UNIVERSITÄT DER BUNDESWEHR MÜNCHEN

FAKULTÄT FÜR BAUINGENIEUR- UND VERMESSUNGSWESEN

On Interoperable Management of Multi-Sensors in Landslide Monitoring Applications

Admire M. Kandawasvika

Vollständiger Abdruck der von der Fakultät für Bauingenieur- und Vermessungswesen der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines Doktor-Ingenieurs (Dr.-Ing.) genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr.-Ing. Wilhelm Caspary
1. Berichterstatter:	Univ.-Prof. Dr.-Ing. Wolfgang Reinhardt
2. Berichterstatter:	Univ.-Prof. Dr. Lars Bernard

Diese Dissertation wurde am 30. April 2009 bei der Universität der Bundeswehr München eingereicht.

Tag der mündlichen Prüfung: 21. September 2009

Abstract

Modern geoscientific monitoring and early warning applications have strong need for reliable, accurate, and timely live sensor data, in order to continuously keep track of the geographic-based events of interest currently happening in their physical world.

Due to advancements of sensor technology resulting in the availability of low-cost and miniaturized sensors, it is possible for a single monitoring application to utilize large numbers of sensors in the order of tens, hundreds or even thousands. The major problem is how to integrate or manage simultaneously these various, heterogeneous sensors within that single application space. Furthermore, how can monitoring applications exchange any sensor systems without being worried about integration problems and conflicts that arise due vendor-specific solutions (i.e. usage of proprietary communication protocols, interfaces, measurements data types and formats). In addition, how can people and applications better understand the sensors they are working with, and make sense of the provided sensor measurements data.

This dissertation proposes a concept for interoperable management of heterogeneous, multi-sensors within landslide monitoring applications. The management focuses on *loose integration* of sensors, which includes sensor connection, command and control, as well as handling of different sensor measurements at syntactic and semantic levels. The concept provides a new perspective for model-based management or integration of sensors and also defines a generic minimal specification of a sensor access and control service (SACS). The SACS tends to complement the existing specifications which are either very general (i.e. global in design) or based on proprietary solutions (e.g. which is the current situation in landslide monitoring applications). A common SACS should enable interoperable integration of sensors in different applications. We also suggest that it is through generic logical sensor models that people and applications can easily integrate and better understand the types of sensors they are working with as well as the acquired measurements data.

The definition and specification of generic and standards-based sensor models, as well as open, interoperable interfaces for sensor access and control services are important first steps towards achieving interoperable management of multi-sensors in landslide monitoring applications.

Adopted for the proof of concept is a sophisticated, open standards-based field-based mobile data acquisition system we have developed for landslide monitoring. This system makes fully use of the possibilities of ubiquitous access – via wireless technologies – to various sources of information. The system's mobile client employs different sensor systems for the capturing of new geometries of features as well as updating geometries of existing features.

CONTENTS

List of Figures

List of Tables

1. Introduction

Due to the increasing number of disasters or accidents as a result of the frequent occurrences of unavoidable natural hazards like landslides, floods, fires, tornadoes or wind storms, etc., there is a strong need to build or deploy effective, efficient and highly reliable monitoring and early warning systems. These systems can help humans to fully understand and respond timely and adequately to expected or abrupt events in order to mitigate their effects. The backbone of such systems is a *sensor network* that provides reliable, accurate and timely live sensor measurements data.

There are many deployments of monitoring and early warning systems being carried out at global or international [EEA, 2004; EU-GMES, 1998][…], national, regional and local levels. The effective realisations of such systems are enabled by advances in sensor technologies (e.g., Micro-Electro-Mechanical Systems (MEMS) and Nanotechnology [http://www.memsnet.org/], etc.), and information and communication technologies (ICT) like interoperability technologies (i.e., Open Geospatial Consortium (OGC) Sensor Web Enablement [OGC-SWE], Ork Ridge National Laboratory (ORNL) Sensor Net [ORNL-SN], Web Services and Service Oriented Architectures [OGC, W3C, OASIS][…], etc.), the Internet and the Web, and high-speed, broadband wireless communication.

With the help of sensor networks, people and machines can monitor environments of interest or even continuously keep track of events happening in their physical world by accessing and/or controlling different types of sensors from their offices, homes or in the field. Due to the availability of low-cost and miniaturized sensors, it is possible for a single monitoring application to utilize large numbers of sensors in the order of tens, hundreds or even thousands. The major problem is how to integrate or simultaneously use these various, heterogeneous sensors within a single application space. Furthermore, how can monitoring applications exchange any sensors without being worried about integration problems due to differences in communication protocols and interfaces. Also, how can people and applications better understand the sensors, and utilise their measurements data with disregard to the differences in data formats. Current research works are investigating these challenges at various levels and from different perspectives (e.g. sensor data collection services, sensor network configuration, and management, etc.), and for different application domains.

This dissertation proposes a concept for interoperable management of heterogeneous, multi-sensors within landslide monitoring applications. The management focuses on *loose integration* (see section 2.1.1), which includes sensor access (connection) and interaction (command and control), as well as handling of sensor measurements. The concept provides a new methodology for model-driven management of sensors and also defines a generic minimal specification of a sensor access and control service (SACS). The SACS tends to complement the existing specifications which are either very general (i.e. global in design) or based on proprietary solutions (e.g. which is the current situation in landslide monitoring applications). It is through use of generic sensor models that people and applications can easily integrate and better understand the types of sensors they are working with as well as the measurements data. Through an open, standards-based or generic framework, it can be possible to connect and control various disparate in-situ and remote sensors. Also, within a sensor network, specialized sensors can provide information about their positions making it even possible for the users to geolocate (georeference) the events happening in the physical world.

1.1. Problem Definition

Modern geoscientific (geodetic, geotechnical, geophysical, etc.) monitoring applications have strong need for reliable, accurate, and timely live sensor data, in order to be always aware or keep track of the geolocatable events of interest happening in the physical world. However, the challenges in deploying sensor networks for those applications are mainly sensor and sensor data management [Stefanidis et al., 2005, Balanziska et al., 2007]. Dealing with various, heterogeneous sensors within a single monitoring application and, at the same time, providing meaningful information to the end user for analysis and decision making is a very challenging task. The main reason attributing to this challenge is that sensor manufacturers support their own proprietary solutions (i.e. non-standard, vendor-specific specifications). Hence, there is need for interoperable integration or management of multi-sensors.

Definition:: Multi-Sensor Integration: Luo et al. defines it as the *synergistic* use of information provided by multiple sensory devices to assist in the accomplishment of a task by a system [Luo et al., 2002].

Definition:: Interoperable Management of Multi-Sensors: Is defined in the dissertation as the open standards-based or generic access and control of multiple sensory devices as well as the synergistic use of data provided by them, in order to accomplish easily and efficiently a given task within a single application space or system.

The dissertation classifies the problem definition into two layers: (1.) sensor systems layer and (2.) sensor measurements data layer. Figure 1 shows the issues of concern that a system developer or integrator faces when integrating multi-sensors within a single application.

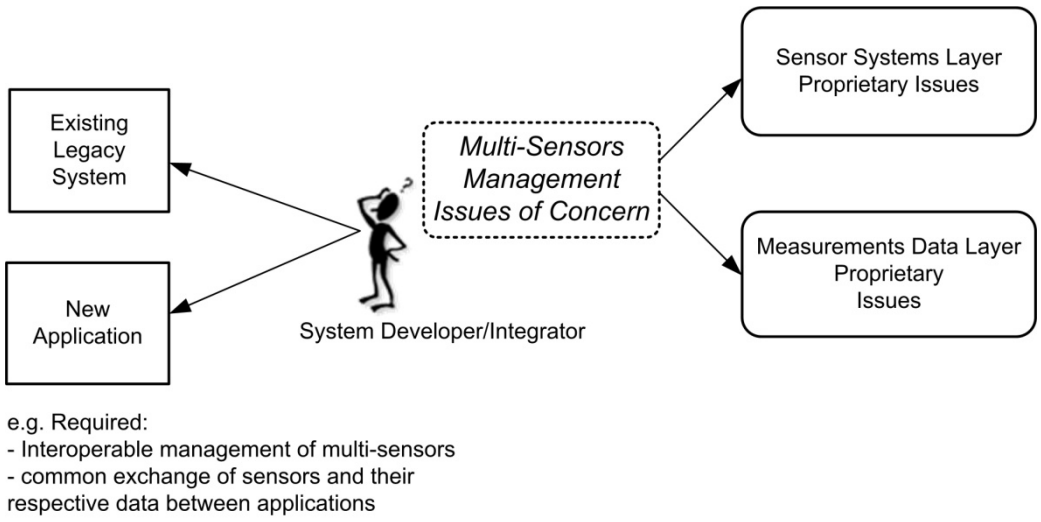

Figure 1: Overview of Issues of Concern in Multi-Sensors Management.

The problem definition matrix shown in the Figure 2 elaborates the above figure.

	Issue	Focus
Sensor Systems Layer	Connection	Resolving differences in vendor-specific *protocols* for connecting sensors.
	Command and control	Resolving differences in vendor-specific *interfaces* for accessing and controlling sensors.
	Sensor types (simple to complex)	Defining sensor level of abstraction (i.e. functional description details)
Sensor Measurements Data Layer	Data types and structures	Handling different data types & structures, i.e. considering syntactical conflicts
	Data representations (formats)	Resolving different data formats.
	Data ontologies	Resolving data misinterpretations due to different meanings.
	Data quality	Considering differences in data quality.

Figure 2: Problem Definition Matrix

Sensor Systems Layer

Due to the advances in sensor technology (MEMS, Nanotechnology), computing (hardware, software and algorithms), and wireless communication, applications can deploy a large number of sensors (ranging from low-cost to very expensive) of different capabilities, and sizes (e.g., smart dusts [Sailor et al., 2005; Warneke et al., 2001], micro-drones [http://www.microdrones.com/], etc.). Various problems can be encountered when communicating with those disparate sensor devices in a single application. The sensor systems layer deals with the problems related to the inherent differences in functional model (capability, behavior and characteristics) of a sensor. This specifically includes the connection and communication (*command and control*) management problems that arise due to discrepancies in the specifications of sensor protocols and interfaces by different manufacturers, which make it not easy for a single application to efficiently integrate (plug-and-play) new sensors.

Connection protocols refer to the communication mechanism of a system at the application or web service layer. For example a sensor system can allow particular protocols like:

- ASCII Remote Procedure Call (RPC) command (e.g. input syntax like:

  $: command code, <parameter$_1$>, <parameter$_2$>, <parameter$_n$>)

- DCOM (Distributed Component Object Model)[Microsoft-DCOM]
- CORBA (Common Object Request Broker Architecture)[OMG]
- Etc.

The command and control interfaces form the *syntax part* of the communication contract between a sensor system and the target sensor application (e.g., sensor manager). Following are the specific problems that this dissertation tries to solve:

- The sensor manufacturers do not exactly define or follow standard interfaces for connecting, commanding and controlling their sensors at the application, service, or presentation layer (e.g. application layer defined by the OSI/ISO model [ISO/IEC-7498-1, 1994], OGC Sensor Web Enablement [OGC-SWE], IEEE P1451 for Smart Transducer Interfaces [IEEE-1451]). Therefore, there is need for a vendor-neutral solution for integrating multi-sensors in landslide monitoring applications.

- Lack of common sensor models that applications can use as single portals or query interfaces for accessing all the relevant information about a particular type of a sensor.

- There is no standard definition of a sensor model available

The current situation is that for each sensor (S), for example, total station (TPS), digital camera, laser scanner, and global navigation satellite system (GNSS) device, a specific software component (SC) is needed to properly manage that single sensor (see Figure 3).

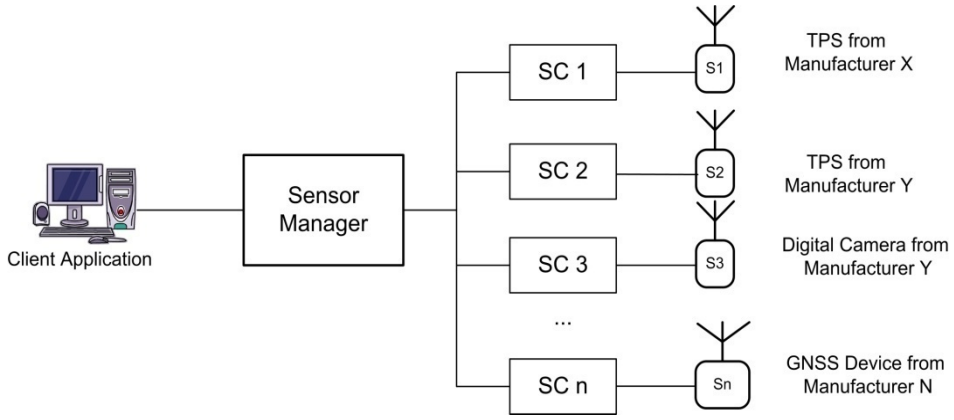

Figure 3: A Specific Sensor Component for each Sensor.

With increase in the number of sensors, the sensor manager can be overwhelmed. The dissertation considers this problem.

Lack of detailed and well-defined sensor models causes system integrators and analysts have problems in fully understanding the sensors and their respective measurements. This makes sensor systems to be treated as “black boxes”. A *black box* refers here to a closed (sensor) system whose descriptive information is not adequate. Following two are examples of black box problems:

- The mapping between the external inputs (i.e. sensor signal input) and the internal behaviour model (e.g., processing model) of the component is unknown.

- The mapping between the internal behaviours and the measurable external results is also unknown.

With increase in complexity of some of the black-boxed sensors, if errors occur sometimes it is difficult to determine the source of errors (i.e. system internal or human failure).

Sensor Measurements Data Layer

Balanziska et al. notes that large-scale sensor networks generate tremendous volumes of priceless data [Balanziska et al., 2007]. Harvesting the benefits of a sensor-rich world is not easy, due to data management challenges. These management problems can generally be split into two levels: *sensor services level* and the *client applications level* (e.g., landslide monitoring applications). The client applications can deal with issues regarding handling of vast amounts of data, detection of outliers using various algorithms, and data fusion. The sensor services can perform the initial or preparatory data management like handling of different *proprietary data structures and types of different dimensions* (spatial including the temporal aspect), *discrepancies in data quality* (determine sensor reliability, failure, etc.) and *complexity of data content formats*.

The sensor measurements layer, which is the other focus of the dissertation, considers only the sensor services level issues. Tao et al states that the data layer forms the backbone of a system and it can serve as a gateway to integrate and fuse observations of spatially referenced sensors [Tao et al., 2003]. Since a single monitoring application can use different sensors (ranging from very simple ones like temperature, pressure, etc., to complex sensors like GNSS receivers, laser scanners, etc.), various data types, such as scalar data (e.g., count, range, distance), vector data (e.g. point, line, polygon), raster and complex matrices, with different dimensions and qualities, have to be handled by the application. *Data types* refer here to data class definitions and *data structures* are compositions of data types (simple, complex or mixed types), for example, data tables, data records, data arrays, terrain, etc. *Data formats* refer to the organization of data, for example, in ASCII-based, XML (text) or binary formats. It is important to note that data definitions do form both the *syntax* and *semantics part* of communication between sensor systems and the targeted applications. Therefore any differences in meaning of same terminology usually lead to data misinterpretation (i.e. ontology conflicts). The dissertation looks at the semantics and syntactic issues, but does not address any data fusion techniques.

To summarise the problem definition, from research analysis and our experiences in a joint project "Advances of Geoservices" (http://www.geoservies.uni-osnabrueck.de/), in which a sophisticated mobile data acquisition client has been developed for landslide monitoring applications, we have identified the following:

- Sensor measurements are archived, but the information about the sensors (i.e. sensor models) used to capture the data is forgotten or does not exist at all. For example, for future analysis of the measurements, the sensor information at the time of observation (e.g. calibration state data, atmospheric conditions, accuracy, position, process methods etc.) can be of vital importance. Roddick et al. points out that the modelling of spatial data sources is as important as the data themselves if such data are to be interpreted correctly [Roddick et al., 2004]. Lack of sensor models will, few years after the measurement campaign, result in measurements "data graveyards".

- Lack of a common data formats. For example, if within the landslide monitoring applications all measurements and observations are modelled using common schema encodings, exchange of the

data between applications can greatly be improved.

- There is no overall conceptual framework for interoperable integration of various, heterogeneous sensors. Such a concept can enable easy plug-and-play of sensors as well as model-based exchange of sensors and their respective measurements among various landslide monitoring applications. This, however, can demand for the development of a sophisticated sensor service that realises such a concept.

1.2. Research Statement and Objectives

The main goal of the dissertation is to propose a concept for interoperable management of heterogeneous, multi-sensors within landslide monitoring applications. The hypothesis statement of the dissertation states that it is only through self-descriptive, complete, open generic or standard-based sensor models that people and software applications can easily integrate (plug-and-use) and better understand the sensors they are working with as well as the measurements data. In addition to adoption of interoperable standards, applications can exchange sensors (via sensor models) and integrate them with no (i.e. zero programming) or minimal adjustment.

The dissertation objectives are as follows:

- Carry out detailed analysis of the needs of landslide monitoring and early-warning applications.
- Define and specify minimal generic logical sensor models.
- Propose a workflow for the development and usage of the logical sensor models.
- Research on methods and technologies for achieving interoperable management of sensors
- Investigate the applicability of sensor standards (e.g. OGC Sensor Web Enablement) for interoperable integration of sensors.
- Propose a model-driven conceptual framework for interoperable management of multi-sensors.
- Validation of the concept in a landslide monitoring application as well as the logical sensor models developed for this application.

A generic logical sensor model can be seen as a vendor-independent, minimal specification of a given sensor type. The model has to be defined by a specific community and it must be possible to use such model within that community in order to support interoperable integration of sensors. It is envisaged that a generic logical model can also be encoded or mapped to a standards-based model using any standard modeling language (e.g. OGC SensorML). The logical sensor model should define and specify the following:

- Common data types (e.g. spatial and temporal) and structures
- Common Interfaces (e.g. command and control)

- Input and output (I/O) parameter definition
- Common protocol (for connection at application layer as well as physical layer)
- Quality information about sensor (e.g. reliability, accuracy, resolution, etc.)
- Other sensor metadata (e.g., Geoposition information of the sensor, calibration information, physical characteristics, measurement capabilities, etc.)
- Sub-sensing devices
- Sub-components (i.e. non-sensing devices like battery or formal descriptions of algorithms)

Figure 4 provides an illustration of the general concept framework of the dissertation.

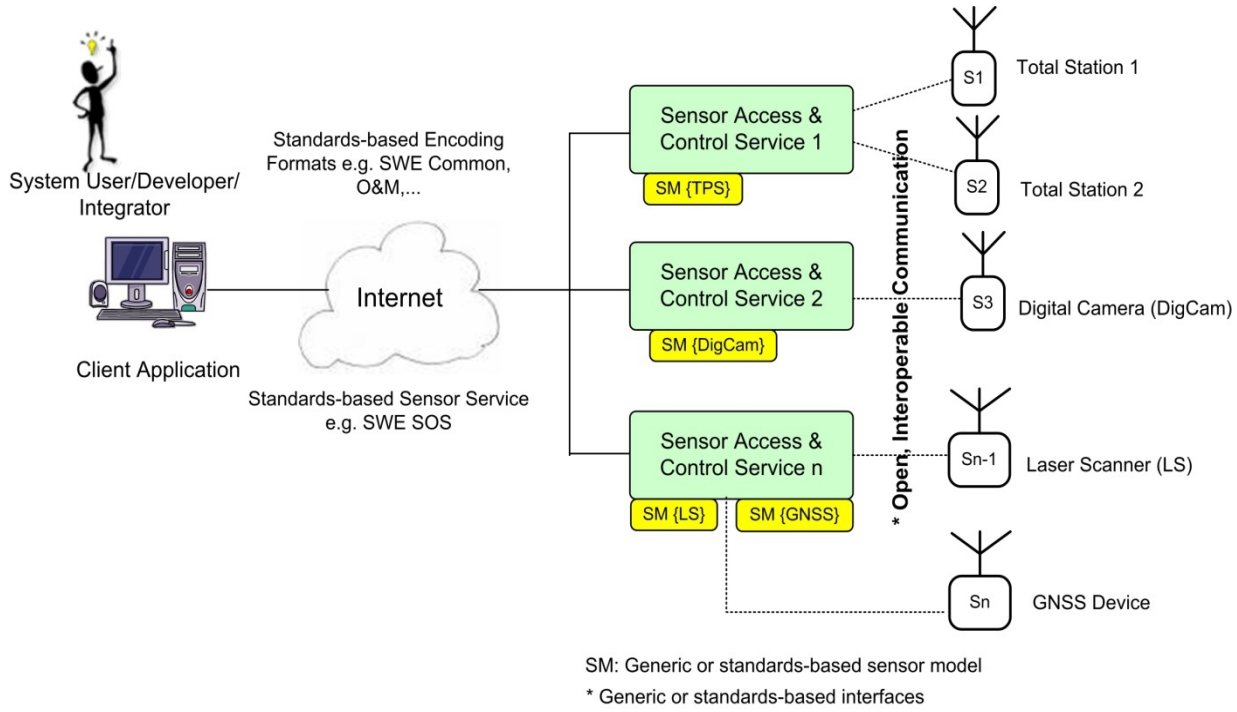

Figure 4: General Concept Framework

The definition and specification of generic or standards-based sensor models (SMs), and interoperable sensor access and control services play an important role towards achieving interoperable management of multi-sensors in landslide monitoring applications.

1.3. Structure of the Dissertation

Figure 5 gives an overview of the layout of the dissertation. Chapter 1 gives the introduction, defines the problem being addressed, the research statement and objectives. This current section explains the dissertation layout.

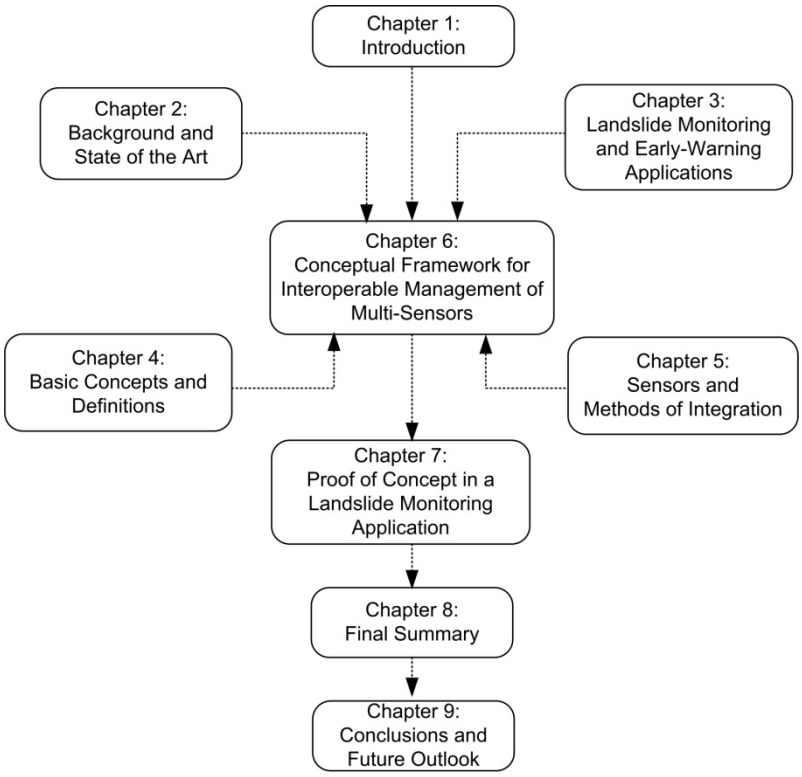

Figure 5: Layout of the Dissertation

Chapter 2 looks at the research background and state-of-the-art in multi-sensors integration. The chapter also covers general interoperability standards as well as sensor specific standards and specifications. Examples of existing sensor-based frameworks are also given. Chapter 3 continues chapter 2 by looking at the current situation in landslide monitoring and early-warning applications, which is the application domain of the dissertation. Chapter 4 explains the basic concepts and definitions that form the basis for the dissertation. Chapter 5 covers the classification and technical specifications of sensors that have been selected for the dissertation. It also documents about sensor integration methods and technologies. Based on the information and knowledge contained in the previous chapters, chapter 6 focuses on the conceptual framework for interoperable management of multi-sensors. Chapter 7 shows the proof of concept within a landslide monitoring application. This is then followed by the final summary in chapter 8. The last chapter 9 contains the conclusions and future outlook.

