19     accessRuleSet.addAccessRule(SPFMonitorTask.class, monitorRule);
20     accessRuleSet.addAccessRule(HighTrustRelationship.class, highTrustRule);
21     accessRuleSet.addAccessRule(LowTrustRelationship.class, lowTrustRule);
22     accessRuleSet.addAccessRule(DomainGroup.class, domainGroupRule);
23 } catch (AccessCriterionClassAlreadyExisting accessCriterionClassAlreadyExisting)
       {
24     accessCriterionClassAlreadyExisting.printStackTrace();
25 }
```

Listing 7.28: Anlegen des Zugriffsregelsatzes, Definition von Zugriffsregeln und ihre Registrierung am Zugriffsregelsatz.

7.3.6.3 Implementierung der Traversierungslogik

Die Implementierung der **Traversierungslogik** wird zum großen Teil in der vom Elternelement `AccessManagementTree` geerbten Methode

```
Set<InformationTag> getAllowedInformationTags(Entity e, Domain context)
```

implementiert. Diese Methode bestimmt für eine gegebene Nutzerentität `e` und einen Domänenkontext `context` alle registrierten `InformationTag`-Instanzen, für die ein Zugriff erlaubt ist. Sie wird schließlich als Basis zur Implementierung der Methoden zur Bestimmung der Sichtbarkeit spezifischer Informationselemente (z. B. `getAccessibleMOs(...)`) verwendet.

In der Implementierung der Methode `getAllowedInformationTags` wird knotenweise gemäß der in Abbildung 7.8 gezeigten Struktur vorgegangen und daher mit dem zuvor implementierten Knoten für **Föderationsbeziehungen** angefangen und wie folgt vorgegangen:

1. Ermittle die Parteien, denen die Nutzerentität `e` angehört, über die zentrale `Federation`-Instanz.

2. Erstelle eine Auswahl aller Föderationsbeziehungen, die für die Nutzerentität relevant sind, d. h., in die eine Partei, der die Nutzerentität angehört, involviert ist. Die Auswahl wird zum Zweck eines im nächsten Schritt notwendigen effizienten Zugriffs in folgender Abbildung gespeichert:

   ```
   Map<Class<? extends FederationRelationship>,
           Set<FederationRelationship>>
   ```

3. Schließlich wird sequenziell für jedes ZVB-Element überprüft, ob Elemente in `relevantFedRelations` der Klasse des Kriteriums entsprechen und eine Partei der die Nutzerentität angehört, an zutreffender Stelle steht (nicht als referenzierendes, sondern als referenziertes Element – im Konzept als Objekt einer Beziehung bezeichnet).

4. Treffen die Überprüfungen des vorherigen Schrittes zu, wird unter Berücksichtigung der Blacklisting- und Whitelistingregelung die Menge der durch die Kriterien (z. B. `ProjectPartners`) und die Menge der damit im Regelsatz verknüpften `InformationTag`-Instanzen berechnet und unter Berücksichtigung der Operation des ZVB-Elements wie folgt angewendet:

   (a) *ADD*: Hinzufügen aller zuvor berechneten relevanten `InformationTag`-Instanzen zum Endergebnis der Funktion.

   (b) *NONE*: Keine Änderungen am Endergebnis der Funktion.

   (c) *OVERRIDE*: Ersetzen des Endergebnisses mit den zuvor berechneten relevanten `InformationTag`-Instanzen.

   (d) *REDUCE*: Entfernen der zuvor berechneten relevanten `InformationTag`-Instanzen vom Endergebnis der Funktion.

Nachdem alle ZVB-Elemente des Knotens für Föderationsbeziehungen abgearbeitet sind, wird geprüft, ob die Nutzerentität selbst der Kontext-Domäne angehört. Im positiven Fall werden die ZVB-Elemente des **Intra-Domänen-Knoten** durchlaufen. Die Domänenzugehörigkeit wird durch Hinzufügen einer domänenspezifischen `InformationTag`-Instanz realisiert, die in folgendem Codeausschnitt

```
allowedInformationTags.add(new InformationTag(context.getId()));
```

zugewiesen wird. Die Abarbeitung der aufgabenspezifischen ZVB-Elemente (z. B. `SPFMonitorTask`) wird wie im Folgenden beschrieben implementiert:

1. Prüfe, ob die Kriterium-Klasse – in diesem Fall eine Ableitung der Klasse `Task` – mit einer im zentralen Föderationsobjekt registrierten Aufgabe übereinstimmt.

2. Überprüfe mithilfe des in Abschnitt 7.3.4 beschriebenen und implementierten Objekts der Klasse `ResponsibilitiesMap`, ob die Nutzerentität für die durch das ZVB-Element beschriebene Aufgabe in der Kontextdomäne zuständig ist. Im positiven Fall wird wie in Schritt 4 der zuvor beschriebenen Abarbeitung im Föderationsbeziehungen-Knoten vorgegangen.

3. Prüfe, ob eine `FSNMOCK`-Klasse mit einem Intra-Domänen-Element verknüpft ist (siehe Abbildung 7.8, Element `NetworkIntrusionDetectionSystemApp` verknüpft mit Knoten `SPFMonitorTask`). Generiere im positiven Fall ein `FSNMOCK`-spezifisches `InformationTag`-Element für jede verknüpfte `FSNMOCK`-Klasse, hier beispielsweise auf Basis des vollen Klassennamens als String (inkl. Package- und Klassenname).

Da alle ZVB-Elemente im Intra-Domänenknoten die Operation *Add* erfüllen, wird der Ergebnissatz des zuvor betrachteten Knotens für Föderationsbeziehungen in jedem Fall ausschließlich erweitert (mindestens um die zuvor beschriebene domänenspezifische `InformationTag`-Instanz.)

Falls die Nutzerentität nicht Teil der Kontextdomäne ist, wird nicht die Abarbeitung des Intra-Domänen-Knotens, sondern die Abarbeitung des **Knotens für Domänenbeziehungen** fortgeführt. Ähnlich wie bei den zuvor beschriebenen Knoten werden alle Elemente (in diesem Fall eines) durchgegangen. Dabei wird zunächst die Menge der für die Kontextdomäne relevanten Domänenbeziehungen ermittelt. Relevante Domänenbeziehungen enthalten mindestens die Kontextdomäne und eine Domäne, der die Nutzerentität angehört. Für jede relevante Domänenbeziehung, die schließlich zur Kriterium-Klasse des ZVB-Elements (hier die Domänenbeziehung `DomainGroup`) gehört, wird Schritt 4 der Abarbeitung im Föderationsbeziehungen-Knoten durchgeführt.

Die, wie hier beschrieben, durch die Methode `getAllowedInformationTags` ermittelten `InformationTag`-Elemente werden in den anderen Methoden für spezifische Informationselemente genutzt. Deren Implementierung muss insbesondere den Kontext erneut berücksichtigen und die zuvor ermittelten `InformationTag`-Instanzen ausschließlich auf Elemente der angefragten Domäne beziehen.

7.4 Kommunikationsmodell

Die beispielhafte Implementierung des Kommunikationsmodells fokussiert sich besonders auf die Beschreibung von Komponenten für Performanzmessungen der Managementplattformkernprozesse in Kapitel 8. Der wesentliche Teil adressiert entsprechend automatische Prozesse, die in erster Linie von einer hohen Performanz des SBI abhängig sind. Wie auch in der Implementierung der Frameworks in Abschnitt 6 werden Teile des Kommunikationsmodells in der beispielhaften Umsetzung – dazu zählen beispielsweise die Datenbankanbindung und Teile der schnittstellenübergreifenden Komponenten – aufgrund ihrer starken Abhängigkeit von einer szenarienspezifischen Umsetzung nicht explizit berücksichtigt, sondern teilweise im Rahmen der SBI- und NBI-Implementierung angesprochen.

7.4.1 Southbound-Interface

Im Rahmen der Anwendung beispielhaft implementierte Komponenten des SBI sind in Abbildung 7.9 gezeigt. Die Implementierung orientiert sich an zwei unterschiedlichen Anwendungsfällen.

- Im ersten Anwendungsfall wird die Abfrage der Netztopologie auf einem Floodlight-SDN-Controller und einem von diesem gemanagten auf Mininet basierenden Netz ausgewertet. Floodlight liefert eine JSON-formatierte Rückgabe über eine HTTP-Schnittstelle, die beispielhaft für drei in einer Reihe verbundene Switches in Listing 7.29 gezeigt wird.

```
1 [{"src-switch": "00:00:00:00:00:00:00:01", "src-port": 2, "dst-switch":
      "00:00:00:00:00:00:00:03", "dst-port": 3, "type": "internal",
      "direction": "bidirectional", "latency": 0},
2  {"src-switch": "00:00:00:00:00:00:00:01", "src-port": 1, "dst-switch":
       "00:00:00:00:00:00:00:02", "dst-port": 3, "type": "internal",
       "direction": "bidirectional", "latency": 0}]
```

Listing 7.29: Beispielrepräsentation der Netztopologie, die über die Floodlight-API abgefragt wird.

  Durch die so erhaltenen Informationen soll über das Managementbeziehungsframework das Modell der zuletzt erfassten Topologie für diese drei Switche aktualisiert werden. Der Anwendungsfall bezieht sich insofern auf die Controller-zu-Switch-Konstellation aus Abbildung 7.4.

- In einem zweiten Anwendungsfall wird von einer via MiniONE realisierten OpenNebula-Instanz[XIX] über ihre XML-RPC-Schnittstelle eine Liste der von ihr verwalteten Virtualisierungshosts abgefragt. Zur Pull-Abfrage wurde der Befehl „onehost list -x“ des OpenNebula beigefügten gleichnamigen Werkzeugs genutzt. Die Rückgabe ist entsprechend in XML formatiert und dient zur Aktualisierung des Modells der OpenNebula-Instanz in der Managementplattform (die Attribute der `FSNMOCK`-Klasse `OpenNebula`) hinsichtlich verfügbarer IT-Ressourcen.

7.4.1.1 Simulation von Netzadaptern und Priorisierungsregeln für Rohdaten

Netzadapter als erster Schritt im Prozess wurden durch die Implementierung einer `Extended-RawDataPusher`-Komponente ersetzt, welche für die Evaluierung `SBIExtRawData`-Instanzen beliebig parallelisiert in die beiden `ExtRawDataQueue` anhängt. Dabei kann das Verhältnis von priorisierten zu nicht-priorisierten Elementen eingestellt werden.

Zur Überprüfung der Priorisierung von zu verarbeiteten `SBIExtRawData`-Instanzen wurden zwei unterschiedliche Priorisierungsregeln implementiert:

- `SourceDCPrioRule` priorisiert anhand des Quell-Datenzentrums einer `SBIExtRawData`-Instanz und bevorzugt hier beispielhaft solche, die aus dem Datenzentrum mit Namen „DCP1“ empfangen wurden. Der Abgleich erfolgt unter Verwendung der zentralen `Federation`-Instanz.

- `NormLabelPrioRule` priorisiert hingegen auf Basis den einer `SBIExtRawData`-Instanz angehängten `NormalizationLabel`-Instanzen und bevorzugt hier beispielhaft solche, die ein Label `new NormalizationLabel("prioritized")` aufweisen.

7.4.1.2 Implementierung von Parsern und Parsed-Priorisierungsregeln

Für die zwei Testanwendungsfälle für das SBI werden zwei unterschiedliche **Parserimplementierungen** benötigt. Der Parser für JSON-Formate basiert auf der dafür gängigen Java-Bibliothek *Jackson*[XX] und konvertiert die empfangenen Rohdaten stets in ein geparstes `JsonNode`-Element, das dann in einer frameworkspezifischen `SBIDataobject`-Instanz gekapselt

[XIX] `https://github.com/OpenNebula/minione`
[XX] `https://github.com/FasterXML/jackson`

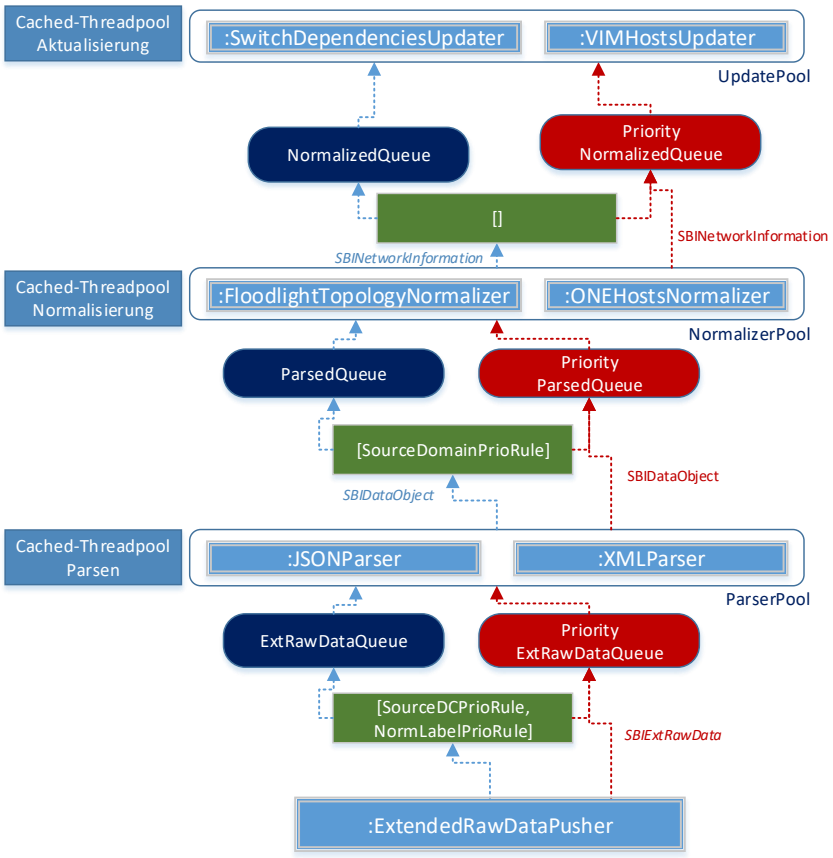


Abbildung 7.9: Beispielhaft implementierte Elemente im SBI-Verarbeitungsprozess.

wird. Die Kernmethode des JSON-Parsers ist in Listing 7.30 gezeigt. Für die im Parser ebenfalls vorgesehene Ermittlung der Quell-MO-Instanz auf Basis des im `SBIExtRawData`-Elements gegebenen Quell-Datenzentrums und der Quell-IP-Adresse wurde für die Evaluierung ein einfaches Mapping `Map<DataCenter, Map<InetAddress, FSNMOC>> dataCenterIpToMOMap` von beiden letzteren Informationen auf eine MO-Instanz in Form einer Klasse `DataCenterIPToMoMapping` implementiert. Der Parser für XML-Formate wurde analog mithilfe der *Java API for XML Processing* (JAXP) implementiert und generiert ein `Document`-Objekt.

Durch das ermittelte Quell-MO können schließlich neue Formen von Priorisierungsregeln angewendet werden. Hier wurde beispielhaft eine Klasse `SourceDomainPrioRule` implementiert. Einer Instanz dieser Klasse kann im Konstruktor jeweils eine beliebige Domäneninstanz übergeben werden. Die Priorisierungsregel prüft schließlich, ob das Quell-MO des Elements `SBIDataobject` in dieser übergebenen Domäne liegt und priorisiert im positiven Fall die weitere Verarbeitung. Die Regel `SourceDomainPrioRule` kann entsprechend vielfach mit unterschiedlichen Domänen instanziiert werden.

7.4.1.3 Implementierung von Normalisierern und Normalized-Priorisierungsregeln

Die hier anwendungsfallbezogen implementierten **Normalisierer** generieren aus der jeweiligen `SBIDataobject`-Instanz `NetworkInformation`-Elemente, die als normalisierte Form der ursprünglichen Daten angesehen werden. Die von Floodlight an dieser Stelle in einem `JsonNode`-Element ausgedrückten Topologiedaten werden entsprechend in das zuvor in Abschnitt 7.2.4 beschriebene Element `NetworkTopology` überführt. Die Daten zur Beschreibung von OpenNebula-Hosts werden pro Host in der Netzzustandsklasse `VIMHostState` beschrie-

ben, die Informationen über jeweils verfügbare und genutzte CPUs, Primär- und Sekundärspeicherressourcen, den Hostnamen sowie der Anzahl auf einem Host jeweils betriebener VMs beschreibt. Mehrere `VIMHostState`-Instanzen werden in einer `VIMHosts`-Instanz gekapselt. Die Zuweisung von Normalisierern zu einem entsprechenden `SBIDataobject`-Element erfolgt jeweils über Normalisierungslabel, den Typ des Quell-MOs (in Form einer `FSNMOCK`-Klasse) und anhand des resultierenden Objekttyps des Parserschritts, wie in Listing 7.31 am Beispiel des Netztopologie-Anwendungsfalls gezeigt wird.

```
1  public SBIDataobject<JsonNode> parse(SBIExtRawData rawData) throws
       ParsingException {
2
3      DataCenterIPToMoMapping moMap = DataCenterIPToMoMapping.getInstance();
4      //...
5      String content = rawData.getRawData();
6      try {
7          JsonNode mapped = objectMapper.readTree(content);
8          SBIDataobject<JsonNode> ret = new SBIDataobject<>(mapped,
9                  rawData.getReceivingTimeStamp(),
10                 moMap.getMo(rawData.getSourceDataCenter(),
11                             rawData.getSourceAddress()),
12                 rawData.getNormalizationLabels());
13         return ret;
14     } catch (JsonProcessingException e) {
15         throw new ParsingException("ParsingError: " + e.getMessage());
16     } catch (MODoesNotExistException e) {
17         throw new ParsingException("Could not find out the source MO of the
               content");
18     }
19 }
```

Listing 7.30: Implementierung der Methode `parse` der Klasse `JSONParser`.

```
1  // Floodlight topology
2  NormalizationLabel<String> floodTopoLabel = new NormalizationLabel<>("floodTopo");
3  sbiRegister.getNormalizerPool().addNormalizerByLabel(floodTopoLabel,
       flTopoNormalizer);
4  sbiRegister.getNormalizerPool().addNormalizerByFSNMOCK(Floodlight.class,
       flTopoNormalizer);
5  sbiRegister.getNormalizerPool().addNormalizerByObjClass(ArrayNode.class,
       flTopoNormalizer);
```

Listing 7.31: Beispielhafte Zuweisung von Normalisierungskomponenten anhand von Normalisierungslabels, der Quell-MO-Klassifizierung und nach geparstem Objekt.

7.4.1.4 Implementierung eines Pools zur Modellaktualisierung

Der letzte Schritt des SBI-Prozesses zur Verarbeitung von Daten der gemanagten Infrastruktur ist nach dem Parsen und Normalisieren die **Aktualisierung** des Datenmodells auf Basis der normalisierten Netzinformationen. Da die Datenbankanbindung in dieser Implementierung nicht im Speziellen berücksichtigt wird, wird die Aktualisierung des Datenmodells nicht wie in Abschnitt 5.6.3.2 aufgezeigt implementiert, sondern direkt. Hier wurde daher, wie in Abbildung 7.9 gezeigt ist, ein weiterer Komponentenpool *UpdatePool* eingeführt, welcher ebenfalls durch das SBIRegister verwaltet wird. Ähnlich wie für Parser oder Normalisierer implementiert jede Update-Komponente `UpdateComponent` eine Methode

```
void update(NetworkInformation netinfo, SBINetworkInformation
        context)throws UpdatingException.
```

Die Methode `update` hat keinen Rückgabewert, sondern greift direkt auf das Datenmodell der Managementplattform zu. Um Race-Conditions durch parallelen Zugriff an dieser Stelle zu unterbinden, wird diese Methode in dieser Evaluation von jeder `UpdateComponent` als

synchronized – für exklusiven Zugriff darauf – markiert. Die Update-Komponentenimplementierung SwitchDependenciesUpdater verarbeitet die zuvor beschriebenen NetworkTopology-Netzzustände und generiert auf dieser Basis ein Modell der Netztopologie der dadurch beschriebenen Teilkomponenten. Dazu werden die alten Managementbeziehungen vom Typ DHNIpNetwork (vgl. Abschnitt 7.2.1) in der zentralen DependencyManager-Instanz (vgl. Abschnitt 6.3.2) entfernt und neue DHNIpNetwork-Instanzen gemäß dem NetworkTopology-Element angelegt. Die Update-Komponentenimplementierung VMHostsUpdater aktualisiert die verfügbaren IT-Ressourcen-Daten der im Szenario beschriebenen OpenNebula-Instanz. Darüber hinaus wird ein Alert Fatal (eine Ableitung der Alarm-Klasse Failure, vgl. Abbildung 7.7) generiert, falls der verfügbare Speicher der OpenNebula-Instanz unter einen Schwellwert von 80% fällt. Dieser generierte Alert dient in Abschnitt 8.2.2 zur Messung der Performanz des Benachrichtigungsprozesses und der Closed-Loop-Automatisierung.

7.4.2 Northbound-Interface

In dieser beispielhaften Modellierung wird die Verwendung der Kernkomponenten des NBI gezeigt:

- Pull-Komponenten zur Nutzung von Plattform- und MOC-Funktionen über das NBI und
- Push-Komponenten zur Benachrichtigung von Managementapplikationen über das NBI.

Die Verwendung weiterer im NBI-Framework beschriebener Komponenten werden hier nicht explizit gezeigt. Teile davon – beispielsweise ein Managementplattform-Verbund – sind zudem bereits in Lua in [181] implementiert.

7.4.2.1 Implementierung eines remoten Pull-Zugriffs

Als zunächst beispielhaft am NBI nutzbare MOC-Funktion wird die in Abschnitt 7.2.3 beschriebene Funktion *addSrcBlockingFlow* eines OpenFlow-Controllers beschrieben. Der Zugriff wird als **remoter Pull-Zugriff** über eine neue Klasse

```
public class ODLBlockMAC extends NBIRemotePullComponent
```

implementiert, da dieser einen lokalen Zugriff einschließt. Die entsprechende MOC-Funktionsinstanz sowie die MOCKFunctionMap-Instanz wird bereits im Konstruktor von ODLBlockMAC übergeben. Die zu implementierende Kernmethode der Klasse ist executeLocal, durch welche die eigentliche Ausführung der MOC-Funktion umgesetzt wird. Darin werden zunächst übergebene Parameter überprüft, insbesondere auch die MO-Instanz, auf der die Funktion ausgeführt wird. Falls alle notwendigen Parameter vorhanden sind, wird für die MO-Instanz ein funktionierender Treiber in der MOCKFunctionMap-Instanz gesucht, wobei in dieser Implementierung ausschließlich der erste funktionierende Treiber ausgeführt wird. Durch ungünstige Klassifikation eines MO, z. B. als OpenDaylight- und als Floodlight-Controller, könnten hier unterschiedliche Implementierungen konkurrieren. In [181] wurde dazu eine Lösung vorgeschlagen und implementiert. Falls eine gefundene Treiberimplementierung für das betrachtete MO gefunden wird, wird diese ausgeführt und das Ergebnis zurückgemeldet. Unterschiedliche Verarbeitungsgrade wurden hier nicht implementiert, da diese keinen Einfluss auf die Wirksamkeit des Konzepts haben, jedoch für Managementanwendungen nützlich sind. Auch wurde in dieser Anwendung eine remote Ausführung bzw. Funktionsweiterleitung nicht berücksichtigt. Das Anlegen der Pull-Komponente erfolgt schließlich über die Instanziierung eines ODL-BlockMAC-Objekts und seine Registrierung am NBI-Register unter einem bestimmten Aufrufkontext, wie in Listing 7.32 gezeigt wird. In diesem Fall würde die Pull-Komponente auf einem HTTP-Kontext „/addSrcBlockingFlow“ registriert und bei Anfrage über das NBI-Register entsprechend zugewiesen.

```
// Create Pull component instance
ODLBlockMAC odlBlockPull = new ODLBlockMAC(
    SampleID.gen(ODLBlockMAC.class),
    "NBI Pull component to execute a block source MAC from an ODL SDN
        controller", // Description
    PrivilegeComponent.NORMAL, // Normal privilege level
    mocf, // Exemplary MOC-Function
    fwdconn, // Dummy forward connector
    mockFunctionMap); // Instance of MOCFunctionMap

// Register by context
HttpPullContext sampleContext = new HttpPullContext("/addSrcBlockingFlow");
nbiRegister.addNBIPullComponent(odlBlockPull);
nbiRegister.setPullContext(sampleContext, odlBlockPull);
```

Listing 7.32: Instanziierung einer remoten Pull-Komponente und ihre Registrierung nach Aufrufkontext am NBI-Register.

7.4.2.2 Implementierung eines Dienstes für remote Benachrichtigungen

Analog wird eine **remote Push-Komponente** implementiert; sie stellt ebenfalls eine Erweiterung ihres lokalen Pendants dar. Sie ist eine wesentliche Komponente für Benachrichtigungen und eine Closed-Loop-Automatisierung, die in Kapitel 8 hinsichtlich ihrer Performanz untersucht werden. In dieser beispielhaften Anwendung wird die Benachrichtigung eines Sicherheitsalarms der Klassifizierung `Fatal` als Push-Benachrichtigung implementiert, deren Auftreten proaktiv an Managementanwendungen gemeldet wird. Die Umsetzung erfolgt durch Ableitung einer entsprechenden neuen, für diesen Zweck vorgesehene Klasse `SecFatalPushComponent` von der Klasse `NBIRemotePushComponent`, die in diesem Fall primär die Serialisierungsmethode `serializeNotification` für die Benachrichtigungselemente wie hier die Klasse `Fatal` implementiert. In diesem Fall wird aus Instanzen dieser Klasse eine JSON-Darstellung generiert, die in Listing 7.33 gezeigt ist.

Danach wird das Pushkriterium durch Ableitung einer Klasse `FatalPushCriterion` von der Klasse `NBIPushCriterion` definiert und mit der Benachrichtigungselementenklasse `Fatal` parametrisiert, wie in Listing 7.34 gezeigt wird. Die Implementierung der Methode `isRelevant` dient zur Filterung *interessanter* Benachrichtigungselemente und drückt in der gezeigten Form aus, dass nicht alle Instanzen des Typs `Fatal` als Push-Benachrichtigungen berücksichtigt werden sollen, sondern ausschließlich solche, die mit dem `AlertCategory`-Element `MALWARE` klassifiziert sind.

```
{ "ID": "Fatal:3155866752613",
  "AlertType": "Sec/Fatal",
  "Description": "Example",
  "AlertCategories": [
    "MALWARE"
  ],
  "Timestamp": "Thu Nov 11 08:30:46 UTC 2021"}
```

Listing 7.33: Beispielhaftes Resultat der in der Klasse `SecFatalPushComponent` implementierten serialisierten Form einer `Fatal`-Instanz.

```
public class FatalPushCriterion extends NBIPushCriterion<Fatal> {
  @Override
  public boolean isRelevant(Fatal element) {
      // Push every fatal element with alert category fatal
      return element.getAlertCategory().contains(AlertCategory.MALWARE);
  }
}
```

Listing 7.34: Implementierung eines Pushkriteriums zum Filtern relevanter Benachrichtigungen des Typs `Fatal`.

Da es sich in diesem Fall um eine remote Push-Benachrichtigung handelt, muss zudem ein geeigneter Konnektor implementiert werden. Zu diesem Zweck wurde auf Basis der Java-nativen Klasse `HttpClient` eine Klasse `HttpConnector` von der Framework-Klasse `Connector` abgeleitet und eine einfache Senden-Methode implementiert, die das serialisierte Benachrichtigungselement an eine URL mittels HTTP-POST-Methode asynchron verschickt.

```
1  public class HttpPushEndpoint extends NBIRemotePushEndpoint {
2    private InetAddress address;
3    private int port;
4    private String context;
5
6    public HttpPushEndpoint(Credentials managementPlatformCredentials, Class
         typeOfConnector, InetAddress address, int port, String context) {
7        super(managementPlatformCredentials, typeOfConnector);
8        this.address = address;
9        this.port = port;
10       this.context = context;
11   }
12   @Override
13   public void send(String element, Connector connector) {
14       if (element != null && connector != null && connector instanceof
             HttpConnector) {
15           HttpConnector httpc = (HttpConnector) connector;
16           httpc.send(address, port, context, element);
17       }
18   }
19 }
```

Listing 7.35: Implementierung einer Klasse zur Beschreibung von HTTP-basierten Empfängerendpunkten für Push-Benachrichtigungen.

```
1  // Push component
2  SecFatalPushComponent sfpPusher = new
       SecFatalPushComponent(SampleID.gen(SecFatalPushComponent.class), "Push
       component for fatal alerts", PrivilegeComponent.NORMAL, Set.of(new
       FatalPushCriterion()), PushPriority.NORMAL);
3  // Sample endpoint 1
4  HttpPushEndpoint sampleEndpoint = new HttpPushEndpoint(null, HttpConnector.class,
       InetAddress.getByName("127.0.0.1"), 8080, "/");
5
6  // Register HTTP connector
7  nbiRegister.addNBIAdapter(new HttpConnector());
8
9  // Register push component
10 nbiRegister.addNBIPushComponent(sfpPusher);
11
12 // Register endpoint with priority
13 nbiRegister.addNotificationDecision(PushPriority.NORMAL, Fatal.class, sfpPusher,
       sampleEndpoint);
14
15 // Sample alert
16 Fatal sampleAlert = new Fatal(Set.of(AlertCategory.MALWARE), "Example", new
       TimeStamp(System.currentTimeMillis()));
17
18 // Exemplary push; usually triggered by SBI on item generation
19 nbiRegister.pushNotification(sampleAlert);
```

Listing 7.36: Registrierung notwendiger Komponenten für eine beispielhafte Push-Benachrichtigung und ihre Ausführung.

Als letztes zu implementierendes Element zum Abschicken einer Push-Benachrichtigung müssen remote Endpunkte als eigentlich Empfänger implementiert werden. Da der Versand über HTTP umgesetzt wird, wird ein dafür spezifischer HTTP-Endpunkt `HttpPushEndpoint` als Ableitung der Klasse `NBIRemotePushEndpoint` implementiert (vgl. Listing 7.35). Die darin zu implementierende Methode `send` definiert, wie die Benachrichtigung an den Endpunkt

dieses Typs geschickt wird. Die Methode wird letztlich durch das NBI-Register mit der passenden Konnektorinstanz aufgerufen, falls ein relevantes Element gesendet wird. Die Klasse `HttpPushEndpoint` ist so implementiert, dass sie für beliebige HTTP-basierte Endpunkte genutzt und mit den verschiedenen Bausteinen einer URL wie der Hostadresse, dem Dienstport und dem HTTP-Kontext instanziiert werden kann.

Nachdem alle relevanten Elemente implementiert sind, müssen diese am NBI-Register, wie in Listing 7.36 gezeigt ist, registriert werden. Dazu werden zunächst die Push-Komponente mit daran gebundener Menge an Push-Kriterien sowie ein beispielhafter Push-Endpunkt instanziiert. Der erste Parameter des Endpunkts beschreibt Zugangsdaten zur Authentisierung am Push-Endpunkt und wurde in diesem Beispiel vernachlässigt, d. h. mit `null` belegt.

Als Nächstes werden die Instanzen am NBI-Register registriert, zunächst eine `HttpConnector`-Instanz und schließlich die zuvor bereits instanziierte Push-Komponente. Der Endpunkt wird im Kontext einer Push-Entscheidung über die Methode `addNotificationDecision` gemäß der im Konzept beschriebenen Struktur aus Abbildung 5.37 registriert. Diese wird dann bei der Push-Priorisierung und -Entscheidung dementsprechend durchlaufen.

7.5 Funktionsmodell

In diesem Abschnitt wird eine beispielhafte Anwendung zentraler Komponenten des Funktionsmodells gezeigt. Die Anwendung bezieht sich insbesondere auf das Anlegen von funktionalen Managementbereichen und eine beispielhafte Umsetzung einer Plattformfunktion. Die Umsetzung automatischer Überwachung und Steuerung mit Fokus auf Quasi-Echtzeit-Performanzkriterien wird dediziert in Kapitel 8 evaluiert.

7.5.1 Funktionale Managementbereiche

Funktionale Managementbereiche unterteilen das Netzmanagement von FSNs in notwendige Managementfunktionen. Sie dienen schließlich als Basis zur feingranulareren Unterteilung durch Aufgabenbereiche aus dem Organisationsmodell.

7.5.1.1 Definition funktionaler Managementbereiche

Wie in Abschnitt 7.3.1 beschrieben wurde, wird sich in dem beschriebenen Szenario insbesondere auf das Security-, Performance-, Fault- sowie VM-Lifecyclemanagement fokussiert. Die Definition der betrachteten funktionalen Managementbereiche wird in Listing 7.37 gezeigt wird. Jeder funktionale Managementbereich zeichnet sich durch eine eindeutige ID sowie eine kurze Beschreibung aus.