2. Background and State Of The Art

This chapter discusses the background and state of the art of the research with respect to managing or integrating heterogeneous systems within a single landslide monitoring application. The first sections of the chapter cover the introduction and issues of concern to be considered when integrating different systems. The last sections focus on interoperability standards and specifications.

2.1. Integration of Disparate Systems

In the literature, the term *integration* is not well defined and it very much depends on the context of domain application (e.g. enterprise business process integration, social integration, economic integration, technology integration, etc.). In this dissertation, the definition context of interest is that of the information and technology systems integration point of view.

Definition1:: Integration: Integration refers to linking together different systems or system components and software applications physically or functionally. This might include joining system components by "gluing" their interfaces together.

Definition2:: Integration: Barkmeyer et al. [Barkmeyer et al., 2003] defines integration as "the engineering process that creates or improves information flows between information systems designed for different purposes. What actually flows between systems is the data, but what is critical to the business process is that all of the *right data* flows in the *right form* for the receiving system, and the people who use it, to interpret the data correctly."

The first definition is adopted for managing different sensor systems in a single application. The second definition is adopted for the sensor data. The transmitted or exchanged data between the sensor and the application are expected to be in "right form", that is the data have to be either already in a standardized format (derived from a common schema) or have to be transformed (data mapping) to be compatible with data types and structures of the receiving system. The sensor models to be defined have to describe the various sensor measurements data, in the "right form" so that people and machines can easily use them, as well as make correct interpretations.

2.1.1. Types of Integration

Before discussing about the problems related to integrating different systems (e.g. sensor systems) in a single application, this section gives an overview on the types (degrees) of integration and highlights the type of integration that is adopted for the dissertation.

- **Self Integration**

Is the process in which a sensor system or system component (sensor, detector, etc.) reconfigures itself within its operating environment (i.e. targeted application space) in order to participate in a *joint action* or *cooperate* with other sensor systems. This process does not involve human intervention or use of third-party tool that builds a *bridge* (see section 5.3.3) or set of *wrappers* (see section 5.3.2) for the purpose of adapting the sensor system or the monitoring application to the new requirements. The prerequisite is that the following technical requirements are strictly adapted:

- Common technology (e.g. communication methods, techniques) for the transmission and reception of the data
- Common protocols for the exchange of data
- Common and exactly defined data types, structures and representations

At the moment, sensor manufacturers do not follow these technical requirements, making self-integration not possible or rather very difficult. The existing interoperability standards and specifications (see section 2.2) try to address some of these technical issues.

- **Loose Integration**

Also termed "weak" or "low" integration, refers to the interaction or communication between two different components via a stable interface. The communicating parties are unaware of each other's internal implementation (information hiding or encapsulation). For example, in GNSS (e.g. GPS) and inertial navigation system (INS) research works [Gebre-Egziabher, 2007; Li and Wang, 2006; …], which intensively started in the early 1980s, loose integration is seen as a process where a GNSS receiver and INS provide their measurements to a *blending* software that actually fuses measurements data using estimators like kalman filter (KF) or extended kalman filter (EKF) [Cui and Chen, 1999;…], unscented kalman filter (UKF) [Julier and Uhlmann, 2004…] or neural networks and neuro-wavelet analysis method [Wang et al., 2006; Chiang and El-Sheimy 2004; Goodall et al., 2005, …]. The key feature of this approach is that both the INS and GNSS receiver remain independent navigators and their respective software components do not fuse the data themselves.

This approach is adopted for the dissertation in trying to achieve interoperable management of disparate sensors within a landslide monitoring application. For example, in a sensor network, sensors can take up the roles of specific servers and the applications interacting with them the roles of clients. Software system integrators can employ loose integration concept to incorporate these sensors into their applications. The software components they develop for accessing and controlling the sensors do not necessarily need to *blend* or *fuse* the measurements data, but do provide the data in a common format. The applications receiving the data have to do the actual fusing of the relevant data themselves depending on their specific needs. The loose integration approach is also being followed by interoperable Web services or service oriented architecture (SOA) applications.

- **Tight Integration**

For a complete discussion, tight integration is also explained here. There are different degrees of tight integration: *close*, *full (seamless)* and *deep (ultra-tight)*. The higher the degree of integration, the less independent are the individual components. If one of the integrated components fails to work then the joint action or functionality fails. Tight integration may include combining hardware components into a single unit. From INS/GNSS experiences found in the literature, tight integration can require development of complex architectures and this approach also makes a GNSS receiver no longer be seen as an independent navigator. However, some researchers [Li and Wang, 2006; Wendel and Metzeger, 2005] […] argue that the next generation of INS/GNSS will be based on ultra-tight integration because of its

improved performance under critical conditions (high dynamics, jamming, and interferences), but this will make interoperability difficult. Also, in cases where hardware integration is required through modification or composition of physical sensor designs, this is not an option for software system developers or integrators, hence the reason for this dissertation not to consider tight integration approach.

2.1.2. Issues of Concern when Integrating Multi-Sensors

There are different issues and aspects related to integration of disparate systems (also applicable to sensor systems or instruments). The International Organisation for Standardization (ISO) Reference Model for Open Distributed Processing (RMODP) [ISO/IEC 10746-1, 1998] explains the viewpoints which can be used to categorize the problems. Using this reference model, integration problems can be viewed from: *the enterprise, the information, the computational, the engineering,* and *the technology* viewpoint. The dissertation considers the last four points of view.

Barkmeyer et al. [Barkmeyer et al., 2003] explores the integration problems in detail in its "concept for automating systems integration" and it concludes that "*technical impediments are interoperability problems, in one way or another, and most of interoperability problems are technical impediments*."

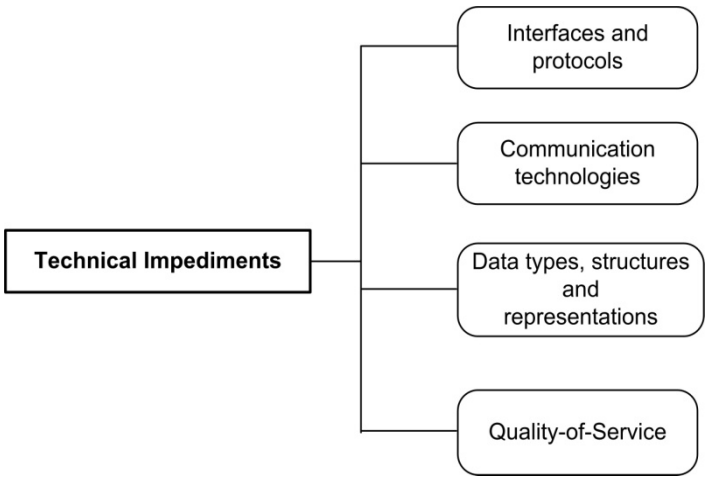

Figure 6: Systems Integration Technical Impediments

Figure 6 gives an overview of the technical problems that arise due to discrepancies in:

- Use of vendor-specific interfaces and protocols
- Discrepancies in the communication technologies used
- Differences in data types, structures and representational formats
- Differences in the quality-of-service of the sensor systems

Examples of communication technologies and mechanisms are SOA, Web services (REST-based, SOAP-based, etc.), remote procedure calls (RPCs) or operation invocation (e.g. Java Remote Method Invocation …), etc.

Following are the discussions of specific issues that are of relevance to the dissertation.

2.1.2.1. Connection conflicts

For two or more systems to communicate, first a connection has to be established. There must be an agreement of the communication protocols and mechanism of interaction to be used. Disagreements in the system supported interaction mechanism lead to *communication mechanism conflict*. If the interaction mechanism is the same, but the underlying specific protocols different, this causes *protocol conflict* [Barkmeyer et al., 2003]. For example, if the client is based on Microsoft's Distributed Component Object Model (DCOM) [Microsoft-DCOM] protocols but the server on Common Object Request Broker Architecture (CORBA) [OMG-CORBA] or other different technology.

In the 1970s and earlier, different hardware (including sensor systems) vendors used to design their own custom hardware protocols for networking or communication. Interaction between the systems was only possible through a lot of integration efforts. In the 1977, the American National Standards Institute (ANSI) working group on Distributed Systems (DISY) proposed a layered model of network architecture, which later (in 1984) became the Open Systems Interconnection (OSI) Reference Model [Stallings, W.R., 1998; ISO/IEC-7498-1, 1994]. The goal of OSI model (also known as the OSI/ISO seven layer model) is to standardise the communications protocols and functions (services) of the defined protocol layers. The model specifies seven layers which are Application, Presentation, Session, Transport, Network, Data Link, and Physical. If systems strictly adopt the OSI model, they can be interoperable at various levels of communication.

Table 1 shows the main elements of the OSI Model (see **Appendix 12.1** for detailed explanation).

OSI/ISO Model		
Layer #	**Name**	**Function**
7	*Application*	Communication with application processes, using protocols for messaging, file transfer etc.
6	*Presentation*	Data representation and encryption; data conversion
5	*Session*	Interhost communication e.g. start and stop sessions
4	*Transport*	End-to-end connections and ensures reliability on data delivery.
3	*Network*	Path determination and logical addressing, i.e. routing of data to different destinations
2	*Data Link*	Physical addressing. It ensures reliable transmission of data packets from node to node based on the destination address.
1	*Physical*	Media, signal and binary transmission. It also includes also wiring/cabling of devices or systems.

Table 1: OSI/ISO Seven Layer Model

However, the ISO model does not concretely provide or specify software or application programming interfaces (e.g. Microsoft Windows Sockets (Winsock), Unix Berkeley Sockets, or Web

application/services interfaces, etc.) and higher level protocols like SOAP, etc. For the dissertation, the higher levels of the OSI model (e.g., **application and presentation layers**) and the extensions to it including the **application programming interfaces** and **Web services** are of importance. Web services do rely on the Internet communication model (which can be seen as a clearly defined subset of the ISO Model).

At the moment, Web services provide a promising solution for resolving the connection conflicts and other software and data integration problems discussed above. Examples are the OGC Open Web Services (OWS), OASIS and partners' Web Service (WS-*) protocols [http://roadmap.cbdiforum.com/]. WS_* is essentially a set of standards to achieve end-to-end security and reliability (quality) of interaction over any underlying protocol, including HTTP, TCP, In-Process communication, etc.

2.1.2.2. Command and Control conflicts

The prerequisite is that the connection conflicts are resolved. Command and control interoperability aims at resolving the problem of using proprietary interfaces when communicating with disparate (sensor) systems. Use of generic or standards-based interfaces (e.g. IEEE P1451 Smart Transducer Interfaces) can resolve these conflicts.

2.1.2.3. Functional conflicts

Refer to the suitability of a system for the specific role that it is assigned to. Examples are *intention conflicts* and *functional scope conflicts*. Intention conflicts happen when the system is used for a purpose or in a way that is not anticipated by the vendor. This may be due to lack of a detailed functional model of the system. Functional scope includes vendor-specified system capabilities and limitations as well as the scope of system behaviour under certain environmental conditions. These details must be provided in both human and machine readable format. Also, the functional specifications must be clear and accurate so as to avoid functional conflicts.

2.1.2.4. Quality-of-Service (QoS) conflicts

They arise if the expectations or requirements of an application (client) are not fulfilled in terms of performance, **quality**, **reliability**, timing, security, etc. QoS is a very important aspect in real-time systems, especially in monitoring applications. Sensor systems can provide information about their quality and reliability via their sensor models. People and machines can then match this information with the application requirements or expectations.

2.1.2.5. Data Integration Conflicts

Data integration can be seen as a process of combining data residing from different sources and providing the user with a unified view of these data [Lenzerini, 2002]. Barkmeyer et al. [Barkmeyer et al., 2003] defines it as a problem that deals with identifying *common information specifications* or *resolving differences between alternatives* (i.e. by comparing other similar, relevant data sources).

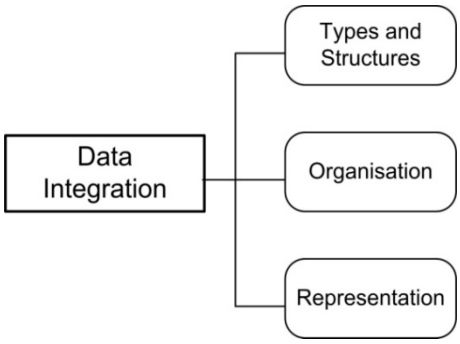

Figure 7: Overview of Important Aspects of Data Integration

Figure 7 shows an overview of important aspects that have to be considered when handling data from heterogeneous data sources. Following is a brief explanation of these aspects:

- Incompatibilities in data type definitions (user defined types (UDT) or primitives (double, float, integer, etc.)), and proprietary data structures (e.g. file structures, table or record structures, page structures)
- Differences in data organisation (e.g. formats)
- Discrepancies in data representation (i.e. at logical or physical level of a data model)

The problem of integrating heterogeneous data sources under a single query interface is not new. In GIS, for example, in monitoring applications and other IT systems, there are still active research works going on in order to find optimal solutions. For example, the Oxford University Computing Laboratory (OUCL) [http://www.comlab.ox.ac.uk], in United Kingdom, is working on a project (2007-2011), funded by the Engineering and Physical Sciences Research Council (EPSRC), investigating "Schema Mappings and Automated Services for Data Integration and Exchange". *Data Exchange* is defined as the problem of inserting data structured under one or more source schemas into a target schema of different structure while reflecting the source data as accurate as possible, and at the same time considering *integrity constraints*. They define *data integration* as a problem of answering queries formulated over a target schema that integrates the information provided by several data sources over one or more source schemas. The laboratory is researching on a new model for using Web service technology for delegating data exchange and integration tasks, and it aims to implement and test new algorithms and methods in that model.

A prominent example dealing with similar problems is the EU Infrastructure for Spatial Information in Europe (INSPRE) [http://www.inspire-europe.en; http://inspire.jrc.it]. The 'Draft Implementing Rules for INSPIRE Transformation Services' (February 2009) discusses *schema transformation* in data interchange and issues about *quality of service*. For more information on the development and research requirements of INSPIRE and its direction, refer to Bernard et al. [Bernard et al., 2005a, 2005b]. Several other related research works can be found on the Internet and in various literature.

However, the idea of using a single query interface and a target schema is adopted in this dissertation. Precisely, a generic logical sensor model can also act as a single query interface as well as a target

schema for the modelled sensor type.

The following conflicts, e.g. data consistency, semantic and syntax, contribute to the issues of concern discussed before.

2.1.2.6. Data consistency conflicts

ISO 19113 Geographic Information - Quality principles defines data consistency as a sub-element of data quality. A significant amount of research work on data quality has been carried out [Caspary, 1993; Caspary, 1992; Joos, 2000; Joos, 1994]. The sub-element *logical consistency* covers different aspects (conceptual, domain, format and topological) of spatial (geographic) data consistency. Wang [Wang, 2008] discusses about logical data consistency in detail and proposes a new methodology for dealing with spatial data consistency using spatial data integrity rules defined in constraint decision tables (CDTs). Data consistency can also be applied to measurements data. For example, the format of different sensor streams can be checked for consistency with the application's underlying sensor data storage (repository) format (e.g. SWE Common data specification, NMEA etc.). This can also include typical plausibility checks.

2.1.2.7. Semantic conflicts

Are the results of lack of common concepts or agreement on terms used to refer to those concepts. The basic requirement is that specific communities or domains create dictionaries with controlled vocabulary or ontologies. The conflicts can be found in the data conceptual model (*conceptualisation conflict*) or in the interpretation of references to object types and instances in the application space, etc. For example, if the measurement values do not specify measurement units, then assumption of different units (e.g. metres, feet …) can lead to misinterpretation. Barkmeyer et al. states that *interpretation conflicts* also exist even when standard interfaces are used. The reason is that standards usually permit "least common denominator" semantics or specifications. This dissertation considers ontologies, but uses very simple approaches like controlled dictionaries for defining terms/concepts used in logical sensor models.

2.1.2.8. Syntactic conflicts

Arise when communicating components use different data types, structures, and organisation to describe identical concepts. Examples of such concepts [Barkmeyer et al., 2003] are:

- Object type
- Action type
- Property type
- Relationship type
- Any instance of the above

The above concepts can be represented or defined differently for the same real world features, for example, a component might use ASCII RPC-based messages and the other SOAP messages to communicate its data. Some of the syntactic conflicts can be resolved through intermediates that perform

protocol transformation and syntax translation. There are tools like Microsoft Visio, XML Spy [Altova] that implement languages like Express-X [ISO/IS-10303-25] and XSLT [W3C], respectively, for name translation and data reorganisation etc. This can also involve some kind of syntactic and structural schema mapping.

Summary:

The above sections have discussed issues and aspects of concern when handling different systems (e.g. sensor systems) as well as managing data from disparate sources (systems). For introductory work in the field of multi-sensors integration, Luo and Kay [Luo and Kay, 1990] discusses about multi-sensor fusion and integration in a more general sense. Luo et al. [Luo et al., 2002] provides a comprehensive overview of multi-sensor fusion and integration related approaches, applications, and future research directions. Other related introductory works [Dasarathy, 1997; Vashney, 1997; Hall and Llinas, 1997] cover different aspects of sensory data fusion and integration techniques. However, these aforementioned research works do not address the use of standards, but provide an insight into the need for interoperability when dealing with multi-sensors within a single application space.

2.2. Interoperability Standards and Specifications

Since the dissertation focuses on interoperable management of multi-sensors, the following subsections discuss the interoperability standards and specifications that can be adopted in order to fulfil the research's main goal.

2.2.1. Definition of Interoperability

ISO 19101 Reference model [ISO 19101] defines interoperability as "the ability of a system or system component to provide information sharing and inter-application co-operative process control. It provides a freedom to mix and match information system components without compromising overall success". Therefore, interoperable management of sensors can ensure that different or same types of sensors can be installed, accessed, and controlled through vendor-neutral means.

2.2.2. Aspects of Interoperability

This subsection covers aspects of interoperability that consider the problems of integration discussed in chapter 2 in section 2.1, according to ISO 19101.

1. **Protocol and Interface Interoperability**: Refers to communication strategy between two systems, at two main levels: *higher* and *lower levels*.

The higher level addresses communication issues between sensing devices and the application consumers at connection, command and control layer. The lower level communication is about physical communication between sensing devices and the respective applications, e.g., handshaking, cabling, wireless, etc. Of importance to the dissertation are the higher level interfaces/protocols.

2. **File System Interoperability**: Requirements for interoperability are common naming conventions (*semantics*), access control, access methods, and file management. Figure 8 shows the steps which need to be taken in order to support native (proprietary) formats of existing legacy systems.

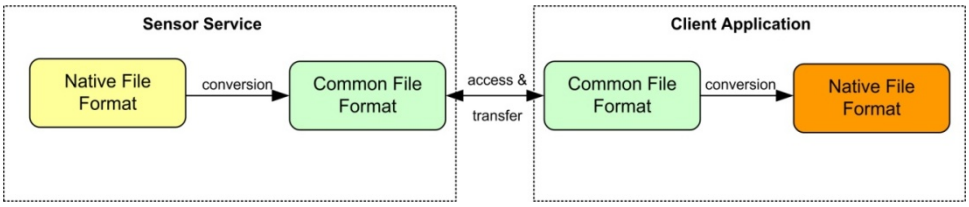

Figure 8: File System Interoperability

3. **Remote Procedure Calls**: Interoperability can be achieved if a specification of a common set of operations that execute procedures on remote systems exists.

4. **Search and Access Databases**: Use of common databases (e.g. Oracle Sensor Data Repository) and common methods of access (e.g. SQL).

5. **Syntactic Interoperability**: Is the ability of different systems to be able to interpret the syntax of the same data in the same way.

6. **Application Interoperability**: Is the ability of applications to use and represent data in the same manner. This requires *semantic interoperability*, which aims to ensure that applications interpret data consistently in the same manner in order to provide the intended representation of that data. For example, *common translators* can be used to convert data from different data sources (e.g., sensors) into the target applications. For details refer to ISO 19101 Reference model.

2.2.3. Elements of Interoperability

Following are the main elements that we consider important in order to ensure interoperable systems or sub-systems:

- common data formats
- common protocols
- common interfaces
- controlled vocabularies, common ontologies or semantics
- common metadata

2.2.4. General Interoperability Standards and Specifications

Standards enable innovation. Following are some of the standards that form the basis of or complement the existing sensor standards.

2.2.4.1. World Wide Web (W3C) Consortium Standards

W3C is a consortium that "develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential" [W3C]. Following are some of the well-known standards that can be exploited in various applications. These standards are the base for sensor-based standards.

Extensible Markup Language (XML)

Is a subset or application profile of the Standard Generalized Markup Language (SGML) [ISO-8879, 1996]. XML specifies a way to structure, describe, and interchange data. It can be used, for example, to describe mathematical formulae (e.g. MathML – a W3C Recommendation, 2001), architectural blueprints, voice-processing in telephone systems, system modelling, etc. It is self-describing and its elements can be nested in a way that is useful to provide the semantics of the data contained. In other words, the context of an element can be sufficient to determine its meaning.

The main advantage of XML is that it is easily readable by both machines and people. One of the major drawbacks is that the more elements are contained in larger documents the less comprehensive they become. For efficient exchange of XML data, W3C has proposed an efficient XML interchange (EXI) format (based on XML Binary Characterization). There are many software tools available for processing XML documents. For more information on XML, refer to the W3C specifications.

Extensible Stylesheet Language (XSL) Family

It defines ways of transformation and presentation of documents. It consists of XSL Transformation (XSLT), XML Path Language (XPath), and XSL Formatting Objects (XSL-FO). XSLT is a language for transforming XML into other output formats like HTML, PDF, other XML formats, etc. XPath is an expression language used by XSLT to access or refer to parts of the XML document. This family can be useful if one wants to convert, for example, proprietary XML-based sensor models into standards-based models (e.g. OGC SensorML).

XML Schema

It provides mechanisms to define and describe the structure, content, and the semantics (to some extent) of XML documents. It allows users or modelers to use in built primitives (*simpleTypes*) and/or define own data types (*complexTypes*) for the elements/components contained in the XML document. If a document has a schema associated with it, then its structure can be validated by a machine (e.g. PC).

XML schema (XSD) can be extended or profiled. There are two common methods for extensibility: *extensibility by type substitution*, and *extensibility by using <any> element*. Type substitution can be realised by using *substitutionGroups*. Elements that belong to the same substitution group can be replaced by any other element in that group. In cases of using <*any*> element, this is just a place holder for any element of any type. Enumerations, patterns, and facets (e.g. constrained domain value spaces) can be used to limit data contents. Since this dissertation focuses on sensor modelling (mainly based on XML schema), it is necessary to explain some of the concepts used in schema modelling.

Content model: Describes the object's or component's structure and contents.

Global Components: These are element declarations, attribute declarations, type definitions, annotations, and groups (e.g. attribute groups, model groups) in a document that can be accessed or seen by other documents. Analogy to global variables in a class defined, for example, by an object-oriented programming language, like Java, C++, etc.

Namespace: is a collection of names (e.g., for element types, attribute names …), identified by a unique resource identifier (URI) reference [IETF-RFC-2396, 1998]. Namespaces avoid name collision (i.e. if two elements with the same name but different types occur in the same document or symbol space).

Simple Type: is a data type without any content model. It defines all attributes and elements that contain only text or restricted only by primitive data type. There are several simple types already built into the XML schema language.

Complex Type: has at least one or more child elements or attributes.

Table 2 shows the main directives that can be used in XML schema modelling for schema declaration and for referencing other external schemas.

Directive	Description
<*schema*>	The root element of a schema document. It tells the parser or reader that this is an XML schema document.
<*include*>	A directive used for accessing components in other schemas which have the same target namespace.
<*import*>	A directive used for accessing components in a schema with a different namespace.
<*redefine*>	A directive used for accessing components in another schema, while simultaneously allowing for modifying (by extension or restriction) zero or more of the components.

Table 2: The Main Directives for Schema Declaration and Inclusion.

For details on XML schema, refer to the W3C specifications.

2.2.4.2. International Organisation for Standardization (ISO) Standards

This subsection describes some of ISO standards that can be used to support interoperable sensor-based applications. The International Organisation for Standardization is a non-governmental organisation working on developing and publishing international standards. It is a network of national standards institutes of 157 countries, one member per country, which aims at bridging the public and private sectors [http://www.iso.org/iso/about.htm].

ISO 19115 – Metadata

The purpose of metadata is to describe a dataset fully so that the users can understand the limitations and assumptions affecting the creation of that data, and be able to evaluate the applicability of the dataset for their intended use. This ISO standard provides a structure for describing digital geographic data. Metadata is a very important part of any sensor model. Sensor metadata characterizes the ‘what’, ‘when’ and ‘where’ of the data. The OGC SensorML standard will fully harmonize itself with this standard in its next versions.