```
1  FunctionDomain securityManagement = new FunctionDomain(
2          SampleID.gen(FunctionDomain.class),
3          "Functional Area for Security Management ");
4  FunctionDomain faultManagement = new FunctionDomain(
5          SampleID.gen(FunctionDomain.class),
6          "Functional Area for Fault Management ");
7  FunctionDomain perfManagement = new FunctionDomain(
8          SampleID.gen(FunctionDomain.class),
9          "Functional Area for Performance Management ");
10 FunctionDomain lifecycleManagement = new FunctionDomain(
11         SampleID.gen(FunctionDomain.class),
12         "Functional Area for VM-Lifecycle Management ");
```

Listing 7.37: Anlegen funktionaler Managementbereiche für Security-, Performance-, Fault- und Lifecyclemanagement.

Nach Anlegen der Managementbereiche ist ihre Registrierung am zentralen Federation-Objekt notwendig. Sie kann über die vom Föderationsobjekt bereitgestellte Methode addFunctionArea umgesetzt werden, die Listing 7.38 zeigt. Jeder funktionale Managementbereich wird einem eindeutigen String-Schlüssel zugewiesen, der verhindern soll, dass funktionale Managementbereiche nicht redundant registriert werden und jeder Bereich einzigartig ist. Um eine Registrierung derselben FunctionDomain-Instanz für unterschiedliche Schlüssel zu unterbinden, ist beispielsweise die ID der FunctionDomain-Instanz als solcher zu verwenden.

```
1 federation.addFunctionDomain("SecurityManagement", securityManagement);
2 federation.addFunctionDomain("FaultManagement", faultManagement);
3 federation.addFunctionDomain("PerformanceManagement", perfManagement);
4 federation.addFunctionDomain("LifecycleManagement", lifecycleManagement);
```

Listing 7.38: Registrierung der funktionalen Managementbereiche am Federation-Objekt.

7.5.1.2 Untergliederung in Aufgabenbereiche

Für eine feingranularere Organisation müssen die Funktionsbereiche in einen oder mehrere Aufgabenbereiche unterteilt werden. Die Aufgabenbereiche, wie sie in Abschnitt 7.3.1 festgelegt wurden, sind ebenfalls zentral in der Federation-Instanz hinterlegt und können an dieser abgefragt werden.

```
1 TaskDomain secperffault = federation.getTaskDomains().get("SecPerfFaultMan");
2 TaskDomain lifecycle = federation.getTaskDomains().get("LifecycleMan");
3
4 securityManagement.addTaskDomain(secperffault);
5 faultManagement.addTaskDomain(secperffault);
6 perfManagement.addTaskDomain(secperffault);
7 lifecycleManagement.addTaskDomain(lifecycle);
```

Listing 7.39: Abfragen von Aufgabenbereichen und Zuweisung zu funktionalen Managementbereichen.

Listing 7.39 zeigt die Abfrage der Aufgabenbereiche und ihre Zuweisung zu den funktionalen Managementbereichen. Demnach bündelt der Aufgabenbereich *SecPerfFaultMan* alle Aufgaben für das Security-, Performance- und Faultmanagement. Das VM-Lifecyclemanagement wird hingegen durch einen eigenen Aufgabenbereich *LifecycleMan* erfüllt.

7.5.2 Framework zur Erweiterung der Managementplattformfunktionalität

Plattformfunktionen werden für die Beschreibung und Umsetzung von Funktionalität genutzt, die nicht direkt mit einem MO in Verbindung stehen. Beispielsweise werden Funktionen der Organisation des Managements wie das Anlegen einer administrativen Domäne, eines neuen Nutzers oder die Vergabe von Verantwortlichkeiten über sie realisiert.

7.5.2.1 Implementierung der Funktionsschnittstelle

Die Funktionsschnittstelle, über die andere Komponenten später auf die eigentliche Plattformfunktion zugreifen können, wird über eine Instanz der Klasse PlatformFunction implementiert. Darin werden die Parameter, Rückgabewerte, eine Kurzbeschreibung sowie zur Identifikation eine eindeutige ID und ein Funktionsname angegeben. Da die Implementierung nicht komponentenspezifisch ist, wird (im Gegensatz zu MOC-Funktionen) ebenfalls der Funktionstreiber direkt im Konstruktor übergeben. In dieser beispielhaften Implementierung wird eine Funktion realisiert, durch die ein neuer **Nutzer erstellt** und einer als Parameter anzugebender **Partei zugewiesen** wird. Listing 7.40 zeigt die Implementierung.

```
PlatformFunction platformFunction = new PlatformFunction(
      SampleID.gen(PlatformFunction.class),
      "addUser", // Function Name
      "A platform function for adding a new user", // Description
      // Parameters
      Map.ofEntries(new AbstractMap.SimpleEntry<>("party", new
          ShortStringType())),
      // Return values
      Map.ofEntries(new AbstractMap.SimpleEntry<>("result", new
          ShortStringType())),
      pfDriver);  // Platform Driver Implementation
```

Listing 7.40: Beispielhafte Implementierung einer `PlatformFunction`-Instanz.

In der Beispielimplementierung wird die Plattformfunktion *addUser* genannt und erwartet einen einzigen Parameter zur Identifikation der Partei, der der neu erstellte Nutzer zugewiesen wird. Als Rückgabe liefert die Plattformfunktion ebenfalls nur einen Wert: Im Fehlerfall wird die Fehlermeldung zurückgegeben, andernfalls die ID des erstellten Nutzers.

```
PlatformFunctionDriver pfDriver = new PlatformFunctionDriver() {
    @Override
    public Map<String, FunctionValue> execute(Map<String, FunctionValue>
        parameter) throws ExecutionError {
        try {

        String partyDescr = parameter.get("party").getValue().toString();
        if (partyDescr == null)
            throw new ExecutionError("error: Party identifier is missing!");
        Optional<Party> p = Federation.getInstance().getParties().stream().filter(
                    party -> party.getDescr().equals(partyDescr)).findFirst();

            if (p.isEmpty())
                return Map.ofEntries(new AbstractMap.SimpleEntry<>("result", new
                    FunctionValue("error: Party not found", new
                    ShortStringType())));

            User u = new User(SampleID.gen(User.class));
            p.get().addUser(u);
            return Map.ofEntries(new AbstractMap.SimpleEntry<>("result", new
                FunctionValue(u.getId().toString(), new ShortStringType())));
        }

        catch (NonConformValueError e) {
            throw new ExecutionError(e.getMessage());
        } catch (EntityAlreadyExistsException e) {
            throw new ExecutionError(e.getMessage());
        } catch (FederationIsNotInitialized e) {
            throw new ExecutionError(e.getMessage());
        }
    }
};
```

Listing 7.41: Beispielhafte Implementierung der Treiberfunktion `PlatformFunctionDriver`.

7.5.2.2 Implementierung der Funktionalität

Die eigentliche Funktionalität der Managementplattform wird über einen Plattformfunktionstreiber bzw. die Klasse `PlatformFunctionDriver` implementiert, wie es beispielhaft in Listing 7.41 gezeigt ist. Die zu implementierende Kernmethode ist `execute`, die auch die Ausführung der Plattformfunktion darstellt. Zunächst wird der benannte Parameter *party* geholt und überprüft, ob er belegt ist. Im Fehlerfall wird eine entsprechende Exception mit Hinweis auf Fehlen des Parameters generiert. Bei gegebenem Partei-Identifikator wird die entsprechende Partei von der zentralen `Federation`-Instanz erfragt. Wird ein entsprechendes `Party`-Objekt

gefunden, wird ein neuer Nutzer angelegt, der Partei hinzugefügt und die ID des Nutzers zurückgegeben.

7.6 Zusammenfassung des Kapitels

In diesem Kapitel wurden die in dieser Arbeit konzipierten Frameworks zur Entwicklung von Managementplattformen in FSNs anhand eines beispielhaften Szenarios angewendet. Die Anwendung fokussiert sich dabei vor allem auf die Demonstration der **Modellierbarkeit** des gemanagten Netzes durch das Informationsmodell sowie darin typischer organisatorischer und funktionaler Aspekte. Im Rahmen der Anwendung der Frameworks des Kommunikationsmodells konnte insbesondere gezeigt werden, dass Komponenten des Benachrichtigungs- und Steuerungsprozesses geeignet für FSNs modellierbar und implementierbar sind. Dabei wurde die Trennung zwischen Komponenten der Frameworks und vom Anwender zu implementierenden Teilen deutlich. Im folgenden Kapitel werden unter anderem, aufbauend auf den in diesem Kapitel erstellten Modellen, Performanzkriterien an die Implementierung evaluiert und bewertet.

Kapitel 8

Evaluierung der Frameworks

Inhalt

8.1 Evaluierungssystem 279
8.2 Performanz von Kernprozessen 280
8.2.1 Sichtbarkeits- und Zugriffsentscheidungen 280
8.2.2 Benachrichtigungsprozess 281
8.2.3 Closed-Loop-Automatisierung 293
8.3 Abgleich der Anforderungen 297
8.3.1 Anforderungen an das Funktionsmodell 298
8.3.2 Anforderungen an das Informationsmodell 300
8.3.3 Anforderungen an das Kommunikationsmodell 300
8.3.4 Anforderungen an das Organisationsmodell 301
8.3.5 Anforderungen an die Umsetzung 302
8.3.6 Bewertung der Erfüllung der Anforderungen 302

In diesem Kapitel werden die Konzepte und ihre Implementierung sowie Anwendung aus Kapitel 7 zunächst hinsichtlich performanzkritischer Kernprozesse des Netzmanagements evaluiert. Auf Basis der Auswertungen werden die zuvor aufgestellten Anforderungen der Performanz in Abschnitt 8.3 bewertet.

8.1 Evaluierungssystem

Das **Testsystem** ist Ubuntu 20.04.2 mit Linux-Kernel 5.11.0. Die Tests wurden, wie die prototypische Implementierung und beispielhafte Umsetzung auch, im Rahmen des ONOS-Workspace entwickelt. Die Ausführung der Tests wird in einer JUnit-Testumgebung durchgeführt und durch das Werkzeug Bazel in Version 3.7.2 gesteuert. Wie in Abschnitt 6.2.1 beschrieben ist, wurden auch die Tests aus Kompatibilitätsgründen zu ONOS mit dem JDK 11 kompiliert.

Als **Hardware-Plattform** für die Testreihen werden bewusst weder Komponenten im Niedernoch im Hochperformanzbereich, sondern herkömmliche Verbraucherkomponenten genutzt, um die Arten und Größe von Einsatzszenarien nicht einzuschränken. Als CPU wird ein AMD Ryzen 7 PRO 4750U mit acht CPU-Kernen und insgesamt 16 Threads genutzt. Der DDR4-Arbeitsspeicher hat eine Größe von 32 Gigabyte und eine Transfer-Rate von 3200 MT/s.

8.2 Performanz von Kernprozessen

Performanzkritische Kernprozesse einer Managementplattform sind in dieser Arbeit vor allem die Überwachung der gemanagten Infrastruktur mit Unterstützung des Benachrichtigungsprozesses sowie ihre aktive und automatisierte Steuerung. Gemeinsam bilden sie einen *Closed-Loop-Zyklus*. Die Datenvisualisierung wurde mit Gnuplot 5.2 erstellt. Die Boxplot-Diagramme wurden gemäß der Standardkonfiguration von Gnuplot erstellt, bei der die *Whiskers* Datenpunkte im 1,5-fachen Interquartilbereich umfassen [209]. Werte, die darüber hinausgehen, sind als Ausreißer markiert.

8.2.1 Sichtbarkeits- und Zugriffsentscheidungen

Sichtbarkeits- und Zugriffsentscheidungen sind keine eigenständigen Kernprozesse im Netzmanagement, jedoch ein wesentlicher koordinierender Bestandteil davon. Sie werden in dieser Arbeit durch den Zugriffsverwaltungsbaum (vgl. Abschnitt 5.5.5.2) gesteuert, welcher jedoch in den Frameworks nur durch eine generelle Struktur vorgegeben ist. Die Implementierung und Performanz sind dagegen stark szenarien- bzw. umsetzungsabhängig, wodurch Zeitmessungen nicht szenarienübergreifend übertragbar sind. Darüber hinaus sind die Struktur des geschaffenen Zugriffsverwaltungsbaums, die Anzahl der Knoten und darin enthaltenen Elemente Größen, die sich auf die Verarbeitungsdauer auswirken.

In dieser Messreihe wird daher die beispielhafte Anwendung aus Abschnitt 7.3.6 als Einzelfall betrachtet. Die zentrale und in dieser Auswertung evaluierte Funktion ist dabei