ISO 19757-3 – Rule-based validation – Schematron

Schematron is a rule-based language that can be used for modelling. It can complement grammar-based languages like XML Schema, RelaxNG, etc. Schematron directives allow constraints to be directly expressed in the schema document, which is a deficit in grammar-based languages like XML Schema. Like any other model, sensor models can have their constraints collocated within schema documents.

2.2.5. Sensor Specific Standards and Specifications

Adapting the objectives of ISO 19101 for the dissertation, sensor-based standards can provide the following benefits:

- Increase common understanding and exploitation of sensors and their respective data.
- Increase the availability, access, integration, and use (plug-and-play) of sensors and their respective data.
- Enhance the exploitation of sensor measurements data in an effective, efficient, and economic way.
- Afford collaboration among different organisations and scientific disciplines through exchange of sensor data and also promote rapid deployment of reliable systems for monitoring natural events (e.g. landslides, windstorms, tsunami …) and valuable or sensitive infrastructure, as well as in other mission-critical applications.
- Etc.

2.2.5.1. OGC Sensor Web Enablement (SWE)

The Open Geospatial Consortium, Inc (OGC) [OGC] is "an international industry consortium of 369 companies, government agencies and universities participating in a consensus process to develop publicly available *interface specifications*. The specifications support *interoperable* solutions that "geo-enable" the Web, wireless and location-based services, and the mainstream IT. The specifications empower technology developers to make complex spatial information and services accessible and useful with all kinds of applications". One of the major initiatives of OGC, and of relevance to this dissertation, is the Sensor Web Enablement (SWE) [OGC-SWE]. The following quotations in a SWE newsletter [ISO/TC 211 Geographic Information/Geomatics, 2005] reflect the vision and the objectives of OGC:

"SWE objectives encompass *specifications for interfaces*, *protocols* and *encodings* that enable discovery, *tasking and access of sensors*, acquisition of sensor data, and discovery and access sensor-processing services. … addresses *self-describing* in-situ sensors and imaging devices, remote-sensing devices, stored data and live sensor feeds, and simulated models that use sensor data."

"The ultimate goal of the SWE is to provide for the processing of raw sensor data into value-added information with semantic descriptions and *link sensors to the network-resident processing services*. This will *make sensor measurements accessible to the spatial data infrastructures* for use by professional decision makers …".

The SWE vision recognises the discussed sensor integration challenges. Figure 9 shows an overview diagram of the SWE concept.

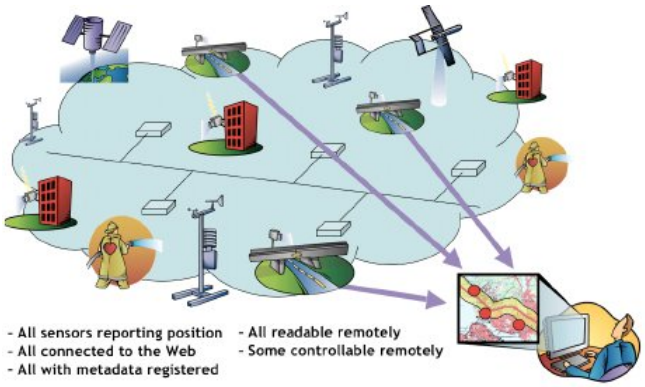

Figure 9: Sensor Web Concept [OGC, 2006]

SWE is being validated within the test-beds under OGC Web Services (OWS) interoperability activities - reference is given to the OGC web site [http://www.opengeospatial.org]

A *Sensor Web* can be imagined or thought of as a "global sensor" that connects web-resident sensing devices and sensor databases, as well as people, machines and other users of these resources [Tao et al., 2003; Kandawasvika and Reinhardt, 2005a]. It can also be seen as a universal network of homogeneous and heterogeneous sensors. These sensors are expected to be discovered, accessed, and tasked online using common interfaces. It is interesting to note that the idea of a Sensor Web can be referred back to the efforts of NASA Sensor Web Applied Research Planning Group. This group defined a Sensor Web as "a system composed of multiple science instrument/processor platforms that are interconnected by means of a communications fabric for the purpose of collecting measurements and processing data for Earth or Space Science objectives", see Figure 10.

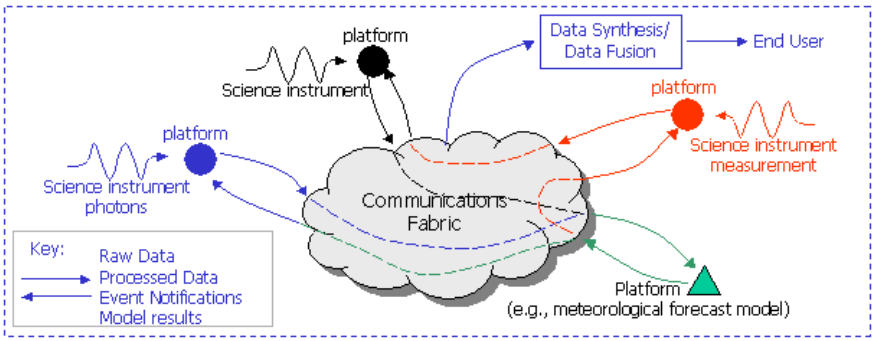

Figure 10: NASA GSFC-ISC Sensor Web Concept [NASA-GSFC-ISC]

NASA collaborates with other OGC members in the development of OGC Sensor Web Enablement standards. This dissertation considers OGC SWE as fundamental in order to achieve interoperable management of multi-sensors in monitoring applications like landslides. Following is a summary of the important issues that motivates the SWE initiative [OGC-SWE; SensorML]:

- The state-of-the-art in sensor deployment is of heterogeneous networks of disparate sensors. These sensors are rarely easily available for use and, if available, the processing of observations from these

sensors can be difficult due to proprietary solutions of the vendors. Interoperability between such 'stovepipe systems' can be achieved usually at high costs and the resulting system may be difficult to maintain and extend.

- The need to *preserve low-level sensor data* and the sensor models required for reprocessing the data in the future.

- The need to make existing or newly deployed private and public sensor systems readily available for use by any application. This is very important in mission-critical, monitoring and disaster prevention or mitigation applications that may greatly benefit from accessing the sensor systems already installed in the region of interest.

- Web-resident sensor systems can enable rapid access to environmental information from the field. Streaming sensor information in *standard formats* facilitates integration, analysis, and creation of various *data views* that are more meaningful to the end user and avoids the drawbacks of locating and accessing incompatible archived data. "Time savings are particularly noticeable in the management of time critical events such as emergency response, advanced warning systems and forecasting" [OGC-SensorML].

- The need to improve decision-making based on high *quality*, near real-time data and information.

For examples of existing and ongoing sensor-based projects see Section 2.2.6. Following are discussions of the relevant SWE specifications and standards.

Modelling languages and specifications:

1. Sensor Model Language (SensorML)

A language for describing *processes* and *processing components* associated with the measurement and post-measurement transformation of observations. It provides an information model and encodings that enable discovery and tasking of Web-resident sensors, and exploitation of sensor observations. SensorML mainly focuses on providing a means to describe the functional models of sensors, rather than the detailed description of hardware design.

Conceptual models:

In SensorML, everything is seen as a *process*. A process is "defined as an activity that takes in one or more inputs, and based on the given parameters and methodologies, generates one or more outputs". For example, sensors and transducers (e.g. detectors, transmitters, actuators, filters, batteries, system clocks, etc.) can be modelled as processes that can be connected to form process chains. It is also envisaged that SensorML processes could be imported and executed by other environments like software supporting IBM's Business Process Execution Language (BPEL), MATLAB Simulink, as well as other SensorML-enabled process execution software.

SensorML processes are categorized into two levels of granularity: atomic processes (*ProcessModel* and *Component*) and composite processes (*ProcessChain* and *System*) – see Figure 11. All these processes are derived from OGC/ ISO Standard [ISO-19136, 2007] Geography Markup Language (GML)

AbstractFeatureType.

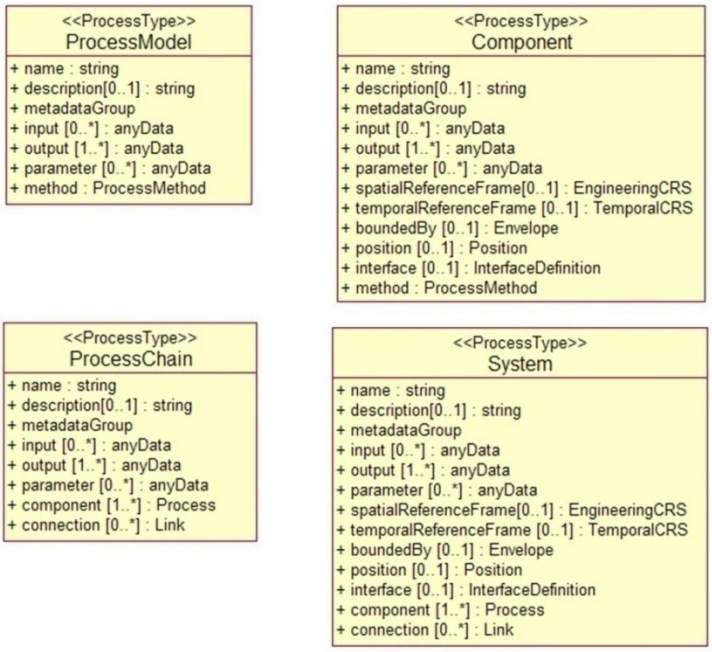

Figure 11: SensorML Conceptual Models [OGC-SensorML, 2007]

The ProcessModel and ProcessChain are defined as non-physical processes, while the Component and System elements are regarded as physical. Physical processes (e.g. sensor system, detector, actuator …) are those where the physical location or physical interface is important. It is important here to briefly explain some of the important elements shown in the figure.

ProcessMethod: Is a method of a process that provides the rules for validating the instances of a process, references or defines an algorithm used for the execution of the process model, and can also include references to the implementations of the software (e.g. algorithm, execution method …) for the on-demand execution of the process. For example, a total station (TPS) can have a module called "CoordTransformation" which can be described as a process method for converting coordinates from one system to another. A process method can also specify its inputs, outputs, parameters, as well as any metadata relevant for that process. For defining the rules of the process method, there are existing rule-based languages that can be used, for example, Schematron etc.

Interface: This element is not fully or clearly defined in the SensorML. The specification only points out that the *InterfaceDefinition* object can be based on OSI Reference Model. For the dissertation, a more detailed interface definition at sensor layer will be given and used for the development of the generic logical sensor models for the selected sensors.

MetadataGroup: It is important for the discovery of processes or sensor systems. It includes information about sensor system or process identification (e.g., unique IDs, URNs …), classification, constraints (e.g., time of use, legal or security constraints …), capabilities (e.g. measurement functions) and characteristics (e.g., sensor response model), contacts (e.g., manufacturer details), as well as history (e.g., deployment date, calibration, maintenance, algorithm corrections or updates …), etc. This element partly considers the ISO 19115 Metadata standard [ISO-19115]. From the dissertation's perspective, some of the elements in

the ‘System’ and ‘Component’ objects, such as coordinate reference systems, interface definition, and localization information can better be encapsulated in the metadata group in order to maintain a clear structure in SensorML instances.

The sensor measurements can be encoded using the SWE Common XML encodings or the OGC Observation and Measurement (O&M) specifications [OGC-O&M-Part1, 2007; OGC-O&M-Part2, 2007]. An observation can be an event (action) or a result of a measurement.

SWE Common Data Types Specification:

The SWE Common specifies the following:

- Primitive data types (complementing those specified in GML)
- General composite data types (e.g., records, arrays, vectors, matrices …)
- Composite types with some semantics (e.g., position, curve, time-aggregates)
- Standard encodings to add semantics, quality information, and constraints

This common data type specification also tries to specify some non-XML array encodings that can be used for the transportation of large volumes of data. From the dissertation’s perspective, this approach can lead to interoperability problems as well as problems regarding development of complex software parsers that can properly handle both XML and non-XML encodings. Standard XML parsers like Apache’s Xerces, MSXML, IBM’s XML Parser for Java, etc., only support ‘pure’ XML rules. In short, more work is needed in order to refine the common data type specification as well as harmonizing it with other encoding specifications like O&M.

Limitations/Disadvantages:

Following are some limitations or disadvantages of the current SensorML specification:

The specification does not provide guidelines or rules for profiling it. It requires that SensorML instances be developed from scratch or by modifying existing documents. The results are usually SensorML instances that are barely interoperable.

The SensorML concept is very generic. The specification has become complex, very flexible (e.g., almost all elements are optional), and verbose. Simonis et al., [Simonis and Echterhoff, 2008] notes that it is not practical to create *universal useable* SensorML documents. Therefore, there is need for each community to develop specific models that are well-defined and applicable to their specific needs. This dissertation recommends that different communities (e.g., geodetic, geophysics, geotechnical, satellite-based remote sensing, etc.) develop specific logical sensor models which can be transformed into any sensor standard like SensorML. Community specific models can support interoperability within that community, and standard-based models can then allow for interoperable exchange of models across communities.

The specification also lacks guidelines for formalizing quality information. There are current research efforts that are focusing on formal description of quality information, for example the INTAMAP project which is developing UncertML [http://www.intamap.org/uncertml]. UncertML is an XML-based

language for describing uncertainties. Further, the ISO is developing a "Guide to Expression of uncertainty in Measurement (GUM)", which can later be exploited in modelling of sensor uncertainties [ISO/IEC-Guide-98-3, 2008].

Summary:

Despite some of the current shortfalls, the SensorML specification provides a fundamental foundation for building systems that can allow for interoperable management of multi-sensors. Definition and development of community specific logical models is a value-added first step in achieving interoperability. Mapping of the community-developed specific models into standards like SensorML can greatly help in identifying improvement needs of the SensorML specification.

2. Observations and Measurements (O&M)

O&M provides a framework (conceptual) and encoding for measurements and *observations*. Its main goal is to enable interoperable access and exchange of observations and measurements using common formats. O&M supports both spatial and temporal data.

An *observation* is an event or an act of observing some physical property or phenomenon, with the goal of producing an estimate of the value of the property. It can have metadata and some quality indicator. A *measurement* is defined as a specialization of an observation made using a sensor or instrument which results in a measured quantity or measurand. In O&M, an observation is modelled as a *Feature* according to the ISO General Feature Model (GFM) [ISO-19109]. A feature is seen as a generic carrier of properties, and defines the domain of an observation. The key properties of an observation are:

- *Feature of interest (FOI)*: is a feature of any type [ISO-19109, 2005; ISO-19101], which is a representation of a real world object being observed. In landslide monitoring, features of interest can be ditches, slope edges or escarpments, cracks, etc.

- *Observed Property*: Identifies or describes the phenomenon for which an observation result provides an estimate of its value. Examples of properties are ditchLocation, ditchDepth, ditchPerimeter, etc.

- *Observation procedure*: It is the description of a process (see SensorML definition) used to generate the result. In the case of observing a ditch, a procedure can be an extensometer, or a measuring tape. In general, an observation procedure can be a sensor or instrument, an analytical procedure, method or algorithm, a simulation program etc.

- *Result*: It is the value generated by the procedure. For example, these can be numerical values assigned to the observed properties. The result of an observation or measurement can be a quantity (measure), category, temporal and geometric value, coverage, or a composite or array of any of these results.

Summary:

The O&M specification also uses elements from the SWE Common data types and SensorML. However, the current version does not implement the ISO 19115 Metadata, but will do so in the future.

3. Transducer Markup Language (TML)

Argon ST IRIS Corporation [http://www.iris.org] is the initial developer of TML, which then was brought into OGC in 2004, at the request of the National Geospatial-Intelligence Agency (NGA), to investigate its applicability in supporting real-time streaming of data to and from sensor systems.

Definition::Transducer: Is an entity that receives a signal as input and produces a modified signal as output. It can either be a sensor or an actuator (refer to Chapter 4.2.1 for definitions). The conversion or modification of an input signal into output (i.e. a measurement) can be modelled mathematically by a *convolution*. A convolution is a technique for determining the system output given an input signal and the system impulse response. Following is an example of a signal convolution model [ECE, 2005], see equation [1]:

$$o(z_0) = \int_a^b i(\propto)\, r(z_0 - \propto)\, d \propto \qquad [eq\ 1]$$

where:

i(α) : is the input signal, as a function of time, space …

$r(z_0 - \alpha)$: is the instrument response, inverted and shifted by z_0

$o(z_0)$: is the output of the signal at $z = z_0$

[a,b] : is the range over which the instrument response is significant

TML defines and specifies the following:

- A set of models that describe the (hardware) response characteristics of a transducer.
- An efficient method for transporting sensor data and preparing it for fusion through *spatial* and *temporal* associations.

Since sensor data are a result of internal processing, the effects of the processing can be modelled as functions. Figure 12 shows a simple signal processing model.

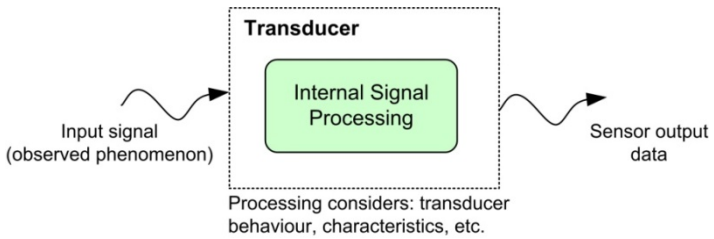

Figure 12: Simple Signal Processing Model

TML response models are formalized XML descriptions of the known hardware behaviours. Examples of these models include:

- Transducer's latency and integration times

- Noise figure or curve
- Spatial and temporal geometries
- Steady-state response
- Impulse response

Using TML, live streams of sensor data can be transported in time-tagged groupings called *TML clusters*. The time-tags can be used as a mechanism for temporal correlation to other transducer data.

Summary:

TML describes information needed to support a logical data structure model. This information can include system calibration, transducer behaviour, operation conditions, and data collection parameters. However, it does not specify a service or streaming platform needed to deliver the TML-encoded data.

Sensor Web Services

"Web Services provide a standard means of interoperating between different software applications running on a variety of platforms and/or frameworks. They can be combined in a loosely coupled way in order to achieve complex operations. Programs that provide simple services can interact with each other in order to deliver sophisticated value-added services" [W3C]. Web Services are also extensible and can provide machine-processable self-descriptions (e.g. Web Service Description Language (WSDL) files). Following are explanations of some of SWE services.

1. Sensor Observation Service (SOS)

"It provides an API for managing deployed sensors and retrieving sensor data - specifically 'observation' data". SOS is observation centric and does not provide interfaces for actually managing deployed sensors. It does, however, support the access and retrieval of observations from sensor repositories. SOS also specifies interfaces to support transactions, i.e. for inserting and updating sensor data repositories. The current "core" operations (mandatory) of SOS are:

- *GetObservation*: For accessing and retrieving sensor observations and measurements data via a spatio-temporal query that can be filtered by phenomena (e.g. observation offerings).
- *GetCapabilities*: Provides the SOS service metadata.
- *DescribeSensor*: Provides detailed description about the sensors and processes (e.g. simulators) generating measurements.

Summary:

SOS capabilities do not directly show the relationship between the sensors (or observation procedures) and their offerings (observed phenomena). An SOS service can greatly benefit from services like sensor service registry (does not currently exist in SWE suite) and sensor management service (which is part of

this dissertation work).

2. Sensor Planning Service (SPS)

It provides interfaces for determining the collection feasibility for a desired set of collection (data acquisition) requests for specific or required sensors/platforms. SPS basically supports data acquisition planning, i.e. user can make collection feasibility plans based on, for example, geographic regions of interest, specific time frames, and possibly choose quality parameters to be delivered together with the collected data.

Summary:

SPS plays an important role in mission planning. However, the actual controlling and configuration of sensors can be part of a sensor management service. A sensor management service should provide SPS with enough information about non-restricted or configurable parameters (e.g. for testing and submitting feasibility requests).

2.2.5.2. IEEE P1451 Smart Transducer Interface Standard

It is a family of standards that define and specify industry-wide, open, common communication interfaces for connecting transducers (sensors and actuators) to microprocessors, instrumentation systems, and control/field networks (e.g. sensor network for monitoring purposes). The core feature of the standards is a Transducer Electronic Data Sheet (TEDS) for sensor self-description. TEDS stores transducer identification, calibration, correction data, and manufacturer related information.

Using the IEEE 1451 Neutral Model, sensors can be accessible to clients (e.g. Web sites) over a Network Capable Application Processor (NCAP) interface. This interface can provide a point of entry to services (e.g. OGC SWE services). Sensors that do not adopt this neutral model can implement a 1451 Wrapper. For the current implementations of the standard, references are given to the Open1451 project [http://open1451.sourceforge.net] and the TEDS Library for Labview by National Instruments [NI-LabVIEW].

2.2.5.3. National Maritime Electronics Association (NMEA)

NMEA [http://www.nmea.org] is an association composed of manufacturers, distributors, dealers, educational institutions, and other parties interested in marine electronics. NMEA standard (e.g. NMEA-0183) defines an electrical interface and data protocol for communications between instrumentation (e.g. GPS, gyrocompass, anemometer, sonar, etc.). NMEA 0183 is a serial interface, but not network based. There is also NMEA 2000 specification, which specifies a network interface and it is based on the OSI Reference Model.

2.2.5.4. ISO 19130 - Sensor and data models for imagery and gridded data

The objectives of the initial standard were to develop standard models for sensor and data for imagery and gridded data [Di et al., 2001]. This first standard exceeded its 5 year deadline in March 2006 and is deleted by the ISO. However, work on a new version of the ISO 19130 is in progress and its focus is on developing standard models that address the orientation and calibration data for digital cameras [Kresse, 2006]. Standard-based geographic or spatial referencing of imagery data will be based on the developed sensor models. The next versions of SensorML standard will try to harmonize with this standard.

2.2.6. Examples of Sensor-based Implementation Frameworks

There are several research efforts going-on in the field of sensor-based services and applications. The following are examples of such efforts.

1. SWE-based 52° North

52° North is a Sensor Web community that focuses on implementations of OGC SWE. Its vision is to enable real time integration of heterogeneous Sensor Webs into spatial information infrastructures. Its current software implementations include O&M encodings, SOS, SPS, Sensor Alert Service, and Web Notification Service [http://52north.org], at various stages of development but not yet mature.

2. SensorNet

Ork Ridge National Laboratory (ORNL), together with its partners: the National Oceanic and Atmospheric Administration (NOAA), OGC, National Institute of Standards and Technology (NIST), US Department of Defense, and several universities and companies, is developing a SensorNet for the real-time detection, identification, and assessment of chemical, biological, radiological, nuclear, and explosive (CBRNE) threats. The SensorNet is based on IEEE 1451, OGC SWE, and other open standards.

3. EU-FP6 SANY (Sensors Anywhere) Integrated Project (IP)

SANY [http://www.sany-ip.eu] is an ongoing project started in September 2006 with a time span of 3 years and is being financed under the EU/FP6 IST programme. It aims to contribute to the European Commission (EC) and the European Space Agency (ESA) on the GMES by improving the interoperability of in-situ sensors and sensor networks. The project's objectives include building a European environment monitoring and risk management infrastructure, syntactic and semantic interoperability, integration of existing sensor technologies, and use of standard interfaces like the OGC SWE.

Schimak et al. [Schimak and Havlik, 2009], in an article on SANY and OGC SWE in risk management applications, states that "none of the currently available and emerging technologies offers *rapid deployment*, *easy maintenance*, *quality assurance*, and *automated data processing* along the whole information processing chain from smart sensors and wireless ad hoc sensor networks, over data loggers and value-added middleware services, to user applications capable of dynamically integrating all available data sources at runtime". This reflects the notion that all the current sensor-based research works (including this dissertation) being carried out from different perspectives to solve similar or different integration problems are in fact important parts of the whole sensor research field. A lot of harmonization work of these various researches will the valuable in the future. On this point, the SANY project has based its work on existing EU projects like GMES [http://ec.europa.eu/gmes], ORCHESTRA [http://eu-orchestra.org], and a U.S. global initiative GEOSS [http://www.epa.gov/geoss].

Some of the current results of the SANY project demonstrate the feasibility of integration of wireless ad hoc sensors into its Sensor Service Architecture (SensorSA) networks. There is a deliverable document version 1 of the "Specification of the Sensor Service Architecture" submitted on the 6th of August 2008 for the public. The SensorSA relies on the OGC SOS; therefore its interfaces will also be influenced by any changes to the SOS. Validation sub-projects for SANY include *air quality management* in urban

areas and around industrial zones; *coastal water management* for predicting bathing water quality and microbiological contamination; and *geoharzard monitoring* in complex urban environment in order to detect structural/architectural instability due to human activities like tunnel excavation.