```
public Set<InformationTag> getAllowedInformationTags(Entity e, Domain
                                context)
```

einer Instanz der Klasse `BaselineAMTree`, welche die ZVB-Implementierung des Evaluationsszenarios darstellt. Die Methode ermittelt für eine Entität in einem Domänenkontext alle sichtbaren und zugreifbaren `InformationTag`-Instanzen und ist daher das Ergebnis der Auswertung einer Zugriffsentscheidung durch den ZVB. Die Zugreifbarkeit auf ein konkretes Informationselement wie eine MOC-Funktion kann durch Abgleich des Vorhandenseins eines erlaubten Tags einfach und, beispielsweise durch Organisation in einer Hashtabelle, sehr effizient bestimmt werden und wird daher in dieser Messung nicht näher berücksichtigt.

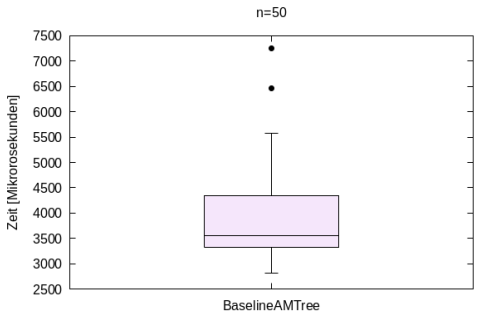


Abbildung 8.1: Dauer zur Ermittlung zugreifbarer `InformationTag`-Instanzen für einen Nutzer über verschiedene Domänen des Evaluierungsszenarios.

Im Testlauf wird in jeder Stichprobe für einen Nutzer aus dem Evaluierungsszenario die Menge zugreifbarer `InformationTag`-Instanzen für eine zufällige Domäne aus der Domänenkreuzorganisationstabelle ermittelt. Auf diese Weise sind Fälle abgedeckt, in denen der Nutzer zu einer Domäne gehört und dann seine Aufgaben einfließen, bzw. der Nutzer nicht Teil von Domänen ist

und folglich Domänenbeziehungen einfließen. Bei 50 zufälligen Stichproben ergibt sich ein vornehmlicher Laufzeitbereich von ca. 2,8 bis ca. 5,5 Millisekunden, wie in Abbildung 8.1 gezeigt wird. Darüber hinaus sind zwei Ausreißer bei ca. 6,5 und 7,3 Millisekunden verzeichnet. Im Mittel dauert eine Durchführung der Funktion so 3,9 Millisekunden für die `BaselineAMTree`-Implementierung eines Zugriffsverwaltungsbaums.

Zu beachten ist, dass die ermittelte Verarbeitungsdauer nicht zwangsläufig auf jeden Nutzerzugriff addiert werden muss und den Zugriff je nach Implementierung nicht zwangsläufig verzögert. Dies ist nur in einem naiven Ansatz der Fall, in dem bei jedem Nutzerzugriff die Menge von für den Nutzer und Domänenkontext die Menge zugreifbarer `InformationTag`-Instanzen über die Methode ermittelt wird. Genauso ist es denkbar, dass für jede Nutzer-Domänen-Konstellation die Mengen erlaubter `InformationTag`-Instanzen vorberechnet werden – beispielsweise periodisch in einem eigenen Thread oder besser aber bei Änderungen von Kriterien (vgl. Abschnitt 5.5.5.2). Zugriffsentscheidungen beeinflussen dann Nutzeranfragen nicht als zusätzlichen zeitlichen Overhead.

8.2.2 Benachrichtigungsprozess

Der erste hier betrachtete Kernprozess des Managements ist der Benachrichtigungsprozess, um unmittelbar und automatisiert über Fehler oder ungewünschte Zustände im Netz zu informieren. Er umfasst alle Schritte von der Erfassung des Netzzustands über das SBI hin zur Benachrichtigung relevanter Managementanwendungen über das NBI der Managementplattform. Dieser Prozess wird in diesem Abschnitt über die in Abbildung 8.2 gezeigten Komponenten unter Laborbedingungen simuliert. Die jeweilige Komponente erlaubt dabei die Anpassung der in dem Kontext gezeigten Evaluierungsparameter, auf die im Folgenden eingegangen wird.

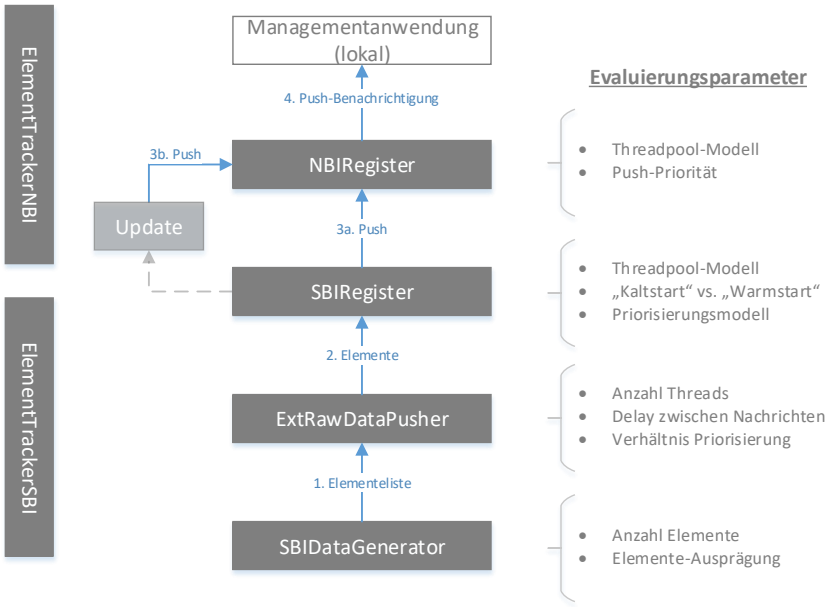


Abbildung 8.2: Übersicht über den Benachrichtigungsprozess mit Evaluierungsparametern.

- **`SBIDataGenerator`**: Um die Ergebnisse konsistent halten zu können, werden `SBIExtRawData`-Elemente über eine eigene zu diesem Zweck implementierte Klasse `SBIDataGenerator` generiert. Als Basiselemente dienen dafür die in Abschnitt 7.4.1 beschriebenen Daten, die einerseits über ein Floodlight- und andererseits über ein OpenNebula-System abgefragt wurden. Auf diese Weise werden gemischt sowohl JSON- als auch XML-

Daten verarbeitet und unterschiedliche Parserimplementierungen genutzt. Die Elemente werden in ihrer Ausprägung sinnvoll variiert, beispielsweise Floodlight-Topologiedaten durch die Variation der Anzahl der Switches (1-50) und Switch-Verbindungen (1-200) sowie die OpenNebula-Host-Daten hinsichtlich der Anzahl der Hosts (1-20) und CPU-Cores (1-100). Darüber hinaus werden Parameterfelder durch zufällig verschieden lange, aber sinnvoll generierte Zeichenketten variiert. Auf diese Weise werden Caching-Effekte in der Implementierung vermieden. Die entstehenden Ereignisse haben schließlich gleichverteilt eine Länge zwischen ca. 500 Byte und ca. 120 Kilobyte und decken daher sowohl kleine als auch sehr große Datenelemente des SBI ab. Das Ergebnis ist eine Liste von unterschiedlichen `SBIExtRawData`-Instanzen, deren Größe kontrolliert gesteuert werden kann. Auf diese Weise kann neben der Variation dieser Parameter die Grundbasis der Daten nachvollziehbar gehalten werden. Wie im erwarteten Praxisfall werden nicht allen, sondern nur einem Teil (hier der Hälfte) der generierten `SBIExtRawData`-Instanzen Normalisierungslabel zugewiesen, die eine Normalisiererauswahl beschleunigen.

- `ExtRawDataPusher`: Die Klasse `ExtRawDataPusher` erlaubt die Variation der Kadenz der Registrierung der zuvor generierten `SBIExtRawData`-Instanzen am SBI-Register. Durch die Anzahl der registrierenden Threads kann der Grad der Nebenläufigkeit gesteuert werden. Darüber hinaus kann pro Thread eine gleichartige Verzögerungszeit ab dem Millisekundenbereich festgelegt werden. Auch das Verhältnis von nicht-priorisierten und zu priorisierten Elementen kann als Prozentsatz angegeben werden. Beispielsweise sagt ein Wert von $0,3$ aus, dass ca. 30 Prozent der am SBI-Register registrierten `SBIExtRawData`-Instanzen priorisiert eingebracht werden sollen.

- `SBIRegister`: Die Evaluierungsparameter des SBI-Registers umfassen das Threadpoolmodell zur Parallelisierung der Schritte des Parsens, Normalisierens und der Aktualisierung des Datenmodells. Das Threadpoolmodell wird jedoch für die drei Teilschritte konsistent gehalten, d. h. die Threadpools für die Aktivitäten im SBI-Register arbeiten stets im gleichen Modus. Darüber hinaus wird zwischen einem *Kaltstart* und einem *Warmstart* unterschieden. Der Kaltstart zeichnet sich dadurch aus, dass die Threadpools nicht durch vorherige Verarbeitung initialisiert und potenziell leer sind. Bei einem Warmstart hingegen werden die Threadpools durch Verarbeitung von Attrappen-Elementen vor den eigentlichen Performanzmessungen aufgebaut und haben bereits einen Satz an Threads angelegt. Der letzte Parameter am SBI ist das Priorisierungsmodell, d. h. die Art der Bevorzugung priorisierter gegenüber nicht-priorisierter Elemente der einzelnen Teilschritte im SBI. Wie in Abschnitt 6.5.1 beschrieben ist, werden im Basisfall 50% der Elemente in den nicht-priorisierten Queues bearbeitet, bevor die priorisierten Queues erneut auf Elemente kontrolliert werden. Die Aktionen der Aktualisierung sind bei der Performanzmessung weiterhin aktiv und dienen auch dem Zweck der Registrierung von Alert-Elementen (vgl. Abschnitt 7.4.1.4), die wesentlich in die Zeitmessung im Schritt *3b* einfließen.

- `NBIRegister`: Die Evaluierungsparameter am NBI-Register ähneln im Threadpoolmodell und der Art des Starts dem des SBI-Registers. Diese können auch an dieser Stelle für Push-Benachrichtigungen angewendet werden. Zusätzlich wird das Verhältnis der Push-Priorität variiert und Benachrichtigungen als *Normal*, *Hoch* oder *Höchst* registriert.

- `ElementTrackerSBI/ElementTrackerNBI`: Diese Klassen wurden zur Zeitmessung des Durchlaufs aller einzelnen Elemente implementiert. Für jedes generierte `SBIExtRawData`-, `SBIDataObject`- und `SBINetworkInformation`-Objekt speichern sie den jeweiligen Zeitstempel der Generierung und den jeweiligen Ableitungspfad. Die Speicherung erfolgt in Javas threadsicheren `ConcurrentHashMap`-Instanzen [210]. Die Zeitmessung einzelner Elemente wird durch die Methode `System.nanoTime()` und daher mit einer theoretischen Genauigkeit, jedoch nicht notwendigerweise einer Auflösung im Nanosekundenbereich umgesetzt [211]. Zur Zeitmessung ist eine Unterscheidbarkeit der Werte im zweistelligen Mikrosekundenbereich notwendig, welche durch diese Funktion am ehes-

ten erfüllt werden kann. Die Klasse bietet schließlich die Funktion zur Berechnung der gemittelten Gesamtlaufzeit aller priorisierten oder nicht-priorisierten Elemente, sowie der Verarbeitungsdauer des jeweiligen Schrittes von einem `SBIExtRawData`-Element zu dem davon abgeleiteten `SBIDataObject`-Element zu dem davon abgeleiteten `SBINetwork-Information`-Element und der Auslösung einer Benachrichtigung. Die Trennung zwischen einem Element-Tracker jeweils für das SBI und das NBI ist ausschließlich zur Vermeidung zyklischer Abhängigkeiten notwendig.

Eine Übersicht über die Evaluierungsparameter mit in dieser Evaluation berücksichtigten Werten ist in Tabelle 8.1 gezeigt. Die Anzahl der verarbeiteten Elemente wird mit einer kleinen Menge von 1.000 Elementen über eine große Menge von 10.000 Elemente (Basisfall) und eine sehr große Menge von 100.000 Elementen getestet. Die Elemente werden parallel entweder mit zwei Threads, fünf Threads oder schließlich zehn Threads am SBI-Register eingebracht. Der Basisfall ist mit zwei Threads ausgelegt, damit den Hauptprozessen der Evaluierung für die anderen Tests mehr CPU-Ressourcen zur Verfügung stehen. Nachrichten werden durch den `ExtRawDataPusher` in erster Einstellung nicht verzögert, im Basisfall jedoch mit einer Verzögerung von einer Millisekunde berücksichtigt, da eine vorhandene, aber geringe Verzögerung eher einem realen Szenario entspricht. Die initiale Priorisierung von Elementen am SBI beträgt im Basisfall 15%, jedoch wird auch ein kleineres Verhältnis von 5% und ein sinnvolles größeres Verhältnis von 30% berücksichtigt. Deutlich größere Verhältnisse sind für eine Priorisierung nicht sinnvoll, da sich dann die meisten Elemente in den priorisierten Queues befinden und sequenziell abgearbeitet werden. Das Threadpoolmodell am SBI und am NBI wird im Basisfall durch einen Cached-Threadpool umgesetzt (vgl. Abschnitt 6.5.1), in dem die Anzahl und Nutzung der Threads durch Java selbst reguliert wird. Alternativ wird ein Ansatz ohne Multithreading (*Single*) sowie ein Threadpool mit fixer Anzahl an Threads (*Fixed*) – in diesem Fall 16, ausgerichtet an der Zahl tatsächlich vom Testsystem parallel ausführbarer Threads – gewählt. Der Betriebsmodus ist im Basisfall als Warmstart ausgelegt, d. h. der eigentlichen Zeitmessung geht eine Menge von immer 1.000 zuvor bearbeiteter, im Inhalt zufällig variierter Elemente mit fixer Priorisierungsrate 15% voraus. Das Priorisierungsmodell am SBI ist wie zuvor beschrieben im Basisfall eine maximale Abarbeitung von 50% der nicht-priorisierten Elemente. Daraufhin wird durch jeden Thread auf Vorhandensein neuer priorisierter Elemente geprüft. Dieses Verhältnis wird in der Evaluation mit 20% bzw. 80% ausgetauscht. Schließlich wird noch das Priorisierungsmodell am NBI variiert. Benachrichtigungen werden im Basisfall als *Normal* priorisiert, in Variation jedoch auch als *Hoch* und *Höchst*.

Parameter	**Ausprägungen**		
Anzahl der Elemente	1.000	**10.000**	100.000
Anzahl Threads `ExtRawDataPusher`	**2**	5	10
Verzögerung pro Nachricht (Prä-SBI)	ohne		**1ms**
Initial priorisierte Elemente am SBI	5%	**15%**	30%
Threadpoolmodell SBI	Single	Fixed (16 Threads pro Pool)	**Cached**
Threadpoolmodell NBI (Push)	Single	Fixed (16 Threads)	**Cached**
Betriebsmodus	Kaltstart		**Warmstart**
Priorisierungsmodell SBI	20%	**50%**	80%
Priorisierungsstufe NBI	**Normal**	Hoch	Höchst

Tabelle 8.1: Übersicht über Parameterausprägungen zur Evaluierung des Benachrichtigungsprozesses. Fett gedruckte Ausprägungen stellen die Basiseinstellung dar, die bei Variation eines anderen Parameters gilt.

Die **Zeitmessung** der beiden hier betrachteten Prozesse in Abbildung 8.2 wird ab Schritt 2, der Registrierung von Einzelelementen am SBI-Register, bis nach Schritt 4, dem Versand der Push-Benachrichtigung, durchgeführt. Die Zeitmessung erfolgt einerseits für den jeweiligen gesamten Durchlauf und gemittelt pro `SBIExtRawData`-Element. Die Zeitmessung der Gesamtdurchläufe wird mit einer Auflösung von Millisekunden gemessen und schließlich durch die Anzahl der verarbeiteten initialen Menge an `SBIExtRawData`-Elemente geteilt. Die Zeitmessung der Einzelelemente erfolgt, wie zuvor beschrieben wurde, durch die Klasse `ElementTracker{SBI, NBI}` mit Unterscheidung zwischen im ganzen Verlauf priorisierten und nicht-priorisierten Elementen. Dabei wird ausschließlich der Teil der Elemente berücksichtigt, die eine **Benachrichtigung ausgelöst** haben, also einen gesamten Durchlauf provozieren. Nicht alle Elemente führen folglich zu einer Benachrichtigung. Auch beide Arten der Zeitmessungen werden über mehrere Durchläufe gemittelt. Die im jeweiligen Test durchgeführten Stichproben sind alle zufällig ausgewählt.

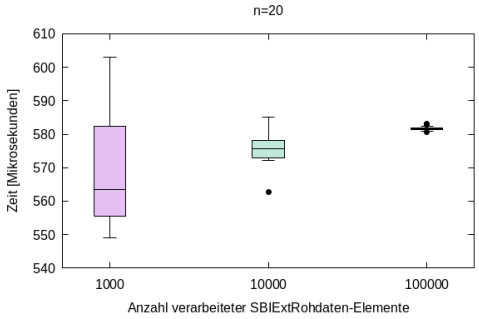
8.2.2.1 Anzahl der `SBIExtRawData`-Elemente

Im Folgenden wird der Einfluss der Anzahl der verarbeiteten `SBIExtRawData`-Instanzen auf den Benachrichtigungsprozess evaluiert. Die Ergebnisse sind im Detail in Abbildung 8.3 illustriert.

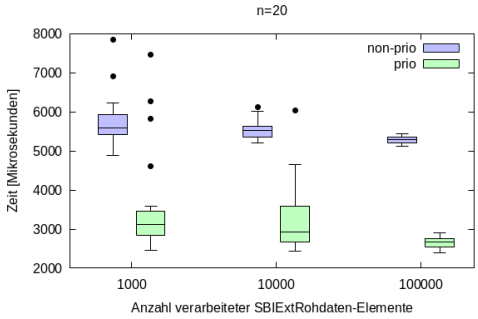
Die über die Anzahl der verarbeiteten `SBIExtRawData`-Instanzen gemittelte Gesamtbearbeitungsdauer ist in Abbildung 8.3a zu sehen. Für 1.000 verarbeitete Elemente ist die Spanne der Ergebnisse dabei in einem Bereich von ca. 50 Mikrosekunden verteilt am höchsten und sinkt mit steigender Elementezahl auf einen Bereich von ca. 20 Mikrosekunden für 10.000 Elemente und 2 Mikrosekunden bei 100.000 Elementen. Die durchschnittliche Bearbeitungsdauer beträgt für 1.000 Elemente ca. 570 Mikrosekunden und steigt bei wachsender Elementezahl nur in sehr geringem Umfang auf ca. 575 und ca. 580 Mikrosekunden an. Auf die Gesamtbearbeitungsdauer hat die Anzahl der verarbeiteten Elemente daher nur einen sehr geringen Einfluss und steigt bei exponentiellem Wachstum der Parametergröße schwach linear an.

Die tatsächliche gemittelte Zeit, in der der Benachrichtigungsprozess aus einem `SBIExtRawData`-Element über die Zwischenformate `SBIDataObject` und `SBINetworkInformation` eine Benachrichtigung generiert, ist hingegen in Abbildung 8.3b veranschaulicht. Dabei ist zu erkennen, dass alle Messreihen mit unterschiedlicher Anzahl an `SBIExtRawData`-Instanzen eine ähnliche Generierungsdauer haben. Diese liegt für nicht-initialpriorisierte Elemente im Durchschnitt bei ca. 5,7 Millisekunden für 1.000 Elemente, bei 5,5 Millisekunden für 10.000 Elemente und 5,3 Millisekunden bei 100.000 Elementen. Initialpriorisierte Elemente hingegen benötigen im Mittel ca. 3,6 Millisekunden für 1.000 Elemente, ca. 3,2 Millisekunden für 10.000 Elemente sowie ca. 2,7 Millisekunden für 100.000 Elemente und benötigen entsprechend halb so lange. Die Priorisierung ist daher unabhängig von den hier betrachteten Größenordnungen an verarbeiteten Elementen effektiv. Auch hier verringert sich mit steigender Anzahl an Elementen der Ergebnisbereich.

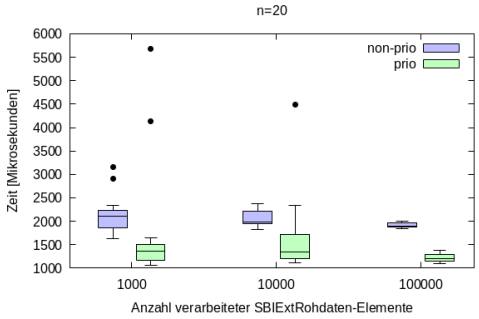
Die zuvor betrachtete tatsächliche Zeit zur Generierung einer Benachrichtigung wurde darüber hinaus an dieser Stelle beispielhaft detaillierter aufgeschlüsselt und verdeutlicht die Bearbeitungszeiten der Teilprozesse Parsen (vgl. Abbildung 8.3c), Normalisierung (vgl. Abbildung 8.3d) und Aktualisierung (siehe Abbildung 8.3e) des SBI. Den größten Teil der Verarbeitung im Benachrichtigungsprozess nimmt das Parsen von Nachrichten ein, ist jedoch für nicht-initialpriorisierte Elemente über die betrachteten Evaluierungsparameter relativ konstant bei einem Bereich von durchschnittlich ca. 1,9 Millisekunden für 100.000 Elemente bis 2,1 Millisekunden für 1.000 Elemente. Priorisierte Elemente liegen zwischen ca. 1,2 Millisekunden (100.000 Elemente) und ca. 1,7 Millisekunden (1.000 Elemente). Auch der zweite Schritt, die Normalisierung der geparsten Elemente ist für nicht-priorisierte Elemente und priorisierte Elemente auf einem relativ gleichbleibenden Wert zwischen ca. 1,8 und 1,7 Millisekunden resp. ca. 1,1 und 1,3 Millisekunden. Der letzte Teilschritt der Aktualisierung liegt für nicht-



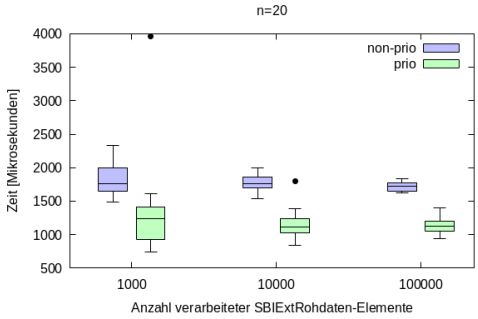
(a) Gemittelte Gesamtbearbeitungszeit pro `SBIExtRawData`-Element.



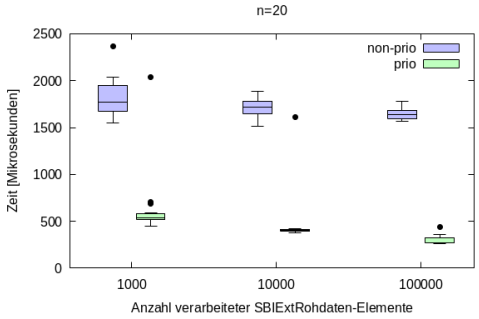
(b) Durchschnittliche Gesamtbearbeitungsdauer pro generierter Benachrichtigung (Schritte 1-3).



(c) Durchschnittliche Verarbeitungsdauer einer `SBIExtRawData`-Instanz zu einer `SBIDataObject`-Instanz (Schritt 1, Parsen).



(d) Durchschnittliche Verarbeitungsdauer einer `SBIDataObject`-Instanz zu einer `SBINetworkInformation`-Instanz (Schritt 2, Normalisierung).



(e) Durchschnittliche Verarbeitungsdauer einer `SBINetworkInformation`-Instanz zu einer Benachrichtigung (Schritt 3, Aktualisierung).

Abbildung 8.3: Einfluss der Anzahl initial verarbeiteten `SBIExtRawData`-Elemente auf die Performanz des Benachrichtigungsprozesses.

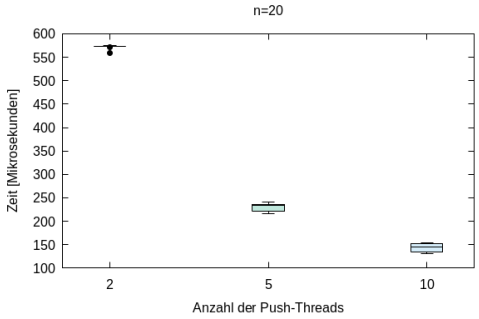
initialpriorisierte Elemente in einem ähnlichen durchschnittlichen Bereich der Verarbeitungsdauer von ca. 1,6 bis 1,8 Millisekunden. Priorisierte Elemente liegen hingegen wesentlich niedriger bei durchschnittlich ca. 0,3 bis 0,6 Millisekunden. Für alle Teilprozesse ist eine leichte Verbesserung bei einer größeren Anzahl verarbeiteter `SBIExtRawData`-Instanzen sowie in den Auswertungen der Gesamtzeit und der Zusammenfassung der Teilschritte eine Verkleinerung des Ergebnisbereichs zu erkennen.

Zusammenfassend kann für den Parameter der Anzahl der verarbeiteten `SBIExtRawData`-Instanzen festgestellt werden, dass *a)* diese keinen wesentlichen Einfluss auf Laufzeiten hat,

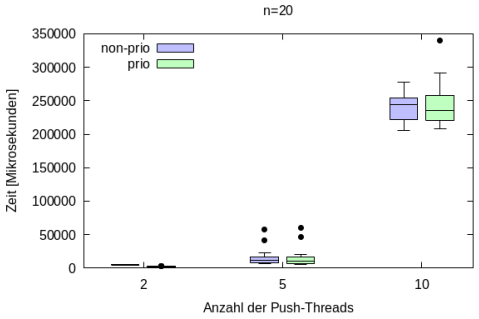
sondern *b)* lediglich eine Schärfung des Ergebnisbereichs für alle Tests bei höherer Anzahl an Elementen herbeiführt. Der erste Aspekt *a)* deutet darauf hin, dass dem Prozess ausreichend Rechenressourcen zur Verfügung standen und alle Elemente in ähnlicher Weise bearbeitet werden konnten. Die zweite Beobachtung *b)* ist für sich genommen kein Ergebnis, das erwartet werden konnte, da mehr verarbeitete Elemente zunächst mehr Potenzial für Ausreißer bieten. Die Schärfung der Ergebnismengen bei steigender Elementezahl kann jedoch durch die Art der Messung pro Stichprobe erklärt werden. Insbesondere für die tatsächliche Generierungsdauer von Benachrichtigungen (vgl. Abbildung 8.3b) und deren Teilschritte bedeuten größere Mengen an verarbeiteten Elementen eine größere absolute Menge generierter Benachrichtigungen. Bei der Bildung des Durchschnittswerts haben Ausreißer dann ein geringeres Gewicht als bei einer kleineren Benachrichtigungsmenge, die bei weniger initialen `SBIExtRawData`-Elementen generiert wird. Ausreißer haben daher bei geringerer Anzahl von initialen `SBIExtRawData`-Instanzen mehr Gewicht in der Auswertung. Auf die gleiche Weise sind ebenfalls die Werte der gemittelten Gesamtlaufzeiten (siehe Abbildung 8.3a) erklärbar.

8.2.2.2 Anzahl der Pusher-Threads

Die Anzahl der Pusher-Threads beeinflusst im Wesentlichen den Grad an Gleichzeitigkeit, in dem `SBIExtRawData`-Elemente am SBI-Register zur Verarbeitung registriert werden. Eine Variierung wirkt sich sehr deutlich auf die Performanz des Prozesses aus, wie in Abbildung 8.4 gezeigt wird. Die Ergebnisse sind dabei sehr konsistent, sodass für diese Experimentreihe eine Anzahl von 20 Stichproben ausreichend ist.



(a) Gesamtbearbeitungszeit für alle Elemente, geteilt durch die Anzahl der verarbeiteten `SBIExtRawData`-Elemente.



(b) Durchschnittliche Gesamtbearbeitungsdauer pro generierter Benachrichtigung.

Abbildung 8.4: Einfluss der Anzahl der Pusher-Threads auf die Performanz des Benachrichtigungsprozesses.

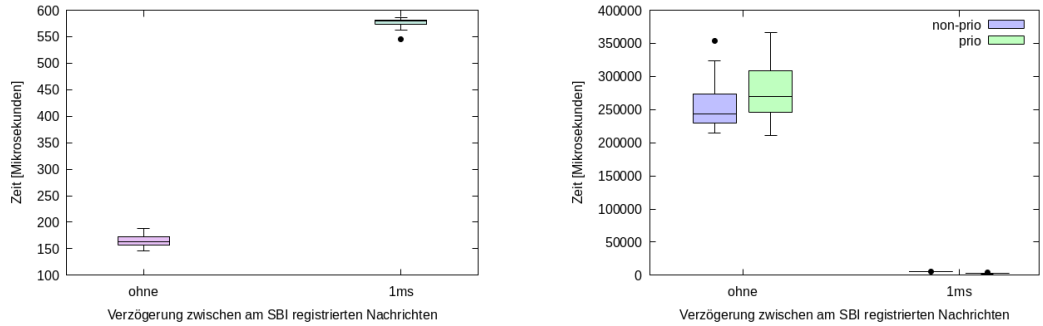
Im Detail zeigt Abbildung 8.4a die durchschnittliche und gemittelte Gesamtdauer zur Bearbeitung von 10.000 Elementen für eine unterschiedliche Anzahl an Pusher-Threads. Die Ergebnisse verdeutlichen, dass der in dieser Arbeit konzipierte und implementierte Prozess vom Empfang eines Datums der gemanagten Infrastruktur bis hin zur Generierung einer Benachrichtigung insgesamt einen hohen Grad an Gleichzeitigkeit bedienen kann und die über alle verarbeiteten Elemente gemittelte Performanz bei höherer Gleichzeitigkeit sogar zunimmt. So konnten mit zwei Pusher-Threads ca. 570 Mikrosekunden erreicht werden, mit fünf Pusher-Threads Werte um 230 Mikrosekunden und mit zehn Pusher-Threads Werte von ca. 140 Mikrosekunden erzielt werden. Die Werte zeigen entsprechend einen logarithmischen Verlauf: Es kann erwartet werden, dass jenseits von zehn Threads eine schrittweise geringere Verbesserung der Performanz bis zu einem Optimum erzielt wird.

Abbildung 8.4b zeigt dagegen die Auswirkung der Anzahl der Pusher-Threads auf die durchschnittliche Zeit, die es braucht, um eine Benachrichtigung zu generieren. Hier zeigt sich, dass sich die Performanz umgekehrt zur zuvor betrachteten gemittelten Gesamtlaufzeit verhält.

Die Generierungszeit einer Benachrichtigung steigt exponentiell mit der Anzahl der Pusher-Threads. Während der Prozess es bei zwei Pusher-Threads schafft, eine Benachrichtigung im Durchschnitt innerhalb von 2,4 Millisekunden für initialpriorisierte und 4,9 Millisekunden für nicht-initialpriorisierte `SBIExtRawData`-Instanzen zu generieren, so benötigt er bei fünf Pusher-Threads 14,9 resp. 15,8 Millisekunden und mit zehn Pusher-Threads 244 resp. 240 Millisekunden. Die Ergebnisse zeigen, dass eine hohe Gleichzeitigkeit der Einbringung von Elementen am SBI den Prozess pro Element einerseits deutlich verlangsamen. Das kann jedoch durch die Bindung von Systemressourcen durch die Pusher-Threads erklärt werden, sodass den Threadpools des NBI und SBI des Frameworks weniger CPU-Ressourcen zur Verfügung stehen. Andererseits wird der hier implementierte Priorisierungsmechanismus bei hoher Gleichzeitigkeit unwirksam. Er müsste für Szenarien verbessert werden, in denen sehr viele Elemente am SBI empfangen und verarbeitet werden. Für eine moderate Gleichzeitigkeit ist der Mechanismus jedoch wirksam und verkürzt die Bearbeitungsdauer bei zwei Threads um mehr als die Hälfte im Gegensatz zu nicht-priorisierten Elementen.

8.2.2.3 Verzögerung pro Nachricht

Eine weitere Dimension zur Steuerung des Grads der Gleichzeitigkeit der Verarbeitung im Prozess ist die Verzögerung jeder `SBIExtRawData`-Instanz bei Einbringen in den Benachrichtigungsprozess. Auch diese Auswertung basiert auf 20 Stichproben. Abbildung 8.5 zeigt die gemittelte durchschnittliche Gesamtbearbeitungszeit für 10.000 Elemente sowie die tatsächliche mittlere Dauer bis zur Generierung einer Benachrichtigung jeweils ohne Verzögerung und mit einer Millisekunde Verzögerung.



(a) Gesamtbearbeitungszeit für alle Elemente, geteilt durch die Anzahl der verarbeiteten `SBIExtRawData`-Elemente.

(b) Durchschnittliche Gesamtbearbeitungsdauer pro generierter Benachrichtigung.

Abbildung 8.5: Einfluss der Verzögerung pro `SBIExtRawData`-Instanz auf die Performanz des Benachrichtigungsprozesses.

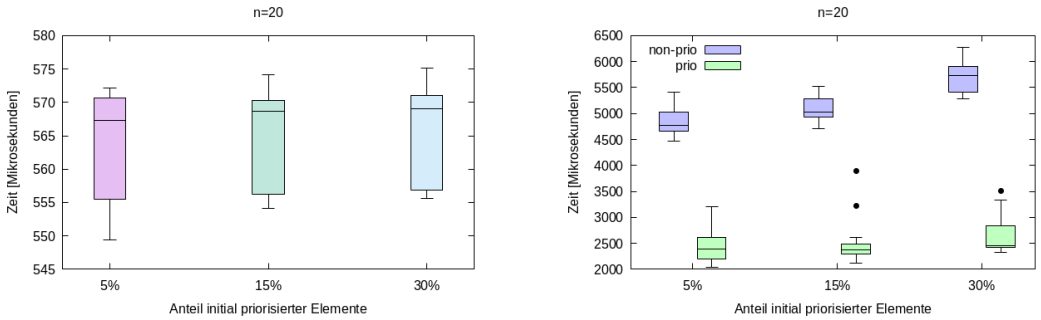
Wie in Abbildung 8.5a gezeigt ist, beläuft sich die durchschnittliche Bearbeitungszeit nach Gesamtbearbeitungsdauer ohne Verzögerung mit relativ hoher Präzision im Durchschnitt auf ca. 160 Mikrosekunden und mit einer Verzögerung von einer Millisekunde im Durchschnitt auf ca. 570 Mikrosekunden. Der rechnerisch erwartete Overhead bei Letzterem würde bei 2 Pusher-Threads und 10.000 Elementen bei 5.000 Millisekunden insgesamt bzw. 500 Mikrosekunden pro Element betragen. Da die tatsächliche Differenz bei ca. 410 Mikrosekunden pro Element liegt, kann davon ausgegangen werden, dass im Fall ohne Push-Verzögerung eine geringe Verzögerung von durchschnittlich 90 Mikrosekunden pro Element durch den Verarbeitungsprozess selbst eingebracht wird.

Abbildung 8.5b hingegen zeigt die tatsächliche durchschnittliche Verarbeitungszeit bis zur Generierung einer Benachrichtigung. Für die Testläufe ohne Push-Verzögerung benötigt die Implementierung im Durchschnitt 255 Millisekunden für nicht-priorisierte Elemente und 277 Millisekunden für priorisierte Elemente. Wie im vorherigen Test mit mehr Pusher-Threads (vgl.

vorheriger Abschnitt) wird bei zu hoher Gleichzeitigkeit der Priorisierungsmechanismus unwirksam. Eine Registrierung von `SBIExtRawData`-Elementen ohne Verzögerung ist dann vergleichbar mit ca. zehn Pusher-Threads mit einer Millisekunde Verzögerung. Im Vergleich zur zuvor betrachteten durchschnittlichen Gesamtbearbeitungsdauer durch die Anzahl der initialen `SBIExtRawData` liegt dieser Wert deutlich höher und zeigt, dass sehr viele Elemente vielmehr gleichzeitig bearbeitet werden und daher erst deutlich später eine Notifikation generieren als es beispielsweise bei serieller Verarbeitung der Elemente der Fall wäre. Die Variante mit zwei Pusher-Threads und einer Millisekunde Verzögerung benötigt hingegen ca. 5,3 Millisekunden für nicht-priorisierte und ca. 2,6 Millisekunden für priorisierte Elemente. Der Effekt des Priorisierungskonzepts ist in diesem Fall deutlich erkennbar.

8.2.2.4 Verhältnis initial priorisierter SBIExtRawData-Elemente

Die Wirksamkeit der Priorisierung ist potenziell auch von der Anzahl initial priorisierter Elemente abhängig. Dabei besteht die Vermutung, dass zu viele priorisierte Elemente den Zeitvorteil priorisierter Elemente unwirksam werden lassen können. Abbildung 8.6 zeigt den tatsächlichen Sachverhalt dazu anhand von 20 zufälligen Stichproben.



(a) Gesamtbearbeitungszeit für alle Elemente, geteilt durch die Anzahl der verarbeiteten `SBIExtRawData`-Elemente.

(b) Durchschnittliche Gesamtbearbeitungsdauer pro generierter Benachrichtigung.

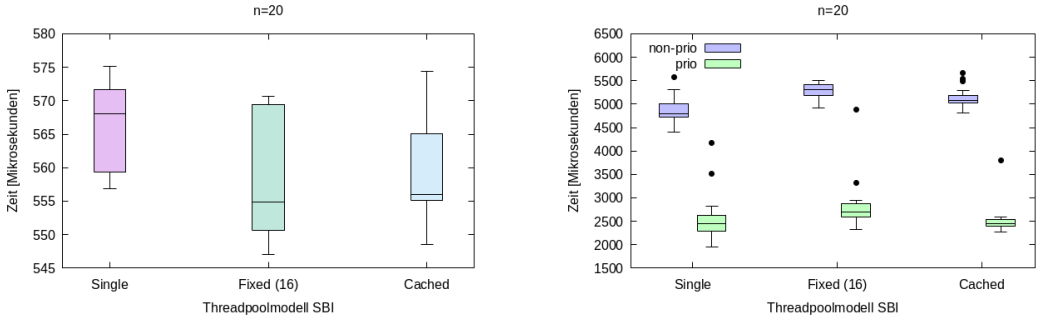
Abbildung 8.6: Einfluss des Anteils der initial priorisierten `SBIExtRawData`-Instanzen auf die Performanz des Benachrichtigungsprozesses.

Hinsichtlich der in Abbildung 8.6a gezeigten pro Element gemittelten Gesamtbearbeitungsdauer ist erkennbar, dass diese nicht wesentlich von der Anzahl der initialpriorisierten Elemente abhängig ist und im Mittel für jeden Parameter bei ca. 565 Mikrosekunden liegt.

Die in Abbildung 8.6b gezeigten Messungen pro Element bis zur Generierung einer Benachrichtigung zeigen für einen steigenden Anteil initialpriorisierter `SBIExtRawData`-Instanzen, dass nur eine leichte Erhöhung der mittleren Bearbeitungszeit auftritt. Bei einem Anteil von fünf sowie von 15 Prozent liegt die Bearbeitungszeit dieser im Mittel bei ca. 2,43 bis 2,47 Millisekunden. Bei einem Anteil von 30 Prozent hingegen bei ebenfalls nicht wesentlich höheren 2,65 Millisekunden. Bei nicht-priorisierten Elementen ist dagegen eine vergleichsweise höhere Steigerung der Bearbeitungszeit erkennbar. So liegt diese bei fünf Prozent bei 4,86 Millisekunden, bei 15 Prozent bei ca. 5,08 Millisekunden und für 30 Prozent bei ca. 5,72 Millisekunden. Entsprechend werden durch die Erhöhung des Anteils initialpriorisierter Elemente insbesondere nicht-priorisierte Elemente benachteiligt.

8.2.2.5 Threadpoolmodell des Southbound-Interface

Es besteht die Annahme, dass Änderungen des Threadpoolmodells des SBI einen wesentlichen Einfluss auf die Performanz des Benachrichtigungsprozesses dieser Arbeit hat. Eine Testreihe mit 20 zufällig gewählten Stichproben dazu ist in Abbildung 8.7 gezeigt.



(a) Gesamtbearbeitungszeit für alle Elemente, geteilt durch die Anzahl der verarbeiteten `SBIExtRawData`-Elemente.

(b) Durchschnittliche Gesamtbearbeitungsdauer pro generierter Benachrichtigung.

Abbildung 8.7: Einfluss des am SBI eingesetzten Threadpoolmodells auf die Performanz des Benachrichtigungsprozesses.

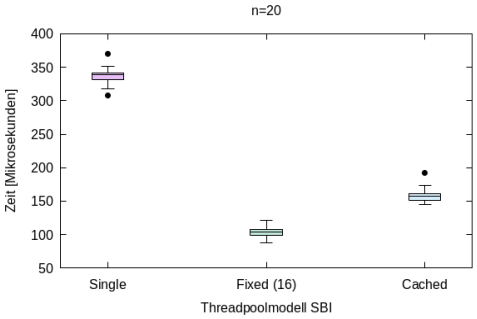
Hinsichtlich der Gesamtlaufzeit (vgl. Abbildung 8.7a) ist gut zu erkennen, dass das verwendete SBI-Threadpoolmodell auf die Gesamtbearbeitungsdauer für 10.000 Elemente keinen großen Einfluss in der aktuellen Konfiguration hat. Starkes Multithreading durch Threadpool mit konstant 16 Threads (ca. 558 Mikrosekunden pro Element) oder auch mit Javas Cached-Threadpoolmodell (ca. 560 Mikrosekunden pro Element) können gegenüber einem Single-Thread-Ansatz (ca. 565 Mikrosekunden pro Element) keinen bedeutsamen Vorteil erreichen.

Ein ähnliches Ergebnis ist bei der durchschnittlichen Dauer zur Generierung einer Benachrichtigung (vgl. Abbildung 8.7b) erkennbar. Für nicht-priorisierte Elemente liegen die durchschnittlichen Zeiten für eine Single-, Fixed- und Cached-Einstellung bei ca. 4,9 zu ca. 5,3 zu ca. 5,1 Millisekunden. In diesem Fall sind die Ansätze mit starkem Multithreading sogar schlechter, was durch eine unnötige Ressourcenbindung durch zu viele Threads erklärt werden kann. Ein ähnliches Ergebnis ist bei priorisierten Elementen mit durchschnittlich ca. 2,5 zu 2,8 und 2,5 Millisekunden zu beobachten. Die Priorisierung von Elementen ist jedoch deutlich bemerkbar.

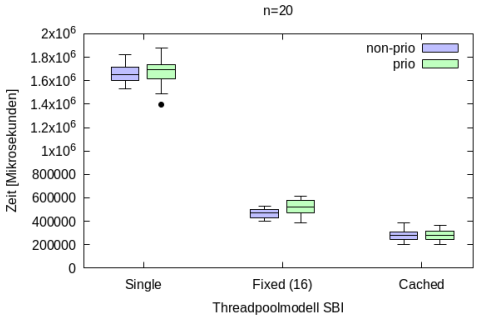
Die Ergebnisse weisen darauf hin, dass ein Singlethreading-Modell im SBI für die aktuelle Konfiguration und insbesondere der durch die Parameter der Pusher-Threads und Push-Verzögerung erzielte Grad an Gleichzeitigkeit ausreichend ist. Dabei ist zu berücksichtigen, dass der Prozess im SBI auch im Single-Modell nicht durch einen einzigen, sondern durch drei Threads behandelt wird, einer zum Parsen, einer zur Normalisierung und einer zur Aktualisierung des Zustands der Managementplattform sowie zum Anstoßen von Benachrichtigungen am NBI-Register.

Zur Stützung dieser Erklärung werden weitere Testreihen mit deutlich höherem Grad an Gleichzeitigkeit in Abbildung 8.8 gezeigt, die mit fünf Pusher-Threads und ohne Push-Verzögerung durchgeführt wurden. Die Gesamtbearbeitungsdauer weist in diesem Fall (vgl. Abbildung 8.8a) einen Vorteil für die Varianten mit starker Parallelisierung auf. Statt wie im Single-Modell-Fall mit ca. 340 Mikrosekunden Laufzeit hat eine Verarbeitung mit konstant 16 Threads eine Laufzeit von ca. 100 bzw. ca. 160 Mikrosekunden pro Element und ist daher dreifach resp. doppelt so schnell. Die weitere Senkung des Wertes zeigt auf, dass die Basiskonfiguration noch keine Limitierung für ein Single-Threadmodell ist.

Bei der Betrachtung der durchschnittlichen Gesamtbearbeitungsdauer zur Generierung einer Notifikation im Analysefall (Abbildung 8.8b) ist zudem eine deutliche Verschlechterung des Single-Modells erkennbar. In diesem wird eine Benachrichtigung im Mittel nach 1,6 Sekunden generiert, im Vergleich zu zwischen 470 und 520 Millisekunden für das Fixed-Threadmodell



(a) Gesamtbearbeitungszeit für alle Elemente, geteilt durch die Anzahl der verarbeiteten `SBIExtRawData`-Elemente.



(b) Durchschnittliche Gesamtbearbeitungsdauer pro generierter Benachrichtigung.

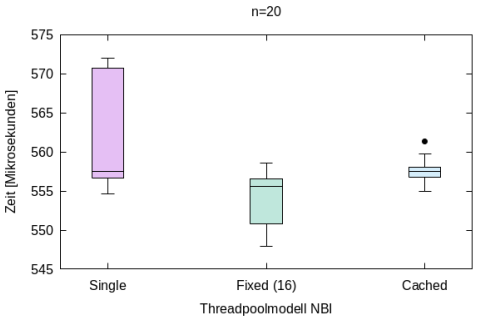
Abbildung 8.8: Verfeinerte Analyse des Einflusses des SBI-Threadpoolmodells auf die Performanz mit fünf Pusher-Threads und ohne Push-Verzögerung.

und ca. 280 Millisekunden für das Cached-Threadmodell. Wie bereits durch vorherige Tests ermittelt, ist das Priorisierungssystem bei diesem hohen Grad an Parallelisierung nicht mehr effektiv.

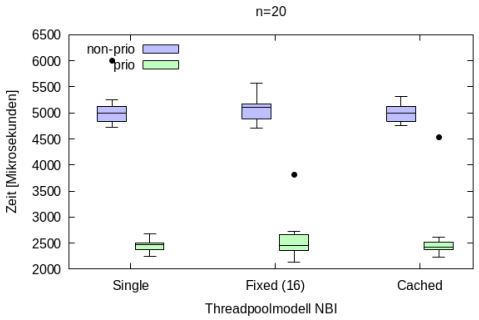
Die Annahme, dass eine sehr hohe Parallelisierung des Prozesses einen wesentlichen Einfluss auf seine Performanz hat, ist daher besonders für Szenarien zutreffend, in denen Elemente der gemanagten Infrastruktur mit einer hohen Kadenz am SBI registriert werden.

8.2.2.6 Threadpoolmodell des Northbound-Interface

Das Threadpoolmodell des NBI hat vermutlich einen deutlich geringeren Einfluss auf den Benachrichtigungsprozess als das Threadpoolmodell des SBI. Die Hypothese basiert auf der Annahme, dass die Menge der jeweils voneinander abgeleiteten Elemente mit jedem Schritt tendenziell geringer wird und folglich nur aus einem Bruchteil der initial am SBI verarbeiteten `SBIExtRawData`-Instanzen über Zwischenformate eine Benachrichtigung generiert wird. Die Auswertung von 20 zufälligen Stichproben ist dazu in Abbildung 8.9 gezeigt.



(a) Gesamtbearbeitungszeit für alle Elemente, geteilt durch die Anzahl der verarbeiteten `SBIExtRawData`-Elemente.



(b) Durchschnittliche Gesamtbearbeitungsdauer pro generierter Benachrichtigung.

Abbildung 8.9: Einfluss des Threadpoolmodells des NBI auf die Performanz des Benachrichtigungsprozesses.

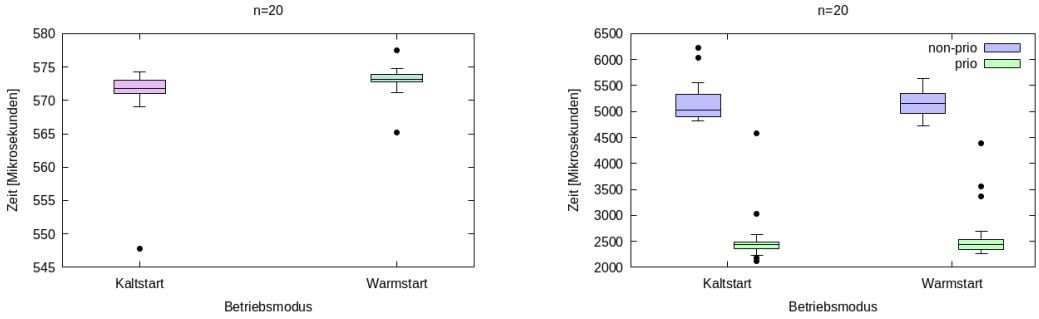
Die in Abbildung 8.9a gezeigte Auswertung zur Gesamtbearbeitungszeit pro Element zeigt eine sehr ähnliche durchschnittliche Bearbeitungszeit von ca. 560 Mikrosekunden pro Element

für alle drei Threadpoolmodelle. Das Single-Modell liegt dabei mit sehr geringer Tendenz über diesem Wert, das Fixed- sowie das Cached-Modell mit schwacher Tendenz darunter. Die beiden letzteren Modelle führen zudem zu einem eingeschränkteren Ergebnisbereich, d. h. zu konsistenteren Durchlaufzeiten.

Die mittleren Zeiten zur Generierung einer Benachrichtigung sind in Abbildung 8.9b dargestellt. Sie sind sehr ähnlich und ergeben für Single-, Fixed- und Cached-Threadpoolmodelle durchschnittliche Zeiten von ca. 5,0 bzw. 5,1 und 5,0 Millisekunden. Die Generierungszeiten für priorisierte Elemente liegen dagegen bei ca. 2,4 für das Single-Modell bzw. 2,5 Millisekunden für die beiden Multithreading-Ansätze. Die Ergebnisse weisen ähnlich wie zuvor bei unterschiedlichen SBI-Threadpoolmodellen darauf hin, dass der durch die Testparameter erreichte Grad an Gleichzeitigkeit am NBI nicht ausreichend ist, um wesentliche Unterschiede feststellen zu können.

8.2.2.7 Betriebsmodus

Durch die Nutzung von insgesamt vier Threadpools im Benachrichtigungsprozess – drei im SBI und einer im NBI – besteht die Annahme, dass der Betriebsmodus des Systems als *Kaltstart* (neue Threadpools) bzw. *Warmstart* (Threadpools waren bereits im Betrieb) ein Kriterium der Performanz ist. Wie die Auswertung von 20 Stichproben in Abbildung 8.10 zeigt, ist dies jedoch nicht der Fall.



(a) Gesamtbearbeitungszeit für alle Elemente, geteilt durch die Anzahl der verarbeiteten `SBIExtRawData`-Elemente.

(b) Durchschnittliche Gesamtbearbeitungsdauer pro generierter Benachrichtigung.

Abbildung 8.10: Einfluss des Betriebsmodus auf die Performanz des Benachrichtigungsprozesses.

Die in Abbildung 8.10a evaluierte Gesamtbearbeitungsdauer pro Element ist für beide Optionen im Mittel beinahe identisch und liegt für einen Kaltstart und einen Warmstart mit wenigen Ausreißern bei ca. 570 Mikrosekunden. Selbst der größte Ausreißer im Kaltstart liegt dabei jedoch maximal 22 Mikrosekunden von diesem Mittel entfernt.

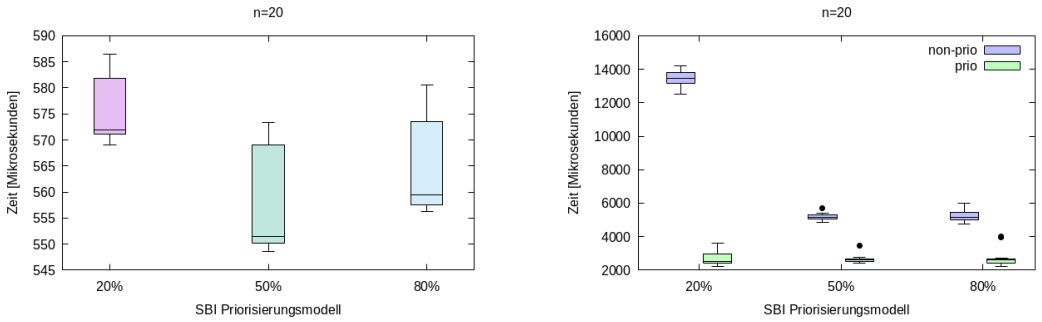
Die mittlere Dauer der Generierung einer Benachrichtigung (vgl. Abbildung 8.10b) weist eine ähnlich kleine Bandbreite an Abweichung auf mit einheitlich im Durchschnitt 5,2 Millisekunden für nicht-priorisierte Elemente. Priorisierte Elemente konnten ebenfalls mit einer sehr geringen Abweichung voneinander von ca. 2,5 Millisekunden (Kaltstart) und 2,6 Millisekunden (Warmstart) keine deutlichen Unterschiede aufzeigen.

Das hier für das NBI und SBI verwendete Cached-Threadpoolmodell scheint daher bei der Zahl von 10.000 verarbeiteten Elementen schnell genug aufgebaut werden zu können, damit keine wesentlichen Unterschiede erkennbar sind. Ein möglicher Effekt ist für kleinere Elementzahlen potenziell stärker erkennbar, die jedoch für produktiv eingesetzte Managementplattformen

irrelevante Szenarien darstellen, da eine Managementplattform in der Regel über viele Monate und Jahre betrieben wird und daher große Datenmengen verarbeiten muss.

8.2.2.8 Priorisierungsmodell am SBI

Die Einstellung des Priorisierungsmodells wird als wichtiges Kriterium angenommen, um priorisierte Benachrichtigungen auf Kosten nicht-priorisierter Benachrichtigungen schneller bearbeiten zu können. In dem hier einfachen implementierten Priorisierungsmodell wird bestimmt, wie groß der Anteil der nicht-priorisierten Elemente in der jeweiligen Queue von einem Thread maximal verarbeitet werden, bis pro Thread geprüft wird, ob neue priorisierte Elemente vorhanden sind. Es besteht die Annahme, dass mit steigendem Anteil priorisierter Elemente kürzer und nicht-priorisierte Elemente länger bis zur Benachrichtigungsgenerierung laufen. Die Ergebnisse der Auswertung sind in Abbildung 8.11 zusammengefasst.



(a) Gesamtbearbeitungszeit für alle Elemente, geteilt durch die Anzahl der verarbeiteten `SBIExtRawData`-Elemente.

(b) Durchschnittliche Gesamtbearbeitungsdauer pro generierter Benachrichtigung.

Abbildung 8.11: Einfluss des SBI-Priorisierungsanteils auf die Performanz des Benachrichtigungsprozesses.

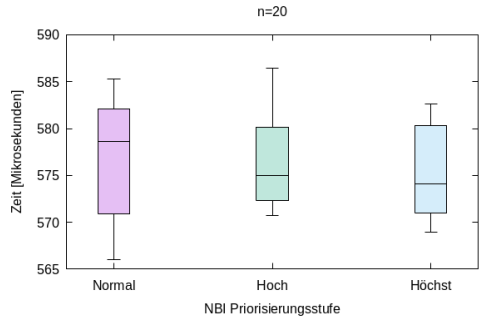
Wie in Abbildung 8.11a gezeigt wird, ist die pro `SBIExtRawData`-Element gemittelte Gesamtlaufzeit in einem kleinen Gesamtbereich von ca. 40 Mikrosekunden. Der Unterschied ist hier entsprechend sehr gering und die durchschnittlichen Laufzeiten pro Element betragen für die Parameterwerte von 20%, 50% und 80% entsprechend ca. 575, 560 und 564 Mikrosekunden. Der geringe Einfluss des Priorisierungsanteils auf die pro Element gemittelte Gesamtlaufzeit der Tests ist jedoch nachvollziehbar und erwartet, da sie sich lediglich auf die Reihenfolge der verarbeiteten Elemente auswirkt.

Ein anderes Bild ergibt sich bei der Betrachtung der durchschnittlich notwendigen Zeit, um auf Basis eines `SBIExtRawData`-Elements eine Benachrichtigung zu generieren. Wie Abbildung 8.11b dazu zeigt, bestehen hier deutliche Unterschiede zwischen priorisierten und nicht-priorisierten Elementen. Bei einer Verarbeitung pro Thread von maximal 20% von nicht-priorisierten Elementen vor erneuter Prüfung auf das Vorhandensein priorisierter Elemente ist der zusätzliche durchschnittliche Zeitaufwand bei ersteren um den Faktor 4,9 höher. Priorisierte Elemente führten entsprechend im Durchschnitt in ca. 2,73 Millisekunden zur Generierung einer Benachrichtigung, nicht-priorisierte Elemente in ca. 13,45 Millisekunden. Bei einem Anteil von 50% nicht-priorisierter Elemente sinkt dieser Faktor deutlich auf 2,0 ab: Priorisierte Elemente führen mit ca. 2,6 Millisekunden doppelt so schnell zu einer Benachrichtigungsgenerierung als nicht-priorisierte Elemente mit ca. 5,2 Millisekunden. Im Vergleich zum vorherigen Modell ist erkennbar, dass sich im Wesentlichen die Verarbeitungsdauer nicht-priorisierter Elemente verändert hat, die mittlere Dauer zur Generierung einer Benachrichtigung durch priorisierte Elemente jedoch nicht. Bei erneuter Erhöhung des Anteils auf 80% bleibt dieser Faktor ebenfalls weitestgehend gleich, bei durchschnittlichen Verarbeitungszeiten von 5,3 und

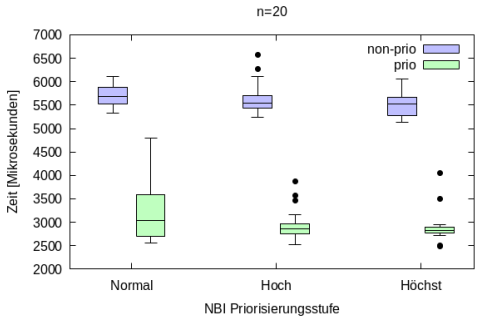
2,7 Millisekunden. Das hier implementierte Priorisierungsmodell scheint sich daher in dieser Gesamtkonfiguration ab einem gewissen Wert zwischen 20% und 50% nicht weiter zu verbessern.

8.2.2.9 Priorisierungsstufe am NBI

Die Auswirkungen der Priorisierungsstufe einer registrierten `NBIPushComponent`-Instanz hat für das Evaluierungsszenario keine starken Auswirkungen auf die Performanz, wie in Abbildung 8.12 gezeigt ist. Die Gesamtbearbeitungsdauer für alle Elemente, gemittelt pro Element und gezeigt in Abbildung 8.12a, bleibt mit hoher Genauigkeit über die unterschiedlichen Parameter gleich bei durchschnittlich ca. 575 Mikrosekunden. Dieses Ergebnis ist erwartbar, da die Priorisierung am NBI nicht den Gesamtbearbeitungsaufwand beeinflusst, sondern nur gewisse Elemente bei der Benachrichtigungsgenerierung bevorzugt.



(a) Gesamtbearbeitungszeit für alle Elemente, geteilt durch die Anzahl der verarbeiteten `SBIExtRawData`-Elemente.



(b) Durchschnittliche Gesamtbearbeitungsdauer pro generierter Benachrichtigung.

Abbildung 8.12: Einfluss der Priorisierung von Push-Benachrichtigungen am NBI auf die Performanz des Benachrichtigungsprozesses.

Geringe erkennbare Auswirkungen hat die Priorisierung von `NBIPushComponent`-Instanzen jedoch auf die durchschnittliche Gesamtbearbeitungszeit pro Element, wie in Abbildung 8.12b gezeigt ist. Diese sinkt über die Einstellung von *Normal*, *Hoch* und *Höchst* stetig von ca. 5,7 auf ca. 5,6 und schließlich ca. 5,5 Millisekunden bei nicht-initialpriorisierten Elementen. Für priorisierte Elemente sinkt die Bearbeitungsdauer ebenfalls stetig in gleicher Reihenfolge von ca. 3,2 auf knapp über 2,9 und schließlich auf knapp unter 2,9 Millisekunden. Die geringen Auswirkungen der Priorisierung sind hier insofern erwartbar, da im Evaluierungsszenario lediglich drei Benachrichtigungsbausteine am NBI registriert sind. Ein deutlich größerer Einfluss wäre für eine weit höhere Zahl an Benachrichtigungsbausteinen über die drei Prioritätenstufen verteilt zu erwarten, da jede Stufe in der Implementierung eine Liste von registrierten Bausteinen darstellt, die nacheinander zunächst für die Einstellung *Höchst*, dann *Hoch* und schließlich *Normal* abgearbeitet werden. Aufgrund des einfachen Priorisierungsmechanismus und da bereits für die hier betrachteten drei registrierten `NBIPushComponent`-Elementen bei jeweils durchschnittlich ca. 750 generierten Benachrichtigung stetige erkennbare Verbesserungen im Erwartungsbereich liegen, wird an dieser Stelle auf eine detailliertere Analyse verzichtet.

8.2.3 Closed-Loop-Automatisierung

Eine Closed-Loop-Automatisierung beinhaltet den zuvor betrachteten Benachrichtigungsprozess, eine Auswertung der daraus erfassten Daten durch Managementanwendungen sowie eine darauf basierende reaktive Steueraktion im gemanagten Netz. Eine beispielhafte Näherung des zeitlichen Aufwands einer Steueraktion im Netz ist im folgenden Abschnitt beschrieben. In Abschnitt 8.2.3.2 werden schließlich alle Teile des Closed-Loop-Automatisierungsprozesses im Gesamtkontext betrachtet und bewertet.

8.2.3.1 Ausführung von Pull-Zugriffen am NBI

Steuerungsaktionen werden über Pull-Zugriffe am NBI der Managementplattform realisiert. Die Ausführungszeit von Pull-Zugriffen ist von drei Teilprozessen abhängig:

- Der Entscheidung über die Zugreifbarkeit eines Nutzers darauf.
- Die Identifikation einer `NBIPullComponent`-Instanz des NBI, die die Ausführung einer Plattformfunktion erlaubt.
- Die eigentliche Ausführung der `NBIPullComponent`-Instanz bzw. ihres Treibers.

Der Aspekt der Zugreifbarkeit und Treffen einer Zugriffsentscheidung wurde bereits in Abschnitt 8.2.1 für alle Informationselemente übergreifend betrachtet und ist daher auch für diesen Zweck anwendbar.

Der zweite Punkt, die Identifikation einer `NBIPullComponent`-Instanz anhand ihrer jeweiligen ID, wird anhand 50 Stichproben ermittelt. Im Framework werden `NBIPullComponent`-Instanzen in einer HashMap mit ihrer jeweiligen ID als Schlüssel verwaltet. Aufgrund dessen kann erwartet werden, dass die Anzahl registrierter `NBIPullComponent`-Instanzen den größten Einfluss auf die Performanz dieses Prozesses hat. Betrachtet wird eine relativ kleine Menge von 100, eine größere Menge von 500 sowie eine große Menge von 1.000 registrierten `NBIPullComponent`-Instanzen. Dabei wird stets der Zugriff auf die in Abschnitt 7.4.2.1 implementierte `ODLBlockMAC`-Instanz gemessen, die genau einmal pro Testszenario registriert ist. Alle restlichen `NBIPullComponent`-Instanzen werden als Attrappe implementiert und registriert. Die Reihenfolge, d. h. auch die Stelle, an der die zugegriffene Pull-Komponente registriert wird, ist zufällig gleichverteilt.

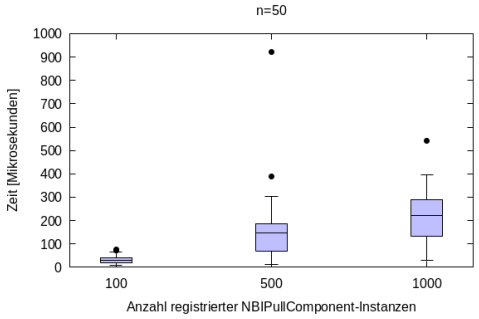
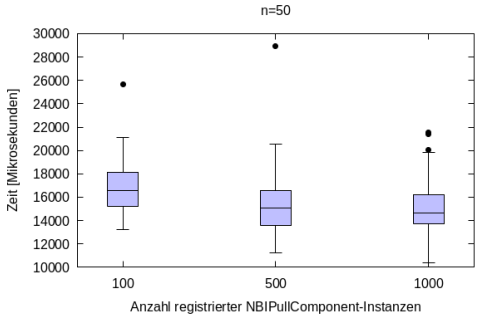


Abbildung 8.13: Dauer zur Identifikation und der Anfrage einer `NBIPullComponent`-Instanz auf Basis ihrer ID.

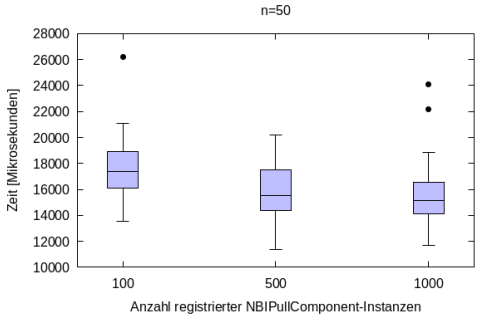
Die Auswertung der Stichproben wird in Abbildung 8.13 gezeigt. Dabei ist gut zu erkennen, dass die Zugriffzeiten nach Anzahl registrierter Komponenten sublinear steigen. Bei 100 registrierten `NBIPullComponent`-Instanzen ist die mittlere Zugriffszeit bei ca. 30 Mikrosekunden, bei 500 registrierten Komponenten im Mittel bei ca. 150 Mikrosekunden und bei 1.000 Komponenten bei ca. 215 Mikrosekunden. Besonders ein Ausreißer bei 500 registrierten Komponenten stellt mit ca. 920 Mikrosekunden jedoch nicht die Regel dar und ist sehr wahrscheinlich durch Hintergrundprozesse (z. B. Garbage-Collection in der JVM) verursacht worden.

Beim letzten Aspekt der Performanz, der eigentlichen Ausführung eines Pull-Zugriffs, wird zwischen zwei Varianten unterschieden, wie Abbildung 8.14 zeigt. Zum einen eine simulierte lokale als auch remote Ausführung. Da die Performanz an dieser Stelle jedoch stark implementierungsabhängig ist und über Funktionen des Frameworks hinausgeht, dienen die Messungen nur als Beispiel und sind nicht auf andere `NBIPullComponent`-Implementierungen

übertragbar. Aus diesem Grund wurden die Tests in einer Laborumgebung mit darunterliegendem MO implementiert. Das heißt, in den betrachteten Beispielen fließen im konkreten Fall Bearbeitungs- und Antwortzeiten des hinter der `ODLBlockMAC`-Instanz angesprochenen OpenDaylight-Systems sowie eine geringe Kommunikationslatenz über das Netz mit ein. Da die Framework-Tests und das OpenDaylight-System jedoch auf demselben physischen Testsystem betrieben werden, ist die Netzlatenz sehr gering.



(a) Ausführzeit eines Zugriffs einer simulierten lokalen Managementanwendung.



(b) Ausführungszeit einer simulierten remoten Zugriffs (ohne zusätzlichen Overhead für Anfrage über das Netz).

Abbildung 8.14: Ausführungszeiten von lokalen oder remoten Pull-Zugriffen für eine `ODLBlockMAC`-Instanz.

Einem lokalen Zugriff geht dabei der zuvor betrachtete Prozess der Identifikation und des Holens einer `NBIPullComponent`-Instanz voraus. Die Performanz für unterschiedlich viele registrierte Pull-Komponenten ist in Abbildung 8.14a gezeigt. Für 100 Komponenten liegt sie im Mittel bei ca. 16,9 Millisekunden, für 500 Komponenten bei ca. 15,4 Millisekunden und bei 1.000 registrierten Komponenten bei ca. 15,2 Millisekunden. Die Ausführungszeiten weisen eine geringfügig sinkende Tendenz für steigende registrierte Pull-Komponentenmengen auf. Ein ähnliches Ergebnis zeigt sich für die Betrachtung simulierter remoter Anfragen, wie Abbildung 8.14b zeigt. Hier sinken die gemittelten Ausführungszeiten ebenfalls stetig von ca. 17,4 Millisekunden auf ca. 15,8 Millisekunden und ca. 15,6 Millisekunden. Dabei ist zu beachten, dass beim Aufruf remoter Pull-Bausteine der Schritt der Identifikation durch einen gegebenen Aufrufkontext ebenfalls enthalten ist. Auf diese Weise lassen sich die geringfügig höheren durchschnittlichen Werte erklären.

Die hier gezeigten Ergebnisse zu Ausführungszeiten von Pull-Komponenten suggerieren sowohl für lokale als auch remote Zugriffe einen schwachen Zusammenhang zur Anzahl registrierter `NBIPullComponent`-Instanzen, insofern dass die gemessenen Zeiten mit größeren Mengen registrierter Komponenten sinken. Weitere Tests haben gezeigt, dass sowohl aufseiten der Framework-Implementierung als auch des OpenDaylight-Systems potenziell Caching-Mechanismen greifen, sodass später ausgeführte Tests im Vergleich zu früheren im Durchschnitt performanter bearbeitet werden können. Die in Abbildung 8.14 gezeigten Tests wurden daher ausschließlich einzeln ausgeführt – d. h. nach einem Test wurde die Framework-Testumgebung neu gestartet – und für jeden Parameterwechsel wurde der OpenDaylight-Controller ebenfalls neu gestartet. Ohne Neustart der Testumgebung und serieller Ausführung der Tests wurden dabei zum Vergleich bei simulierten lokalem Zugriff Ausführungszeiten von bis zu 11,6 Millisekunden erreicht. Da jedoch eine unabhängig von der Zahl registrierter `NBIPullComponent`-Instanzen konstante Ausführungszeit erwartbar ist, wird angenommen, dass weitere unerkannte Caching- oder Optimierungsmechanismen in der JVM die weiterhin sich verbessernde Performanz verursachen.

Parameter	Ausprägungen
Initial priorisierte Elemente am SBI	5%
Threadpoolmodell SBI	Cached
Threadpoolmodell NBI (Push)	Single
Priorisierungsmodell SBI	50%
Priorisierungsstufe NBI	Höchst

Tabelle 8.2: Optimale Konfiguration des Testsystems für Closed-Loop-Automatisierung nach Generierungsdauer pro Benachrichtigung.

8.2.3.2 Bewertung Closed-Loop-Automatisierung

Die Bewertung der Performanz eines Closed-Loop-Automatisierungszyklus erfordert die Berücksichtigung, dass die beschriebenen Testreihen nicht rein die Performanz der Frameworkbestandteile umfassen, sondern durch anwendungsfallspezifische Komponenten stark beeinflusst werden. Entsprechend besteht durchaus Optimierungspotenzial bei den Prozessen über die im Framework vorgegebenen Komponenten hinaus, wodurch einzelne Schritte darin – wie beispielsweise das zeitaufwändige Parsen im SBI, für das hier eine dafür übliche Java-Bibliothek genutzt wurde – auch wesentlich effizienter gestaltet werden kann. Die folgende Bewertung der Performanz des Closed-Loop-Automatisierungszyklus bezieht sich daher auf die vorher teilweise szenarienspezifisch ausgeprägten Testreihen. Des Weiteren ist zu beachten, dass die hier feingranular durchgeführte Art der Zeitmessung, insbesondere am SBI, ebenfalls Overhead induzieren, wodurch die tatsächlichen Ergebnisse tendenziell geringfügig besser sind. Dagegen wurde in der Implementierung keine Datenbankanbindung berücksichtigt, die je nach genutztem Datenbanksystem in einer Managementplattform zu einer Verschlechterung der Werte beitragen kann.

Die insgesamt erwartete Dauer t_{Ges} eines Closed-Loop-Automatisierungszyklus ist durch die Gleichung 8.1 gezeigt. Sie ergibt sich als Summe der Dauer des Benachrichtigungsprozess t_B, der Dauer der Auswertung durch eine Managementanwendung t_A und der Dauer der reaktiven Steuerung t_R. Bei einer remoten Managementanwendung kommt zudem die Netzlatenz der Benachrichtigung t_{N1} sowie der Steuerung t_{N2} hinzu, welche bei einer lokalen Managementanwendung wegfallen. Sinnvolle Werte dazu werden für die Evaluierung im Folgenden hergeleitet.

$$t_{Ges} = t_B + t_{N1} + t_A + t_R + t_{N2} \tag{8.1}$$

Da die Performanz des Benachrichtigungsprozesses, wie zuvor durch die verschiedenen Testreihen gezeigt werden konnte, wesentlich von einer optimierten Einstellung der Parameter abhängig ist, wird eine solche im Folgenden daraus für das Evaluierungssystem abgeleitet. Die Optimierung wird dabei entgegen einer möglichst kurzen Dauer zur Generierung einer Benachrichtigung durchgeführt und in Tabelle 8.2 zusammengefasst. Die Anzahl der verarbeiteten `SBIExtRawData`-Elemente spielt dabei wie gezeigt werden konnte keine große Rolle und wird nicht berücksichtigt. Genauso wird die Anzahl der Pusher-Threads sowie der künstlich erzeugte Delay bei Nachrichtenregistrierung und der Betriebsmodus vernachlässigt, da diese Parameter in realen Szenarien nicht steuerbar sind.

Der Anteil initial priorisierter `SBIExtRawData`-Objekte hat, wie gezeigt werden konnte, keinen wesentlichen Einfluss auf die Performanz. Tendenziell ist aber erkennbar, dass kleinere initialpriorisierte Anteile die Performanz sowohl von priorisierten als auch nicht-priorisierten Elementen geringfügig verbessern, wodurch hier ein Anteil von 5% als Optimum der Messreihen angenommen wird. Da der Grad an Gleichzeitigkeit, in der Elemente am SBI der Managementplattform erfasst werden, nicht komplett steuerbar ist, sondern von der Größe und Zusammen-

setzung der gemanagten IT-Infrastruktur abhängt, kann das *Single*-Threadpoolmodell am SBI nicht als optimale Einstellung gesehen werden, auch wenn sie vermutlich in vielen Szenarien ausreichend ist. Javas *Cached*-Threadpoolmodell hat im Vergleich zu dem *Fixed*-Modell bei normaler als auch hoher Last bessere Resultate erzielt und wird daher auf Basis der Testreihen als optimale Lösung für die Testumgebung angesehen. Ein ähnliches Bild ergab sich für das NBI-Threadpoolmodell, wobei hier im Vergleich das *Single*-Modell ebenfalls sehr gute Ergebnisse erzielte. Ein möglicher Grund, der am NBI für das *Single*-Modell spricht, ist die deutlich niedrige Last, die am NBI-Pushdienst verglichen mit dem SBI-Verarbeitungsprozess auftritt. Da das SBI hier als starker Filter dient – für die Testreihe aus Abschnitt 8.2.2.6 wurden aus 10.000 initialen `SBIExtRawData`-Instanzen lediglich im Durchschnitt ca. 748 Benachrichtigungen generiert – ist eine *Single*-Lösung als sehr guter Kompromiss zwischen Ressourcenverbrauch und Performanz anzusehen. Wie gezeigt werden konnte, tritt für das Priorisierungsmodell am SBI ab einer Einstellung von 50% keine weitere deutliche Verbesserung auf. Für geringere oder höhere Werte verbessert sich die Benachrichtigungsgenerierungszeit von priorisierten Elementen nicht wesentlich. Hinzu kommt, dass sich die Performanz bei geringeren Werten für nicht-priorisierte Elemente lediglich verschlechtert, ohne dass ein positiver Effekt daraus entsteht. Insofern wird das Optimum an dieser Stelle auf 50% geschätzt. Hinsichtlich des Priorisierungsmodells am NBI wird die höchste Performanz für Benachrichtigungen erkennbar, deren entsprechender `NBIPushComponent`-Instanz in der Prioritätenstufe *Höchst* liegen.

Für die ausgewählten optimalen Parameter liegt die Dauer zur Generierung einer Benachrichtigung unter Berücksichtigung der Standardkonfiguration (vgl. Abschnitt 8.2.2) der Testreihen im Mittel in einem Bereich zwischen ca. 2,4 und ca. 2,9 Millisekunden für initialpriorisierte `SBIExtRawData`-Elemente, welche für das Testsystem und Evaluierungsszenario als Optimum angesehen werden kann. Die Dauer des aktiven Eingreifens durch einen Pull-Zugriff auf der Plattform liegt, wie im vorherigen Abschnitt gezeigt wurde, in einem Bereich zwischen 11,6 Millisekunden (mit Caching-Effekten auf dem MO) und 17,4 Millisekunden. Die Zugriffsentscheidungen für einen Pull-Zugriff können, wie in Abschnitt 8.2.1 beschrieben wurde, äußerst effizient gestaltet werden und werden vernachlässigt. Die Auswertung der Benachrichtigung durch eine Managementanwendung umfasst ebenfalls das Parsen dieser und schließlich die eigentliche Auswertung und ist daher dem Benachrichtigungsprozess ähnlich. Als pessimistische Obergrenze werden daher für diesen Schritt die Werte des Benachrichtigungsprozesses verwendet, d. h., $t_A = t_B$. Da die Netzlatenzen sehr stark variieren, werden diese nicht berücksichtigt – d. h. es wird die Verarbeitung durch eine lokale Managementanwendung angenommen. Die mit diesen Werten geschätzte Dauer eines Closed-Loop-Automatisierungszyklus für das betrachtete Szenario beläuft sich daher zwischen ca. 16,4 bis ca. 23,2 Millisekunden.

8.3 Abgleich der Anforderungen

In diesem Abschnitt wird das Konzept unter Berücksichtigung der Implementierung und Evaluation anhand der zuvor im Kapitel 3 aufgestellten Anforderungen bewertet. Eine Übersicht der Bewertung ist in Tabelle 8.3 gezeigt. Die darin separat aufgeschlüsselten nicht-funktionalen Anforderungen werden in den folgenden Abschnitten im Kontext des Teilmodells beschrieben, dem sie zugeordnet werden können.

Das Bewertungsschema wird im Vergleich zur Bewertung der bestehenden Managementplattformen aus Abschnitt 4.6 geringfügig angepasst. Da sich die Bewertung auf einzelne Frameworks und keine fertige Managementplattform bezieht, wird zwischen den folgenden Erfüllungsgraden unterschieden:

- **Dunkelblau**: Die Erfüllung einer Anforderung ist von den Frameworks unterstützt und explizit vorgesehen.

Nicht-funktionale Anforderungen	
NF.1	Quasi-Echtzeitfähigkeit
NF.2	Nachvollziehbarkeit von Aktionen
NF.3	Komponentenagnostischer Zugriff
NF.4	Plattformunabhängigkeit
NF.5	Leichtgewichtigkeit
NF.6	Datenaktualität
NF.7	Logische Zentralisierung
NF.8	Agentenloses Management
NF.9	Skalierbarkeit
NF.10	Resilienz und Fehlertoleranz
NF.11	Verteilte Systemarchitektur
NF.12	Sparsamkeit der API-Kommunikation
NF.13	Sichere Kommunikation
NF.14	Mandantenfähigkeit
NF.15	Föderationsbeziehungen

Anforderungen an das Funktionsmodell	
F.1	Automatisierung der Kernprozesse
F.2	Adaptierbarkeit Managementfunktionen
F.3	Unterbrechungsfreier Betrieb
F.4	Selbstkonfiguration
F.5	Domänenverwaltung
F.6	Aufgabenbasierte Übersicht und Zugriff
F.7	Anwendungsbereich von MAPPs
F.8	Parametrisierung von MAPPs
F.9	Zustandsoperationen durch MAPPs
F.10	Priorisierter Informationsaustausch
F.11	Manipulationsschutz der Plattform

Anforderungen an das Informationsmodell	
I.1	Berücksichtigung Ressourcenabhängigkeiten