2.3. Summary

The main focus of this chapter was to discuss the background and state of the art of multi-sensor integration. It has discussed the problems and conflicts faced when handling heterogeneous systems within a single application space. The chapter has given brief and concise discussions of relevant interoperability and sensor-based specifications and standards that serve as the foundation of the dissertation and these can be applied to resolve the discussed problems. Examples of the current implementations of sensor-based frameworks have been shown.

The following chapter discusses the requirements for sensors and the current situation in landslide monitoring and early-warning applications, which are selected for the dissertation case study.

3. Landslide Monitoring and Early-Warning Applications

Monitoring of geo-environmental variables and understanding their causes and possible effects at different scales is very important. The knowledge gained from monitoring activities enables the prevention of loss of human lives, avoidance of economic losses and damages to valuable infrastructures (e.g., roads, buildings, etc.). Over the last decade, ground instability (including subsidence and landslides) has cost the UK insurance industry more than €500 million a year (ref: GMES Services Element under http://www.esa.int).

An effective and efficient monitoring system is expected to warn responsible authorities (e.g., geoscientists, evacuators …) in ample time. This section looks at the main aspects of landslide monitoring system and early-warning applications, which include requirements for data, requirements for sensors, importance of sensor networks, and issues related to management of heterogeneous sensors together with their respective measurements data. The last subsection gives examples of existing landslide monitoring systems that have been analyzed.

3.1. Requirements Analysis for Data

In order to accurately know about the requirements for sensors within application, it is important first to analyze and understand the data requirements of that particular application.

Table 3 gives an overview of the compiled possible data (i.e. based on our research analysis and experience) that can be required by a landslide monitoring application.

Application DataDomain Category	Data Examples	Example of Usage
Surveying Data	Topographic maps (ATKIS-Data, ALK-Data)	For identifying nearby infrastructure e.g., buildings, roads, dams, etc. For locating natural features like rivers.
	Digital Elevation Models (DEMs)	Show change in surface topography Derivable products: Slope gradient, aspect, curvatures, active areas,…
Remote Sensing Data (e.g. Photogrammetry/Satellite imaging)	Aerial images, orthophotos	Photos of ground deformation
	Satellite images: - Radarsat(C-Band) - JERS(L-Band) - Spot-V - IKONOS - Hybrid Radar Satellite images	For deriving morphological maps showing features such as scarps, landslide crowns, terraces, slump blocks, concave surfaces of rupture, drainage lines, and the slide body
Geological data	cross-sections and long profiles of sub-surface structures	Geologic determination of areas susceptible to landslides
	3D geological models	
	geological map	*Geological development history of the landslide area*
	Boreholes\Drillholes data, Sub-surface blocks data	For determining the subsurface profiles, …
Geotechnical data	Scalar data (expansion values of wire or rod extensometers, inclinometer data, etc.)	For determination of linear expansions or movements of features within the landslide area.
Geophysical data	Electrical and seismic refraction profiles	For deriving landmass movements
	Georadar data (see: http://www.georadar-gbr.com/), sound data	Georadar: for detecting any changes in the properties of interest in the underground.
Meteorological data	Temperature and rainfall data	For correcting other measurements data or for use in simulations in which rainfall and temperature can be influencing parameters
Hydrological data	Groundwater seepage (from rivers → only of importance when rivers are suspected to influence the landslide), levels of groundwater table in the boreholes/drillholes	For locating hydrological effects of rivers, watersheds, etc. on the landslide activity

Table 3: Overview of Data Required by a Landslide Monitoring Application

Landslide monitoring applications require different data from various sources, depending on the type of the landslide being monitored. The data can be of different formats, resolutions, dimensions (e.g., spatial, thematic); with different temporal characteristics, and quality. Usage of these different data in a single landslide monitoring can demand much work in converting it to the required target application format.

Also to enable reliable decision-making, accurate and up-to-date data must be readily available (i.e. in existing databases or from live sensors). This is a common problem faced by current monitoring applications.

3.2. Observable Properties and Measurable Parameters

In landslide monitoring, there are quite a number of parameters that have to be measured in order to accurately model the dynamics of earth mass movements or slope failure. The important observables include slope movements/displacements (based on the fixed-point observations) in the horizontal and vertical; velocity, rate of acceleration and directions of movements; sound measurements, changes in the levels of groundwater table, magnitude and variations of pore water pressure (rate of infiltration of water into a slope), surface elevations, depths of geological features (ditches, cracks, etc.), and area extent (e.g., for landslide zonation, geometry of observed features, …).

3.3. Requirements Analysis for Sensors

Definition:: Sensor: Is defined as any device, instrument or transducer that is capable of converting a measure (a quantity or parameter) into a signal (e.g. electrical, optical or mechanical) carrying information. At a very abstract level, a sensor system can also be seen as a complete sensor.

For the landslide displacements, in many cases, the accuracy expected is in the centimeter range [Gili et al., 2000]. The degree of slope instability is usually used to judge the frequency of sensor data acquisition (query time-interval) and the number and types of sensors needed for complete coverage of the monitored area. Following is a list of some of the important sensors that can be used for landslide monitoring [Kane et al., 2007; Kandawasvika and Reinhardt, 2005a; Mikkelsen, 1996; Beck and Kane, 1996; Dunnicliff, 1993]. The list is also a product of detailed analysis of the data requirements and several discussions with landslide monitoring experts.

3.3.1.1. Geodetic/Surveying/Remote Sensing Systems

Figure 13 shows a list of terrestrial-based and satellite-based sensing systems.

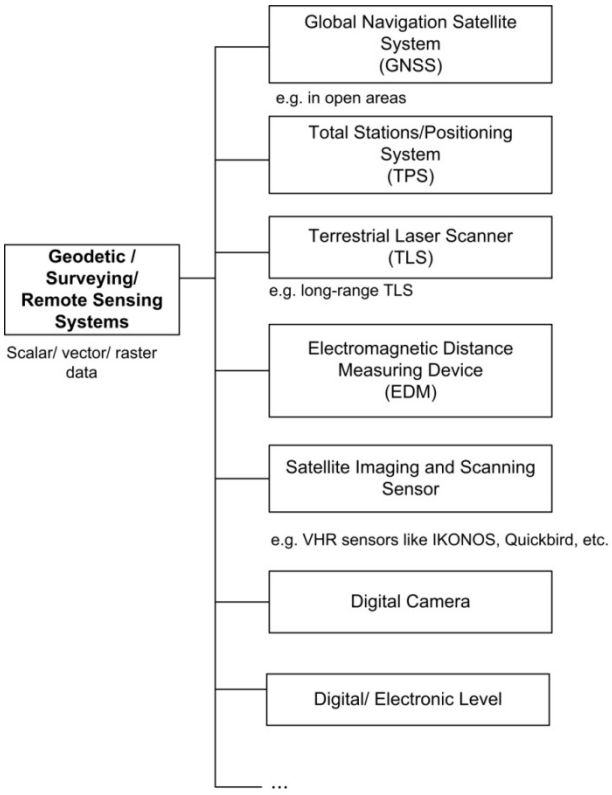

Figure 13: Important Geodetic/Surveying/Remote Sensing Systems

Global Navigation Satellite Systems

There are GNSS receivers available that support single or integrated navigation and positioning technologies like the global positioning system (GPS), Galileo, and GLONAS. Currently, GPS receivers are the most commonly used GNSS system in landslide monitoring applications. Figure 14 shows some examples of the GPS receivers.

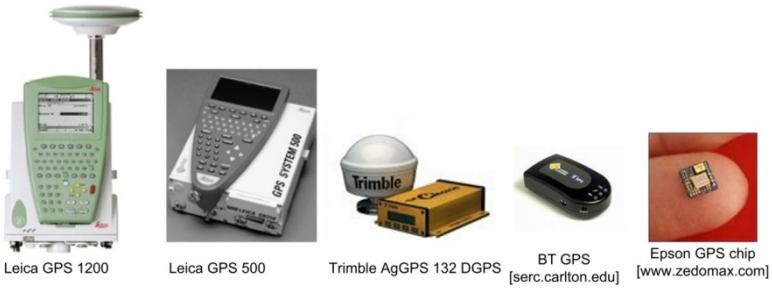

Figure 14: Examples of High Cost to Low Cost GPS receivers

GPS receivers greatly differ in size ranging from large receivers to tiny, embeddable chips. Wired and wireless receivers (with or without differential corrections capabilities) are available on the market, ranging from high cost to low cost. Time-tagged positions of control points observed by GPS receivers provide information about the surface displacements/movements. For accurate or reliable positioning, the signals from GPS satellites to the receiver should not be blocked (e.g., by high buildings, dense trees, etc.) or indirect (multi-path effects). Sub-centimeter accuracies can be achieved, but that largely depends on the mode of operation, type of the receivers and quality of antennae. For example, in landslide areas with

crowded vegetation, low-cost, very simple GPS cannot be reliable and might not even work.

Total Stations/Positioning Systems (TPS)

Figure 15: Examples of Total Stations

Surveying total (positioning) stations or electronic theodolites have greatly improved in capabilities. The current trend is toward wireless, reflector-less, fully automated or motorised instruments. Use of the automated tracking recognition (ATR) function enables the instrument to self-track measured targets. Some of the manufacturers like Leica Geosystems (Leica) and Trimble have developed total solution instruments integrating GPS.

TPSs provide time-tagged positions and other raw measurements (e.g., vertical and horizontal angles, slope distances …) of remote targets/points for determining displacement vectors of surface movements. In the absence of faster terrain modelling instruments like laser scanners, the total stations can be used to acquire DEMs or slopes. The placement of TPS depends upon line-of-sight and stability of the location. In most cases, the total stations are stationed far away from the landslide area, thereby avoiding any danger to the observers and the instrument.

Terrestrial Laser Scanning (TLS) instruments

Figure 16: Examples of Laser Scanners

The terrestrial laser scanning (also called *high definition surveying*) enables fast, non-contact measurements of objects or surfaces of interest. Different scanners with different characteristics and capabilities exist on the market. The 3D point-cloud data from laser scanners can be used to generate Digital Surface Models (DSMs) or Digital Elevation Models (DEMs) and 3D geometric models, for e.g., scans of interesting landslide features, such as ditches, cracks etc. Georeferencing of the point-cloud data can be done using geosensors like GPS positions or other geodetic instruments. For landslide monitoring, long-range terrestrial laser scanners (TLS), capable of measuring distances of several hundreds of meters,

are employed [Scaioni et al., 2004]. Long-range TLS uses the time-of-flight (TOF) method for measurement.

Digital/Electronic levels

The digital levels are used for measuring slope or surface elevations. In some cases, they are also used to monitor water levels in a slope.

Electromagnetic Distance Measurements (EDMs) and Measuring tapes

Measuring tapes and handheld EDMs are used for measuring distances on the site. The main advantage of handheld, reflectorless EDMs is that the observer can take measurements from safe positions without being in contact with the object being observed.

3.3.1.2. Satellite imaging and scanning sensors

Use of very high-resolution (VHR) satellite imagery, such as IKONOS, Quickbird, etc., is common in landslide monitoring. Satellite data can be used to determine the distribution of slope instability factors, such as geomorphology, lithology, and land use [Mantovani et al. 1996]. VHR images can provide centimeter accuracies, e.g. in the panchromatic bands. Satellite Interferometric Synthetic Aperture Radar (InSAR) techniques based on point scatterers extraction can be used in ground movement detection and monitoring [Ferretti et al., 2001].

Also, ground based or terrestrial InSAR techniques allow for the determination of relative displacements at different times with an accuracy of few millimeters [Scaioni et al., 2004].

3.3.1.3. Digital Cameras

Digital images taken at specific timestamps record the changes in terrain or slope appearance (e.g., new cracks, ditches or gaps, etc.) due to debris flow, vegetation collapse or other landslide effects. Digital cameras can be integrated with other devices like TLS in order to provide color texture of the observed objects, GNSS receivers for geopositioning the images, etc.

3.3.1.4. Geotechnical and geophysical sensors

Figure 17 shows geotechnical and geophysical sensors.

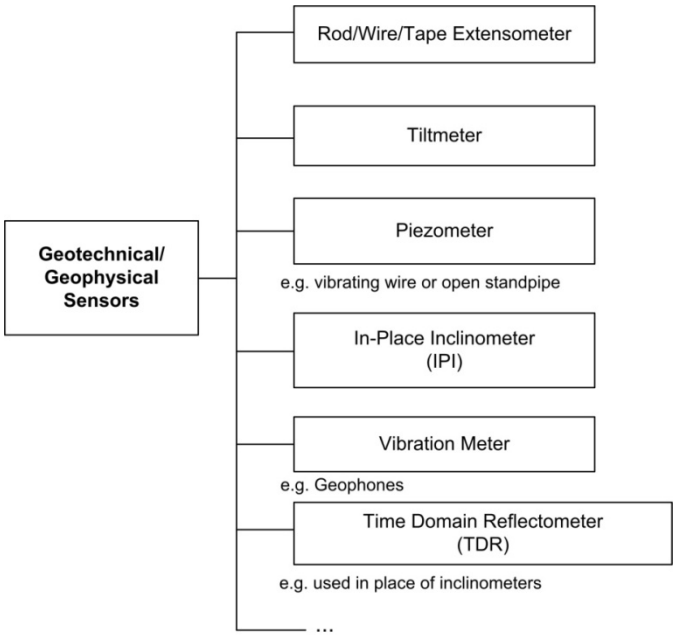

Figure 17: Important Geotechnical and Geophysical sensors

Extensometers (wire, tape or rod)

tape extensometer
[www.geotechsystems.com.au]

rod extensometer
[www.slopeindicator.com]

Observe expansions or contractions of the objects (e.g., ditches, cracks, slopes, etc.). The distance measurements can be corrected for temperature variations. Extensometers can also use potentiometers for measuring the linear movements.

Piezometers (vibrating wire or open standpipe)

They are used for measuring the groundwater levels. Kane and Beck [Kane and Beck, 2000] indicates that for accurate groundwater measurements, two piezometers are required - one for recording the atmospheric pressure and the other down-hole pressure. The difference between the two readings gives the actual water level.

Water levels are important for the determination of water pore pressure. High water pressures indicate that the slope is very sensitive to failure.

In-Place Inclinometers (IPI)

They can detect new movement, accelerations of movement, and the movement direction. These instruments are usually installed in drilled holes or boreholes.

Tiltmeters

Tiltmeters are mounted at the ground surface and are used to determine the angle of tilt of a slope.

Time Domain Reflectometers

They are used for measuring the relative magnitude and rate of displacement of a slope and also for monitoring the water levels. However, the direction of movement and absolute movement values cannot be ascertained by a time domain reflectometer (TDR) [Beck and Kane, 1996]. This type of instrument can be used in place of inclinometers for obtaining quick measurements.

Geophones or other vibration meters

These instruments can detect the ground vibrations caused by the slide movements. The higher the vibrations the more risk of slope failure.

3.3.1.5. Meteorological sensors

Figure 18 gives a short list of meteorological sensors.

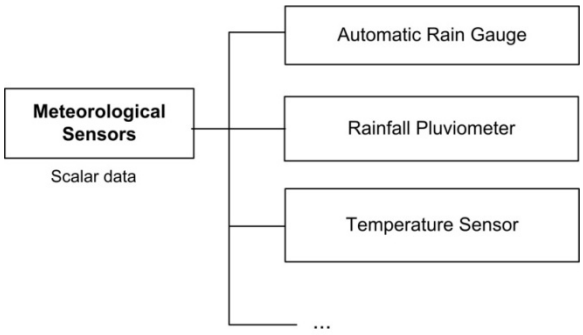

Figure 18: Examples of some of the Meteorological Sensors

Automatic rain gauges and rainfall pluviometers

These instruments are used to record precipitation.

Temperature sensors

Temperature values are necessary for correcting the measurements of other instruments.

Table 4 gives a summary overview of the discussed sensor systems and their data outputs (can either be raw or pre-processed).

Application Domain Category	**Sensor System**	**Data Output**	**Data Dimension**	**Data Type Category**	
Geodetic/ Surveying/ Remote Sensing System	*Global Navigation Satellite System (GNSS) receiver*	Positions, Coordinates Velocity	Spatial	Vector	Data stream (ASCII/ Binary)
		Time	Temporal	Scalar	
		Frequency	Thematic		
	Total Station (TPS)	Raw measurements (distances, angles, height differences)	Thematic	Scalar	
		Coordinates	Spatial	Vector	
	Electronic Distance Measurement (EDM)	Distances	Thematic	Scalar	
	Terrestrial Laser Scanner (TLS)	Coordinates with/without colour/intensity information (i.e. point-cloud data)	Spatial	Vector (Arrays of vectors)	
	Digital Camera, Satellite Imaging and Scanning System	Raw data (arrays of pixels)	Spatial/ Thematic	Raster (Pixel arrays/streams of digital numbers-DNs)	
		Images (single or multi-bands; panchromatic or multi-spectral bands)	Spatial/ Thematic	Raster (Matrices)	
	Digital/Electronic Level	Height differences	Thematic	Scalar	
Geotechnical/ Geophysical Sensor	*Rod/Wire/Tape Extensometer*	Distances	Thematic	Scalar	
	Tiltmeter, Inclinometer	Angles	Thematic	Scalar	
	Piezometer	Pressure	Thematic	Scalar	
	Time Domain Reflectometer (TDR)	Raw (voltages; impedance/resistance)	Thematic	Scalar, Data stream	
		Motion/movement	Thematic	Scalar and/or vector	
	Vibration meter	Motion/movement	Thematic	Scalar and/or vector	
	Geophone	Sound	Thematic	Scalar	
Meteorological Sensor	*Temperature Sensor*	Temperature	Thematic	Scalar	
	Pressure Sensor	Pressure	Thematic	Scalar	
	Automatic Rain Gauge, Rainfall Pluviometer	Rainfall/ precipitation	Thematic	Scalar	

Table 4: Overview of Sensor Systems and their Measurement Data Output

As shown in the table, the sensor outputs are further categorized into respective data measurements dimension and type category like scalar, vector, data streams, etc. *Vector quantities* are data values that

have both magnitude and direction whereas *scalar quantities* have only magnitude and no direction association. A *data stream* can be seen as a composition of an ordered pair (s,Δ) where *s* is a sequence of tuples and Δ a sequence of time intervals [Wiki]. The tuples can be composed of only scalars, only vectors or both. For example, temperature, pressure or sound scalar data can be streamed by sensors at specific time intervals, with or without the information about the geographical position where the measurements are being taken. Also raw measurements and data streams can be subject to post-processing. In short, the categorization provided in the table is valuable input for sensor classification (see Chapter 5.1) and for the actual modelling of the sensors (see Chapter 6.2).

For effective use of sensors in landslide monitoring applications, sensors/sensor systems are usually deployed in form of sensor networks. Therefore, it is also important to discuss about sensor networks and their importance in landslide monitoring. A simple architecture is given which shows the main components we consider important for a landslide monitoring system.

3.4. Sensor Networks

Sensor networks form the core part of a monitoring system. A sensor network can be seen as the data acquisition system (DAS) of a monitoring application. The sensor network data can be transmitted (wirelessly or wired) to a control/base station for subsequent processing and, if necessary, for early warning (see Figure 19). It is important to note that advances in technology (computing, communication and information) and related economical factors as well as physical factors like landslide scale (size) and type of landslide determine the sensor requirements (selection).

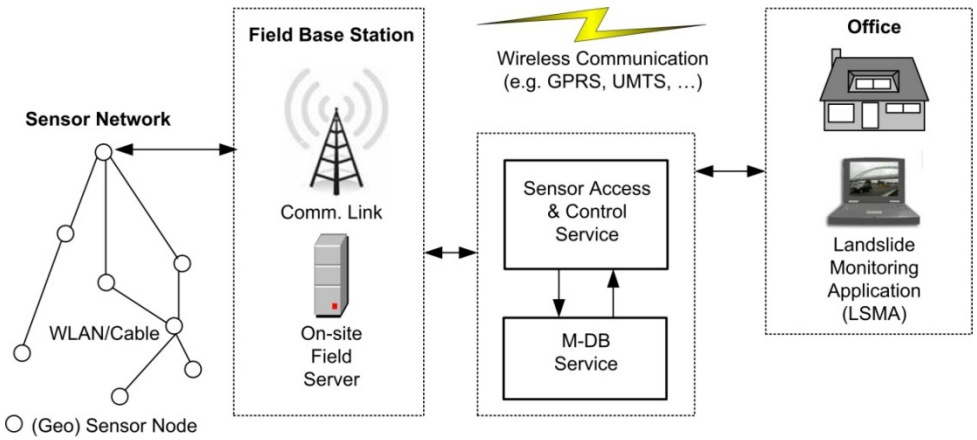

Figure 19: A Simple Architecture for a Sensor Network-based Monitoring System

In the above figure, geosensor nodes can have leading nodes (i.e. group leaders) that are responsible for communicating with the base station. A *geosensor* can loosely be defined as a sensor that is capable of acquiring or collecting geographic (spatial) data and it's locatable in a spatio-temporal space.

A geosensor node has the following responsibilities:

- Georeferencing of other sensor nodes that are not location-aware (e.g. some of the geotechnical sensors (e.g. extensometers …) and meteorological sensors (e.g. temperature, water gauges, etc.)).
- Sensing the object or area of interest, and deliver the data at the times or intervals pre-configured by

the sensor access and control service (SACS), see Chapter 6.3.

- Internal pre-processing of the data (i.e. measured signal) into other transmittable format (e.g. analogue-to-digital conversion).

- Communicating with other sensor nodes, for e.g., when there is need to transmit data to the leading node via other sensor nodes or direct to the base station.

The SACS relies on the measurements database. Both SACS and the M-DB can be running on the field server. It is also important that the sensor network operates in real time, especially in times of danger, since the acquisition of the data has to be available at the right time. The acquired data should accurately reflect the current events occurring in the physical world. However, the integration of sensors into the sensor network is usually done offline (i.e. not in real-time).

3.5. Management of Disparate Sensors and their Measurements Data

Applications using multiple sensors can greatly benefit from the individual contributions of each sensor, since different sensors are inherently designed to sense different aspects of phenomena. A single sensor is inadequate most of the times, but the cooperation of many sensors usually creates usable results [Bill, 2008]. In general, the benefits of multi-sensor integration are:

- redundancy (having more than minimum required sensors, thereby improving the accuracy and reliability of the collected data),

- complementarity (if some sensors fail or cannot observe particular properties of interest, other sensors can complement them),

- timeliness (real-time data acquisition), and

- reduction in costs of data acquisition (e.g. by having more than adequate complementary sensors, if errors occur due to some faulty sensors there is no need to carry out the measurements campaign again. It can be sufficient only to exclude the faulty sensor measurements).

Landslide monitoring applications can utilize various, different sensors (see section 3.3) and data (see section 3.1). Addressing challenges concerning management of disparate sensors and their measurements data caused by usage of proprietary (vendor-specific) solutions are very important research issues [Jellema, 2008; Walter et al., 2008; Balanziska et al., 2007; Stefanidis et al, 2005; Nittel et al., 2005; Dantu et al., 2004; etc.] and are the core issues of the dissertation. The early sections of this chapter have discussed the problems or conflicts that can lead to these challenges. Walter et al. [Walter et al., 2008] also notes that current early warning systems are still being treated as “black boxes” – they are not open or not standardized.

The following subsections discuss monitoring systems we have analysed and also highlight their respective approaches to sensor and measurement data management.

3.5.1.1. Leica Geodetic Monitoring System (GeoMoS)

System description: Leica's GeoMoS system is one of the common systems used in different geoscientific monitoring applications [http://www.leica-geosystems.com]. It has two main components: the *monitor* and the *analyzer*. The monitor is responsible for controlling sensors, data acquisition, tolerance checking, and event management. Offline work like data analysis, post-processing, and visualisation is done using the analyzer. GeoMoS uses an *SQL Database* for the storage of measurements data.

Sensors integration: With GeoMoS, it can be possible to integrate various sensors like meteorological, geotechnical, and geodetic sensors. As a solution to multi-sensor integration, Leica has developed a *generic sensor manager*. The manager relies on simple XML sensor description files. The description files for GeoMoS contain sensor information, such as protocols, commands, properties, and configuration. Sippel 2001 concludes that by reading these sensor descriptions, a sensor manager can integrate easily different types of sensors into an application. However, the problem with this approach is that there are no common generic or standard-based schemas defined for each type of sensors. Therefore, the solution can easily work only for vendor-specific software application.

The measurements data can be exported using formats like ASCII, DGN, WMF, and Microsoft Excel format.

Software libraries and Application Programming Interfaces (APIs): Proprietary interfaces like GeoCOM, GeoC++ and GSI (ASCII commands).

Communication mechanism: Various communication technologies are used for sensor control and data acquisition, for e.g., wireless local area network (WLAN), cable, bus system and other radio links.

3.5.1.2. GPS-based Online Control and Alarm System (GOCA)

System description: GOCA is a deformation analysis system which has been developed by EuroNav Entwicklungsgesellschaft mbH and the University of Karlsruhe (Hochschule für Technik), Germany [http://www.goca.info]. The system comprises of GPS receivers, communication hardware, control and communication software as well as software for deformation analysis and alarm activation. About 25 installations have been deployed world wide.

Sensors integration: Mainly supports GPS receivers. It can also integrate some of the classical local positioning sensors (LPS), such as total stations for real-time deformation monitoring. For integrating the sensor data, vendor-specific data interface "GKA" (for GNSS/GPS and LPS data), and the standard NMEA interface are used. For connecting the sensors, vendor-specific solution has been used.

Software libraries and Application Programming Interfaces (APIs): Proprietary interfaces based on Microsoft Foundation Classes (MFC) libraries and American National Standards Institute (ANSI) C language.

Communication mechanism: System internal communication is via radio (Real Time Kinematic (RTK) connection by means of a radio modem).

3.5.1.3. GeoSens Felsmonitor (FeMon)

System description: GEOsens [http://www.geosens.de] is a debris or mass movement monitoring system. [http://www.leica-geosystems.com]. The system comprises of the field sensors (e.g. movement detection sensors), base station, communication link; signal alarm receivers (e.g. Global System for Mobile (GSM) communication modem plus a micro-controller, for e.g., for changing the traffic robots to red in times of danger), and the monitoring software (ADIOS).

Sensors integration: Supports movement detection sensors like extensometers, which can transmit their data to the base station via radio link or cable. The measurements data is encoded in ASCII and the alarm message from the base station to the responsible authorities is normal SMS text. The sensor data are usually stored in Microsoft Access format.

Software libraries and Application Programming Interfaces (APIs): Proprietary interfaces based on Microsoft Windows 32 API and C++ or Delphi 6 for programming.

Communication mechanism: The system uses integrated services digital network (ISDN) for the transmission of measurements data and alarm message.

3.6. Summary

The chapter has covered the analysis needs and current situation in landslide monitoring applications. In addition, we have already developed own categories for different sensor systems and their measurements data output. This categorization is important for sensor model development. This chapter has also shown the proprietary approaches used by different monitoring applications in order to handle multi-sensors.

4. Basic Concepts and Definitions

This chapter covers the basic concepts and definitions that are important for the dissertation.

4.1. Modelling of Sensor Systems (Instruments)

For better understanding and effective use of physical devices, sensors or sensor systems (or any other data source) in an application, there is need for creating models that fully describe those systems. In general, modelling can be performed at three main levels of abstraction which are: *conceptual*, *logical*, and *physical* modelling.

4.1.1. Conceptual modelling

Conceptual modelling can be defined as creating an abstract description of some part of the real world together with a set of related concepts. For example, the human brain is capable of interpreting or recognising objects of the physical world by using each person's *conceptual models* of those objects. The models may already be installed in the subconscious part of the brain. Those models may be built based on several factors such as the level of knowledge of the person, culture, living environment, experience, etc. This example can be applicable to any other system. For a sensor-based application, the environment (e.g. phenomena to be observed), sensor network, sensors, sensor data, and other related concepts have to be modelled.

ISO 19101 Geographic Information – Reference model [ISO-19101, 2002] states that a *conceptual model* is a model that defines concepts of a *universe of discourse*. The universe of discourse is defined as "a view of the real or hypothetical world that includes everything of interest". In the sensor world, the universe of discourse can be defined by a sensor network or even a single sensor together with the sensed environment.

In order to formalise a conceptual model (i.e. create a conceptual schema), a *conceptual schema language* is required. A conceptual schema language provides a uniform method and format (i.e. conceptual formalism) for describing information, in such a way that both people and machines can read, understand and even process it. A conceptual formalism provides a set of modelling concepts such as rules (semantic, syntactic …), constraints, inheritance mechanisms, events, functions, processes, etc. [ISO-19101]. Examples of schema languages are the unified modelling language (UML) with or without object constraint language (OCL) [OMG], object definition language (ODL), EXPRESS [ISO-10303-11, 2004], IDEFX, and several others.

Following are explanations of some of the conceptual modelling concepts [Barkmeyer et al., 2003]:

1. **Functional modelling**

   Functional modelling identifies all the functions that a system is designed to perform, including the required inputs and the generated outputs. The following aspects are part of functional modelling:

   - Functions
   - Activities

- Information flow (I/O)
- Events
- Interactions
- Time
- Resources

The capabilities of the system are reflected in the functional model.

2. **Activity modelling**

   It is a process in which the activities or interrelated tasks of a system are defined in form of a partially-ordered graph (e.g. sequence diagram). The inputs and outputs of the tasks together with their relationships are modelled. The temporal component (time) of these tasks is important. The activity model shows 'what' a system does and 'when'.

3. **Process modelling**

   It can be defined as a *specialised* form of functional or activity modelling. Unlike the functional modelling, process modelling clearly specifies the behaviour of a system, usually at interface level. It describes the following:

   - How exactly does a system respond to a certain stimulus (physical input or event)
   - What specific actions the system takes in response to that stimulus
   - What are the results of the performed actions and the level of reliability associated with the generated outputs.

   The basic unit of modelling is a *process,* which can be seen as an activity that takes specific input(s), and depending upon the conditional parameters and implemented algorithms or methodologies, it then generates output(s).

4.1.2. Logical modelling

Logical modelling produces models with sufficient details required for implementing the models in target environments. For example, in conventional GIS, logical modelling of data sources outputs data models (include features/entities, relationships among those entities, and other associated concepts) that are detailed enough for physical implementation, but still not include or rather independent of the actual implementation environment details. This approach is adopted for the logical sensor models. However, when modelling a sensor system (instrument), the modelled components should not necessarily reflect one-to-one description of the physical device's components as depicted by the manufacturer's blueprint. The level of abstraction should only show the components deemed relevant for enabling people and applications to carry out their tasks using the modelled sensor. The basic idea is to reflect the logic (e.g. functional model - behaviour, purpose, usage, constraints, etc.) of the sensor as needed by the application in order to effectively and efficiently fully exploit that sensor.

For logical modelling, schema languages, like Schematron, XML Schema, RelaxNG, and several others, are usually used.

4.1.3. Physical modelling

Physical modelling maps the logical models to lower levels and includes the implementation details (e.g. specific hardware platform, storage devices or modules, operating system, specific programming language, etc.). In the case of databases, like Oracle's spatial database, Microsoft's SQL server, postgres, etc., physical models show how data are actually stored in physical structures like tables, clusters, etc. and also describe the methods for accessing and manipulating those structures. With the help of the manufacturers of sensors, it can be possible to develop physical models (e.g. XML based) that reflect one-to-one mapping of that manufacturer's blueprint.

4.2. Sensor Model

In the literature, there is no clear or unambiguous definition of a sensor model. Durrant-Whyte [Durrant-Whyte, 1998] defines a sensor model as "an abstraction of the actual sensing process that describes the information a sensor is able to provide, how this information is limited by the environment, how it can be enhanced by the information obtained from other sensors, and how it may be improved by the active use of the physical sensing device." Durrant-Whyte also points out that "the key to intelligent fusion of disparate sensory information is to provide an effective model of sensor capabilities". However, Durrant-Whyte does not exactly define what an `effective´ model is.

In the OGC Sensor Model Language (SensorML) standard [OGC-SensorML, 2007], the definition is adopted from the remote sensing community which defines it as "a type of a *location model* that allows one to georegister observations from a sensor". A location model is then defined as a "model that allows one to locate objects in one local reference frame relative to another reference frame".

The dissertation does not only see a sensor model as a *location model*, but a more detailed logical model. Figure 20 graphically shows the composition of a sensor model from this dissertation's perspective (own definition). We therefore define a *sensor model* as a logical model that comprises a set of models that includes information and data model (e.g. includes observation and measurements models with semantics …), communication model (higher and lower-level definition of protocols and interfaces), state or behaviour model, computational model (functional aspects), metadata model (include location model and other metadata information), and other dependencies (e.g. subcomponents like CPU, storage devices, battery …).

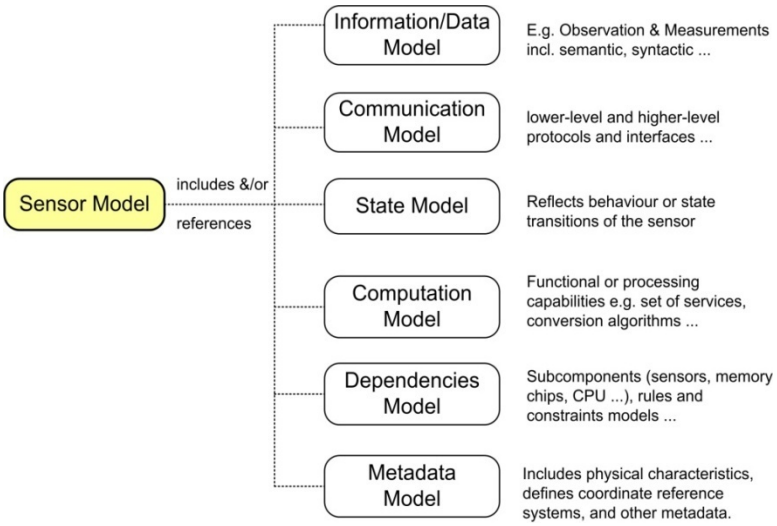

Figure 20: Definition of a Sensor Model

***Information/ data model*:** Describes the sensor data input and output. The output can be composed of sensor measurements and observations together with semantic information. The data types and structures supported by the sensor contain some syntactical information.

***Communication model*:** Describes the *interfaces*, *protocols*, and interaction mechanism for communicating with the sensor system. Higher levels interfaces/protocols are used, in the dissertation, to refer to the connection, command and control interfaces. Lower levels interfaces are defined as those needed for establishing the physical communication with the sensor systems (reference is given to the OSI/ISO model).

- *Interface*

  In computer science, an interface is defined as an abstraction of a software component. In programming languages, like Java, C++, etc., it refers to an abstract type which specifies a kind of "contract" that classes implementing that abstract type should abide to. According to the dictionary of military and associated terms by the US Department of Defence [Dict.-US-DoD, 2005], an interface "is a boundary or point common to two or more similar or dissimilar command and control systems, sub-systems, or other entities against which or at which necessary information flow takes place". The latter definition is adopted for this dissertation.

- *Protocols*

  Following are some of the definitions that can be applied to protocols:

  - Defines the mapping of the interface specifications to the physical implementation of the communication message units and the *rules* for formation, transmission, receipt, acknowledgement and sequencing of messaging units, without regard to their content.

  - Specifies a set of rules that guarantee interoperability among implementations of various communication mechanisms.

  Examples of protocols that can be used to support interoperable communication are:

- HTTP (HyperText Transfer Protocol) [W3C-HTTP]
- SOAP [W3C-SOAP, 2007] (originally stands for “Simple Object Access Protocol”[W3C-SOAP, 2000])
- FTP (File Transfer Protocol)[IETF-RFC-959, 1985]
- SMTP (Simple Mail Transfer Protocol)[IETF-RFC-2821, 2001]
- Representational State Transfer (REST): originated from dissertation by Fielding [Fielding, 2000]
- SQL (Structured Query Language) call level interface [ISO/IEC-9075-1, 2008]
- Etc.

***State model*:** Describes information about the state or condition of the sensor at the time of measurements. This includes information about the sensor calibration parameters and other configurable system parameters that change the behavior of the sensor system.

***Computation model*:** Describes the functional, measurement or processing capabilities of the sensor. This includes the set of services the sensor system provides (e.g. data conversion, various measurement functions, etc.).

***Dependencies model*:** Describes the entities or data/information that the sensor system depends upon in order to operate and fulfill the task at hand. This includes internal sensor components like sub-sensing devices, power/batteries; and non-physical components like rules and constraints model (i.e. govern the behavior of sensor), etc.

***Metadata model*:** Documents the data about the sensor system, which includes the location model and summaries of the details provided by all other above discussed models. This model contains information that is necessary for discovery and mining of sensor systems. Metadata also provide valuable information for evaluation and binding (i.e. integration) of a sensor into a sensor network. Sensor registry services can use this model as the entry point or portal for searching application relevant sensor systems.

It is very important to note that the level of detail (abstraction) in a sensor model is usually determined by the application domain needs. By analysing sensor specifications (i.e., functional, hardware and software) from different sensor manufacturers and matching them with the application domain requirements, it is possible to define generic interfaces that are common for each given sensor type. The workflow for the realization of generic models is given in Chapter 6.1. However, the development of generic interfaces is an iterative process that involves several tests and modifications within a given application (e.g. landslide monitoring).

4.2.1. Sensor System

It is a composition of physical sub-sensing devices (e.g., sensors, actuators, detectors) and non-physical, non-sensing sub-components (e.g., computational algorithms, filters). All parts of the sensor system that can be modelled are regarded as the *components* of that system. Figure 21 shows an example of a sensor system.

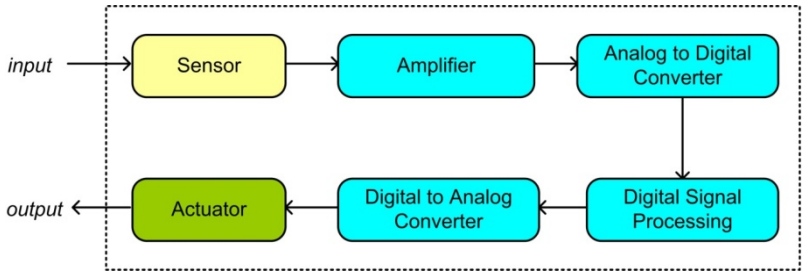

Figure 21: Example of a Sensor System

Note that a complete single sensor (e.g. comprising an array of detectors) can also be regarded as a sensor system. This definition of a sensor system is considered when modelling sensors (i.e. defining the sensor components, etc.) – see Chapter 6.2.3. Also the terms measuring instrument or device can be used to refer to a sensor system.

- ***Sensor***

  It's a device that responds to a physical stimulus (e.g., thermal energy, electromagnetic energy, acoustic energy, pressure, magnetism, or motion) by producing a *signal*, usually electrical. A signal being any detectable transmitted energy that can be used to carry information [ATIS].

- ***Actuator***

  It's a type of a transducer that can be seen as a simple element that converts a signal to some real world action or event. For example, if a landslide movement being measured by an extensometer exceeds a pre-defined threshold, a signal is send to a nearby road traffic light which has an actuator that causes the light to turn red, thereby stopping the traffic using that road.

- ***Component***

  It's any part, entity or object of a sensor system. It applies to components that are either physical (e.g. a detector) or non-physical (e.g. a software module for coordinate transformation).

4.2.2. Quality Concepts

A sensor model has to consider or include the relevant concepts of quality. *Quality* is about relevance, timeliness, completeness, accuracy, accessibility, clarity, punctuality, consistency, cost efficiency, integrity, neutrality, robustness, soundness, and security [Schouppe, 2008]. A formal definition is adopted from the ISO 19113 Quality Principles standard [ISO-19113, 2002] that defines it as the "totality of characteristics of a product that bear on its ability to satisfy stated and implied needs". Quality is a very important aspect of any model. For the dissertation, the following concepts of quality are of importance.

4.2.2.1. Accuracy

"It is the closeness of agreement between a *test result* and the accepted reference value [ISO-3534-1, 1993; ISO-19113, 2002]". A test result can either be an observation or a measurement. In other words, accuracy can be defined as "the degree of closeness of an observation or a derived quantity to the 'true' – but unknown – value" [ICSM, 1996; Wolf and Ghiliani, 1997]. The true value is usually estimated through computations and measurements. Following are some of the expressions that are used to describe accuracy.

1. *Standard deviation*: is a measure of spread or dispersion of a set of values (also called r*adial error*). It is the square root of variance. Sigma (σ) is usually used to represent standard deviation.

2. *Root Mean Square (RMS)*: It is the square root of the arithmetic mean of the squares of a set of numbers. It is sometimes used as a synonym for standard deviation.

3. *Two Distance Root Mean Squared (2DRMS)*: It is twice the radial error of the standard deviation.

4. *Circular Probable Error (CEP)*: It is the value of the radius of a circle containing 50% of the position estimates and the centre of the circle being the actual position.

5. *Spherical Error Probable (SEP)*: It is the spherical equivalent of CEP with a radius containing 50% of 3D position estimates.

4.2.2.2. Precision

It is a measure of closeness among a series of individual measurements or values. It can also be defined as the degree of closeness and consistency of repeated measurements of the same quantity to each other. Note that a measurement can be precise but not accurate or it can be accurate but imprecise.

4.2.2.3. Resolution

It is the smallest count or change that a sensor can detect in the quantity that is being measured. The resolution is related to the precision with which the measurement can be made.

4.2.2.4. Reliability

It is a quality of measure. It generally determines the 'consistency' and 'repeatability' of a measurement about the measured property.

For sensor networks, the *quantitative theory of reliability* initially developed for the geodetic networks by Baarda [Baarda 1967, 1968] can be considered. From the theory, an observation network should be able to *resist systematic* and *gross errors* in order to increase its reliability [Staudinger, 2000]. This is also applicable to sensor networks.

4.3. Summary

This chapter has covered the basic concepts and definitions that are fundamental to the dissertation. It has explained the modelling concepts, and defined terms and elements that are important for sensor modelling. It is important to note that we have also provided our own definition of a sensor model.

5. Sensors and Methods of Integration

In this chapter, the first sections define a classification scheme and explain simplified component and functional models of sensor systems that have been selected for our landslide monitoring application scenario. The last sections discuss the methods and techniques that can be utilised to integrate sensors into existing legacy or new applications.

5.1. Classification of Sensor Systems

A classification scheme can serve as an important part of a sensor registry service (e.g. a portal, a catalogue or a Universal Discovery, Description, and Integration (UDDI) service). Users (people and software) can easily discover sensors of interest or relevance based on a given classification scheme and/or other selection variables.

In general, it is rather difficult to have a common way of classifying sensors. There are several possible classification schemes for grouping sensors like by their behaviour (e.g., passive or active), by locality (e.g., remote or in-situ), by construction (e.g., MEMS), by measurement techniques (e.g., piezoelectric, capacitive, thermoelectric), by measuring principles (e.g., phase comparison, pulse), by what they measure/sense (e.g., optical, ultrasonic, thermal, etc.), by level of automation (none, semi, or full), and frequency of measurements.

For the dissertation, the taxonomy of the sensors is according to the *sensor output* – measurement data dimension and type (refer to Table 4 for details).

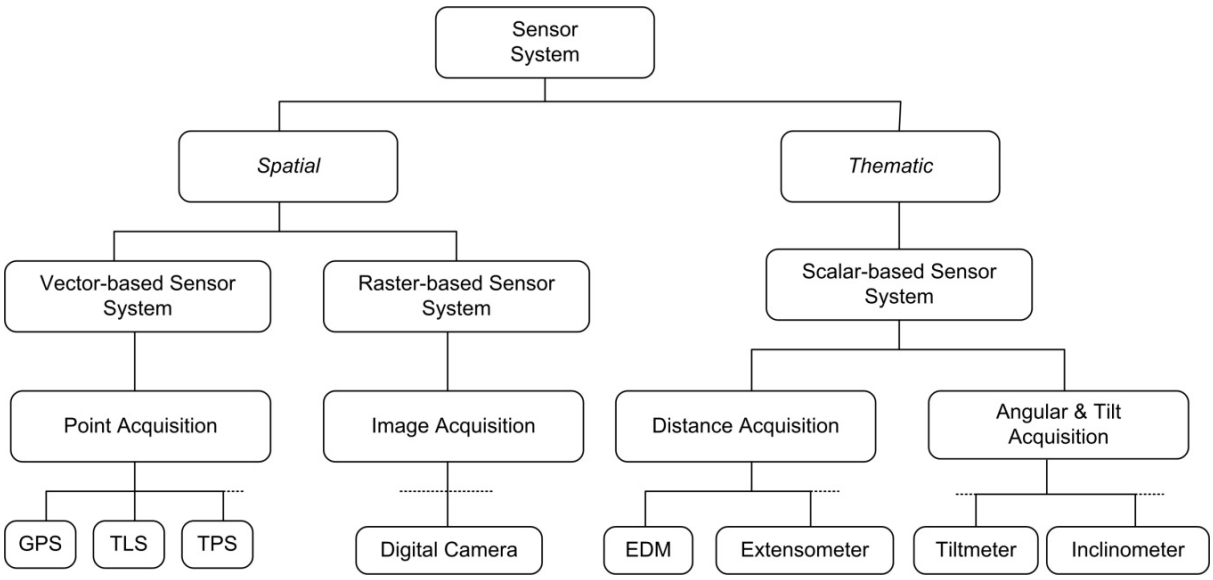

Figure 22: Sensors Classification Scheme

Figure 22 shows a specific classification scheme of sensors that is defined for dissertation. The example measurement types can be points (2D, 3D), images (e.g. multispectral images), distances; angular and tilt measurements (e.g. based on motion, vibration, or leveling).

5.2. Simplified Functional and Component Models of Sensors

The following subsections explain the basic component and functional models (including the input and output (I/O)), as well as the working principles of the selected sensors. These models are necessary for the definition and specification of logical sensor models.

5.2.1. Point Acquisition Sensors

5.2.1.1. Total Stations/Positioning Systems (TPS)

A TPS is a complex sensor system composed of several electrical, mechanical and optical components. Figure 23 shows a simplified model of a TPS sensor, which reflects the basic opto-electronic concept and interconnection of different sub-sensors and other dependencies [Becker et al., 2000].

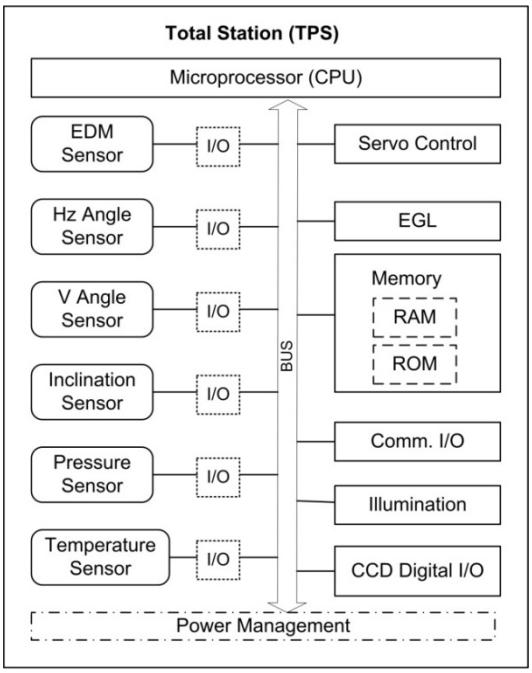

Figure 23: Simplified Functional Model of a Total Station or Electronic Tacheometer.

The in-built sensors include a distance measuring unit (EDM), angle sensors (Hz - horizontal, V - vertical), inclination sensor, etc. The communication I/O interface allows the sensor system to be able to communicate with external devices (e.g. PC or controller via wireless (e.g., Bluetooth, WLAN), cables (e.g., serial interface), or data cards and storage drives (e.g., PCMCIA drives, OMNI)).

For the processing of the raw measurements data of the TPS into final measurements, calibration data from the manufacturer and/user are needed to provide corrective quantities. Calibration data includes, for example, tilting-axis, vertical-index (circle) error corrections, scale factor, zero correction, etc.