I.2	Geographische Ressourcenverteilung
I.3	Adressierbarkeit von MOs
I.4	Modellierbarkeit FSN-spezifischer MO-Typen
I.5	Normalisierung von MO-Funktionen
I.6	Abbildung von IT-Ressourcen-Besitz
I.7	MO-Register
I.8	Modellierbarkeit von Managementbeziehungen
I.9	Abhängigkeit Netze und Netzkomponenten
I.10	Berücksichtigung von Inter-MO-Abhängigkeiten
I.11	Modellierbarkeit Netzereignisse und -Zustände
I.12	Modellierbarkeit von Alarmen
I.13	Normalisierung von Netzinformationen
I.14	Abhängigkeit von Ereignissen zu MOs

Anforderungen an das Kommunikationsmodell	
K.1	Flexibilität der Architektur
K.2	Schnittstelle zu MAPPs (API)
K.3	Schnittstelle zur Infrastruktur (SBI)
K.4	Adaptierbarkeit Informationsaustauschmodells
K.5	Integrierbarkeit von Agenten
K.6	Flexibilität der Datenbankanbindung
K.7	Dritt-Managementplattformen
K.8	API: Erweiterbarkeit
K.9	API:Netzzugriff auf API
K.10	API: Netzunabhängigkeit
K.11	API: Push- und Pull-Mechanismen
K.12	API: Zustandsbehaftete Kommunikation
K.13	API: Bulk-Funktion
K.14	API: Sichere Kommunikation
K.15	API: Authentifizierung und Autorisierung API
K.16	SBI: Erweiterbarkeit Protokolle
K.17	SBI: Erweiterbarkeit Austauschformate
K.18	SBI: Push- und Pull-Mechanismen
K.19	SBI: Authentifizierung und Autorisierung
K.20	SBI: Sichere Kommunikation

Anforderungen an das Organisationsmodell	
O.1	Heterogene Managementkonzepte
O.2	Administrative Domänen
O.3	Zugriffssteuerung
O.4	Nutzer- und Funktionskennungen
O.5	Flexibilität
O.6	Zugriffsrechte Managementanwendungen
O.7	Funktionale Aufgabenorganisation
O.8	Netzweite Vorgaben
O.9	Organisatorische Übersicht
O.10	Domänenüberschneidung
O.11	Domänenspezifische Rollen und Aufgaben
O.12	Domänenübergreifende Aktivitäten
O.13	Zentrale / dezentrale Verwaltung von Domänen
O.14	Domänenspezifische Organisation

Anforderungen an eine frameworkspezifische Umsetzung	
U.1	Dokumentation
U.2	Unterstützung der Implementierung
U.3	Langlebigkeit des Frameworks
U.4	Erweiterbarkeit des Frameworks
U.5	Grad der Designfreiheit
U.6	Nutzerfreundlichkeit

Tabelle 8.3: Übersicht über die Erfüllung von Anforderungen des Konzepts dieser Arbeit (Dunkelblau: erfüllt und vorgesehen; Hellblau: erfüllt, aber abhängig vom Entwickler; Grau: nicht entscheidbar; Weiß: nicht erfüllt).

- **Hellblau**: Die Erfüllung einer Anforderung ist von den Frameworks unterstützt und vorgesehen, jedoch abhängig von der Umsetzung durch den Entwickler einer Managementplattform.
- **Grau**: Die Erfüllung einer Anforderung ist auf Basis des Konzepts, der Implementierung und Evaluation nicht entscheidbar.
- **Weiß**: Die Anforderung wird durch die Frameworks nicht unterstützt.

8.3.1 Anforderungen an das Funktionsmodell

In diesem Abschnitt werden die Anforderungen an das Funktionsmodell und diesbezüglich nicht-funktionale Anforderungen im Einzelnen bewertet.

8.3.1.1 Vollständig unterstützte Anforderungen

Die Kernprozesse des Managements, die Überwachung und Steuerung, werden im Konzept explizit unterstützt (vgl. Abschnitt 5.7.3, **F.1**). Wie in der quantitativen Evaluierung in Abschnitt 8.2 gezeigt werden konnte, kann das Konzept zur Automatisierung im niedrigeren

zweistelligen Millisekundenbereich in Quasi-Echtzeit umgesetzt werden (**NF.1**). Weitere wichtige funktionale Eigenschaften des Konzepts konnten vollständig erfüllt werden: Durch eine Java-basierte Implementierung (vgl. Abschnitt 6.2.1) sind die Konzepte plattformunabhängig einsetzbar (**NF.4**). Die Frameworks sind beispielhaft im Kontext der Entwicklungsumgebung von ONOS implementiert, sind jedoch nicht damit verwoben und einfach portierbar. Die Frameworks wurden nicht vollständig, jedoch zum großen Teil implementiert und weisen eine Quelltextgröße von weniger als 4 Megabyte auf, sodass sie äußerst leichtgewichtig sind (**NF.5**).

Auch unter Aspekten der Erweiterbarkeit der Funktionalität der Managementplattform werden die meisten Anforderungen vollständig erfüllt. Funktionale Managementbereiche sind an das jeweilige Szenario vollständig adaptierbar (vgl. Abschnitt 5.7.1, **F.2**) und legen die Grundlage für ein aufgabenbasiertes Netzmanagement (**F.6**). Mit den Frameworks für Plattformfunktionen, das in Abschnitt 5.7.2 beschrieben wird, kann die Funktionalität einfach erweitert werden. Darauf basiert auch eine erweiterbare Möglichkeit für administrative Funktionen wie das Domänenmanagement (**F.5**), das durch die Domänen-Kreuzorganisationstabelle aus Abschnitt 5.5.2.3 explizit unterstützt wird. Das darauf basierende Framework für MOC-Funktionen aus Abschnitt 5.4.3 bietet im Konzept für einen komponentenagnostischen Zugriff auf Funktionen von MOs (**NF.3**). Sie erlauben auch einen Zugriff auf die Datenmodelle über das NBI (**F.9**) durch Managementanwendungen. Die Steuerung des Zugriffs auf Managementanwendungen wird in Abschnitt 5.7.4 beschrieben und erlaubt ihre gleichartige Behandlung analog zu herkömmlichen MOs (**F.7**). Die Funktionalität des priorisierten Informationsaustauschs (**F.8**) wurde inhärent in Abschnitt 5.6.2.8 vorgesehen. Die Parametrisierung von Managementanwendungen (**F.8**) wird durch das Framework für einen mandantenfähigen Zugriff aus Abschnitt 5.5.5 durch eine sehr feingranulare Betrachtung von Informationselementen unterstützt. Managementanwendungen werden entsprechend durch den zugreifenden Nutzer parametrisiert und die Managementplattform gibt der Anwendung nur Informationen preis, die auch durch den Nutzer einsehbar sind.

8.3.1.2 Teilweise unterstützte Anforderungen

Teilweise unterstützte Anforderungen sind insbesondere von der endgültigen Umsetzung durch den Entwickler abhängig. Dazu zählen die Funktionen des NBI für einen *a)* priorisierten Informationsaustausch (**F.10**) und *b)* die Nachvollziehbarkeit von Aktionen (**NF.2**). Beide Aspekte können durch die geeignete Implementierung von Plattformfunktionen umgesetzt werden. Die Nachvollziehbarkeit kann durch Protokollierung einer Plattformfunktion bei ihrem Aufruf umgesetzt werden.

Ein Konzept für den unterbrechungsfreien Betrieb (**F.3**) wird nur teilweise im Konzept besprochen. Für diesen Aspekt ist insbesondere das Framework für Managementobjektklassen und MOC-Funktionen explizit ausgerichtet, die anderen Frameworks des Informations- und Organisationsmodells hingegen nicht. Durch eine geeignete architekturelle Umsetzung (z. B. als Microservices) und Parallelbetrieb mehrerer Versionen einer Managementplattform ist jedoch auch ein unterbrechungsfreier Betrieb für die in diesem Konzept entwickelten Frameworks möglich.

8.3.1.3 Nicht erfüllte Anforderungen

Nicht erfüllte Anforderungen sind insbesondere der Manipulationsschutz (**F.11**) und die Selbstkonfiguration (**F.4**) der Managementplattform. Beide sind vielmehr auf Ebene einer vollimplementierten Managementplattform zu berücksichtigen und im Kontext objektorientierter Frameworks nicht effektiv lösbar.

8.3.2 Anforderungen an das Informationsmodell

Im Kontext des Informationsmodells konnten alle Anforderungen erfüllt werden. Das Framework für Managementobjektklassen aus Abschnitt 5.4.1 erlaubt nicht nur die Modellierung dieser, sondern unterstützt sie auch durch eine vorgegebene Vererbungsstruktur (**I.4**, **I.3**). Durch das in Abschnitt 5.4.3 beschriebene Framework können MOC-Funktionen dynamisch modelliert und strukturiert nach Typ assoziiert werden. Ihre Normalisierung findet über Vererbungsprinzipien statt (**I.5**). Abhängigkeiten zwischen MOs werden über das Framework für Managementabhängigkeiten in Abschnitt 5.4.2 modelliert (**I.8**), das auch die Vernetzung der Komponenten abbildet (**I.9**). Die Managementbeziehungen können abgefragt und als Kriterium im Netzmanagement genutzt werden (**I.1**, **I.10**). Aspekte der geographischen Verteilung und von Ressourcenbesitz werden dagegen als Teil des Organisationsmodells und im Kontext der Datenzentren-Struktur in FSNs beschrieben (**I.2**, **I.6**). Ein Register für MOs wurde zwar nicht im Konzept, aber darauf aufbauend in der Implementierung der Frameworks umgesetzt (vgl. Abschnitt 6.3.1, **I.7**).

Das Framework für Netzinformation in Abschnitt 5.4.4 erlaubt ihre Modellierung (**I.11**) und bringt sie in unterschiedlicher Weise mit MOs in Verbindung: Zum einen als Quelle einer Netzinformation und zum anderen als damit verbundene Infrastrukturkomponente (**I.14**). Im Verarbeitungsprozess von Daten, die am SBI der Managementplattform gesammelt und verarbeitet werden, dienen Netzinformationen als grundlegende Normalisierungsstruktur vor der eigentlichen Aktualisierung von Datenmodellen (**I.13**). Alarme sind in ähnlicher Weise modellierbar, wie in Abschnitt 5.4.5 beschrieben wird (**I.12**).

Die Erfüllung der Anforderung der Datenaktualität fußt auf der Erfüllung der Umsetzung und Quasi-Echtzeitfähigkeit der Kernprozesse des Managements (vgl. Abschnitt 8.3.1.1, **NF.6**).

8.3.3 Anforderungen an das Kommunikationsmodell

Die Anforderungen an das Kommunikationsmodell werden zum Großteil von den Frameworks erfüllt. Einige Anforderungen werden vom Konzept teilweise erfüllt und sind abhängig von der jeweils konkreten Umsetzung. Letzteres liegt dabei nicht an Unzulänglichkeiten des Konzepts, sondern an Abhängigkeiten zu anderen Komponenten wie MOs und ihren Schnittstellen.

8.3.3.1 Vollständig unterstützte Anforderungen

Unter architekturellen Gesichtspunkten als Teil des SBI werden durch das Konzept die meisten Anforderungen erfüllt. In Abschnitt 5.6.2.6 werden Konzepte zur logischen Zentralisierung und der Skalierbarkeit der Managementplattform vorgestellt (**NF.7**). Gleichzeitig ist die Managementplattform inhärent als verteiltes System konzipiert und erlaubt eine skalierbare Lösung (**NF.11**, **NF.9**). Über MOC-Funktionen kann die Lösung zudem komplett agentenlos funktionieren, abhängig von der gemanagten IT-Infrastruktur können Agenten aber auch eingesetzt werden (**NF.8**, **K.5**). Die Frameworks schränken die Managementplattform nicht in ihrer Flexibilität ein (**K.1**).

Die Konzepte sehen ein detailliert ausgearbeitetes SBI vor (siehe Abschnitt 5.6.1, **K.3**), das eine flexible Erweiterbarkeit in Hinsicht genutzter Protokolle und Kommunikationsmechanismen (vgl. Abschnitt 5.6.1.3, **K.16**, **K.18**) und Datenformate (vgl. Abschnitt 5.6.1.1, **K.17**) vorsieht.

Auch das NBI wird über Frameworkkonzepte geeignet beschrieben (**K.2**). Wie die Abschnitte 5.6.2.1 bis 5.6.2.4 beschreiben, ist die über das NBI zugreifbare API flexibel erweiterbar (**K.8**) und von lokalen Managementanwendungen über eine objektorientierte Schnittstelle (**K.10**) oder über beliebige Netzprotokolle von remote nutzbar (**K.9**). Beide Arten der Anbindungen unterstützen Push- und Pull-Kommunikation (**K.11**).

Die in vielen bestehenden Managementplattformen als West- oder Eastbound-Interface bezeichnete Schnittstelle wird hingegen über das NBI abgedeckt (**K.7**). Eine Trennung zu Managementanwendungen oder Nutzern erfolgt durch den in Abschnitt 5.6.2.7 beschriebenen Privilegienbaustein. Des Weiteren wird in diesem Konzept die Schnittstelle zu Datenbanken beschrieben. Durch die Abstrahierung von Speicheroperationen (siehe Abschnitt 5.6.3.1) kann eine für den jeweiligen Anwendungsfall geeignete Datenbanklösung angebunden werden (**K.6**).

8.3.3.2 Teilweise unterstützte Anforderungen

Die Resilienz (**NF.10**) als architektureller Aspekt einer Managementplattform ist wesentlich von der Umsetzung abhängig. Die Frameworks erlauben eine Implementierung als Monolith, genauso wie als verteiltes System; letzteres wird jedoch explizit unterstützt. Aus dem gleichen Grund ist eine sichere Kommunikation (**NF.13**) sowie die Umsetzung von Informationsaustauschmodellen (**K.4**) von der jeweiligen Umsetzung abhängig, da sie nicht durch die Frameworks forciert werden.

Aufseiten des SBI muss sich eine Managementplattform vor allem nach den von MOs bereitgestellten Schnittstellen richten können. Die in dieser Arbeit beschriebenen Frameworks erlauben eine sichere Anbindung von MOs (**K.20**, **K.19**), müssen jedoch auch eine Kommunikation mit von MOs bereitgestellten unsicheren Schnittstellen ermöglichen. In diesem Fall ist beispielsweise eine agentenbasierte Lösung oder die Absicherung der Kommunikation über sichere Kanäle in Erwägung zu ziehen.

Auch das NBI bietet ausreichend Freiheiten und erlaubt, forciert aber keine sichere Anbindung von Managementanwendungen (**K.14**, **K.15**). Genauso sind API-Funktionen abhängig von der jeweiligen Umsetzung: Eine sparsame Kommunikation (**NF.12**) ist abhängig vom gewählten Datenaustauschformat und dem eingesetzten Protokoll. Eine zustandsbehaftete Kommunikation (**K.12**) und Bulk-Operationen (**K.13**) hängen von der konkreten Implementierung von MOC- und Plattformfunktionen ab.

8.3.4 Anforderungen an das Organisationsmodell

Das Konzept erfüllt beinahe alle Anforderungen an das Organisationsmodell. Lediglich eine Anforderung konnte nur teilweise erfüllt werden.

8.3.4.1 Vollständig unterstützte Anforderungen

Das Konzept zur Mandantenfähigkeit ist grundlegend und anders als bei bestehenden Managementplattformen sehr feingranular gestaltet (**NF.14**). Die Möglichkeit dazu bietet das in Abschnitt 5.5.5 beschriebene System zur Zugriffssteuerung (**O.3**), das auch den Zugriff auf Managementanwendungen als MOs behandeln kann (**O.6**). Die Organisation der Föderation selbst wird auf Basis einer funktionalen Aufgabenorganisation (siehe Abschnitt 5.7.1, **O.7**) und der Einbeziehung von Föderationsbeziehungen (wie Vertrauensbeziehungen) gestaltet, wie in Abschnitt 5.5.1.2 beschrieben wurde (**NF.15**).

Das konzeptionelle Kernelement zur Strukturierung der Organisation sind administrative Domänen (siehe Abschnitt 5.5.2, **O.2**). Über das in Abschnitt 5.5.4.2 beschriebene Konzept zur individuellen Nutzung von Rollen (**O.11**) und ihrer Zuweisung von Zuständigkeiten innerhalb einer Domäne werden heterogene Managementkonzepte der Parteien in derselben Föderation unterstützt (**O.1**). Netzweite Vorgaben werden dahingehend umgesetzt, dass grundlegende Managementfunktionen sowie -Aufgabenbereiche zentral in der Föderation festgelegt werden (**O.8**). Die Umsetzung der Zuständigkeiten dafür ist dann flexibel innerhalb der jeweiligen Domäne gestaltbar (**O.5**) oder kann durch eine föderationsweite Vorgabe modelliert werden. Die föderationsweite Zuständigkeitszuweisung von Aufgaben zu Rollen ist dann gültig, wenn

sie nicht auf Ebene einer Domäne überschrieben wird. Die jeweiligen Aufgaben werden dementsprechend von Nutzern oder Funktionskennungen belegt (**O.4**, **O.14**). In Abschnitt 5.5.2.3 wird explizit eine Möglichkeit zur feingranularen Behandlung von Domänenüberschneidungen (**O.10**) auf MO-Ebene beschrieben. Die Behandlung erfolgt durch Neuzuweisung von Zuständigkeiten für sich überschneidende MOs in den jeweiligen Domänen. Die Ermittlung der Zuständigkeiten und Aufgabenübersicht kann über diese Zuweisungen ermittelt werden (**O.9**).

Der Zugriff auf administrative Funktionen wird im Konzept über Plattformfunktionen umgesetzt und kann wie herkömmliche Zugriffe über das NBI bereitgestellt werden. Über den Authentifizierungsbaustein aus Abschnitt 5.6.4.1 kann der Zugriff auf diese Funktionen festgelegt werden und beispielsweise auch eine zentrale oder dezentrale Verwaltung forciert werden (**O.13**).

8.3.4.2 Teilweise unterstützte Anforderungen

Die im Konzept beschriebenen Frameworks des Organisationsmodells und des Informationsmodells erlauben die Ausnutzung von Abhängigkeiten von MOs oder organisatorischen Abhängigkeiten. Domänenübergreifende Aktivitäten (**O.12**) können von diesen abhängig gemacht werden: Beispielsweise, ob ein Hypervisor auf Ebene 0 abgeschaltet werden soll, selbst wenn darauf noch VMs auf höheren Virtualisierungsebenen aktiv sind. Diese Funktionalität ist jedoch von der Umsetzung durch den eigentlichen Entwickler einer Managementplattform auf Basis der Frameworks notwendig und wurde nicht im Rahmen dieser Arbeit implementiert.

8.3.5 Anforderungen an die Umsetzung

Die Anforderungen an die Umsetzung wurden größtenteils vollständig erfüllt. Lediglich die Erfüllung der zwei Anforderungen der Langlebigkeit und Nutzerfreundlichkeit der Frameworks kann nicht auf Basis der Evaluierung entschieden werden.

8.3.5.1 Vollständig unterstützte Anforderungen

Das Konzept der Frameworks bietet bereits eine detaillierte Dokumentation zum Zweck und der Umsetzung der Frameworks. Eine feingranularere Dokumentation wurde darüber hinaus in der Implementierung der Frameworks umgesetzt und kann als JavaDoc aus den Klassen generiert werden (**U.1**). Die Implementierung wird soweit möglich durch eine explizite Typisierung und einige bereits vorgefertigte Klassen und Beispiele unterstützt. Insbesondere die in Kapitel 7 durchgeführte Anwendung der Frameworks bietet eine Vorlage ihrer Nutzung (**U.2**). Die Frameworks bieten jedoch an vielen Stellen ausreichend Freiheiten zur Erweiterung (**U.4**, **U.5**).

8.3.5.2 Nicht-entscheidbare Anforderungen

Die Anforderungen der Langlebigkeit (**U.3**) und Nutzerfreundlichkeit (**U.6**) der Frameworks sind nicht im Rahmen dieser Arbeit entscheidbar. Für ersteres ist die Beobachtung der Entwicklung von Technologien in SNs und darin zukünftig eingesetzter Netzkomponenten notwendig. Die Entscheidung der Nutzerfreundlichkeit erfordert hingegen die Anwendung der Frameworks auf die Entwicklung von Managementplattformen durch unabhängige Entwickler auf bestimmte Anwendungsfälle von FSNs. Die Implementierung einer Managementplattform selbst ist jedoch nicht Fokus dieser Arbeit.

8.3.6 Bewertung der Erfüllung der Anforderungen

Wie in den letzten Abschnitten erläutert wurde, können die Frameworks die Anforderungen an Managementplattformen in FSNs beinahe vollständig erfüllen. Aktuell **nicht erfüllbare Anforderungen** stellen die Selbstkonfiguration (**F.4**) und Schutz vor Manipulation (**F.11**) dar. Beide

Anforderungen weisen jedoch eine generell geringe Gewichtung für ein erfolgreiches Netzmanagement auf. Ihre Nichterfüllung wirkt sich zudem nicht negativ auf die Beantwortung der Forschungsfragen aus.

Lediglich teilweise erfüllte Anforderungen von **essenzieller Bedeutung** für das Netzmanagement in FSNs betreffen größtenteils eine sichere Kommunikation am NBI (**K.14**, **K.15**), SBI (**K.19**, **K.20**) und innerhalb der Managementplattform-Architektur (**NF.13**). Da die Frameworks jedoch als voneinander unabhängig betrachtet werden, und nur einzelne Frameworks in Managementplattformen integrierbar sind, ist die Kommunikation außerhalb des Einflussbereichs der Frameworks und vielmehr davon abhängig, ob die Basis-Managementplattform eine sichere Kommunikation erlaubt. Auch muss sich eine Managementplattform besonders am SBI nach den Schnittstellen von gemanagten Systemen richten und mit diesen kommunizieren können. Aus beiden Gründen kann eine Absicherung der Schnittstellen einer Managementplattform über Frameworks kaum über den Status einer teilweisen Erfüllung hinausgehen. Daneben ist die Anforderung der domänenübergreifenden Aktivitäten (**O.12**) essenziell, jedoch hier nur teilweise erfüllt. Die Frameworks unterstützen jedoch derartige Funktionalität, die durch Entwickler entsprechend anwendungsfallabhängig ergänzt werden muss und kann. Schließlich kann auch die Resilienz und Fehlertoleranz (**NF.10**) nicht durch die Frameworks sichergestellt werden. Entsprechend muss der Entwickler auf Redundanz von Systemen und Daten achten. Die Nutzbarkeit der Frameworks ist durch die lediglich teilweise Erfüllung dieser Anforderungen entsprechend nicht eingeschränkt.

Kapitel 9

Fazit

Inhalt

9.1 Beantwortung der Forschungsfragen 305
9.1.1 Allgemeine Fragestellungen 305
9.1.2 Informationsmodell 306
9.1.3 Funktionsmodell 307
9.1.4 Organisationsmodell 308
9.1.5 Kommunikationsmodell 309
9.2 Zusammenfassung dieser Arbeit 309
9.3 Ausblick auf weiterführende Forschungsfragen 312

In diesem Kapitel werden im folgenden Abschnitt 9.1 zunächst die zu Beginn dieser Arbeit in Abschnitt 1.3 formulierten Forschungsfragen dieser Dissertation aufgegriffen und auf Basis der Erkenntnisse dieser Arbeit beantwortet. Danach fasst Abschnitt 9.2 diese Arbeit zusammen und hebt daraus gewonnene Erkenntnisse hervor. Abschnitt 9.3 geht schließlich auf weiterführende Forschungsfragen ein, die an diese Arbeit angeschlossen werden können.

9.1 Beantwortung der Forschungsfragen

Die Forschungsfragen wurden gemäß ihrer Zuordenbarkeit zu einem der vier Teilmodelle des Managements gruppiert. Darüber hinaus wurden allgemeine Fragestellungen beantwortet.

9.1.1 Allgemeine Fragestellungen

Die Beantwortung der allgemeinen Forschungsfragen ist zum grundlegenden Verständnis zur Erfüllung der Aufgabe *Netzmanagement* in FSNs notwendig. Ihre Beantwortung erfolgte daher bereits vergleichsweise früh in dieser Arbeit.

F1: Welche Eigenschaften im Betrieb unterscheiden (F)SNs von HN? Die Grundlage zur Beantwortung der Forschungsfrage sind die angewendeten Paradigmen in FSNs: SDN postuliert eine zentrale Steuerbarkeit. Virtualisierung und NFV postulieren dagegen den grundsätzlichen Einsatz von virtualisierten Netzkomponenten (vgl. Abschnitt 2.1). Daraus wurden in Abschnitt 3.1 weitere Eigenschaften und Betriebscharakteristika abgeleitet. Kernaspekte, die den Betrieb von SNs zu HNs unterscheiden, sind eine grundsätzlich **unmittelbare** und **remote Steuerung** hochflexibler Netze. In einer Föderation gibt es darüber hinaus organisatorische Unterschiede wie **unabhängige Parteien** mit **unterschiedlichen Managementkonzepten**.

F2: Wie wirken sich diese Eigenschaften auf Managementplattformen in FSNs aus? Managementplattformen müssen Betriebscharakteristika in FSNs technisch in Modell und Funktion unterstützen. Die Schwerpunkte der Modellierung liegen im Informationsmodell zur Abbildung der Netze, Komponenten, Informationen und Managementbeziehungen darin. Im Organisationsmodell liegen sie in der Abbildung der Föderation und von Strukturen zur Koordination des Netzmanagements. Aufgrund einer starken Heterogenität in FSNs können Managementplattformen kaum mehr als fertige Lösungen betrachtet werden, sondern müssen **anwendungsfallspezifisch adaptierbar** sein. Das SBI stellt im Kommunikationsmodell in dieser Hinsicht die größte Herausforderung dar, da es mit vielen nicht-standardisierten Schnittstellen von gemanagten Komponenten umgehen muss. Dies begründet die Notwendigkeit zur Flexibilität des Funktionsmodells, das Funktionen von und für die gemanagte IT-Infrastruktur modellieren und normalisieren können muss. Alle Ausprägungen werden in der Anforderungsanalyse in den Abschnitten 3.6.3 bis 3.6.8 zusammengefasst.

F3: Welche Funktionen und Eigenschaften müssen Managementplattformen in FSNs besitzen? Welche Aspekte davon müssen flexibel anwendungsfallspezifisch anpassbar, welche fix sein? Notwendige Funktionen einer Managementplattform gehen aus der **Anforderungsanalyse** hervor, die in Abschnitt 3.6.4 zusammengefasst wurde. Eigenschaften von Managementplattformen in FSNs wurden im Rahmen der nicht-funktionalen Anforderungen in Abschnitt 3.6.3 zusammengefasst. Die **Hot-Spot-Analyse** in Abschnitt 3.6.2 beschreibt für alle vier Teilmodelle flexibel anpassbare Komponenten. Die Funktionalität einer Managementplattform in FSNs hängt vor allem am Kernprozess der **Überwachung** und der **Steuerung** von FSNs. Wie im Konzept gezeigt werden kann, sind darin zu verarbeitende Elemente hochflexibel und können weitestgehend über Schnittstellendefinitionen unterstützt werden. Die Koordinierungslogik des Prozesses kann hingegen weitestgehend fix implementiert sein.

F4: In welche Teilframeworks lassen sich die vier Teilmodelle einer Managementarchitektur in FSNs gruppieren und welche Schnittstellen gibt es zwischen ihnen? Die Teilframeworks basieren vor allem auf der Hot-Spot-Analyse in Abschnitt 3.6.2. Das **Informationsmodell** lässt sich in fünf Frameworks unterteilen: Für Managementobjektklassen, Managementbeziehungen, MOC-Funktionen, Netzinformationen und für Alarme. Das **Organisationsmodell** lässt sich in weitere fünf Frameworks unterteilten: Zur Modellierung von Föderationen und Föderationsbeziehungen, für administrative Domänen, für Nutzer und Rollen, für Managementaufgaben und letztlich für die Zugriffskoordination und Mandantenfähigkeit. Das **Kommunikationsmodell** kann gemäß seiner Schnittstellen in drei Frameworks, für das SBI, das NBI und die Datenbankanbindung unterteilt werden. Darüber hinaus ist ein Hilfsframework für zentrale Komponenten notwendig. Das **Funktionsmodell** kann in zwei weitere Frameworks für Managementbereiche und Managementplattformfunktionalität unterteilt werden. Die Kernprozessautomatisierung kann auf Basis der zuvor genannten Frameworks implementiert werden.

9.1.2 Informationsmodell

Die Fragen des Informationsmodells sind hochspezifisch für die Umsetzung von Modellen der gemanagten IT-Infrastruktur. Sie wurden daher erst im Konzept in Kapitel 5 geklärt.

F5: Wie wirkt sich die Dynamik in FSNs auf das Informationsmodell aus? Die Dynamik in FSNs bezieht sich zum einen auf eine hohe Heterogenität an Netzkomponenten. Diese wirkt sich insofern auf das Informationsmodell aus, dass Elemente des Informationsmodells **flexibel** an den jeweiligen Anwendungsfall eines FSN **adaptierbar** sein müssen. Die Unterstützung von Entwicklern bei der Modellierung ist daher ein zentraler Aspekt. Zum anderen bezieht sich die Dynamik auf die remote unmittelbare Steuerung von FSNs. Sie begründet die Anforderung nach **Datenaktualität** (**NF.6**) und **Quasi-Echtzeitfähigkeit** der Kernprozesse (**NF.1**) in einer Managementplattform.

F6: Welche Managementinformationen sind in FSNs zu berücksichtigen und wie hängen sie zusammen? Wie im Informationsmodell in Abschnitt 5.4 beschrieben wird, stellen Netzkomponentenklassen bzw. ***MOCs*** in dieser Arbeit die zentrale Managementinformation dar. Sie können verschiedene Komponentenausprägungen aufweisen, die durch *MOC-Klassifikationen* (**MOCKs**) modelliert werden. MOCKs lassen sich in die in Abschnitt 5.4.1.1 beschriebenen generischen Elemente mit jeweils komponentenspezifischen **Attributen** untergliedern und müssen davon ausgehend anwendungsfallspezifisch spezialisiert werden. Zwischen MOs müssen FSN-spezifische **Managementbeziehungen** (vgl. Abschnitt 5.4.2), wie paradigmenspezifische Steuerungsabhängigkeiten, modelliert werden können. Informationen, die mit MOs in Verbindung stehen, aber keine Attribute darstellen, werden als ***Netzinformationen*** bezeichnet. Ein MO kann eine Quelle einer Netzinformation sein oder damit in Verbindung stehen. Eine Netzinformation kann aber auch die Zustandsänderung eines MO bewirken, wie es im SBI-Prozess zur Normalisierung von Informationen der gemanagten IT-Infrastruktur in Abschnitt 5.6.1.2 ausgenutzt wird. **Alarme** als letzte wesentliche Gruppe werden stets von der Managementplattform generiert. Vermeintliche, von Komponenten der gemanagten IT-Infrastruktur direkt generierte Alarme, werden dagegen zunächst wie Netzinformationen behandelt.

F7: Wie lassen sich diese Informationen einfach modellieren und wie können neue, bisher unberücksichtigte Informationen in Verarbeitungsprozesse integriert werden? Für die Modellierung wurden unterschiedliche Konzepte verwendet, die meist auf **Vererbung**, **Assoziation** und **Klassifikation** aus der objektorientierten Programmierung basieren. Die Vorgabe einer **Vererbungshierarchie** dient vor allem zur Normalisierung (z. B. bei der Modellierung von MOC-Klassifikationen in Abschnitt 5.4.1.1) von Attributen und Funktionen und als unterstützende Anleitung für Entwickler. **Assoziationen** verknüpfen Elemente aus unterschiedlichen Frameworks miteinander. Die so einerseits für sich stehenden Frameworks können so auch als ganzheitlicher Ansatz verwendet werden. Die **Klassifikation** als Spezialisierung von Assoziationen wird innerhalb von Frameworks eingesetzt (z. B. die Klassifikation von MOCs mit MOCKs oder von MOCKs mit MOC-Funktionen) und stützt die Flexibilität des Modellierungsansatzes. Durch eine objektorientierte Umsetzung und die Definition von **Schnittstellenklassen** können unberücksichtigte Informationen in Verarbeitungsprozessen einheitlich gehandhabt werden.