Data output:

TPS system output data includes: distances, heights (elevations), angles, orientations, coordinates, geometries (points, lines, and areas), etc. Most manufacturers provide the output data as ASCII data blocks or files, e.g., Topcon's proprietary format, Leica's Geo Serial Interface (GSI) formats. Figure 24 shows an example of GSI 16 character data format. Within each data sequence, as shown in the example,

codes are used to identify the data, e.g. point number (11), horizontal angle (21) and vertical angle (22). Note that different manufacturers use different definitions for the data blocks.

```
110001+000000000PNC0055 21.002+0000000013384650 22.002+0000000005371500
110002+000000000PNC0056 21.002+0000000012802530 22.002+0000000005255000
110003+000000000PNC0057 21.002+0000000011222360 22.002+0000000005433800
110004+000000000PNC0058 21.002+0000000010573550 22.002+0000000005817600
110005+000000000PNC0059 21.002+0000000009983610 22.002+0000000005171400
       |←  16 char.  →|
```

GSI16 Datablock Structure:

Pos.1-2:	Word Index (WI)	e.g. "11"; WI code
Pos.3-6:	Information related to data	e.g. "0002"; number of lines
Pos.4:	Sign	e.g. + or -
Pos.8-23:	GSI16 data (16 digits)	e.g. "000000000PNC0058"; Pointnumber
Pos.16/24:	Blank (=separating character)	

Figure 24: Leica GSI 16 Format Example [Leica Geosystems]

Interfaces:

For accessing and controlling the TPS, different manufacturers provide their own software libraries (interfaces and protocols), e.g., Leica's GSI protocols, GeoCOM (ASCII RPC, Visual Basic, and C++), and GeoC++ [Leica]; Topcon's C++ SDK GTS-720/GPT-7000 [Topcon], etc.

As an example, Figure 25 shows a client and server architecture used for communicating with a Leica TPS.

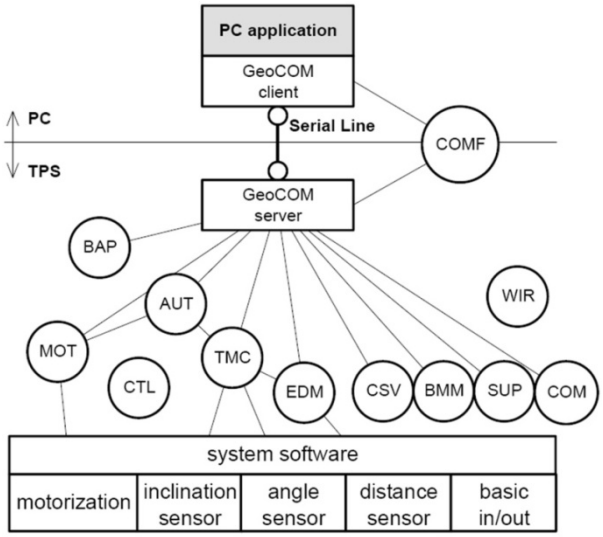

Figure 25: Overview of Client/Server Architecture [Leica]

Table 5 provides a brief key to the above diagram.

Component	Description
AUT	Automisation. Examples of functions are Automatic Target Recognition (ATR), Positioning, Change Face, etc.
BAP	Basic Applications, e.g. for measuring
BMM	Basic Man Machine functions, e.g. basic I/O control, alarming, etc.
COM	Communication
EDM	Electronic Distance Meter
CSV	Central Service, which provides system *get* and *set* functions
MOT	Motorization; controls the movement of the TPS system
TMC	Theodolite Measurement and Calculation
WIR	WI (Word Index) Registration. It provides GSI recording functions

Table 5: TPS Components

The client (e.g. PC) communicates with the server (the total station –TPS) via a serial line. GeoCOM is based on SUN Microsystems' Remote Procedure Call (RPC). For supporting other platforms (e.g. other programming languages), the ASCII RPC-based protocol can be used. Each command or RPC call has an identification number, which is send to the TPS together with other request parameters. Basically, the GeoCOM implements a *request-response model*, that means a request and response pair has to be complete before another communication is allowed (i.e. synchronous communication).

5.2.1.2. Global Positioning Systems

A GPS receiver is a sophisticated system that takes satellite radio signals as input and computes position, velocity, time, etc. For positioning (3D, time), a minimum of 4 satellite signals is required. Figure 26 shows a simplified model of a GPS receiver.

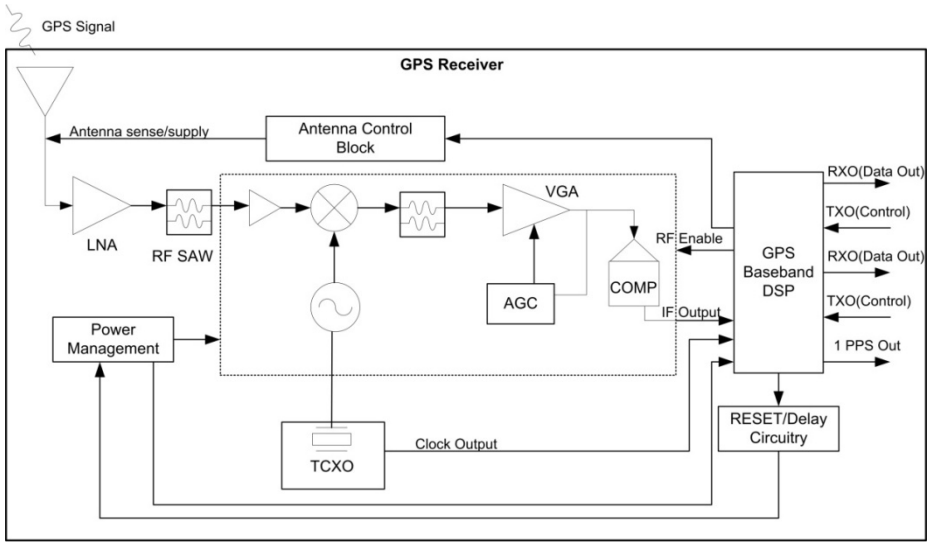

Figure 26: Simplified GPS Receiver Solution Block Diagram [Maxim, 2005]

Table 6 briefly explains the main components shown in the GPS receiver block diagram.

Component	Description
Low Noise Amplifier (LNA)	Analog radio amplifier. The first stage in a GPS front-end receiver.
Radio Frequency (RF) Surface Acoustic Wave (SAW)	It's a filter used to select desired frequencies
Automatic Gain Control (AGC)	A closed-loop control system that is used to keep amplifier *gain* constant under varying signal strength conditions. It is used in the intermediate frequency (IF) portion of the receiver. Gain is usually defined as a ratio of signal input to signal output.
Variable Gain Amplifier (VGA)	Is controlled by either AGC loop or external signal.
Mixer	A device or component that accepts as input 2 different frequencies and outputs a mixture of signals at several frequencies.
Baseband Digital Signal Processing (DSP) chip	Processes the digital data streams. Baseband interface includes external control command (TX), GPS data out (RX), and a one pulse per second (PPS) signal – synchronized with GPS clock.
Temperature Compensated Crystal Oscillator (TCXO)	Reference clock. It supplies the baseband with time information.

Table 6: GPS Receiver Components

Data output:

GPS main output streams consist of position (2D, 3D) data, velocity, time (GPS time), and information about the quality of the data. The quality information is given in form of geometric dilution of precision (GDOP). Following are the components of GDOP:

- Position Dilution of Precision (PDOP) or spherical DOP for 3D positions.
- Horizontal Dilution of Precision (HDOP) for 2D positions, e.g. geographical coordinates (latitude, longitude)
- Vertical Dilution of Precision (VDOP) for height information
- Time Dilution of Precision (TDOP) for time information

GDOP elements give information about reliability (i.e. poor or good), but not accuracy.

The GPS streams (i.e. message sentences) are usually encoded in NMEA format (e.g. NMEA 0183). Each NMEA sentence starts with a "$" and ends with a carriage return (CR)/line feed (LR).

NMEA allows hardware manufacturers to define their own proprietary sentences. All the GPS standard sentences are identified by two letters "GP" in the message ID, and proprietary sentences with a letter "P". Mixing the two outputs leads to a non-standard output. Following in Figure 27 is an example of a standard sentence.

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*42

GGA : Global Positioning System Fix Data
123519 : Fix taken at 12:35:19 UTC
4807.038,N : Latitude 48 deg 07.038' N
01131.000,E : Longitude 11 deg 31.000' E
1 Fix quality: 0 = invalid
1 = GPS fix
2 = DGPS fix
08 : Number of satellites being tracked
0.9 : Horizontal dilution of position
545.4,M : Altitude, Meters, above mean sea level
46.9,M : Height of geoid (mean sea level) above WGS84 ellipsoid
(empty field) : time in seconds since last DGPS update
(empty field) : DGPS station ID number
*42 : the checksum data, always begins with *

Figure 27: NMEA 0183 Sentence Example

Furthermore, some manufacturers provide binary proprietary interface protocols, which demand a developer to have thorough understanding of the bit or byte structures before communicating with the GPS.

Interfaces:

Different manufacturers as well as software vendors use their own APIs, depending on the development platform, to communicate with GPS devices. There are several open source and commercial libraries (C/C++, Java, etc.) available, e.g., JavaGPS [http://javagps.sourceforge.net], Garmin I/O SDK C/C++ [Garmin], etc.

As an example, the Garmin SDK is based on proprietary Garmin Device Interface Specification [Garmin, 2004] which specifies the serial protocols (e.g. serial data packet format, handshaking …), USB protocol and drivers, application protocols (e.g. device commands), and specific data types (based on C language). The communication protocol is based on a "stop and wait" concept, like the RPC procedure discussed for the TPS. The sender (e.g., GPS) has to wait for a reception acknowledgement from the receiver (e.g. PC) before sending new data.

On the physical communication layer, wireless (e.g. Bluetooth), and special serial cables (e.g. electrical and mechanical connections, such as standard DB-9 or DB-25) can be used.

5.2.2. Image Acquisition Sensors

5.2.2.1. Digital Cameras

Digital cameras are continuously advancing in their design and functionality. Most of the modern monitoring applications are employing digital cameras or Digicams to capture images and videos, which provide vital visual evidence about the observed events. Figure 28 shows a simplified functional model of a Charge Coupled Device (CCD) digital camera.

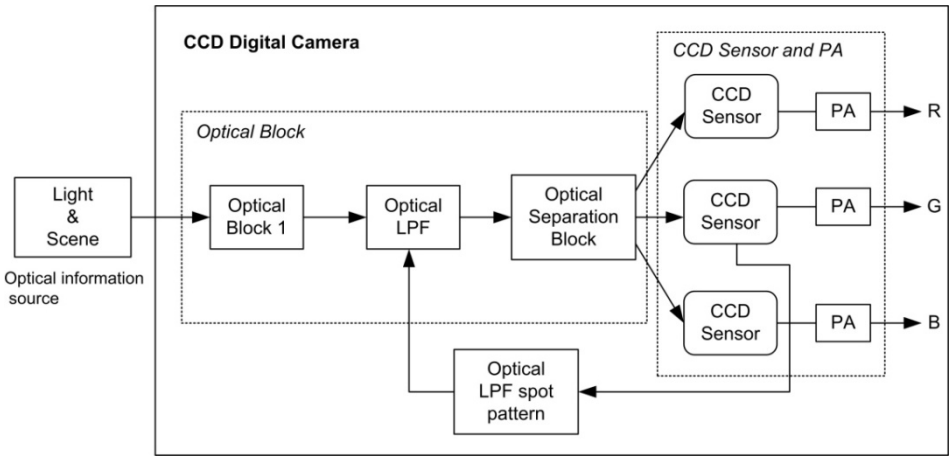

Figure 28: Simplified Functional Model of a CCD Digital Camera

The optical block 1 consists of zoom lens, colour filter, optical lowpass filter (LPF), and a splitting block (e.g., prism). Pre-amplifiers (PA) are mounted at the end, for manipulating the output signal. The CCD sensor pixel outputs are characterized by a *pixel function model*, which describes the pixel dimensions, integration time, pixel sensitivity (or responsivity), and the spectral distribution of light on the pixel surface. Krasnjuk et al. states that a *camera functional model* can be used to determine camera characteristics [Krasnjuk et al., 1994]. Based on this model, internal analysis of image sensors processes can be possible.

Data output:

Images and video data can be captured in different formats or as raw digital data output with different bit sizes, e.g. 8-bit, 12-bit, and 16-bit, etc. Compact digital cameras produce only jpeg images. Video data can be streamed at different frame rates. Existing standard compression techniques can be applied on the data output.

Interfaces:

Different software APIs exist that implement proprietary or standard specifications. Example of the commonly used standard specifications are the IEEE 1394 (Firewire or i.Link) interface-based digital camera specification (also called IIDC or DCAM specification) [http://www.1394ta.org], and the Camera Link Interface standard for digital cameras and frame grabbers specification [http://www.machinevisiononline.org]. Examples of common open source software implementing these specifications are the Camwire IEEE 1394 based camera C API [http://kauri.auck.irl.cri.nz], and the digital camera library (libdc) [http://sourceforge.net/projects/iidcapi].

For physical linking, common interfaces like USB, Serial, Firewire, Ethernet, etc. are used.

5.2.3. Distance Measurement Sensors

5.2.3.1. Electronic Distance Meters (EDMs)

They are usually classified according to type of carrier wave. Electro-optical instruments use light or infra red (IR) waves, and microwave instruments are based on radio waves.

EDMs can also be further categorized based on the distance ranges. Short-to-medium range EDMs are

those that use carrier waves based on Gallium Aluminium Arsenide (GaAIAs) diodes, and long range use HeNe laser ('red' gas laser), etc. To illustrate an example of an EDM model, a simplified diagram of an electro-optical distance meter using analogue phase measurement is given in Figure 29 [Rüeger, 1996].

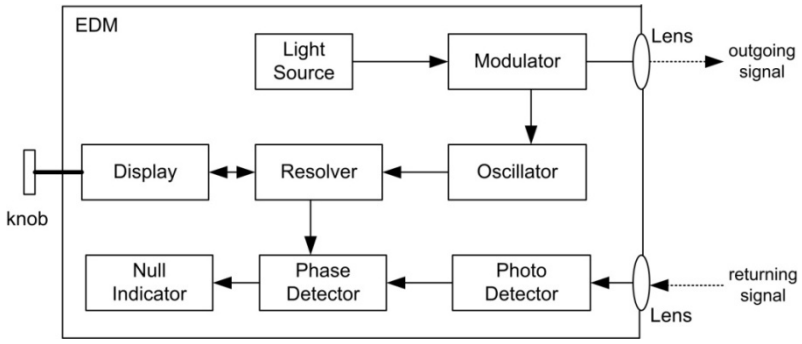

Figure 29: Simplified Electro-optical EDM using Analogue Phase Measurement

Table 7 explains the EDM components.

Component	Description
Modulator	Varies the amplitude of the carrier wave by the intensity modulation. The oscillator determines the modulation frequency.
Photo Detector	Transforms the light beam's intensity variations into current.
Resolver	Shifts the phase of the reference signal
Phase Detector	Provides phase comparison between the returning signal and the reference signal. The result of detection is shown on the null indicator.

Table 7: Electro-optical EDM Components

Data output:

Distances (e.g. horizontal or slope distances).

Interfaces:

Different manufacturers provide different software interfaces.

5.2.3.2. Time Domain Reflectometry (TDR) Instrument

TDR can be used to locate and monitor slope failures. It was originally intended for locating breaks and faults in communication and power lines.

TDR records the impedance change (or disturbance) in a coaxial cable which causes some pulse energy to be reflected back to the TDR instrument. It is similar to the radar measurement principle. The distance to the point of impedance can be calculated based on the propagation velocity of the signal and the time of travel. Figure 30 shows the components of a TDR.

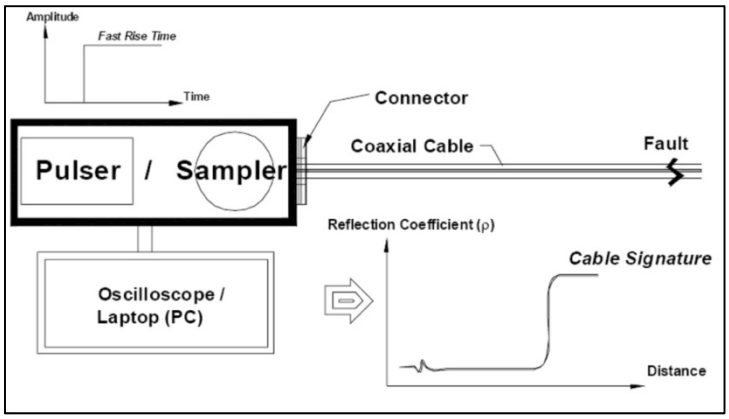

Figure 30: Time Domain Reflectometry Cable Tester [Kane et al., 1996]

On the instrument display (oscilloscope), the reflected signal is shown in waveform. A 'spike' in the cable signature (wave peak) indicates the magnitude and the rate of displacement of slope movement.

Data output:

Voltage output is converted into displacement data.

Interfaces:

Kane et al. experiences show that specialized software is required to process the raw data, and that in order to write code for the data loggers connected to TDRs, manufacturer specific software is necessary [Kane and Beck, 1999].

5.2.4. Angular and Tilt Measurement Sensors

5.2.4.1. Tiltmeters

Tiltmeters are commonly used for geotechnical measurements, such as determining the direction of slope movement and mechanism of movement (e.g. slumping, slope creep, settlement, etc.). They do produce continuous measurements of angular movement (angular position and rotational movement).

Figure 31 shows an example of a tiltmeter which consists of excitation electrodes, pick-up electrode, and a gas bubble.

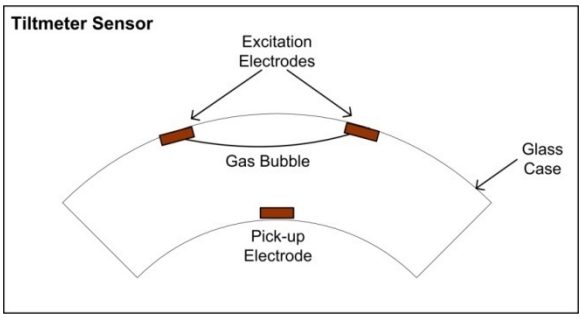

Figure 31: Tiltmeter Sensor Diagram

The movement of the excitation electrodes, due to tilt, causes the AC resistance between the pick-up electrode and each excitation electrode. This change in resistance is converted to angular movement.

Data output:

Raw output: When the tiltmeter is directly connected to an output device (e.g., data indicator, logger, or recorder) and the signal is not conditioned (e.g. no amplification).

Voltage output: The signal is conditioned, and the sensor produces a nominally 1 Volt (dc) full scale voltage output, which is proportional to the tilt.

Interfaces:

Usually data loggers or recorders provide interfacing between tiltmeters and other devices like PCs. Physical communication can be via serial interface, USB interface, wireless, Ethernet, telephone/cell modem, etc. Different manufacturers provide specific API for writing code.

5.3. Integration Methods and Technologies

5.3.1. Transformation on Connectors

Bliudze et al., defines *connectors* as a basic concept for modelling coordination (interaction) between components [Bliudze et al., 2007]. Connectors can also be seen as mediators. Current research efforts are focusing on formalising connectors (i.e. common specifications and architectures, algebraic formalisation …); and on semi-automatic and fully automatic generation of connectors in order to resolve protocol and data type mismatches. With respect to architectural connections, common methods are procedure calls, pipes, event broadcasting, and use of shared variables (e.g., data).

Figure 32 illustrates an example of a connector and its specification.

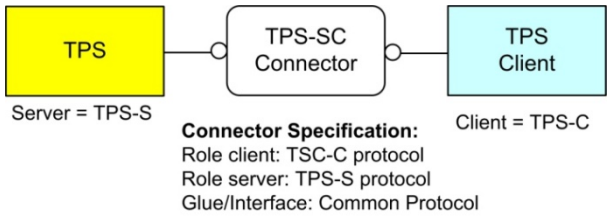

Figure 32: TPS Connector and its specification

The specification in the example defines a TPS connector type, which is basically a set of roles and a gluing interface. If, for example, an existing TPS client (e.g. plug-in or standalone application) tries to communicate with a new TPS whose communication protocols (e.g., physical connection, commands, data etc.) do not agree, the connector has to perform transformation on one or both components (e.g., the TPS client or TPS server and, if possible, automatically updates the respective configuration files or protocol registers).

Transformations on connectors can be seen as a pattern or template class for adapting component interaction mechanism (i.e. pattern identification and modification). The transformations may include changes in the implemented technology, data structures, protocols and representations. Following are examples of a data transformation:

- *Data compression*: Encoding and decoding data at both ends of communication. Both ends should be able to understand the compression method used.

- *Swapping byte order*: Changing the order of bytes depending on the execution platform or expectations of the data receiving component. For example, Endianness (little-endian or big-endian).

- Etc.

As a summary, modifying existing or developing new connectors can be complex, costly and requires guru level of expertise [Spitznagel et al., 2001].

5.3.2. Wrapping on Connectors

At the moment, the only solution to resolve the discussed proprietary problems is through implementing *software wrappers* [Rowe, 2008; Kandawasvika and Reinhardt, 2005b; Smuda, 2005; Feldman, 2003;].

In computer programming, software wrappers can generally be defined as *adapters* that allow classes or components that have incompatible interfaces to work together. Currently, the concept of *zero configuration* [IETF, 2001] and *zero programming* is still a vision with respect to deployment of sensor systems, but can be a reality if interoperability standards are strictly followed. As shown in Figure 33, sensors of the same type e.g. total stations (TPSs) from manufacturer X and Y have to be integrated separately and the same applies to other sensor types.

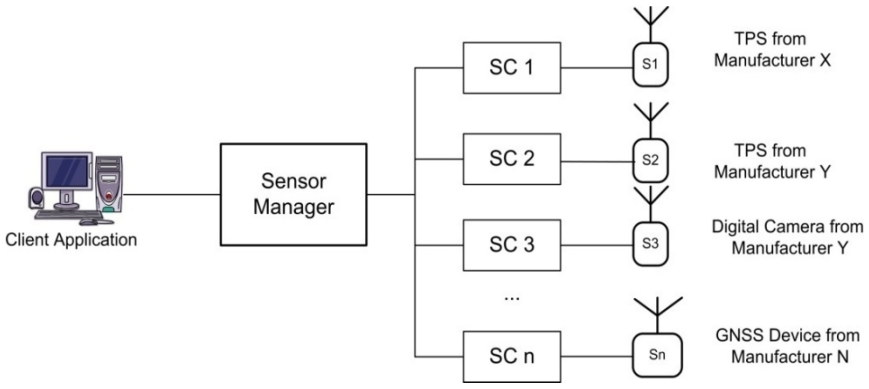

Figure 33: Specific Sensor Component for each Sensor.

The sensor manager component has to scale up to different complex levels as the number of sensors increases. If the wrapping concept is not coupled with open, standards-based (i.e. vendor-neutral) solutions, difficulties in sensor management arise.

Software wrappers allow interaction between disparate components by encapsulating or suppressing the differences (e.g. communication protocols, data types etc.) and exposing only common interfaces and data. Also transformations can be performed on connectors but more advanced. The transformations involve, for example, mapping of data formats (e.g. from a native format to a common format, see Figure 8), and mapping of interface signatures, specifically parameters and return types.

Ontology (semantics) mapping can also be done, for example, using common reference ontologies. Simplified ontologies can be recorded in form of translation tables or XML-based dictionaries. For syntax transformation, the wrappers can include model-specific reasoning, auxiliary databases or mathematical

functions so as to produce correct translations of references. For example, data type mapping between different implementation languages, like Java and C/C++ components, can be realized using equivalence or standard translation tables.

Table 8 shows an example of an equivalence table for mapping Java primitive types to their platform-dependent native types.

Java Type	Native Type	Type Signature	Description
boolean	jboolean	Z	unsigned 8 bits
byte	jbyte	B	signed 8 bits
char	jchar	C	unsigned 16 bits
short	jshort	S	signed 16 bits
int	jint	I	signed 32 bits
long	jlong	J	signed 64 bits
float	jfloat	F	32 bits
double	jdouble	D	64 bits
fully-qualified-class		L fully-qualified-class	
type[]		[type	
method type		(arg-types) ret-type	

Table 8: Java Primitive Types and Native Equivalents [JNI-Specification]

Following are some of the commonly used methods, and technologies tools for realizing the wrapping concept.

Java Native Interface (JNI): This is a very low-level interface which allows Java code that runs inside a Java Virtual Machine (JVM) to interoperate with applications and libraries written in other programming languages (e.g. C, C++, Assembly …). JNI is complex and is not easy to implement. There are, however, a number of open source libraries and tools that use JNI to provide semi-automatic or automatic generation of wrappers. Examples of open source tools are Java COM Bridge (JACOB), COM4J, jSegue, Simplified Wrapper and Interface Generator (SWIG), etc. These tools are at various stages of development and the interfaces (wrappers) generated by the tools may require moderate to heavy manual editing.

Component-based Interfaces: Microsoft's Component Object Model (COM), and .NET Assemblies; CORBA, Java Beans, etc.

Web Services: Unlike time-consuming and difficult component-based technologies mentioned above, Web services provide interoperable means for integrating different components (e.g. in form of middleware). Examples are .NET based Web services, Java-based Web services, etc. For example, a sensor library or connector can be wrapped into a Web service so as to provide its measurements in standard-based formats like SWE Common, O&M etc. Besides, the Web service wrapper can implement standard communication interfaces which can allow applications or web service clients to interact with the targeted sensors in a common way.