F8: Wie können Informationen in heterogenen FSNs normalisiert und vergleichbar gemacht werden? Elemente, deren Normalisierung notwendig ist, sind **MOCs** in ihren Attributen und Funktionen, sowie Netzinformationen. Für die Normalisierung von MOCs wird im Konzept dieser Arbeit eine **Vererbungshierarchie** für Netzfunktionen (MOCKs) genutzt. Allgemeinere MOC-Attribute und -Funktionen können so wiederverwendet werden. Die Umsetzung letzterer wird getrennt von der **Funktionsdefinition** in **Treibern** umgesetzt, sodass eine MOC Funktionen erben kann, bei Bedarf die eigentliche Umsetzung der MOC-Funktion jedoch auch adaptiert werden kann. Auf diese Weise können MOC-Funktionen gleichartig genutzt werden, da ihre **Parameter und Rückgabe** einheitlich definiert werden. Auch wird in dem Konzept dieser Arbeit die Normalisierung von über das SBI empfangener Daten vorgesehen. Dazu werden **Netzinformationen** als generellste Art von Informationselementen genutzt.

9.1.3 Funktionsmodell

Die Fragen des Funktionsmodells können größtenteils erst im Konzept in Kapitel 5 geklärt werden. Lediglich Frage F10 lässt sich bereits in den beschriebenen Szenarien in Kapitel 3 anhand von Beispielen teilweise beantworten.

F9: Wie kann Netzmanagement in a priori unbekannten Umgebungen unterstützt werden? Die technische Unterstützung des Netzmanagements unbekannter Netzumgebungen wird durch eine hohe **Flexibilität** in der Beschreibung unbekannter Gegebenheiten realisiert. Diese umfassen die gemanagte Umgebung (Informationsmodell), die Organisation und die

Schnittstellen der Managementplattform zur gemanagten IT-Infrastruktur (SBI des Kommunikationsmodells). Über das Funktionsmodell können zudem anwendungsfallspezifisch notwendige Plattformfunktionen beschrieben werden. Die Festlegung von **Schnittstellen** zwischen Framework-Elementen erlaubt ihre nahtlose Integration in die Prozesse der Überwachung und Steuerung.

F10: Welchen Einfluss hat der Übergang zu FSNs auf Managementfunktionen, Funktionsabhängigkeiten und Anforderungen an Managementplattformen selbst? Wie in den Szenarien (vgl. Abschnitte 3.3.2.5, 3.4.6 und 3.5.2.1) gezeigt wurde, können Managementfunktion **unterschiedlich interpretiert** werden. Auch sind nicht immer alle Managementfunktionen notwendig (beispielsweise das Abrechnungsmanagement in Szenario 1). Im Konzept dieser Arbeit stellen Managementfunktionen vor allem allgemeine **Strukturierungselemente** dar, von denen feingranulare Aufgabenbereiche und Aufgaben im Netzmanagement abgeleitet werden. Wie in Abschnitt 5.7.1.2 beschrieben, können Managementfunktionen dann von **disjunkten** oder sich **überschneidenden** Aufgabenbereichen erfüllt werden. Der Einfluss auf Anforderungen an Managementplattformen wirkt sich daher auf die Notwendigkeit aus, Managementfunktionen anwendungsfallspezifisch **modellieren** zu können.

F11: Wie kann der Funktionsumfang einer Managementplattform dynamisch erweitert werden? Für die Erweiterung des Funktionsumfangs einer Managementplattform wird im Konzept ein eigenes Framework in Abschnitt 5.7.2 beschrieben. Dabei wird wie bei MOC-Funktionen zwischen der **Funktionsdefinition** als Schnittstelle, die Parameter und Rückgabe einer Funktion beschreibt, sowie ihrer eigentlichen **Implementierung** unterschieden. Im Gegensatz zu MOC-Funktionen, für die für unterschiedliche MOC-Klassifikationen unterschiedliche Treiber existieren können, gibt es jedoch für jede Plattformfunktion genau eine Implementierung. Im Konzept wird zudem zwischen **internen** und über die **API** bereitgestellte Funktionen unterschieden. Eine Plattformfunktion kann, muss aber nicht für Managementanwendungen zugreifbar sein.

9.1.4 Organisationsmodell

Die Fragestellungen des Organisationsmodells werden größtenteils im Konzept dieser Arbeit geklärt.

F12: Wie ist die Föderations- und Organisationsstruktur in FSNs grundlegend aufgebaut und modellierbar? Die Struktur einer Föderation wird in den Szenarien beispielhaft beschrieben und in Abschnitt 5.5.1 modelliert. Die Organisationsstruktur wird in dieser Arbeit insbesondere durch die Definition von **Funktionsbereichen** (vgl. Abschnitt 5.7.1), diese wiederum durch **Aufgabenbereiche** und **Aufgaben** zentral für die gesamte Föderation vorgegeben (siehe Abschnitt 5.5.4). Aufgaben werden wiederum mit einer Rolle verknüpft. Die eigentliche Umsetzung des Netzmanagements passiert in **administrativen Domänen**, innerhalb derer Nutzer im Kontext einer Rolle für eine bestimmte Aufgabe verantwortlich sind. Diese Verantwortlichkeit wirkt sich neben anderen Kriterien auf die Zugriffsprivilegien eines Nutzers auf Plattformfunktionen aus (siehe Abschnitt 5.5.5).

F13: Wie können Zugriffsberechtigungen auf Informationen in FSNs geeignet realisiert werden? Abschnitt 5.5.5 beschreibt die Antwort auf diese Frage. Darin werden nicht nur feingranulare **Informationselemente** definiert, deren Zugriff gesteuert wird, sondern insbesondere auch FSN-spezifische **Kriterien** des Zugriffs: Föderationsbeziehungen, Domänenzugehörigkeit und Aufgabenzuweisungen sowie Domänenbeziehungen. Zur Unterstützung unterschiedlicher Anwendungsfälle sind diese Kriterien in einem als Framework realisierten **Entscheidungsbaum** benutzerdefiniert anordenbar.

F14: Wie können unterschiedliche Organisationsprinzipien der Föderationspartner in einer Plattform berücksichtigt werden? Organisationsprinzipien des Netzmanagements werden endgültig innerhalb jeder **administrativen Domäne** festgelegt. Für jede Domäne können Zuständigkeiten von Rollen für Aufgaben neu festgelegt und föderationsweite Vorgaben dazu lokal *überschrieben* werden (vgl. Abschnitt 5.5.4.2).

9.1.5 Kommunikationsmodell

Die Beantwortung der Forschungsfragen des Kommunikationsmodells basiert insbesondere auf den Szenarien (F15 und F16). Frage F17 hingegen wird erst im Konzept geklärt.

F15: Welche Netzkomponenten und Richtungen müssen bei der Kommunikation berücksichtigt werden? Die eigentlichen technischen Akteure mit einer Managementplattform wurden bereits in Abschnitt 3.1.2.1 beschrieben. Am **SBI** stellen aus dem SDN-Paradigma insbesondere SDN-Controller und SDN-Anwendungen gemanagte Netzkomponenten dar. Aus dem NFV-Paradigma darüber hinaus VIMs, VNFMs, und NFVOs. Am **NBI** werden allgemein Managementanwendungen als Akteure betrachtet, die entweder über das Netz oder eine lokale Programmierschnittstelle mit der Managementplattform kommunizieren. Nicht-FSN-spezifische Systeme (z. B. andere Managementplattformen, Ticketsysteme oder NIDS) müssen gemäß ihrem Verwendungszweck am SBI als MO oder am NBI als Managementanwendung berücksichtigt werden.

F16: Wie muss eine Managementplattformarchitektur gestaltet sein, damit sie ein zuverlässiges Management von FSNs erlaubt? Die Frage nach einer universell geeigneten Architektur einer Managementplattform lässt sich nicht final beantworten. Sie ist vielmehr **anwendungsfallabhängig**. Klassische FSNs, wie sie in dieser Arbeit betrachtet werden, sind jedoch geographisch verteilt, sodass auch die Anforderung nach einer **verteilten Systemarchitektur** (**NF.11**) in Abschnitt 3.6.3 postuliert wurde. Im Konzept dieser Arbeit werden daher auch unterschiedliche Arten der Interoperation von Managementplattforminstanzen in einem Verbund in Abschnitt 5.6.2.6 mit jeweiligen Vor- und Nachteilen vorgeschlagen.

F17: Wie können nicht-standardisierte Schnittstellen (insb. von Netzkomponenten) in einer Managementplattform handhabbar werden? Nicht-standardisierte Schnittstellen umfassen Protokolle und Datenaustauschformate, die insbesondere in einer heterogenen gemanagten IT-Infrastruktur nicht einschränkbar sind. Eine Managementplattform muss daher beide Aspekte jeweils einzeln modellieren und **dynamisch** miteinander **koppeln** können, damit nicht jede Protokoll-Datenformat-Kombination separat implementiert werden muss. Dieser Aspekt erlaubt eine Handhabbarkeit der Heterogenität von Schnittstellen von MOs und wird insbesondere im Kontext des SBI in Abschnitt 5.6.1 geklärt. Am NBI gibt dagegen vielmehr die Managementplattform selbst die Formate und Protokolle für Nutzer homogen vor.

9.2 Zusammenfassung dieser Arbeit

Die vorliegende Arbeit leitet Konzepte zur Entwicklung geeigneter Plattformen für das Management in föderierten softwarebasierten Netzen (FSN) her und formalisiert sie über Softwareframeworks. Die Herleitung wird über einen systematischen Ansatz verfolgt, in dem zunächst **Ausprägungen von FSNs** untersucht und in Form von **Szenarien** verdeutlicht werden. Eine darauf basierende **Anforderungs-** und **Hot-Spot-Analyse** dient zur Bewertung bestehender Ansätze und vor allem Managementplattformen. Auf Basis der daraus gewonnenen Erkenntnisse wurden **Frameworks** für unterschiedliche Teilbereiche des Netzmanagements konzipiert, implementiert und evaluiert.

Ausprägungen von FSNs werden in dieser Arbeit unter organisatorischen und technischen Gesichtspunkten betrachtet. **Organisatorische Charakteristika** können zunächst grundlegend in Hinblick auf **Ausprägungen von Föderationen** variieren. Diese wurden auf Basis bestehender Arbeiten aus den verwandten Bereichen des Managements Virtueller Organisationen aus dem Grid-Computing, dem föderierten Identitätsmanagement und dem interorganisationalen Fehlermanagement erstellt. In FSNs konnten mehrere dieser Charakteristika als statisch oder irrelevant begründet und fünf neue Charakteristika definiert werden: Die Relation zwischen administrativen Domänen, die Domänenstruktur, die Zielsetzung der Föderation, die Zusammensetzung der Infrastruktur und die Art der Infrastrukturnutzung. Stets gültige Charakteristika des Betriebs von FSNs ergeben sich allein durch ein gemeinsames kooperatives Netzmanagement der Partner: Es müssen unterschiedliche Managementkonzepte der Parteien, Beziehungen zwischen ihnen (z. B. Vertrauen) und lokale Entscheidungsauswirkungen berücksichtigt werden. Wesentliche **technische Charakteristika** in FSNs sind ihre ebenfalls durch die Kooperation mehrerer unabhängiger Parteien verstärkte Heterogenität in Komponenten, ihren Schnittstellen, Managementinformationen und Funktionen. Auch sind sie geographisch verteilt und somit, sowie durch Virtualisierung, meist sehr groß und machen einen skalierbaren Ansatz unbedingt notwendig.

Die charakteristischen Ausprägungen flossen in drei unterschiedliche FSN-Szenarien derart ein, dass jede Ausprägung berücksichtigt wurde. **Anforderungen** an eine Managementplattform für FSNs konnten zum einen von den zuvor aufgestellten allgemeinen Betriebscharakteristika in FSNs, zum anderen von den Szenarien abgeleitet werden. Die Szenarien haben darüber hinaus flexible Bausteine der Frameworks, die sogenannten **Hot-Spots**, in den vier Teilmodellen verdeutlicht: Im Funktionsmodell umfassen sie die Modellierung von Plattformfunktionen und funktionalen Managementbereichen, und im Kommunikationsmodell insbesondere das SBI, aber auch die API und die Datenbankanbindung. Besonders spezifische Teilmodelle für FSNs stellen das Informationsmodell und das Organisationsmodell dar. Das Informationsmodell muss die Modellierung von Managementobjektklassen, Managementbeziehungen, Netzinformationen und Alarmen ermöglichen. Eine besondere Herausforderung stellt die Modellierung von Funktionen von Managementobjekten dar, da Netzkomponententypen wie SDN-Controller in unterschiedlichen Implementierungen ähnliche Funktionen bereitstellen. Im Organisationsmodell müssen anwendungsfallspezifisch Föderationen und Domänen, Nutzer und Rollen sowie Managementaufgaben modellierbar sein. Die teilmodellübergreifenden Bausteine müssen zudem Kernprozesse des Netzmanagements unter Berücksichtigung von Mandantenfähigkeit unterstützen.

Die technische Unterstützung des Netzmanagements von FSNs ist in den betrachteten **bestehenden Managementplattformen** unterschiedlich ausgeprägt. Defizite bestehen jedoch in jedem der vier Teilmodelle und keine der Lösungen ist als solche einsetzbar. Im Funktionsmodell fehlt vor allem die Unterstützung flexibler Funktionsbereiche; sie werden in keinem der Ansätze explizit einbezogen und sind meist nur teilweise berücksichtigt. Im Informationsmodell bieten die Plattformen meist bereits einen erweiterbaren Modellansatz an, jedoch werden FSN-spezifische MOs nicht oder nur teilweise berücksichtigt. Anforderungen an das Kommunikationsmodell werden größtenteils erfüllt. Die größten Lücken bestehen demnach im Organisationsmodell: Bestehende Plattformen erlauben keine flexible Modellierung einer Föderation und unzureichende koordinierende Strukturen sowie an FSNs angepasste Zugriffsverfahren auf Informationselemente und Funktionen der Managementplattform. Heterogene Managementkonzepte und administrative Domänen werden nicht ausreichend unterstützt. Domänenüberschneidungen werden ebenfalls von keinem der Ansätze geeignet behandelt.

Das **Konzept** in dieser Arbeit beschreibt Frameworks zur Entwicklung von Managementplattformen in FSNs gemäß der zuvor durchgeführten Hot-Spot-Analyse. Die Frameworks können zum einen genutzt werden, um bestehende Plattformen in einzelnen Aspekten für das Management von FSNs zu komplementieren und sind zu diesem Zweck jeweils für sich verwendbar. Zum anderen sind die Frameworks zueinander kompatibel und weisen definierte Schnittstel-

len zueinander auf. Sie können daher auch verwendet werden, um anwendungsfallspezifisch geeignete Managementplattformen oder Teilmodelle von Grund auf zu implementieren.

Das **Informationsmodell** wird durch fünf Frameworks unterstützt: Das Framework für **Managementobjektklassen** (MOC) muss die Dynamik von Komponenten in FSNs abbilden können. Dazu wurden MOCs jeweils in Netzkomponente (z. B. System oder Anwendung) und Netzfunktion (z. B. SDN-Controller) getrennt. Einer Netzkomponente werden über Klassifikation eine oder mehrere Netzfunktionen zugewiesen. Die Modellierung von Netzfunktionen wird durch eine vorgegebene, und für FSNs geeignete Ableitungsstruktur unterstützt. Davon getrennt wurde ein Framework für **MOC-Funktionen** entwickelt, das MO-Funktionen für Netzfunktionen normalisieren kann. Die Normalisierung wurde durch Vererbungsprinzipien und die Trennung von Funktionskopf und -implementierung erreicht. Funktionsköpfe werden Netzfunktionen zugewiesen und an ihre Spezialisierungen im Vererbungsbaum weitergegeben. Die Funktionsimplementierung kann ebenfalls geerbt oder adaptiert werden. Das Framework für **Managementbeziehungen** muss für FSNs spezifisch sein. Dazu wurde eine systematische Struktur aus vertikalen und horizontalen Abhängigkeiten hergeleitet und in ihrer Funktion vereinheitlicht. Konkrete Managementbeziehungen wurden auf Basis einer Analyse modelliert. Das Framework für **Netzinformationen** dient insbesondere zur Normalisierungen von Informationen aus der gemanagten IT-Infrastruktur. Das Framework für **Alarme** erlaubt die benutzerdefinierte Beschreibung von Alarmklassen und ihrer Kritikalität rein auf Basis von Vererbungsprinzipien. Netzinformationen werden genauso wie Alarme zudem in einen gesamtorganisatorischen Kontext gesetzt; Zuständigkeiten sind inhärent festlegbar.

Das **Organisationsmodell** kann ebenfalls durch fünf unterschiedliche Frameworks beschrieben werden: Das Framework für **Föderationen** dient zur Modellierung der Föderationsstruktur aus Parteien und Datenzentren. Darüber hinaus werden in dieser Arbeit generische und modellierbare Föderationsbeziehungen als Managementinformationen eingeführt, die in bisherigen Arbeiten in der Regel nur in Form von Vertrauen zwischen Parteien betrachtet wurden. Im Framework für **administrative Domänen** können ebendiese modelliert werden. Auch hier werden Beziehungen zwischen Domänen in dieser Arbeit als Managementinformationen eingeführt. Darüber hinaus wird ein Konzept zur feingranularen Behandlung von Domänenüberschneidungen vorgeschlagen. Es basiert auf einer Neuverteilung von Verantwortlichkeiten. Das Framework für **Managementaufgaben** erlaubt die Modellierung von föderationsumfassenden Verantwortlichkeiten sowie ihre Verfeinerung innerhalb von Domänen. Auf diese Weise können heterogene Managementkonzepte der Föderationspartner gehandhabt werden. Das Framework für **mandantenfähigen Zugriff** definiert davon betroffene Informationselemente und erlaubt die benutzerdefinierbare Zugriffsbestimmung auf Basis von FSN-spezifischen Kriterien der Föderationsbeziehungen, Domänenzugehörigkeit und Zuständigkeiten von Nutzern darin (aus dem Framework für **Nutzer- und Rollen**) sowie Domänenbeziehungen.

Die Frameworkkonzepte für das **Kommunikationsmodell** sind nach ihrer Kommunikationsrichtung für das Southbound-Interface (SBI), Northbound-Interface (NBI) und die Datenbankanbindung unterteilt. Zudem wurden zentrale Bausteine wie Netzadapter identifiziert. Im **SBI** wird ein Prozess beschrieben, über den heterogene Datenformate, empfangen über heterogene Schnittstellen und Protokolle, in das Datenmodell integriert werden können. Darin werden Schnittstellencontainer für Daten in unterschiedlichem Verarbeitungsgrad (d. h. im Roh-, Objekt- und einem Normalisierungsformat) definiert, durch den der Prozess flexibel erweitert werden kann. Netzinformationen aus dem Informationsmodell werden zur Normalisierung verwendet. Im **NBI** werden Konzepte zur Modellierung lokaler und remoter Push-Benachrichtigungen bzw. Pull-Zugriffe beschrieben, die in das Gesamtkonzept integriert sind. Sie bieten eine kontrollierte Möglichkeit zur Benachrichtigung von Managementanwendungen über Änderungen im Modell bzw. erlauben das Durchreichen von Plattformfunktionen an diese. Darüber hinaus wird das in bestehenden Managementplattformen meist als East- oder Westbound-Interface zur Anbindung externer Managementplattform-Instanzen kontrolliert über das NBI umgesetzt. Auf diese Weise können Systemkomponenten wiederverwendet wer-

den. Das Framework für die **Datenbankanbindung** sieht eine Abstraktion von Datenbankoperationen vor, durch die die Art der in einer Managementplattform genutzten Datenbank nicht eingeschränkt wird.

Im **Funktionsmodell** werden zwei weitere Frameworks eingeführt: Erstens ein Framework zur Modellierung von **funktionalen Managementbereichen** zur Beschreibung von Managementfunktionen. In den in dieser Arbeit beschriebenen Szenarien wurde die Organisation anhand dieser strukturiert. In der Umsetzung manifestiert sich diese Strukturierung durch die Möglichkeit, Managementfunktionen in mehrere Aufgabenbereiche und diese wiederum in feingranulare Aufgaben zu unterteilen. Zweitens wird ein Framework zur Erweiterung der **Funktionalität von Managementplattformen** beschrieben. Ähnlich wie für MOCs werden Plattformfunktionen anhand ihrer Parameter und ihrem Rückgabewert beschrieben. Im Kontext des Funktionsmodells wurde darüber hinaus der Prozess zur **Überwachung und Steuerung** konzipiert. Dafür ist jedoch kein eigenständiges Framework vorgesehen, sondern der Prozess wird implizit als Teil der Bausteine der Frameworks insbesondere des Kommunikationsmodells beschrieben.

Im Rahmen der Arbeit wurden die Frameworkkonzepte zum großen Teil in einer **Implementierung** realisiert. Die Umsetzung erfolgte aus Gründen der Nutzbarkeit und Plattformunabhängigkeit in Java. Im Rahmen einer **beispielhaften Anwendung** der Implementierung wurden die vier Teilmodelle für ein Evaluierungsszenario implementiert. Auf diese Weise konnte die Eignung der Frameworks zur Modellierung der gemanagten IT-Infrastruktur, organisatorischer und funktionaler Aspekte gezeigt werden. Beispielhaft wurden zudem Schnittstellen des SBI und NBI umgesetzt.

Die Kernprozesse des Netzmanagements wurden in einer dedizierten **Evaluierung** hinsichtlich ihrer Performanz in einer Laborumgebung auf handelsüblicher Verbraucherhardware überprüft. Im **Überwachungsprozess** wurden realistische Informationen vonseiten einer gemanagten IT-Infrastruktur modelliert und in gewissem Umfang zufällig variiert. Sie wurden durch gängige Parser-Implementierungen interpretiert und über Netzinformationen normalisiert. Darüber hinaus wurden beispielhafte Auswertungen implementiert, sodass ein gewisser Bruchteil automatisierte Push-Benachrichtigungen ausgelöst hat. Getrennt davon wurden im **Steuerungsprozess** die übliche Zugriffs- und Ausführungszeiten realistischer Pull-Operationen über das NBI evaluiert. Unter Berücksichtigung einer vernachlässigbaren Zeit zur Ermittlung der **Zugriffsentscheidungen** liegt die Zeit eines Closed-Loop-Zyklus im niedrigeren zweistelligen Millisekundenbereich. Das Konzept und die Implementierung erlauben entsprechend auch für nicht-optimierte Implementierungen niedrige Reaktionszeiten.

9.3 Ausblick auf weiterführende Forschungsfragen

In der vorliegenden Arbeit wurden alle Bausteine beschrieben, die zur Implementierung einer geeigneten Managementarchitektur in föderierten softwarebasierten Netzen in Form einer Managementplattform notwendig sind. In den in dieser Arbeit betrachteten Szenarien wurde jedoch stets davon ausgegangen, dass eine Managementplattform in FSNs für diese logisch zentralisiert ist und die komplette föderierte IT-Infrastruktur überwacht und steuert. Dennoch sind auch andere Szenarien denkbar, deren Untersuchung im Rahmen dieser Arbeit nicht ausführlich betrachtet wurde. Im Folgenden werden vier weiterführende Fragestellungen genannt.

- **Netzmanagement in Multi-Föderationen.** In dieser Arbeit wurde die Föderation mehrerer Parteien betrachtet. Dennoch sind auch Szenarien denkbar, in denen über mehrere selbst föderierte IT-Infrastrukturen kooperiert wird, beispielsweise eine Kooperation über Forschungsprojekte. Möglicherweise können einige Lösungen aus dieser Arbeit hierauf übertragen werden, jedoch wurden derartige Szenarien und damit neu aufkommende Problemstellungen nicht explizit berücksichtigt.

- **Föderation heterogener Managementplattformen.** Die Kopplung von heterogenen Managementplattformen an die Managementinfrastruktur wurde in dieser Arbeit nicht im Detail betrachtet. Damit verbundene Fragestellungen sind beispielsweise, wie Informationen unterschiedlich umgesetzter Teilmodelle, beispielsweise MOCs, Netzinformationen oder auch Plattformfunktionen normalisiert werden können. Gerade in FSNs ist diese Fragestellung unter der Annahme von Bedeutung, dass Föderationspartner die Funktionen eigener Managementplattformen auch in der Föderation nutzen möchten.

- **Platzierung verteilter Managementplattformen.** In dieser Arbeit wird in Hinblick auf die Unterstützung von Skalierbarkeit der Managementplattform ein verteiltes System vorgeschlagen. Eine Managementplattform wird dabei in einem *logischen* Datenzentrum betrieben, das nicht unbedingt einem tatsächlichen Datenzentrum entspricht, sondern letzteres beispielsweise weiter unterteilt oder mehrere tatsächliche Datenzentren zusammenfasst. Eine geeignete Platzierung von Managementplattformen in einer föderierten IT-Infrastruktur wurde in dieser Arbeit jedoch nicht evaluiert und stellt eine offene Forschungsfrage dar.

- **Integration von Unterstützungssystemen.** Das Zusammenspiel von Managementplattformen und unterstützenden Lösungen wie Trouble-Ticket-, Schwachstellenmanagement- oder SIEM-Systemen und ihre Bedeutung für die Automatisierungsmöglichkeiten in FSNs müssen noch näher betrachtet werden. Beispielsweise bieten gerade SNs Möglichkeiten für eine automatisierte Behebung detektierter Schwachstellen in Netzkomponenten. Die Berücksichtigung organisatorischer Aspekte genauso wie Managementbeziehungen zwischen Systemen muss jedoch im Detail untersucht werden, um die Zuverlässigkeit des Netzes weiterhin zu gewährleisten.

Weitere auf dieser Arbeit aufbauende Fragestellungen ergeben sich durch offene Lücken, die sich im Rahmen dieser Dissertation ergeben haben.

- **Selbstkonfiguration und Manipulationsschutz.** Der Abgleich der Anforderungen an Managementplattformen in FSNs mit den Konzepten aus dieser Arbeit hat gezeigt, dass zwei funktionale Aspekte nicht gelöst werden konnten. Das sind zum einen die Möglichkeit zur Selbstkonfiguration von Managementplattformen sowie ihr Manipulationsschutz. Beide Anforderungen schränken die Beantwortung der Forschungsfragen dieser Arbeit nicht ein, sind jedoch in bestimmten Szenarien (z. B. FSNs als IT-Infrastrukturen für medizinische Anwendungen) notwendigerweise zu berücksichtigen.

- **Priorisierte Nachrichtenverarbeitung.** Wie in Abschnitt 8.2.2 gezeigt werden konnte, ist die Echtzeitfähigkeit der Überwachung insbesondere von geeigneten Priorisierungsmechanismen bei der Verarbeitung von Elementen der IT-Infrastruktur abhängig. Unter bestimmten Bedingungen, z. B. vieler gleichzeitig eintreffender Überwachungsdaten der gemanagten IT-Infrastruktur, ist die Effektivität des in dieser Arbeit berücksichtigten Priorisierungsansatz nicht mehr gegeben. Eine nähere Untersuchung dieser Bedingungen und die Entwicklung geeigneter Priorisierungsmechanismen stellen daher wichtige weiterführende Forschungsgebiete dar.

Abkürzungen

3GPP 3rd Generation Partnership Project.

A&AI Active and Available Inventory.
AAA Authentication, Authorization and Accounting.
AAF Application Authorization Framework.
AD Active Domain.
ADEV Managementanwendungsentwickler.
AMQP Advanced Message Queuing Protocol.
APEX Adaptive Policy Execution.
API Application Programming Interface.

BSS Business Support System.

CADF Cloud Auditing Data Federation.
CIMI Cloud Management Infrastructure Interface.
CLAMP Control Loop Automation Management Platform.
CLI Command Line Interface.
CNF Containerized Network Function.
CORD Central Office Re-architectured as a Datacenter.
CPU Central Processing Unit.
CRUD Create, Read, Update, Delete.
CSV Character-Separated Values.

DCAE Data Collection, Analytics and Events.
DFN Deutsches Forschungsnetz.
DKOE Domänenkreuzorganisationseintrag.
DKOEI Domänenkreuzorganisationseintragsindex.
DKOT Domänenkreuzorganisationstabelle.
DMaaP Data Movement as a Platform.

EM Element Management.
ETSI European Telecommunications Standards Institute.
ETSI GS ETSI Group Specification.

FCAPS Fault-, Configuration-, Accounting-, Performance- und Securitymanagement.
FEDCON Federated Control Framework.
FIDDLE Federated Infrastructure Discovery and Description Language.
FIM Föderiertes Identitätsmanagement.
FSN Föderiertes softwarebasiertes Netz.
FSNMOC FSN-Managementobjektklasse.
FSNMOCK FSNMOC-Klassifikator.

GMS Global System for Mobile Communications.
GPRS General Packet Radio Service.
GUI Graphical User Interface.

HN Herkömmliches Netz.
HTTP Hypertext Transfer Protocol.
HTTPS Hypertext Transfer Protocol Secure.

ID Identifikator.
IDS Intrusion Detection System.
IEC International Electrotechnical Commission.
IM Integriertes Management.
IoT Internet of Things.
IP Internet Protocol.
ISO International Organisation for Standardization.
ISP Internet Service Provider.
IT Informationstechnik.
ITU-T ITU Telecommunication Standardization Sector.

JAX-RS Jakarta RESTful Web Services.
JAXP Java API for XML Processing.
JDK Java Development Kit.
JSON JavaScript Object Notation.
JVM Java Virtual Machine.

KNF Kubernetes Network Function.
KVM Kernel-based Virtual Machine.

LCM Lifecycle Management.
LDAP Lightweight Directory Access Protocol.

LMAPP Lokal angebundene Managementanwendung.
LXC Linux Containers.

MAC Media Access Control.
MANO Management and Orchestrierung.
MAPP Managementanwendung.
MD-SAL Model-Driven Service Adaptation Layer.
MedG Medizinisches Gerät.
MIB Management Information Base.
MO Managementobjekt.
MOC Managementobjektklasse.
MOCK siehe FSNMOCK (Synonym).
MWN Münchener Wissenschaftsnetz.

NBI Northbound-Interface.
NETCONF Network Configuration Protocol.
NF Netzfunktion.
NFSN Nicht-FSN-spezifische Systeme.
NFV Network Functions Virtualization.
NFV-IFA NFV Interfaces and Architecture.