5.3.3. Specification of Middleware

Software middleware (or bridges) can be used to bridge the differences between different components by acting as links, adaptors, brokers, or intermediaries. Middleware can be more sophisticated than connectors. It can be a composition of transformed connectors or wrappers and can also perform advanced interface transformations based on particular conversion rules. Different technologies can be used to realise middleware for interoperable integration of disparate components, e.g., technologies like SOA, Web services, component-based architectures (COM, CORBA ...), etc. Note that current Web services and SOA applications can achieve interoperability by being *message-oriented* and/or *event-based*.

5.4. Summary

The sensor component and functional models described in this chapter are general enough to provide the basic understanding of the respective sensor systems, but they can be modified (extended or further simplified) depending on the application domain needs, and the level of understanding (abstraction) required about the sensor models and their respective measurements data.

Middleware and wrappers can be seen as intermediaries that convert different technologies, protocols, data structures or representations, in order to enable communication between non-interoperable resources. The following chapter will cover the conceptual framework of the dissertation, based upon the information in this chapter and the preceding ones.

6. Conceptual Framework for Interoperable Management of Multi-Sensors

This chapter consists of the conceptual framework of the dissertation. Sensor models form the core part of the proposed concept. The models are divided into two types: the generic logical sensor models (GLSMs) and the standards-based sensor models (SBSMs). GLSMs should be developed by specific communities (e.g. environmental and infrastructure monitoring community comprising geodetic/surveying/remote sensing systems manufacturers, geotechnical sensors manufacturers, application developers, users, etc.), and the developed models should enable interoperable management and exchange of sensors within that community domain. In addition, for intercommunity or cross-domain applications, the GLSMs can be transformed to standards-based sensor models (SBSMs) using standard languages like OGC SensorML, etc.

The first sections of this chapter discuss the proposed model-driven approach (MDA) for integrating multi-sensors into an application, followed by the workflow for development of generic logical sensor models. The last sections of this chapter cover the proposed generic sensor-based access and control service (SACS) that loosely integrates the developed sensor models via plug and play modules. It also discusses SACS components and how SACS fits into or extends the current OGC SWE framework.

6.1. Integration of Sensor Models in an Application

In order to achieve interoperable management of multi-sensors as well as minimum integration efforts and time in deploying those sensors into hosting applications, we recommend the use of sensor plug-ins that expose generic or standard-based interfaces for connection, command and control. All the information and knowledge about the sensors being handled by the sensor manager component should be obtained from the logical sensor models. The integration conflicts due to differences in vendor-specific protocols, interfaces and data formats can be wrapped by the sensor manager component (also called *sensor wrapper* in this dissertation). Following is a simplified architecture for sensor model integration.

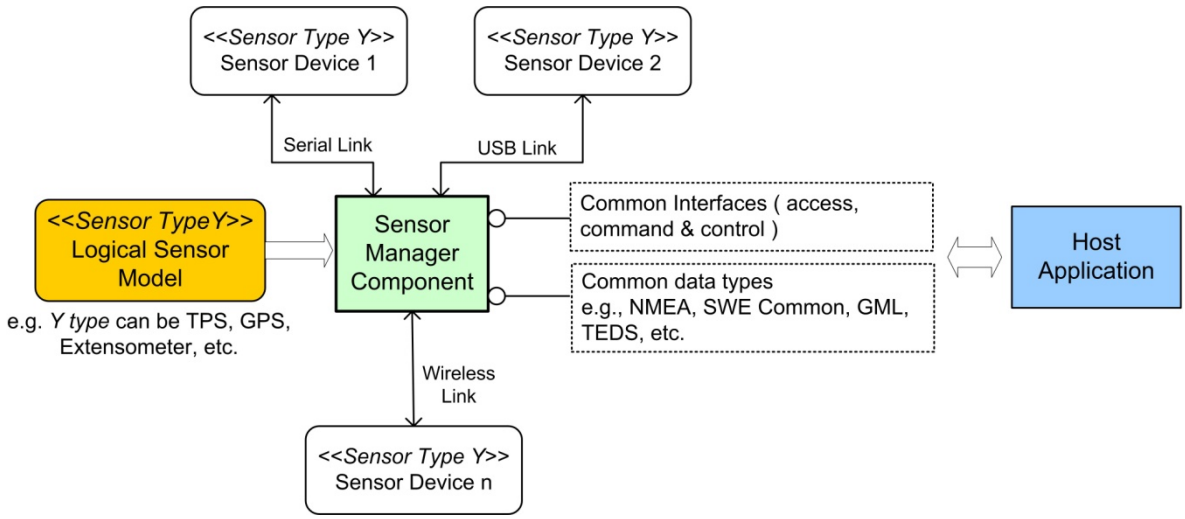

Figure 34: Sensor Model Integration Architecture.

As illustrated in Figure 34, a sensor manager component is responsible for handling *sensor types* (e.g., type Y can be a GNSS receiver, TPS, extensometer, etc.), thereby avoiding the need to have a specific sensor component for each and every single sensor. The sensor manager component is model-driven. It gets all the knowledge about the physical sensing device from an instance document of a logical sensor model of that particular sensor type. In the figure, sensor devices employ different means of communication mechanism (e.g. serial, USB, wireless, etc.). The sensor manager component makes use of the available low-level communication wrappers (e.g. USB, Serial, Wireless wrappers, etc.) to physically link with the respective sensors. The required linking information can also be accessed via the logical sensor model.

Different data types and structures can be wrapped by the sensor plug-in, which should only output results that are exactly defined using the common data types (e.g., OGC SWE Common, NMEA, TEDS, TML-based formats, GML, etc.) to the host application (e.g. a landslide monitoring application). Vendor-specific communication interfaces can be wrapped into common interfaces (e.g., community-based generic interfaces, and standard-based interfaces like IEEE P1451 Smart Transducer Interface Standard).

In order to bind with an existing physical sensing device, a sensor manager component can make use of any of the two paths, see Figure 35. The first path relies on generic logical sensor model (GLSM) (like the ones we have developed, see Section 6.2.3) and the second path uses standards-based logical model (SBSM). The SBSM can be derived from mapping the GLSM using standard modelling languages like OGC SensorML, etc.

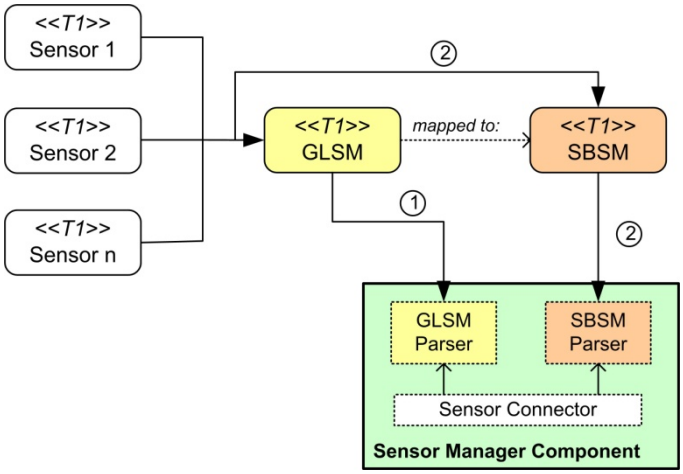

Figure 35: Sensor Manager Component Parsers for GLSM and SBSM.

A sensor connector (or *sensor component wrapper (SCW)*) uses the respective XML parsers for accessing the sensor model information. Specific communities can employ their own developed parsers to exchange sensors, but in order to achieve higher degree of interoperability (i.e., easy exchange of sensor models across multiple application domains), it is better to use standards-based parsers. However, in cases where the open, international standards are still immature and the respective parsers not available or early in development, we recommend the first path.

Note also that standard modelling languages can be very dynamic (i.e. change rapidly) in their development and those that are mature like OGC SensorML can be complemented or even replaced by

new ones. Therefore, the sensor manager component should be able to use community-level generic models (e.g., GLSMs), which can be stable for a long time and are not influenced by rapid changes in standard modelling language specifications. The developed generic logical models can then be mapped to stable versions of the standard specifications at any time.

If the available sensor software libraries use different programming languages (e.g. C/C++) from the language of the target/host application (e.g. Java), then sensor component wrappers can also resolve these differences (e.g. syntax conflicts) using, for example, translation tables and/or existing software wrapping tools (see previous chapter). Figure 36 shows an example of a sensor wrapping concept.

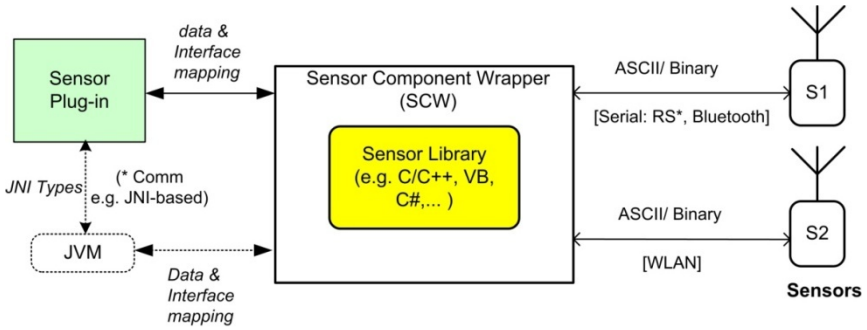

Figure 36: Sensor Wrapping Concept

In this case, the sensor plug-in communicates with a sensor API/library via a Java Virtual Machine, which facilitates the conversion between Java and non-Java (native) interfaces. For other languages, there may not be any need for an intermediate virtual machine. For the dissertation, the term *sensor plug-in* is loosely defined as any software add-on or component needed for accessing and controlling a sensor. It can be implemented in any language.

In summary, in order to support a very flexible and interoperable framework for multi-sensors management, we propose that the various sensor manager components (plug-ins) be hosted in a generic sensor access and control service. This sensor service can be realised based on service-oriented architecture (SOA) and/or Web service concept (see Section 6.3).

6.2. Proposed Workflow for Sensor Model Development and Usage

Figure 37 shows a general overview of a sensor development cycle. Similar to any IT system development, the first phase involves analysing the requirements of the monitoring applications and then analysing the selected sensor system in order to understand and model its components, functionality, applicability, limitations, and behaviour, to mention a few.

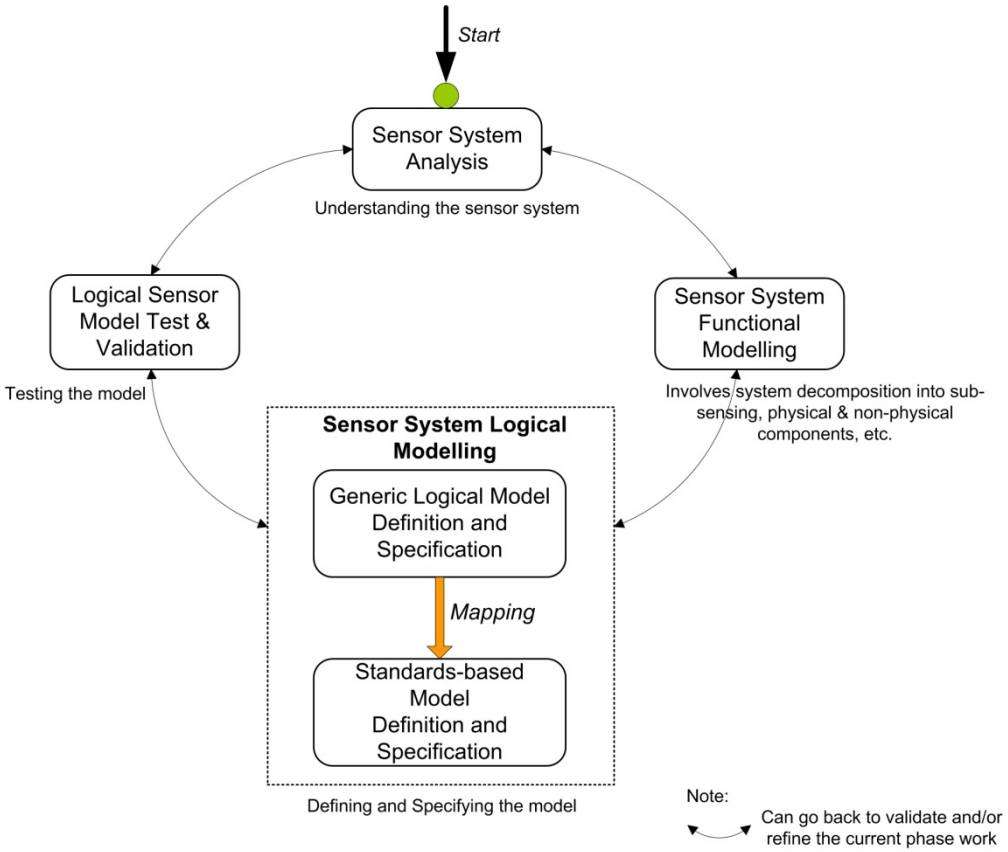

Figure 37: Proposed Sensor Model Development Cycle

The second phase involves sensor system functional modeling, which is then followed by logical modeling. The last phase addresses testing and validating the developed logical model in an application.

6.2.1. General Analysis of Sensor Systems

Understanding of the sensor systems (physical sensing devices or instruments) is the first step in conceptual modelling. Figure 38 lists some of the important items that have to be analysed.

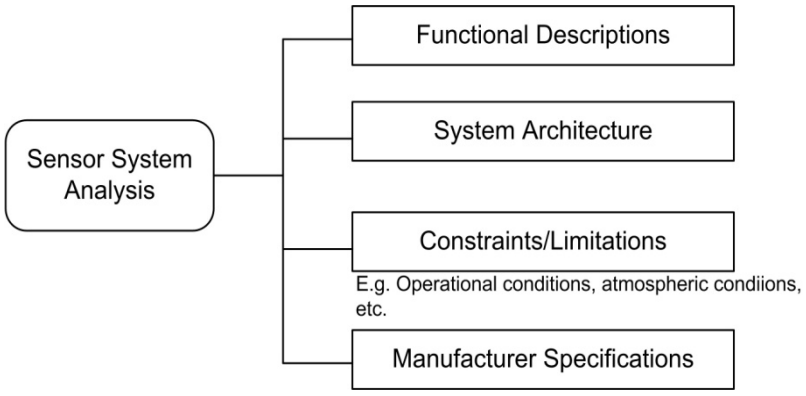

Figure 38: Phase I: Sensor System Analysis

Figure 39 shows the level of abstraction that is derived from the analysis and used for the sensor conceptual models.

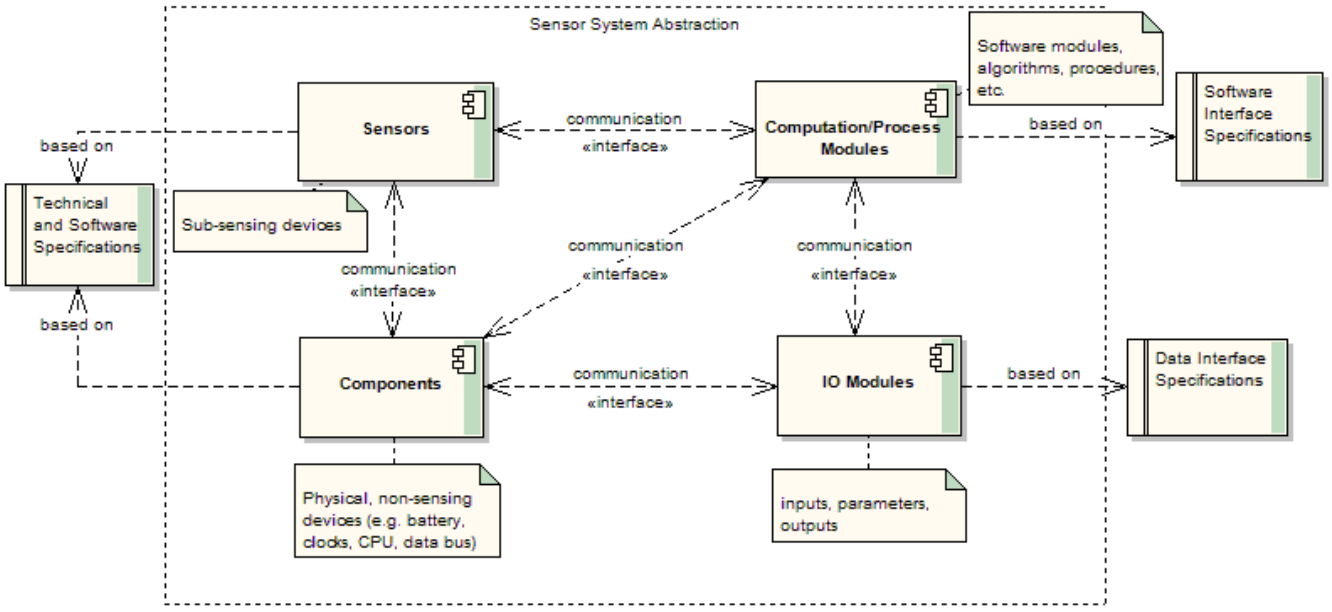

Figure 39: High-level Abstraction of Sensor

The input resources (e.g., specifications) required for the modelling are illustrated in the diagram. Following are the items to consider during the sensor system analysis.

6.2.1.1. Functional Descriptions

The functional descriptions of a sensor system may include descriptions of algorithms, process methods or mathematical models, as well as information about intended uses or applications of the system.

6.2.1.2. System Architecture

The system architecture should enable a good overview of the system's component model, interaction mechanisms between the sub-components and their dependencies, data flows (I/O), etc.

6.2.1.3. Constraints and Limitations

It is also important to comprehend the system constraints and limitations, such as operational conditions (e.g. storage and operating temperature ranges), atmospheric conditions (e.g. weather effects), etc.

6.2.1.4. Manufacturer Specifications

Sensor technical specifications from different manufacturers for each sensor type/class of interest should be analysed in detail. The main objective is to identify common patterns and derive common denominator specifications that match the application domain requirements. Each derived specification can be used as an abstract base for a generic, common specification for each sensor type.

6.2.1.5. Available Software APIs and Libraries

From the software API specifications of different manufacturers, it is necessary to identify the syntactical information, e.g., language of implementation (e.g. C, C++, Java, VB, etc.). The language of implementation usually determines the data types and structures used, as well as the interface and method definitions. This information also plays a role in developing common sensor specifications.

If any information about implementation algorithms is available, this should also be captured for later use in the logical modelling.

6.2.2. Sensor System Functional Modelling

Functional and component-based modelling of a sensor system is based on the information and knowledge acquired from the detailed needs analysis. Figure 40 lists the activities or aspects that fall under functional modelling.

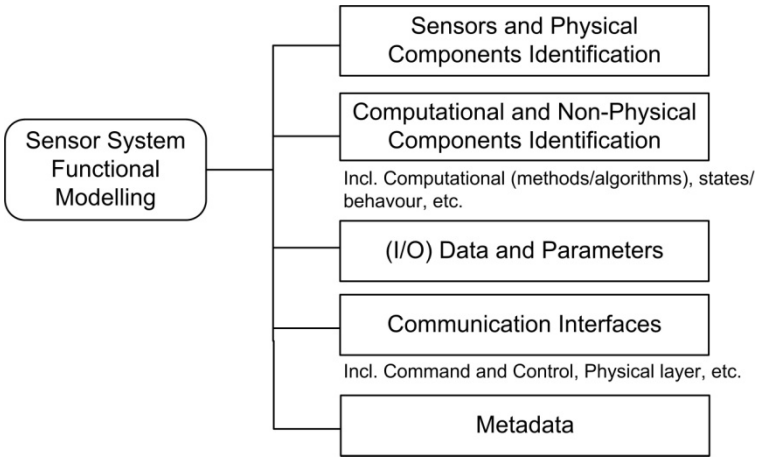

Figure 40: Phase II: Sensor System Functional Modelling

Formally, the function models can be expressed using modelling languages like UML. Figure 41 gives a simplified example of a TPS sensor conceptual model.

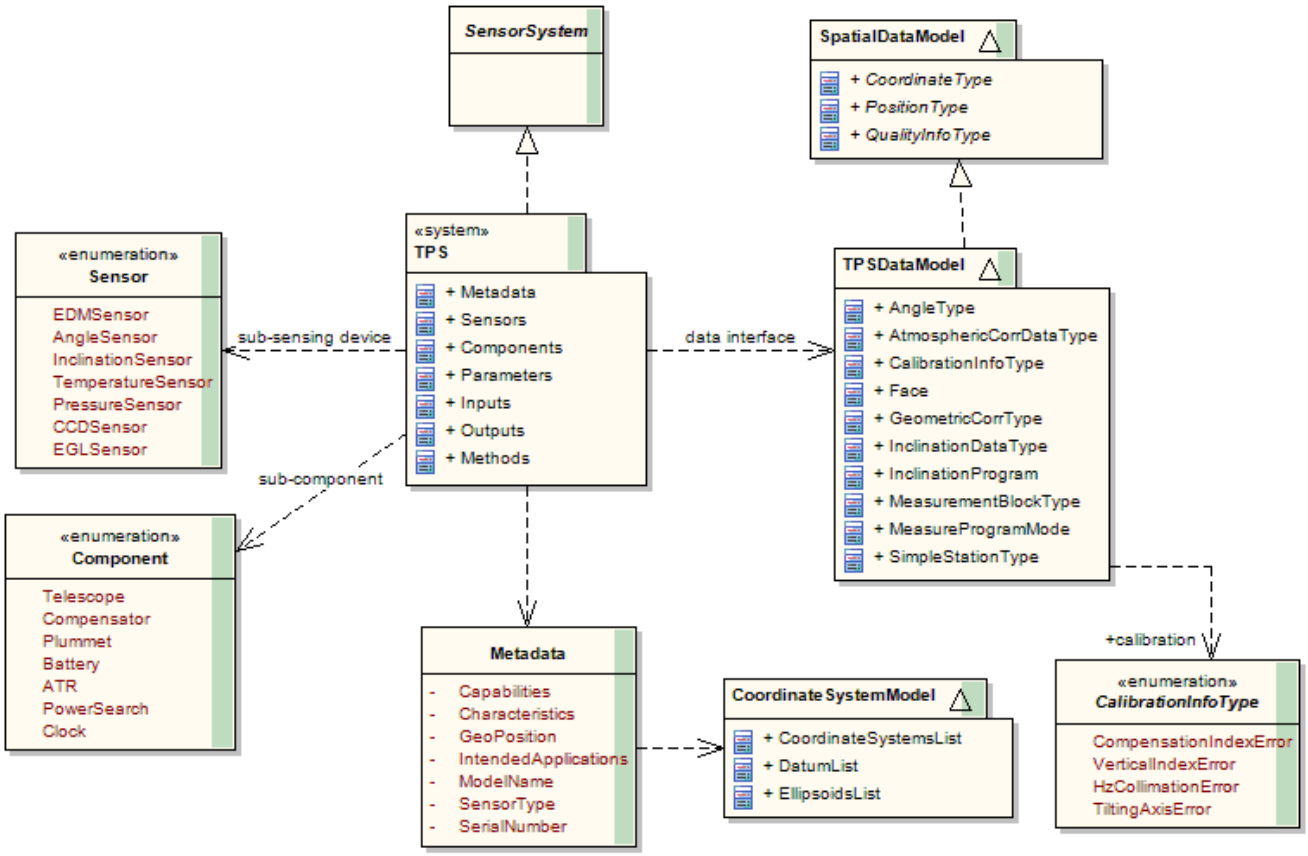

Figure 41: TPS Sensor Conceptual Model

The data model (e.g., TPSDataModel) should provide a generic data interface that can be used for the sensor system input/output (I/O) definition. This model also serves as the base for defining common command and control software interfaces. The sensor metadata model also uses the data model definition to describe its offerings (i.e., measurements data and functions). Figure 42 shows a conceptual model for a GNSS sensor (i.e. GPS receiver).