NFV-SOL NFV Solutions.
NFV-TST NFV Testing.
NFVI NFV Infrastructure.
NFVO NFV Orchestrator.
NIDS Network Intrusion Detection System.
NIST National Institute of Standards and Technology.
NMAN Netzmanager.
NMAPP Netzgebundene Managementanwendung.
NML Network Markup Language.
NSD Network Service Descriptor.
NSI Network Slice Instance.
NSOK Network and Security Objects Kit.
NSR Network Service Record.
NST Network Slice Template.
NVGRE Network Virtualization Using Generic Routing Encapsulation.
NVW Komponenten zur Netzverkehrweiterleitung.

OASIS Organization for the Advancement of Structured Information Standards.
OCCI Open Cloud Computing Interface.
ODL OpenDaylight.
OID Object Identifier.
ONAP Open Network Automation Platform.
ONF Open Network Foundation.
ONOS Open Network Operating System.
OOF ONAP Optimization Framework.
OOP Objektorientierte Programmierung.
OPNFV Open Platform for NFV.
OSM Open Source MANO.
OSS Operations Support System.
OVSDB Open vSwitch Database Management Protocol.

PADM Managementplattformadministrator.
PC Personal Computer.
PCAP Packet Capture.
PDEV Managementplattformentwickler.
PNF Physical Network Function.
POF Protocol Oblivious Forwarding.

QEMU Quick Emulator.

RAM Random Access Memory.
RBAC Role-based Access Control.
REST Representational state transfer.
RFC Request for Comments.
RO Resource Orchestration.
RPC Remote Procedure Call.
RZ Rechenzentrum.

SBI Southbound-Interface.
SDK Software Development Kit.
SDN Software-Defined Networking.
SDNA SDN-Anwendung.
SDNC SDN-Controller.
SDNI SDN-Infrastruktur.
SF Sicherheitsfunktion.
SLA Service Level Agreement.
SMI Structure of Management Information.
SN Softwarebasiertes Netz.
SNMP Simple Network Management Protocol.
SOAP Simple Object Access Protocol.
SSH Secure Shell.
STIX Structured Threat Information Expression.

TAXII Trusted Automated Exchange of Intelligence Information.
TLS Transport Layer Security.
TOSCA Topology and Orchestration Specification for Cloud Applications.
TOTP Time-based One-time Password.

UDP User Datagram Protocol.
URI Unified Resource Identifier.
URL Uniform Resource Locator.

VCA VNF Configuration and Abstraction.
VES Virtual Functions Event Streaming Model.
VIM Virtualized Infrastructure Manager.
VIRS Virtualisierungssystem.
VLAN Virtual Local Area Network.
VM Virtuelle Maschine.
VNF Virtual Network Function.
VNFD Virtual Network Function Descriptor.

VNFM VNF Manager.
VNFR Virtual Network Function Record.
VO Virtuelle Organisation.
VPN Virtual Private Network.
VXLAN Virtual Extensible LAN.

WP Work Package.

XACML Extensible Access Control Markup Language.
XML Extensible Markup Language.

YANG Yet Another Next Generation.
YIN YANG Independent Notation.

ZSM Zero Touch Network & Service Management.
ZVB Zugriffsverwaltungsbaum.

Literatur

[1] M. Chiosi, D. Clarke, P. Willis u. a., *Network Functions Virtualization*, Okt. 2012. Adresse: https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/White%20Papers/NFV_White_Paper1_2012.pdf.

[2] J. G. Herrera und J. F. Botero, „Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management*, Jg. 13, Nr. 3, S. 518–532, 2016.

[3] Open Networking Foundation, *Software-Defined Networking:The New Norm for Networks*, Apr. 2012. Adresse: http://opennetworking.wpengine.com/wp-content/uploads/2011/09/wp-sdn-newnorm.pdf.

[4] G. Pujolle, *Software Networks: Virtualization, SDN, 5G, Security*. John Wiley & Sons, 2015.

[5] F. De Turck, R. Boutaba, P. Chemouil, J. Bi und C. Westphal, „Guest Editors' Introduction: Special Issue on Efficient Management of SDN/NFV-Based Systems—Part I," *IEEE Transactions on Network and Service Management*, Jg. 12, Nr. 1, S. 1–3, 2015.

[6] European Telecommunications Standards Institute, *ETSI GS NFV 002 V1.2.1 (2014-12). Network Functions Virtualisation (NFV); Architectural Framework*, Dez. 2014. Adresse: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf.

[7] A. Clemm, *Network management fundamentals*. Cisco Press, 2006.

[8] H.-G. Hegering, S. Abeck und B. Neumair, *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. ISBN 1-55860-571-1, Morgan Kaufmann Publishers, 1999.

[9] C. A. Joseph und K. H. Muralidhar, „Integrated network management in an enterprise environment," *IEEE Network*, Jg. 4, Nr. 4, S. 7–13, 1990.

[10] J. Case, R. Mundy, D. Partain und B. Steward, *Introduction and Applicability Statements for Internet Standard Management Framework*, Dez. 2002. Adresse: https://tools.ietf.org/html/rfc3410.

[11] H. Reiser und S. Metzger, *Das Münchner Wissenschaftsnetz (MWN) Konzepte, Dienste, Infrastruktur und Management*, Jan. 2019. Adresse: https://www.lrz.de/services/netz/mwn-netzkonzept/mwn-netzkonzept-2019.pdf.

[12] R. Moreno-Vozmediano, R. S. Montero und I. M. Llorente, „Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures," *Computer*, Jg. 45, Nr. 12, S. 65–72, 2012.

[13] J. S. Turner und D. E. Taylor, „Diversifying the internet," in *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, IEEE, Bd. 2, 2005, 6–pp.

[14] A. Bhardwaj und C. R. Krishna, „Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey," *Arabian Journal for Science and Engineering*, S. 1–17, 2021.

[15] M. Mahalingam, D. Dutt, K. Duda u. a., *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*, Aug. 2014. Adresse: https://datatracker.ietf.org/doc/html/rfc7348.

[16] P. Garg und Y. Wang, *NVGRE: Network Virtualization Using Generic Routing Encapsulation*, Sep. 2015. Adresse: https://datatracker.ietf.org/doc/html/rfc7637.

[17] Open Networking Foundation, *OpenFlow Switch Specification, Version 1.5.1 (Protocol version 0x06)*, März 2015. Adresse: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[18] S. Li, D. Hu, W. Fang u. a., „Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Network*, Jg. 31, Nr. 2, S. 58–66, 2017.

[19] P. Bosshart, D. Daly, G. Gibb u. a., „P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, Jg. 44, Nr. 3, S. 87–95, 2014.

[20] B. Pfaff und B. Davie, *The Open vSwitch Database Management Protocol*, Dez. 2013. Adresse: https://datatracker.ietf.org/doc/html/rfc7047.txt.

[21] M. Schiffers, „Management dynamischer Virtueller Organisationen in Grids," Dissertation, Ludwig-Maximilians-Universität München, 2007. Adresse: https://edoc.ub.uni-muenchen.de/7352/1/Schiffers_Michael.pdf.

[22] D. Pöhn, „Architektur und Werkzeuge für dynamisches Identitätsmanagement in Föderationen," Dissertation, Ludwig-Maximilians-Universität München, 2016. Adresse: https://edoc.ub.uni-muenchen.de/20203/1/Poehn_Daniela.pdf.

[23] P. Marcu, „Architekturkonzepte für interorganisationales Fehlermanagement," Dissertation, Ludwig-Maximilians-Universität München, 2011. Adresse: https://edoc.ub.uni-muenchen.de/13106/1/Marcu_Patricia.pdf.

[24] M. Hamm, „Eine Methode zur Spezifikation der IT-Service-Managementprozesse Verketteter Dienste," Dissertation, Ludwig-Maximilians-Universität München, 2009. Adresse: https://edoc.ub.uni-muenchen.de/10264/1/Hamm_Matthias.pdf.

[25] R. J. Wirfs-Brock und R. E. Johnson, „Surveying Current Research in Object-Oriented Design," *Communications of the ACM*, Jg. 33, Nr. 9, S. 104–124, 1990.

[26] R. E. Johnson, „Frameworks=(Components+Patterns)," *Communications of the ACM*, Jg. 40, Nr. 10, S. 39–42, 1997.

[27] G. Froehlich, H. J. Hoover, L. Liu und P. Sorenson, „Hooking into Object-Oriented Application Frameworks," in *Proceedings of the 19th international conference on Software engineering*, ACM, 1997, S. 491–501.

[28] R. E. Johnson, „Documenting Frameworks using Patterns," in *conference proceedings on Object-oriented programming systems, languages, and applications*, 1992, S. 63–76.

[29] J. van Gurp und J. Bosch, „Design, Implementation and Evolution of Object Oriented Frameworks: Concepts and Guidelines," *Software: Practice and Experience*, Jg. 31, Nr. 3, S. 277–300, 2001.

[30] M. Mattsson, „Evolution and Composition of Object-Oriented Frameworks," Diss., Blekinge Institute of Technology, 2000.

[31] D. Parsons, A. Rashid, A. Speck und A. Telea, „A 'Framework' for Object Oriented Frameworks Design.," in *TOOLS (29)*, 1999, S. 141–151.

[32] C. Lorenz, D. Hock, J. Scherer u. a., „An SDN/NFV-enabled enterprise network architecture offering fine-grained security policy enforcement," *IEEE communications magazine*, Jg. 55, Nr. 3, S. 217–223, 2017.

[33] International Organization for Standardization und International Electrotechnical Commission, *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management framework*, Nov. 1989.

[34] C. Ashford, „The OSI Managed-object Model," in *European Conference on Object-Oriented Programming*, Springer, 1993, S. 185–196.

[35] A. Westerinen, J. Schnizlein, J. Strassner u. a., „Terminology for Policy-Based Management," en, RFC Editor, Techn. Ber. RFC3198, Nov. 2001. DOI: 10.17487/rfc3198. Adresse: https://www.rfc-editor.org/info/rfc3198 (besucht am 28. 03. 2019).

[36] M. Steinke und W. F. Hommel, „Overcoming Network and Security Management Platform Gaps in Federated Software Networks," in *2018 14th International Conference on Network and Service Management (CNSM) (CNSM 2018)*, Rome, Italy, Nov. 2018.

[37] P. Mell und T. Grance, *The NIST Definition of Cloud Computing*, Sep. 2011. Adresse: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[38] A. Marsico, A. Reid, F. Ramón und G. García, *OSM in Action*, Juli 2021. Adresse: https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_in_Action.pdf.

[39] U. Blumenthal und B. Wijnen, *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*, Dez. 2002. Adresse: https://tools.ietf.org/html/rfc3414.

[40] *Project SENDATE-PLANETS*, Juni 2020. Adresse: https://www.celticnext.eu/sendate-planets/ (besucht am 27. 12. 2021).

[41] *Project Achievements*, SEcure Networking for a DATa center cloud in Europe, Sep. 2019. Adresse: https://bscw.celticnext.eu/pub/bscw.cgi/d26620/SENDATE-Planets-final_lq.pdf.

[42] M. Hoffmann, M. Jarschel, R. Pries u. a., „SDN and NFV as Enabler for the Distributed Network Cloud," *Mobile Networks and Applications*, Jg. 23, Nr. 3, S. 521–528, 2018.

[43] Bundesministerium für Gesundheit, *Glossar*, Mai 2018. Adresse: https://www.bundesgesundheitsministerium.de/service/begriffe-von-a-z/e/e-health.html (besucht am 20. 05. 2019).

[44] Bundesärztekammer, *Telemedizin*, Dez. 2015. Adresse: https://www.bundesaerztekammer.de/aerzte/telematiktelemedizin/telemedizin/ (besucht am 21. 05. 2019).

[45] Z. Jin und Y. Chen, „Telemedicine in the Cloud Era: Prospects and Challenges," *IEEE Pervasive Computing*, Jg. 14, Nr. 1, S. 54–61, 2015.

[46] C. O. Rolim, F. L. Koch, C. B. Westphall, J. Werner, A. Fracalossi und G. S. Salvador, „A Cloud Computing Solution for Patient's Data Collection in Health Care Institutions," in *2010 Second International Conference on eHealth, Telemedicine, and Social Medicine*, IEEE, 2010, S. 95–99.

[47] D. C. Klonoff, *Fog Computing and Edge Computing Architectures for Processing Data From Diabetes Devices Connected to the Medical Internet of Things*, 2017.

[48] V. J. Sharp und K. J. Kreder, „Medical Equipment: Leasing vs. Buying," in *The Complete Business Guide for a Successful Medical Practice*, Springer, Okt. 2015, S. 133–148.

[49] Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften, *Dienstleistungs- und Gebührenkatalog 2022*, Dez. 2021. Adresse: https://www.lrz.de/wir/regelwerk/dienstleistungskatalog.pdf (besucht am 02. 05. 2022).

[50] I. Ruiz-Agundez, Y. K. Penya und P. G. Bringas, „A Flexible Accounting Model for Cloud Computing," in *2011 Annual SRII Global Conference*, IEEE, 2011, S. 277–284.

[51] M. Nkosi, A. Lysko, L. Ravhuanzwo, T. Nandeni und A. Engelberencht, „Classification of SDN distributed controller approaches: a brief overview," in *2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, IEEE, 2016, S. 342–344.

[52] S. Chisholm und D. Romascanu, *Alarm Management Information Base (MIB)*, Sep. 2004. Adresse: https://www.rfc-editor.org/rfc/rfc3877.txt (besucht am 02. 05. 2022).

[53] J. Case, M. Fedor, M. Schoffstall und J. Davin, *A Simple Network Management Protocol (SNMP)*, Mai 1990. Adresse: https://tools.ietf.org/html/rfc1157.

[54] J. Case, K. McCloghrie, M. Rose und S. Waldbusser, *Manager-to-Manager Management Information Base*, Apr. 1993. Adresse: https://tools.ietf.org/html/rfc1451.

[55] J. Case, K. McCloghrie, M. Rose und S. Waldbusser, *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)*, Jan. 1996. Adresse: https://tools.ietf.org/html/rfc1905.

[56] OpenDaylight Project, *SNMP Plugin User Guide*, 2018. Adresse: https://docs.opendaylight.org/en/stable-fluorine/user-guide/snmp-plugin-user-guide.html (besucht am 16. 02. 2022).

[57] A. Campanella, *SNMP*, Apr. 2017. Adresse: https://wiki.onosproject.org/display/ONOS/SNMP (besucht am 16. 02. 2022).

[58] *Libvirt-snmp*, Juni 2011. Adresse: https://wiki.libvirt.org/page/Libvirt-snmp (besucht am 16. 02. 2020).

[59] K. McCloghrie und M. Rose, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, März 1991. Adresse: https://tools.ietf.org/html/rfc1213.

[60] H. Asai, M. MacFaden, J. Schoenwaelder, K. Shima und T. Tsou, *Management Information Base for Virtual Machines Controlled by a Hypervisor*, Okt. 2015. Adresse: https://tools.ietf.org/html/rfc7666.

[61] K. McCloghrie, D. Perkins, J. Schoenwaelder u. a., *Structure of Management Information Version 2 (SMIv2)*, Apr. 1999. Adresse: https://tools.ietf.org/html/rfc2578.

[62] R. Enns, M. Bjorklund, J. Schoenwaelder und A. Bierman, *Network Configuration Protocol (NETCONF)*, Juni 2011. Adresse: https://tools.ietf.org/html/rfc6241.

[63] M. Bjorklund, J. Schoenwaelder, P. Shafer, K. Watsen und R. Wilton, *Network Management Datastore Architecture (NMDA)*, März 2018. Adresse: https://tools.ietf.org/html/rfc8342.

[64] M. Bjorklund, *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, Okt. 2010. Adresse: https://tools.ietf.org/html/rfc6020.

[65] M. Bjorklund, *The YANG 1.1 Data Modeling Language*, Aug. 2016. Adresse: https://tools.ietf.org/html/rfc7950.

[66] A. Clemm, J. Medved, R. Varga, N. Bahadur, H. Ananthakrishnan und X. Liu, *A YANG Data Model for Network Topologies*, März 2018. Adresse: https://datatracker.ietf.org/doc/html/rfc8345.

[67] A. Clemm und E. Voit, *Subscription to YANG Notifications for Datastore Updates*, Sep. 2019. Adresse: https://datatracker.ietf.org/doc/html/rfc8641.

[68] Z. Shelby, K. Hartke und C. Bormann, *The Constrained Application Protocol (CoAP)*, Juni 2014. Adresse: https://tools.ietf.org/html/rfc7252.

[69] A. Bierman, M. Bjorklund und K. Watsen, *RESTCONF Protocol*, Jan. 2017. Adresse: https://tools.ietf.org/html/rfc8040.

[70] European Telecommunications Standards Institute, *Zero touch network & Service Management (ZSM)*, 2022. Adresse: https://www.etsi.org/technologies/zero-touch-network-service-management (besucht am 29. 04. 2022).

[71] European Telecommunications Standards Institute, *Zero-touch network and Service Management (ZSM); Reference Architecture*, Aug. 2019. Adresse: https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/002/01.01.01_60/gs_ZSM002v010101p.pdf.

[72] European Telecommunications Standards Institute, *Zero-touch network and Service Management (ZSM); Requirements based on documented scenarios*, Okt. 2019. Adresse: https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/001/01.01.01_60/gs_ZSM001v010101p.pdf.

[73] European Telecommunications Standards Institute, *Zero-touch network and Service Management (ZSM); General Security Aspects*, Juli 2021. Adresse: https://www.etsi.org/deliver/etsi_gr/ZSM/001_099/010/01.01.01_60/gr_ZSM010v010101p.pdf.

[74] European Telecommunications Standards Institute, *Zero-touch network and Service Management (ZSM); Terminology for concepts in ZSM*, Okt. 2019. Adresse: https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/007/01.01.01_60/gs_ZSM007v010101p.pdf.

[75] European Telecommunications Standards Institute, *Zero-touch network and Service Management (ZSM); Means of Automation*, Mai 2020. Adresse: https://www.etsi.org/deliver/etsi_gr/ZSM/001_099/005/01.01.01_60/gr_ZSM005v010101p.pdf.

[76] J. Durand, M. Andreou, D. Davis, G. Pilz u. a., *Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol*, An Interface for Managing Cloud Infrastructure, Juli 2016. Adresse: https://www.dmtf.org/sites/default/files/standards/documents/DSP0263_2.0.0.pdf.

[77] R. Nyrèn, A. Edmonds, A. Papaspyrou, T. Metsch und B. Parák, *Open Cloud Computing Interface – Core*, Sep. 2016. Adresse: https://www.ogf.org/documents/GFD.221.pdf.

[78] M. Drescher, B. Parák und D. Wallom, *OCCI Compute Resource Templates Profile*, Apr. 2015. Adresse: https://www.ogf.org/documents/GFD.222.pdf.

[79] G. Katsaros, *Open Cloud Computing Interface – Service Level Agreements*, Sep. 2016. Adresse: https://www.ogf.org/documents/GFD.228.pdf.

[80] R. Nyrén, A. Edmonds, T. Metsch und B. Parák, *Open Cloud Computing Interface - HTTP Protocol*, Sep. 2016. Adresse: https://www.ogf.org/documents/GFD.223.pdf.

[81] R. Nyrén, F. Feldhaus, B. Parák und Z. Šustr, *Open Cloud Computing Interface – JSON Rendering*, Sep. 2016. Adresse: https://www.ogf.org/documents/GFD.226.pdf.

[82] B. Jordan, R. Piazza und T. Darley, *STIX™ Version 2.1*, Juni 2021. Adresse: https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.pdf.

[83] B. Jordan und D. Varner, *TAXII™ Version 2.1*, Juni 2021. Adresse: https://docs.oasis-open.org/cti/taxii/v2.1/os/taxii-v2.1-os.pdf.

[84] M. Rutkowski u. a., *Cloud Auditing Data Federation (CADF) - Data Format and Interface Definitions Specification*, Juni 2014. Adresse: https://www.dmtf.org/sites/default/files/standards/documents/DSP0262_1.0.0.pdf.

[85] Distributed Management Task Force, *Open Source Projects Using DMTF Technologies*, 2022. Adresse: https://www.dmtf.org/standards/opensource (besucht am 02. 05. 2022).

[86] R. Cohen, G. Chung, M. Rutkowski, S. Martinelli und M. John, *Cloud Audit Data Federation - OpenStack Profile (CADF-OpenStack)*, Apr. 2015. Adresse: https://www.dmtf.org/sites/default/files/standards/documents/DSP2038_1.1.0.pdf.

[87] J. Medved, R. Varga, A. Tkacik und K. Gray, „OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, IEEE, 2014, S. 1–6.

[88] OpenDaylight Project, *Introduction*, Apr. 2022. Adresse: https://docs.opendaylight.org/en/latest/getting-started-guide/introduction.html (besucht am 02. 05. 2022).

[89] Apache Software Foundation, *Apache Karaf Container 4.x - Documentation*, Aug. 2021. Adresse: http://karaf.apache.org/manual/latest/ (besucht am 16. 02. 2022).

[90] OpenDaylight Project, *Magnesium*, März 2020. Adresse: https://www.opendaylight.org/what-we-do/current-release/magnesium (besucht am 16. 02. 2022).

[91] OpenDaylight Project, *Controller*, März 2022. Adresse: https://docs.opendaylight.org/projects/controller/en/latest/dev-guide.html (besucht am 02. 05. 2022).

[92] OpenDaylight Project, *MD-SAL Release 9.0.0-SNAPSHOT*, März 2022. Adresse: https://docs.opendaylight.org/_/downloads/mdsal/en/latest/pdf/ (besucht am 02. 05. 2022).

[93] OpenDaylight Project, *Setting Up Clustering*, Apr. 2022. Adresse: https://docs.opendaylight.org/en/latest/getting-started-guide/clustering.html (besucht am 02. 05. 2022).

[94] OpenDaylight Project, *Security Considerations*, Apr. 2022. Adresse: https://docs.opendaylight.org/en/latest/getting-started-guide/security_considerations.html (besucht am 02. 05. 2022).

[95] OpenDaylight Project, *Authentication, Authorization and Accounting (AAA) Services*, 2022. Adresse: https://docs.opendaylight.org/projects/aaa/en/latest/dev-guide.html (besucht am 16. 02. 2022).

[96] Administrator, A. Indermitte u. a., *ONOS*, Nov. 2020. Adresse: https://wiki.onosproject.org/ (besucht am 18. 02. 2022).

[97] A. Koshibe, Y. Wang u. a., *System Components*, Sep. 2016. Adresse: https://wiki.onosproject.org/display/ONOS/System+Components (besucht am 18. 02. 2022).

[98] L. Prete und J. Halterman, *Forming a Cluster*, Sep. 2018. Adresse: https://wiki.onosproject.org/display/ONOS/Forming+a+cluster (besucht am 07. 02. 2020).

[99] L. Prete, *Managing ONOS applications*, Jan. 2017. Adresse: https://wiki.onosproject.org/display/ONOS/Managing+ONOS+applications (besucht am 18. 02. 2022).

[100] D. Smyth, *Configuring TLS for inter-controller communication*, Juni 2018. Adresse: https://wiki.onosproject.org/display/ONOS/Configuring+TLS+for+inter-controller+communication (besucht am 09. 12. 2021).

[101] A. Koshibe, *Interacting with ONOS*, Dez. 2014. Adresse: https://wiki.onosproject.org/display/ONOS/Interacting+with+ONOS (besucht am 18. 02. 2022).

[102] T. Vachuska und A. Koshibe, *Configuring Authentication - CLI, GUI & REST API*, Juli 2016. Adresse: https://wiki.onosproject.org/pages/viewpage.action?pageId=4162614 (besucht am 16. 02. 2022).

[103] A. Koshibe und T. Vachuska, *Appendix B: REST API*, März 2019. Adresse: https://wiki.onosproject.org/display/ONOS/Appendix+B%3A+REST+API (besucht am 17. 02. 2022).

[104] P. Joshi und T. Vachuska, *Security-Mode ONOS*, Jan. 2020. Adresse: https://wiki.onosproject.org/display/ONOS/Security-Mode+ONOS (besucht am 17. 01. 2022).

[105] V. Thomas und P. Kandi, *Device Driver Subsystem*, Nov. 2016. Adresse: https://wiki.onosproject.org/display/ONOS/Device+Driver+Subsystem (besucht am 16. 02. 2022).

[106] Anuket, *Platform Overview*, 2021. Adresse: https://docs.anuket.io/en/latest/release/overview.html (besucht am 05. 02. 2022).

[107] A. Tierno, G. García, P. Montes Moreno u. a., *OpenMANO*, Aug. 2021. Adresse: https://github.com/nfvlabs/openmano (besucht am 12. 04. 2022).

[108] *SONATA NFV Platform*. Adresse: https://www.sonata-nfv.eu/content/sonata-nfv-platform (besucht am 05. 02. 2022).

[109] *SONATA Project*. Adresse: https://www.sonata-nfv.eu/content/sonata-project (besucht am 05. 02. 2022).

[110] *5GTANGO Project*. Adresse: https://www.sonata-nfv.eu/content/5gtango-project (besucht am 05. 02. 2022).

[111] A. Reid, A. González, A. Armengol u. a., *OSM Scope, Functionality, Operation and Integration Guidelines*, Feb. 2019. Adresse: https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Scope_and_Functionality.pdf.

[112] ETSI OSM, *5. OSM Usage*, 2020. Adresse: https://osm.etsi.org/docs/user-guide/latest/05-osm-usage.html (besucht am 12. 04. 2022).

[113] ETSI OSM, *1. OSM Quickstart*, 2020. Adresse: https://osm.etsi.org/docs/user-guide/latest/01-quickstart.html (besucht am 12. 04. 2022).

[114] ETSI OSM, *How to configure your environment to develop with OSM*, 2020. Adresse: https://osm.etsi.org/docs/developer-guide/02-developer-how-to.html (besucht am 12. 04. 2022).

[115] ETSI OSM, *6. OSM platform configuration*, 2020. Adresse: https://osm.etsi.org/docs/user-guide/latest/06-osm-platform-configuration.html (besucht am 12. 04. 2022).

[116] M. Beierl, *OSM Development Introduction*, März 2021. Adresse: http://osm-download.etsi.org/ftp/osm-9.0-nine/OSM-MR10-hackfest/presentations/OSM-MR%2310%20Hackfest%20-%20HD4.1%20OSM%20Development%20Intro.pptx.pdf.

[117] A. Reid, A. Marsico, A. ElSawaf, G. García, F. Ramón und S. Sheikh, *OSM Deployment and Integration*, Feb. 2020. Adresse: https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Deployment_and_Integration.pdf.

[118] ETSI OSM, *ANNEX 4: NBI API Description*, 2020. Adresse: https://osm.etsi.org/docs/user-guide/latest/12-osm-nbi.html (besucht am 12. 04. 2022).

[119] ETSI OSM, *11. ANNEX 3: OSM Information Model*, 2020. Adresse: https://osm.etsi.org/docs/user-guide/latest/11-osm-im.html (besucht am 12. 04. 2022).

[120] European Telecommunications Standards Institute, *ETSI GR NFV 003 V1.5.1 (2020-01). Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV*, Jan. 2020. Adresse: https://www.etsi.org/deliver/etsi_gr/NFV/001_099/003/01.05.01_60/gr_NFV003v010501p.pdf.

[121] D. Lete, E. Jacob, H. Khalili u. a., „White Paper: OAV Architectures," Mai 2021.

[122] European Telecommunications Standards Institute und 3rd Generation Partnership Project, *ETSI TS 123 501 V16.6.0 (2020-10). 5G; System architecture for the 5G System (5GS) (3GPP TS 23.501 version 16.6.0 Release 16)*, Okt. 2020. Adresse: https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf.

[123] ETSI OSM, *KNF Onboarding Walkthrough (Work in Progress)*, 2019. Adresse: https://osm.etsi.org/docs/vnf-onboarding-guidelines/07-knfwalkthrough.html (besucht am 12. 04. 2022).

[124] ETSI OSM, *Developer Guides for Specific OSM Modules*, 2020. Adresse: https://osm.etsi.org/docs/developer-guide/03-developer-how-to-for-modules.html (besucht am 12. 04. 2022).

[125] *XOS Overview*, Juli 2020. Adresse: https://guide.xosproject.org/ (besucht am 12. 02. 2022).

[126] *XOS Modeling Framework*, Juli 2020. Adresse: https://guide.xosproject.org/dev/xproto.html (besucht am 12. 02. 2022).

[127] *Core Models*, Juli 2020. Adresse: https://guide.xosproject.org/core_models.html (besucht am 13. 02. 2022).

[128] *Tutorial*, Juli 2020. Adresse: https://guide.xosproject.org/tutorials/basic_synchronizer.html (besucht am 14. 02. 2022).

[129] *Installing XOS*, Juli 2020. Adresse: https://guide.xosproject.org/install.html (besucht am 13. 02. 2022).

[130] *The xosapi Library*, Juli 2020. Adresse: https://guide.xosproject.org/dev/xosapi.html (besucht am 13. 02. 2022).

[131] *Synchronizer Design*, Juli 2020. Adresse: https://guide.xosproject.org/dev/sync_arch.html (besucht am 13. 02. 2022).

[132] *Synchronizer Reference*, Juli 2020. Adresse: https://guide.xosproject.org/dev/sync_reference.html (besucht am 13. 02. 2022).

[133] *Security Policies*, Juli 2020. Adresse: https://guide.xosproject.org/security_policies.html (besucht am 13. 02. 2022).

[134] *XOSSH*, Juli 2020. Adresse: https://guide.xosproject.org/dev/xossh.html (besucht am 14. 02. 2022).

[135] OpenStack contributors, *Install Guide*, Jan. 2022. Adresse: https://docs.openstack.org/install-guide/InstallGuide.pdf (besucht am 27. 01. 2022).

[136] Lifecycle management, *Install Guide*. Adresse: https://www.openstack.org/software/project-navigator/deployment-tools (besucht am 02. 02. 2022).

[137] OpenStack Foundation, *Ceilometer Documentation, Release 18.1.0.dev6*, Apr. 2022. Adresse: https://docs.openstack.org/ceilometer/latest/doc-ceilometer.pdf (besucht am 02. 05. 2022).

[138] OpenStack Foundation, *Monasca Documentation, Release 7.1.0.dev6*, Mai 2022. Adresse: https://docs.openstack.org/monasca-api/latest/doc-monasca-api.pdf (besucht am 02. 05. 2022).

[139] OpenStack Foundation, *Welcome to Vitrage documentation!* Mai 2020. Adresse: https://docs.openstack.org/vitrage/latest/ (besucht am 27. 01. 2022).

[140] OpenStack Foundation, *Using Templates*, Juli 2020. Adresse: https://docs.openstack.org/vitrage/latest/contributor/vitrage-templates.html (besucht am 27. 01. 2022).

[141] OpenStack Foundation, *Aodh Documentation, Release 14.1.0.dev6*, Apr. 2022. Adresse: https://docs.openstack.org/aodh/latest/doc-aodh.pdf (besucht am 02. 05. 2022).

[142] OpenStack Foundation, *Blazar Documentation, Release 9.1.0.dev7*, Apr. 2022. Adresse: https://docs.openstack.org/blazar/latest/doc-blazar.pdf (besucht am 02. 05. 2022).

[143] OpenStack Foundation, *Masakari service overview*, Jan. 2021. Adresse: https://docs.openstack.org/masakari/latest/install/overview.html (besucht am 27. 01. 2022).

[144] OpenStack, *Introduction to TLS and SSL*, März 2022. Adresse: https://docs.openstack.org/security-guide/secure-communication/introduction-to-ssl-and-tls.html (besucht am 29. 04. 2022).

[145] OpenStack Foundation, *Choosing a hypervisor*, Nov. 2018. Adresse: https://docs.openstack.org/arch-design/design-compute/design-compute-hypervisor.html (besucht am 28. 01. 2022).

[146] OpenStack Foundation, *Opendaylight - SDN controller*, Nov. 2018. Adresse: https://docs.openstack.org/kolla-ansible/train/reference/networking/opendaylight.html (besucht am 28. 01. 2022).

[147] OpenStack Foundation, *Welcome to the Heat documentation!* Mai 2021. Adresse: https://docs.openstack.org/heat/latest/ (besucht am 30. 01. 2022).

[148] OpenStack Foundation, *Welcome to Tacker Documentation*, Apr. 2019. Adresse: https://docs.openstack.org/tacker/latest/ (besucht am 30. 01. 2022).

[149] OpenStack Foundation, *OpenStackSDK Documentation, Release 0.62.0.dev121*, Apr. 2022. Adresse: https://docs.openstack.org/openstacksdk/latest/doc-openstacksdk.pdf (besucht am 02. 05. 2022).

[150] C. Bryant, J. Halterman, D. Szumski u. a., *Monasca API*, Nov. 2019. Adresse: https://github.com/openstack/monasca-api/blob/master/docs/monasca-api-spec.md (besucht am 30. 01. 2022).

[151] OpenStack, *Keystone Documentation, Release 21.1.0.dev12*, Apr. 2022. Adresse: https://docs.openstack.org/keystone/latest/doc-keystone.pdf (besucht am 02. 05. 2022).

[152] OpenStack, *Policies*, März 2022. Adresse: https://docs.openstack.org/security-guide/identity/policies.html (besucht am 29. 04. 2022).

[153] ONAP, *Introduction*, Apr. 2022. Adresse: https://docs.onap.org/en/latest/guides/onap-developer/architecture/onap-architecture.html (besucht am 02. 05. 2022).

[154] L. Deng, H. Deng und S. Terrill, *Harmonizing Open Source and Open Standards: The Progress of ONAP*, 2019. Adresse: https://www.onap.org/wp-content/uploads/sites/20/2019/04/ONAP_HarmonizingOpenSourceStandards_032719.pdf (besucht am 21. 02. 2022).

[155] B.-W. Jun, *ONAP Next Generation Security & Logging Architecture*, Jan. 2022. Adresse: https://wiki.onap.org/pages/viewpage.action?pageId=103417456 (besucht am 25. 02. 2022).

[156] ONAP, *OOM Cloud Setup Guide*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-oom/en/latest/oom_cloud_setup_guide.html (besucht am 02. 05. 2022).

[157] M. O'Brien, *Cloud Native Deployment*, Apr. 2022. Adresse: https://wiki.onap.org/display/DW/Cloud+Native+Deployment (besucht am 02. 05. 2022).

[158] ONAP, *Offered APIs*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-externalapi-nbi/en/latest/offeredapis/offeredapis.html (besucht am 02. 05. 2022).

[159] ONAP, *DMaaP Data Router API*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-dmaap-datarouter/en/latest/apis/data-router-api.html (besucht am 02. 05. 2022).

[160] ONAP, *TLS Support*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-dcaegen2/en/latest/sections/tls_enablement.html (besucht am 02. 05. 2022).

[161] ONAP, *ONAP MultiCloud Architecture*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-multicloud-framework/en/latest/MultiCloud-Architecture.html (besucht am 02. 05. 2022).

[162] ONAP, *Architecture*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-dcaegen2/en/latest/sections/architecture.html (besucht am 02. 05. 2022).

[163] ONAP, *Root Model*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-modeling-modelspec/en/latest/ONAP%20Model%20Spec/im/Root/Root.html (besucht am 02. 05. 2022).

[164] ONAP, *Modeling Spec Release Notes*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-modeling-modelspec/en/latest/Release-notes/release-notes.html (besucht am 02. 05. 2022).

[165] ONAP, *ONAP Vnfd Model*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-modeling-modelspec/en/latest/ONAP%20Model%20Spec/im/VNF/Vnfd.html (besucht am 02. 05. 2022).

[166] ONAP, *NetworkServiceDescriptorModel*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-modeling-modelspec/en/latest/ONAP%20Model%20Spec/im/NSD/NSD.html (besucht am 02. 05. 2022).

[167] ONAP, *Enhanced Nested Service Model*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-modeling-modelspec/en/latest/ONAP%20Model%20Spec/im/Service/ServiceModel-Nested%20Service.html (besucht am 02. 05. 2022).

[168] ONAP, *VES Model*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-modeling-modelspec/en/latest/ONAP%20Model%20Spec/im/VES/VES%20index.html (besucht am 02. 05. 2022).

[169] ONAP, *8.7. Service: VES Event Listener 7.1.1*, Okt. 2021. Adresse: https://docs.onap.org/projects/onap-vnfrqts-requirements/en/latest/Chapter8/ves7_1spec.html (besucht am 23. 02. 2022).

[170] ONAP, *License Management Model*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-modeling-modelspec/en/latest/ONAP%20Model%20Spec/im/License/LicenseModel.html (besucht am 02. 05. 2022).

[171] ONAP, *Business Interaction Model*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-modeling-modelspec/en/latest/ONAP%20Model%20Spec/im/Common/BusinessInteraction.html (besucht am 02. 05. 2022).

[172] ONAP, *AAF - Application Authorization Framework*, 2022. Adresse: https://docs.onap.org/projects/onap-aaf-authz/en/latest/index.html (besucht am 02. 05. 2022).

[173] ONAP, *Policy Framework Architecture*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html (besucht am 02. 05. 2022).

[174] ONAP, *Architecture*, Apr. 2022. Adresse: https://docs.onap.org/projects/onap-aai-aai-common/en/latest/platform/architecture.html (besucht am 02. 05. 2022).

[175] M. Liyanage, I. Ahmad, J. Okwuibe u. a., „Software Defined Security Monitoring in 5G Networks," *A comprehensive guide to 5G security*, Jg. 1, 2018.

[176] European Telecommunications Standards Institute, *ETSI GS NFV-IFA 015 V3.4.1 (2020-06). Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on NFV Information Model*, Juni 2020. Adresse: https://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/015/03.04.01_60/gr_NFV-IFA015v030401p.pdf.

[177] I. Oliver, S. Panda, K. Wang und A. Kalliola, „Modelling NFV concepts with ontologies," in *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, IEEE, 2018, S. 1–7.

[178] J. M. Sánchez Vı lchez, I. G. Ben Yahia, C. Lac und N. Crespi, „Self-modeling based diagnosis of network services over programmable networks," *International Journal of Network Management*, Jg. 27, Nr. 2, e1964, 2017.

[179] F. A. Lopes, L. Lima, M. Santos, R. Fidalgo und S. Fernandes, „High-level modeling and application validation for SDN," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2016, S. 197–205.

[180] J. van der Ham, F. Dijkstra, R. Łapacz und J. Zurawski, *Network Markup Language Base Schema version 1*, Mai 2013. Adresse: https://www.ogf.org/documents/GFD.206.pdf.

[181] M. Steinke und W. Hommel, „FEDCON: An embeddable Framework for Managing MOC Functions and Interfaces in Federated Software Networks," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, IEEE, 2021, S. 107–115.

[182] OpenDaylight Project, *OVSDB User Guide*, Jan. 2022. Adresse: https://docs.opendaylight.org/projects/ovsdb/en/latest/ovsdb-user-guide.html (besucht am 24. 01. 2021).

[183] M. Steinke und W. Hommel, „A Data Model for Federated Network and Security Management Information Exchange in Inter-Organizational IT Service Infrastructures," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2018.

[184] European Telecommunications Standards Institute, *ETSI GS NFV-IFA 027 V4.2.1 (2021-05). Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Performance Measurements Specification*, Mai 2021. Adresse: https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/027/04.02.01_60/gs_NFV-IFA027v040201p.pdf.

[185] European Telecommunications Standards Institute, *ETSI GS NFV-TST 008 V3.3.1 (2020-06). Network Functions Virtualisation (NFV) Release 3; Testing; NFVI Compute and Network Metrics Specification*, Juni 2020. Adresse: https://www.etsi.org/deliver/etsi_gs/NFV-TST/001_099/008/03.03.01_60/gs_NFV-TST008v030301p.pdf.

[186] *Interface Alarm*, Dez. 2021. Adresse: http://api.onosproject.org/2.7.0/apidocs/org/onosproject/alarm/Alarm.html (besucht am 02. 05. 2022).

[187] A. Willner, „Semantic-based Management of Federated Infrastructures for Future Internet Experimentation," Dissertation, Technische Universität Berlin, 2016. Adresse: https://depositonce.tu-berlin.de/bitstream/11303/5327/8/willner_alexander.pdf.

[188] A. Willner, R. Loughnane und T. Magedanz, „FIDDLE: Federated Infrastructure Discovery and Description Language," in *2015 IEEE International Conference on Cloud Engineering*, IEEE, 2015, S. 465–471.

[189] S. Son, H.-H. Choi, B. T. Oh, S. W. Kim und B. S. Kim, „Cloud SLA relationships in multi-cloud environment: models and practices," in *Proceedings of the 8th International Conference on Computer Modeling and Simulation*, 2017, S. 1–6.

[190] R. Latif, S. H. Afzaal und S. Latif, „A novel cloud management framework for trust establishment and evaluation in a federated cloud environment," *The Journal of Supercomputing*, Jg. 77, Nr. 11, S. 12 537–12 560, 2021.

[191] C. Bezemer und A. Zaidman, „Challenges of reengineering into multi-tenant SaaS applications," *Technical Report Series TUD-SERG-2010-012*, 2010.

[192] N. Muhammad, N. Boucke und Y. Berbers, „Using the Parallelism Viewpoint to Optimize the Use of Threads in Parallelism-intensive Software Systems," in *In proceedings of the International Conference on Software and Computing Technology (ICSCT 2010)*, IEEE, 2010.

[193] D. B. Hoang und M. Pham, „On Software-defined networking and the design of SDN Controllers," in *2015 6th International Conference on the Network of the Future (NOF)*, IEEE, 2015, S. 1–3.

[194] D. Box, D. Ehnebuske, G. Kakivaya u. a., *Simple Object Access Protocol (SOAP) 1.1*, Mai 2000. Adresse: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (besucht am 05. 05. 2022).

[195] S. Tarkoma, *Publish / Subscribe Systems: Design and Principles*. John Wiley & Sons, 2012.

[196] G. Gardikis, I. Koutras, G. Mavroudis u. a., „An integrating framework for efficient NFV monitoring," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, IEEE, 2016, S. 1–5.

[197] T. Kohler, F. Dürr und K. Rothermel, „ZeroSDN: A Highly Flexible and Modular Architecture for Full-Range Distribution of Event-Based Network Control," *IEEE Transactions on Network and Service Management*, Jg. 15, Nr. 4, S. 1207–1221, 2018.

[198] J. Halterman, *Cluster Configuration in Owl (1.14)*, Nov. 2018. Adresse: https://wiki.onosproject.org/pages/viewpage.action?pageId=28836788 (besucht am 07. 02. 2020).

[199] T. Kim, S.-G. Choi, J. Myung und C.-G. Lim, „Load balancing on distributed datastore in opendaylight SDN controller cluster," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, IEEE, 2017, S. 1–3.

[200] M. T. González-Aparicio, M. Younas, J. Tuya und R. Casado, „A New model for Testing CRUD Operations in a NoSQL database," in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, IEEE, 2016, S. 79–86.

[201] R. Deari, X. Zenuni, J. Ajdari, F. Ismaili und B. Raufi, „Analysis And Comparision of Document-Based Databases with Relational Databases: MongoDB vs MySQL," in *2018 International Conference on Information Technologies (InfoTech)*, 2018, S. 1–4. DOI: 10.1109/InfoTech.2018.8510719.

[202] M. Steinke, I. Adam und W. Hommel, „Multi-Tenancy-Capable Correlation of Security Events in 5G Networks," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, IEEE, 2018, S. 1–6.

[203] A. Campanella und S. Condon, *Using an IDE with ONOS 1.14 or higher (Bazel build)*, Apr. 2020. Adresse: https://wiki.onosproject.org/pages/viewpage.action?pageId=28836246 (besucht am 23. 03. 2022).

[204] Oracle, *Class LinkedBlockingQueue<E>*, Jan. 2021. Adresse: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/LinkedBlockingQueue.html (besucht am 23. 03. 2022).

[205] Oracle, *Interface BlockingQueue<E>*, Jan. 2021. Adresse: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/BlockingQueue.html (besucht am 23. 03. 2022).

[206] Oracle, *Class Executors*, 2018. Adresse: https://cr.openjdk.java.net/~iris/se/11/latestSpec/api/java.base/java/util/concurrent/Executors.html#newCachedThreadPool() (besucht am 23. 03. 2022).

[207] OpenDaylight Project, *Flow Examples*, März 2022. Adresse: https://docs.opendaylight.org/projects/openflowplugin/en/latest/users/flow-examples.html (besucht am 02. 05. 2022).

[208] OISF, *Eve JSON Format*, Okt. 2020. Adresse: https://suricata.readthedocs.io/en/suricata-6.0.0/output/eve/eve-json-format.html (besucht am 28. 03. 2022).

[209] T. Williams, C. Kelley u. a., *gnuplot 5.2*, Dez. 2019. Adresse: http://www.gnuplot.info/docs_5.2/Gnuplot_5.2.pdf (besucht am 05. 12. 2021).

[210] Oracle, *Class ConcurrentHashMap<K,V>*, März 2022. Adresse: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html (besucht am 02. 05. 2022).

[211] Oracle, *Class System*, Juni 2020. Adresse: https://docs.oracle.com/javase/7/docs/api/java/lang/System.html (besucht am 18. 11. 2021).

Abbildungsverzeichnis

1.1 Überblick über die Architektur mit beispielhaften Komponenten in FSNs. 3
1.2 Vorgehensweise in dieser Arbeit. 10

2.1 Arten der Systemvirtualisierung im Vergleich zu einem traditionellen System ohne Virtualisierung [14]. 13
2.2 Dreischichtige SDN-Referenzarchitektur mit jeweiligen Komponenten [3]. 14
2.3 NFV-Referenzarchitektur [6]. 16
2.4 Generelle Zusammenhänge von Komponenten im Netzmanagement basierend auf [8]. 32

3.1 Übersicht und Gruppierung der Akteure im Netzmanagement in FSN (Hellblau: Gruppierungsknoten; Dunkelblau: Akteure). 39
3.2 Exemplarische mehrebnige IT-Infrastruktur für Szenario 1. Die Ebenen beschreiben aufeinander aufbauende Virtualisierungsschichten. 45
3.3 Beziehungen und generelle Infrastruktur der Föderationspartner bei SNs in medizinischen Anwendungen. 58
3.4 FSN-Infrastruktur und Komponenten am Beispiel eines konkreten Patienten. . . 60
3.5 Domänenstruktur in Szenario 2. 63
3.6 Übersicht über Föderations-, Infrastruktur- und Organisationsaspekte in Szenario 3. 72

4.1 Genereller Aufbau eines CADF-Ereignisses mit Beispielbelegungen [84]. 105
4.2 Architektur von OpenDaylight 12 [90]. 107
4.3 Übersicht über die Architektur und Dienste von ONOS [97]. 110
4.4 ONOS Subsysteme und Kommunikationswege [97]. 110
4.5 OSM-Systemarchitektur und -Dienste [116]. 115
4.6 Logische Sicht der Architektur von OpenStack am Beispiel einiger Dienste [135]. 124
4.7 Architektur von ONAP, basierend auf [153]. 128

5.1 Kernprozesse des Managements (Weiße Box: Teil-Prozess; Blaue Box: Aktivität). 136
5.2 Prozess der Datenakquise innerhalb des Überwachungsprozesses (Weiße Box: Teil-Prozess; Blaue Box: Aktivität). 137
5.3 Prozess der Datenauswertung innerhalb des Überwachungsprozesses (Weiße Box: Teil-Prozess; Blaue Box: Aktivität). 138
5.4 Prozess der Anfrage innerhalb des Steuerungsprozesses (Weiße Box: Teil-Prozess; Blaue Box: Aktivität). 139
5.5 Aktivitäten des Prozesses der Umsetzung innerhalb des Steuerungsprozesses. . 139
5.6 Modellierungsbeispiel mit Bedeutung der verwendeten Elemente. 140
5.7 Managementobjektklassen (MOCs) und MOC-Klassifikationen (MOCKs) mit Ableitungsbeziehungen. Nur nicht-abstrakte MOCKs können zur Klassifikation von MOCs genutzt werden. Die Klassifikation wird durch dynamische Zuweisung über eine Verbindungsklasse `Klassifikator` durchgeführt. 143

5.8 Übersicht über das vererbungsbasierte Modell für Managementbeziehungen. Die Klassen Abhängigkeit, AVertikal und AHorizontal können nicht direkt instanziiert werden. Die Klassen AHNEthernet und AHNIP gehören nicht zum Framework, sondern stellen Beispielableitungen dar. 151
5.9 Vertikales Durchlaufen mit Funktionen hoch / runter in Kind-Klassen von AVertikal. Die Kanten repräsentieren die Abhängigkeitsobjekte. 151
5.10 Illustration der Funktionen nächsteSub und nächsteObj am Beispiel von Netzabhängigkeiten AHNetz zwischen Switches und Instanzen. Die Kanten verbildlichen die Abhängigkeitsobjekte. 152
5.11 Darstellung des MOCTypeModels aus [181]. Durchgezogene Linien implizieren primäre Eltern-Kind-Vererbung, Strichlinien sekundäre Vererbungsbeziehungen. 154
5.12 Klassifizierung von FSNMOCK-Instanzen mit MOCFunktion-Instanzen. 155
5.13 Zusammenhang zwischen einem MOCK, einer MOC-Funktion und einem MOC-Funktionstreiber. Der MOC-Funktionstreiber implementiert eine MOC-Funktion für ein MOCK und ist für davon abgeleitete MOCKs anwendbar. 157
5.14 Darstellung vom Funktionsaufruf am MOC-Funktionstreiber für eine generische Klasse T über das SBI-Register zum jeweiligen Konnektor. Die grünen Elemente stellen Schnittstellenklassen zum Aufruf bzw. der Rückgabe der jeweiligen Komponente dar. 158
5.15 Netzereignis und Netzzustand erben Attribute von Netzinformation und vererben sie selbst wiederum an benutzerdefinierte Subklassen. 162
5.16 Alarm-Kernelement AlertType mit einem beispielhaft davon abgeleiteten Alarmsystem. Abbildung basierend auf [183]. 163
5.17 Übersicht über Elemente zur Beschreibung von Föderationen und darin vorkommenden Beziehungen. 165
5.18 Elemente zur Beschreibung einer administrativen Domäne (Domäne) sowie von Beziehungen zwischen Domänen (Domänenbeziehung). Ein Verbund von administrativen Domänen (Domänenverbund) wird als spezielle Domänenbeziehung davon abgeleitet. 167
5.19 Funktionsprinzip der Domänenkreuzorganisationstabelle zur effizienten Verwaltung von Domänenüberschneidungen. Für jedes DKOE wird ein Index-Wert DKOEI berechnet, der eine effiziente Suche erlaubt. 169
5.20 Elemente zur Beschreibung von Nutzer-Entitäten und -Rollen in einer Managementplattform. 171
5.21 Das Element Aufgabenbereich fasst mehrere Aufgabe-Elemente innerhalb einer Föderation zusammen. Das Aufgabenbereich-Element ist an das Föderation-Element (grau gekennzeichnet) aus dem Framework für Föderationsmodelle geknüpft. 173
5.22 Zuständigkeit stellt eine Richtlinie zur Ausprägung organisatorischer Strukturen dar. Sie wird entweder auf die gesamte Föderation oder auf einzelne Domänen angewendet. 174
5.23 Beispielhafte Definition von Aufgaben und Rollen, mit einer föderationsweiten Zuweisung von Zuständigkeiten und einer separaten Zuweisung für Domäne d1. Die farbliche Kodierung der Assoziationen dient der Übersichtlichkeit. 175
5.24 Beispielhafter Zugriffsverwaltungsbaum als Überprüfungspfad mit Kriterien als ZVB-Knoten und jeweiligen Folgen von ZVB-Elementen. Die Farben verdeutlichen die Kriteriengruppen. 178
5.25 Beispiel relevanter Kriterien (visuell hervorgehoben) der Sichtbarkeits- und Zugriffsermittlung im Zugriffsverwaltungsbaum auf ein Informationselement *X* für einen fiktiven Nutzer *Alice*. 180
5.26 Übersicht über zentrale und schnittstellenspezifische Komponenten des Kommunikationsmodells. Die Komponente *Funktionen* ist Teil des Funktionsmodells und beschreibt Funktionen der Managementplattform. 181

5.27 Architektursicht auf Komponenten, Schnittstellen und Datencontainer (SBIErweiterteRohdaten, SBIDatenobjekt und SBINetzinformation) im SBI. Gestrichelte Linien deuten einen Datenfluss mit jeweiligem Datencontainer an. . . 182
5.28 Funktionsweise des Parser-Pools als Tabelle: Parser (hier beispielhaft) werden über Datenformat-Instanzen ausgewählt. 183
5.29 Funktionsweise des Normalisierer-Pools, illustriert an einem Beispiel. Die Verwaltung der Normalisierer wird durch die drei Kriterien der FSNMOCK, Objekttypen und Normalisierungsmarken begründet. Die Farben verdeutlichen die unterschiedlichen Auswahlkriterien. 185
5.30 Vereinfache beispielhafte Darstellung der Weiterreichung einer Normalisierungsmarke (orange) von einer MOC-Funktionsdefinition an einen Konnektor (Pull-Zugriff) zur gemanagten Infrastruktur. Dieser generiert aus den empfangenen Daten eine SBIErweiterteRohdaten-Instanz mit der jeweiligen Normalisierungsmarke. 186
5.31 Illustration der Funktionen des SBI-Registers: Konnektor-Auswahl (grün), Auswahl des Verarbeitungsgrads der Rückgabe (rot), Bypass-Möglichkeit bei Speicherung (violett). Die Nummerierung der Übergänge zeigt die Bearbeitungsreihenfolge auf. 187
5.32 Ausschnitt wichtigster Komponenten des NBI und ihrer Zusammenhänge. 189
5.33 Architektur eines Managementplattform-Verbunds, basierend auf [181]. Managementplattformen kommunizieren darin über ihr jeweiliges NBI, über das auch Managementanwendungen (MAPPs) innerhalb der Datenzentren an Managementplattformen angebunden sind. 196
5.34 Sequenzdiagramm zur Illustration eines remoten Pull-Zugriffs im Managementplattform-Verbund. Managementplattform MP_A leitet die Anfrage an die zuständige Managementplattform MP_B weiter und gibt die Rückgabe an die anfragende Managementanwendung zurück. 198
5.35 Sequenzdiagramm zur Verdeutlichung des Ablaufs des Abonnements von Push-Benachrichtigungen und die Weiterleitung von Push-Benachrichtigungen im Verbund. 199
5.36 Beispielumsetzung von Privilegien durch die abstrakte Klasse PrivilegienBaustein. 200
5.37 Effiziente Verwaltung zentraler Komponenten von Push-Benachrichtigungen am NBI. Prioritätenklassen (grün) verweisen auf Klassen von Informationselementen aus den jeweiligen NBIPushKriterium-Instanzen, diese auf NBI-Bausteine und diese wiederum auf Callback-Funktionen bzw. remote Endpunkte. 201
5.38 Übersicht über Komponenten der Datenbankanbindung. 203
5.39 Komponenten zur Überführung eines SBINetzinformation-Elements zu Modifikationsanfrage-Elementen. 204
5.40 Ablauf der Überwachung von Speicherelementen durch die Datenzugriffsverwaltung und Benachrichtigung des NBI. 206
5.41 Mit der Klasse FSNMOCK verknüpftes Credentials-Element und davon beispielhaft abgeleitete spezifische Authentifizierung-Informationstypen. 207
5.42 Anfrage eines geeigneten Konnektors für NBI-Push- und SBI-Pull-Funktionalität über das NBI- respektive SBI-Register und damit je verknüpfte Konnektoren. . . 209
5.43 Integration von Schnittstellen in das SBI und NBI. Das SBI- und NBI-Register verweisen auf genutzte Schnittstellen und installieren eine Callback-Funktion. . 210
5.44 Die Konkretisierung und Organisation von Funktionsbereichen wird durch die bereits definierten Aufgabenbereiche aus dem Organisationsmodell vorgenommen. Eine Funktionsbereich-Instanz kann auf beliebig viele Aufgabenbereich-Instanzen verweisen. 211
5.45 Beschreibung von Plattformfunktionen. Die Zugreifbarkeit auf eine Plattformfunktion wird an eine oder mehrere Rollen gekoppelt. Plattformfunktionen werden durch Treiber implementiert, die als Elternklasse der Treiber für MOC-Funktionen dienen. 213

5.46 Vereinfachter Ablauf einer automatisierten Überwachung (grüne Kanten) und darauf aufbauender automatisierter Steuerungsoperationen (blaue Kanten). . . 214

6.1 Verzeichnisstruktur der Implementierung und Einbettung in ONOS. 222
6.2 Paketstruktur der Southbound-Interface-Implementierung. 231
6.3 Vereinfachter grundlegender Ablauf des Prozesses zur Verarbeitung von Daten des SBI. 232

7.1 Übersicht über das Baseline-Evaluationsszenario. 242
7.2 Übersicht über die resultierenden Klassifikator-Klassen zur Beschreibung der MOs des Anwendungsszenarios. 243
7.3 Beispielhafte Klassifizierung einer MO-Repräsentation als OpenDaylight-Controller und von diesem implementierte Funktionen als OpenFlow- und OVSDB-Controller. 246
7.4 Beispielhafte Steuerungs- und Netzabhängigkeiten zwischen MOs. 247
7.5 Klassenstruktur der Erweiterungen von Abhängigkeiten. 248
7.6 Durch MOC-Funktion `addSrcBlockingFlow` generierter Flow in einer Open vSwitch-Instanz in einer Mininet-Testumgebung. 252
7.7 Beispielhaft implementiertes Alarm-Schema aus [183]. Die blaue Box entspricht dem Frameworkelement, weiße Boxen sind davon abgeleitete Klassen. 255
7.8 Umzusetzender Zugriffsentscheidungsbaum für das Evaluationsszenario. 264
7.9 Beispielhaft implementierte Elemente im SBI-Verarbeitungsprozess. 270

8.1 Dauer zur Ermittlung zugreifbarer `InformationTag`-Instanzen für einen Nutzer über verschiedene Domänen des Evaluierungsszenarios. 280
8.2 Übersicht über den Benachrichtigungsprozess mit Evaluierungsparametern. . . 281
8.3 Einfluss der Anzahl initial verarbeiteten `SBIExtRawData`-Elemente auf die Performanz des Benachrichtigungsprozesses. 285
8.4 Einfluss der Anzahl der Pusher-Threads auf die Performanz des Benachrichtigungsprozesses. 286
8.5 Einfluss der Verzögerung pro `SBIExtRawData`-Instanz auf die Performanz des Benachrichtigungsprozesses. 287
8.6 Einfluss des Anteils der initial priorisierten `SBIExtRawData`-Instanzen auf die Performanz des Benachrichtigungsprozesses. 288
8.7 Einfluss des am SBI eingesetzten Threadpoolmodells auf die Performanz des Benachrichtigungsprozesses. 289
8.8 Verfeinerte Analyse des Einflusses des SBI-Threadpoolmodells auf die Performanz mit fünf Pusher-Threads und ohne Push-Verzögerung. 290
8.9 Einfluss des Threadpoolmodells des NBI auf die Performanz des Benachrichtigungsprozesses. 290
8.10 Einfluss des Betriebsmodus auf die Performanz des Benachrichtigungsprozesses. 291
8.11 Einfluss des SBI-Priorisierungsanteils auf die Performanz des Benachrichtigungsprozesses. 292
8.12 Einfluss der Priorisierung von Push-Benachrichtigungen am NBI auf die Performanz des Benachrichtigungsprozesses. 293
8.13 Dauer zur Identifikation und der Anfrage einer `NBIPullComponent`-Instanz auf Basis ihrer ID. 294
8.14 Ausführungszeiten von lokalen oder remoten Pull-Zugriffen für eine `ODLBlockMAC`-Instanz. 295

Tabellenverzeichnis

2.1 Harmonisierte Übersicht über Charakteristika von Föderationen aus der Literatur über thematisch verwandte Bereiche (Blau gefärbte Zellen entsprechen einer Berücksichtigung). . . . 21
2.2 Relevanz und Ausprägung identifizierter Charakteristika von Föderationen in SNs (Blau: verschiedene Ausprägungen; Grau: keine Relevanz; Weiß: statische Ausprägung). . . . 22

3.1 Vorlage einer Morphologie von Föderationen in SNs zur Einordnung der Szenarien. 44
3.2 Einordnung von Szenario 1. . . . 56
3.3 Rollen im Netzmanagement in Szenario 2. . . . 65
3.4 Einordnung von Szenario 2. . . . 71
3.5 Einordnung von Szenario 3. . . . 77
3.6 Nicht-Funktionale Anforderungen an Managementplattformen. . . . 82
3.7 Anforderungen an das Funktionsmodell. . . . 83
3.8 Anforderungen an das Informationsmodell. . . . 86
3.9 Anforderungen an das Kommunikationsmodell. . . . 88
3.10 Anforderungen an das Organisationsmodell. . . . 90
3.11 Anforderungen an die Umsetzung der Frameworks. . . . 92

4.1 Bewertung bestehender Ansätze auf Basis der Anforderungen an Managementplattformen in FSNs (Dunkelblau: erfüllt; Hellblau: teilweise erfüllt; Weiß: nicht erfüllt). . . . 134

5.1 Standardattribute generischer MOCs sowie MOCKs in FSNs. Einrückungen entsprechen einer Vererbungshierarchie. . . . 146
5.2 Notwendige Attribute in Netzereignissen und -Zuständen. . . . 161

7.1 Berücksichtigung von Charakteristika des Evaluationsszenarios (Dunkelblau: Baseline-Charakteristika; Hellblau: berücksichtigte Alternativen; Weiß: nicht-berücksichtigte Ausprägungen). . . . 240

8.1 Übersicht über Parameterausprägungen zur Evaluierung des Benachrichtigungsprozesses. Fett gedruckte Ausprägungen stellen die Basiseinstellung dar, die bei Variation eines anderen Parameters gilt. . . . 283
8.2 Optimale Konfiguration des Testsystems für Closed-Loop-Automatisierung nach Generierungsdauer pro Benachrichtigung. . . . 296
8.3 Übersicht über die Erfüllung von Anforderungen des Konzepts dieser Arbeit (Dunkelblau: erfüllt und vorgesehen; Hellblau: erfüllt, aber abhängig vom Entwickler; Grau: nicht entscheidbar; Weiß: nicht erfüllt). . . . 298

Anhang A

Veröffentlichungen im Kontext dieser Arbeit

Im Folgenden werden die themenbezogenen wissenschaftlichen Veröffentlichungen aufgelistet, die während der Arbeit an dieser Dissertation entstanden sind. Publikationen ohne direkten Bezug zu dieser Dissertation werden nicht aufgelistet.

2018

Titel:	A Data Model for Federated Network and Security Management Information Exchange in Inter-Organizational IT Service Infrastructures
Autoren:	Michael Steinke, Wolfgang Hommel
Konferenz:	IEEE/IFIP Network Operations and Management Symposium (NOMS)

Titel:	Multi-Tenancy-Capable Correlation of Security Events in 5G Networks
Autoren:	Michael Steinke, Iris Adam, Wolfgang Hommel
Konferenz:	IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)

Titel:	Overcoming Network and Security Management Platform Gaps in Federated Software Networks
Autoren:	Michael Steinke, Wolfgang Hommel
Konferenz:	International Conference on Network and Service Management (CNSM)

Titel:	Policy-Based Network and Security Management in Federated Service Infrastructures with Permissioned Blockchains
Autoren:	Michael Grabatin, Wolfgang Hommel, Michael Steinke
Konferenz:	International Symposium on Security in Computing and Communication

2021

Titel:	FEDCON: An embeddable Framework for Managing MOC Functions and Interfaces in Federated Software Networks
Autoren:	Michael Steinke, Wolfgang Hommel
Konferenz:	IFIP/IEEE International Symposium on Integrated Network Management (IM)