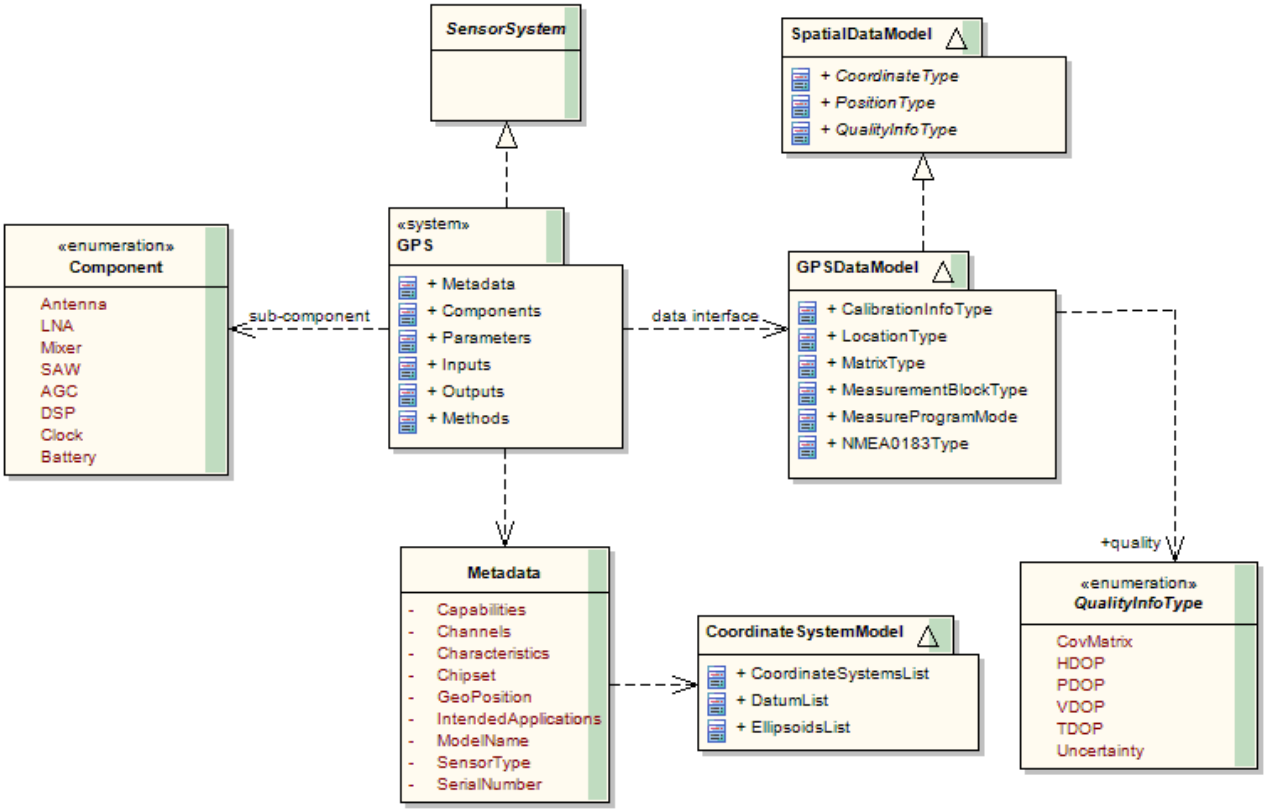

Figure 42:GPS Sensor Conceptual Model

The above conceptual models form the basis for defining logical sensor models. For more details about the model elements, see Section 6.2.3. Following is an explanation of the steps taken.

6.2.2.1. Identification of Sensors and other Physical Components

The first step should clearly identify the sub-sensing devices and other physical sub-components of the system. These entities should be abstracted in a way that reflects the functional model of the system at a higher level. By higher level, we mean that the components should not map one-to-one system components as at the hardware/physical design level. Refer to Chapter 5.2 for details about the basic sensor functional and component models.

6.2.2.2. Identification of Computational and other Non- Physical Sensing Components

From detailed analysis of the needs together with other sensor information from the manufacturers and users, non-physical components like computational algorithms and methods, calibration procedures, etc., can be identified. Connections between the individual components can be described using connectors or process chains.

6.2.2.3. Input and Output Data (Messages) and Parameters

The next step is to derive a *common set* of data definitions from the derived specifications for each sensor type. Bear in mind that for the system internal communication (i.e. for access, command, and control) of the sensors, the derived common data types/definitions could be mapped back to their respective native data types/definitions. Therefore, any available complex data structures should clearly expose their nested primitive data types. It is only through primitive data types that *syntax transformation* can be effectively and easily performed. For example, the Java Virtual Machine (JVM) uses a standard translation table (see Table 8) to map data types or structures from other languages into java native types using mostly primitive data types.

6.2.2.4. Communication Interfaces

The same approach for abstracting data types and structures can be applied to command and control interfaces (also referred to, in this dissertation, as higher-level communication interfaces).

6.2.2.5. Metadata

Metadata information is very important for sensor mining and discovery. The metadata information includes sensor system applications or usage, sensor system offerings (e.g. measurements and operations), physical characteristics; associations or connections between system internal entities (e.g., sub-sensors, sub-components, etc.), and the respective outputs of these individual sub-entities, etc. Information about the manufacturer and sensor model author is also important.

6.2.3. Definition and Specification of Generic Logical Sensor Models

Phase III formalizes the functional modelling step by defining and specifying generic logical models and, at the same time, providing more details about the sensor system. In this subsection, the TPS and partly the GPS sensor systems are used as examples in order to clarify the sensor logical modelling concepts. The TPS is a complex sensor system that comprises a number of sub-sensors and sub-components, hence the reason to use it as an example. The GPS represents an example of data streaming sensors. The diagram in Figure 43 shows logical sensor modelling phase elements.

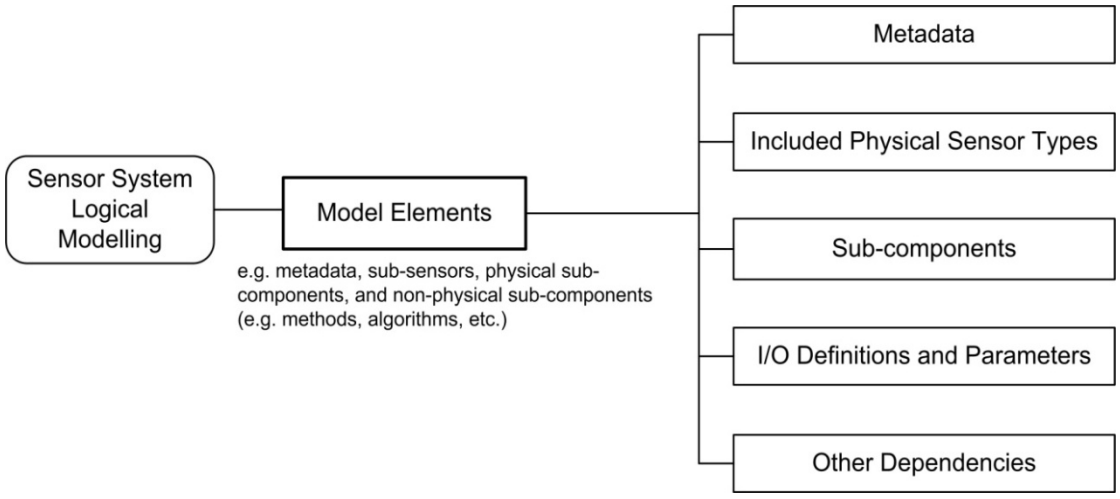

Figure 43: Sensor System Logical Modelling

6.2.3.1. Model Elements

The sensor system sub-sensing devices, sub-components, non-physical processes like mathematical algorithms, operation methods, etc., are modelled as schema model elements. These model elements, together with sensor metadata, I/O definitions and parameters, form a composite generic logical model. Examples of TPS and GPS logical schema models are shown in Figure 44 and Figure 46.

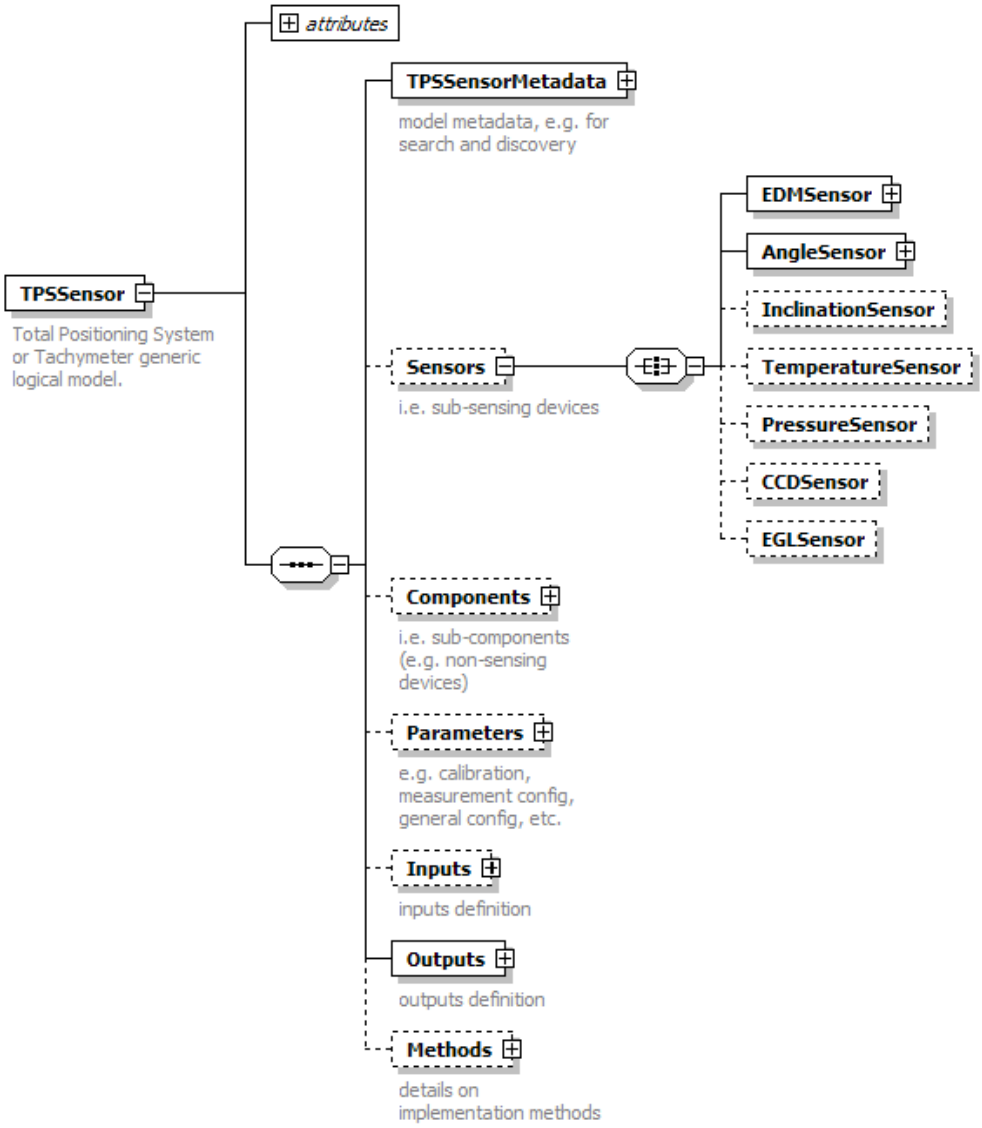

Figure 44: TPS Generic Logical Model (Schema Snapshot)

In Figure 44, the TPS generic logical model consists of sub-sensing devices like the Electronic Distance Measuring unit, angle sensors, inclination or tilt sensor, meteorological sensors (e.g. temperature and pressure sensors), a CCD sensor, and electronic guiding light. These sub-sensors can also further define and specify their own metadata, I/O definitions, parameters and methods. The level of granularity depends on the amount of detail required by the sensor application. For the generic models of the dissertation, the first level of the hierarchy is considered sufficient for most of those applications that only want to use the sensors, but not analyzing the internal processes of the respective sensors.

The first element of the logical model indicates the type of the sensor system (i.e. TPSSensor). The ‘metadata’ element has a capabilities section which contains sub-elements that define the interfaces for accessing and controlling the sensors as well as the types of measurements data returned by the sensors. The ‘parameters’ element shows all the variables that can be used to configure and control the sensors. The ‘inputs’ and ‘outputs’ define the I/O data part, and the ‘methods’ element describes the sensor model dependencies like sensor algorithms, procedures, or references to other implementation details. Refer to **Appendices 12.2.1** and **12.2.2** for details on the respective TPS and GPS schema instance files.

Figure 45 shows a sample of the TPS schema definition. The schema includes other schemas, like for generic data types (*tpsDataTypes.xsd*), interface (*tpsInterfaceDefinition.xsd*), and conditions definitions (*tpsConditionsDefinition.xsd*). These definitions are important inputs for accessing and controlling the sensors.

```
<xs:import namespace="http://www.w3.org/1999/xlink"
           schemaLocation="C:\kaad\ogc_schemas\xlink\1.0.0\xlinks.xsd"/>
<xs:include schemaLocation="tpsDataTypes.xsd"/>
<xs:include schemaLocation="tpsInterfacesDefinition2.xsd"/>
<xs:include schemaLocation="tpsConditionsDefinition.xsd"/>
<xs:element name="TPSSensor">
  <xs:annotation>
    <xs:documentation>Total Positioning System or Tachymeter generic logical model.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="TPSSensorMetadata" type="MetadataType">
        <xs:annotation>
          <xs:documentation>model metadata, e.g. for search and discovery</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Sensors" minOccurs="0">
        <xs:annotation>
          <xs:documentation>i.e. sub-sensing devices</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:all>
            <xs:element name="EDMSensor" type="EDMSensorType"/>
            <xs:element name="AngleSensor" type="AngleSensorType"/>
            <xs:element name="InclinationSensor" type="InclinationSensorType" minOccurs="0"/>
            <xs:element name="TemperatureSensor" type="TemperatureSensorType" minOccurs="0"/>
            <xs:element name="PressureSensor" type="PressureSensorType" minOccurs="0"/>
```

Figure 45: Schema Part of TPS Generic Logical Model

The contents of the above schema is also shown in a tabular form, see Table 9.

Sensor System Type: TPSSensor		
Model Component Category	***Name***	***<Abstract Type> Type***
Metadata	TPSSensorMetadata	MetadataType
Included Sensor Types	EDMSensor	EDMSensorType
	AngleSensor	AngleSensorType
	InclinationSensor	InclinationSensorType
	TemperatureSensor	TemperatureSensorType
	PressureSensor	PressureSensorType
	CCDSensor	CCDSensorType
	…	…

Table 9: Schema Part of TPS Generic Logical Model

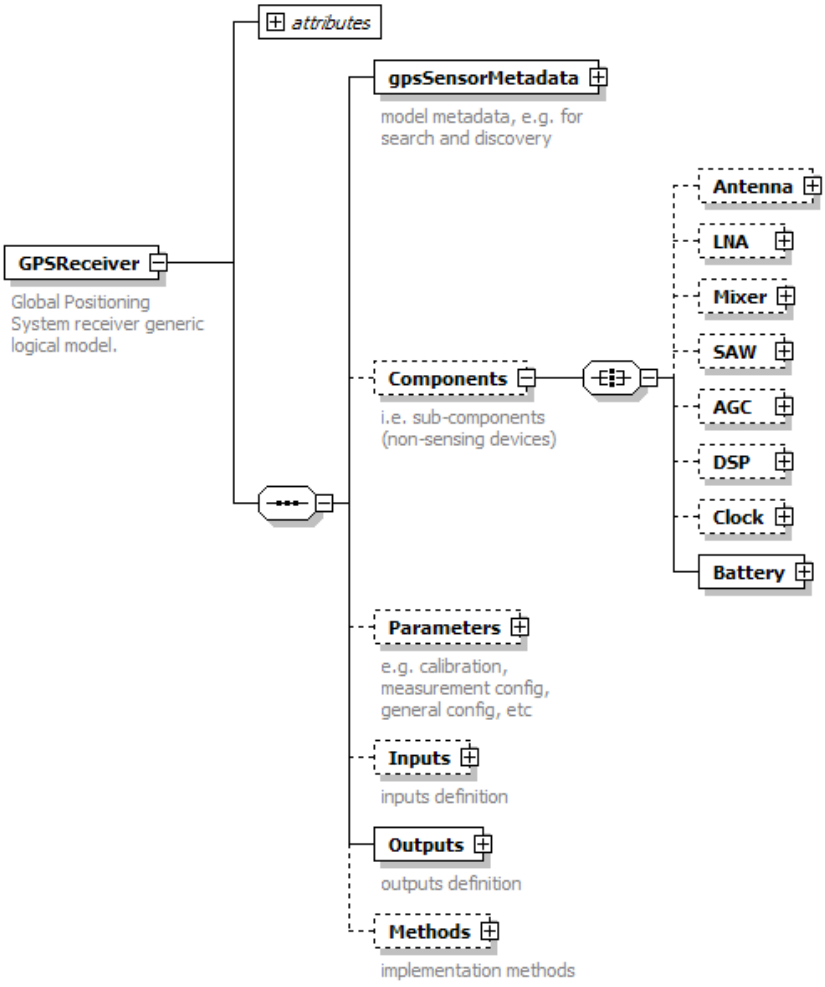

Figure 46: GPS Generic Logical Model (Schema Snapshot)

Figure 46 illustrates the GPS model. In this case, it has no physical sub-sensing devices, but sub-components that are responsible for the internal processing of the GPS signals into position, velocity, and time data, etc. However, if a sensor system like digital camera, TPS, etc. have an integrated GNSS receiver then the GNSS becomes a sub-sensing device of that sensor system.

6.2.3.2. Data Types and Structures

The generic sensor logical model specifies a common set of data types and structures (e.g. in a separate data definition XSD schema file). This common set forms the *data contract* part of the sensor type interface definition. The data contract can later be mapped into standards-based encodings like SWE Common Data Types, O&M, GML, etc. in order to facilitate data interoperability across communities. Table 10 shows an example of the TPS generic data types and structures.

complexType	**InclinationDataType**	ann:inclination data
complexType	**AngleType**	ann:angle data
simpleType	**BYTE**	ann:single byte range 0-255
simpleType	**RC_TYPE**	ann:return code type
simpleType	**SYSTIME**	ann:time since the system is powered on [ms]
simpleType	**EDM_MODE**	ann:Possible measuring modes of EDM
simpleType	**InclinationProgram**	ann:Inclination measurements
simpleType	**MeasureProgAction**	ann:Measuring Program Action
simpleType	**MeasureProgMode**	ann:Basic Measuring Program Mode
simpleType	**DateTime**	ann: Date and Time
complexType	**Coordinate1Type**	ann:coordinates data
complexType	**PositionType**	ann: Position type
complexType	**tpsQualityInfoType**	ann: Quality information type
complexType	**AtomspericCorrDataType**	ann:atmospheric correction data e.g. used for final distance computations
complexType	**SimpleStationType**	ann:basic station definition
complexType	**CoordinateType**	ann:coordinates data
simpleType	**Face**	ann:telescope face
complexType	**MatrixType**	ann: Matrix Data Type
complexType	**GeometricCorrType**	ann:geometric corrections e.g. used for final distance corrections.
complexType	**EDMSignal**	ann:EDM Signal Data Type
complexType	**MeasurementASCIIBlockType**	ann:For encoding measurements data
element	**MeasurementBlock**	ann:e.g. can be used to implement Leica's GSI Dataformats
complexType	**CalibrationInfoType**	ann: Calibration Info Data Type
element	**CalibrationInfo**	ann: Calibration Element
simpleType	**DatumList**	ann:List of projections
simpleType	**CoordinateSystemList**	ann:List of Coordinate Systems
simpleType	**EllipsoidList**	ann:List of Ellipsoids

Table 10: TPS Generic Data Types Examples.

For any GPS receiver, the NMEA data format can be used, but it is also possible to convert it into other data standard formats like SWE Common Data Types. This can be recommended in cases where proprietary GPS sentences are also encoded in the NMEA data streams.

We also recommend that, if possible, the generic data type and structure definitions be strictly based on the standard XML data types in order to facilitate easy and interoperable mapping to other XML-based standards.

6.2.3.3. Interfaces

Interfaces form the *interaction or communication contract* between or among different systems or components. In order to facilitate interoperable communication at higher-level (e.g. access, command and control), generic or standard-based interface definitions should be used. An interface definition can be provided in-line (i.e., in the same sensor XSD schema definition) or off-line (in a separate interface definition file (IDC)). The interface definition discussed in this subsection can be used to extend and complete the rudimentary interface definition provided by the OGC SensorML standard.

Following is an example of a minimum interface definition that can be used to model a sensor systems' communication interfaces. As shown in Figure 47, the *InterfaceCollection* element is the root and we have grouped the specified interfaces (*groupNames*) into *connection*, *control*, *sensorInfo* (i.e. metadata), and *messaging* (e.g. notification, failure or exception) interfaces.

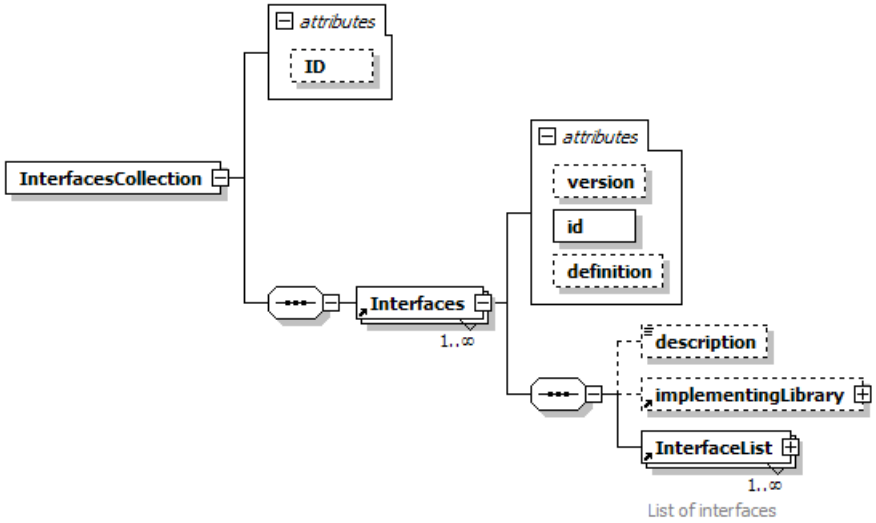

Figure 47: Interface Collection Definition

Following is an instance example using the above defined interface definition.

```
<InterfacesCollection xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="E:\phardmasy\dvpt\case_study_poc\sensors_msmts_modeling\SensorsGeneri
cModels\tps_24.05.08\tpsInterfacesDefinition2.xsd" ID="TPS_MinInterfaceSpecification">
  <Interfaces id="TPS_GeoCOM_MinSpec">
  <description>This document section contains a minimum list of interfaces provided by GeoCOM used
in integrating the TCA1800 total station (TPS) in geotech client. … In order to access the
interfaces documented, please use the following syntax: for example - tps_geocom.COM_Init(...)
  </description>
   <implementingLibrary name="tps_geocom.dll" path="C:/kaad/geotech/Geotech_2005-11-21_dvpt"/>
      <InterfaceList groupName="Connection">
       <description> The interfaces used for configuring and communicating with the TPS.
       </description>
        <interface name="Initialization">
         <return>getRC_OK</return>
       </interface>
       <interface name="OpenConnection">
         <Parameters>
          <parameter name="com_port" role="input" type="string" default="COM1"/>
          <parameter name="com_baud_rate" role="input" type="long" default="19200"/>
          <parameter name="num_retries" role="input" type="integer" default="1"/>
         </Parameters>
         <return>getRC_OK</return>
       </interface>
       ...
   <InterfaceList groupName="Control">
    <description>Interfaces for controlling the total station (TPS).</description>
      <interface name="MeasureDistanceAngle">
       <Parameters>
         <parameter name="progMode" role="input" type="MeasureProgMode"/>
         <parameter name="Hz" role="output" unit="rad" type="double"/>
         <parameter name="V" role="output" unit="rad" type="double"/>
         <parameter name="slope_Dist" role="output" unit="m" type="double"/>
       </Parameters>
       <return>getRC_OK</return>
      </interface>
      ...
```

Figure 48: TPS Communication Interfaces Example

In this example, the *implementingLibrary* element documents the implementation software library and its URI or file location. For example, the 'tps_geocom.dll' is a Java-based software wrapper we have created for the GeoCOM C/C++ API. Any Java-based TPS sensor plug-in can use the linked library to access the interfaces defined in the interface definition file.

A tabular presentation of the example contents is shown below.

Interfaces: TPS_GeoCOM_MinSpec						
Implementing Library	*Name*	*Path*				
	tps_geocom.dll	C:/kaad/geoetch/Geotech_2005-11-21_dvpt				
Interface Group: Connection						
Interface Name	*Description*	*Parameters*				
		Name	*Role*	*Type*	*Unit*	*Default*
Initialization	Initializes GeoCOM environment variables	-	-	-	-	-
OpenConnection	Opens communication channel between sensor system (TPS) and client application.	port baudrate	input input	string long	- -	Com1 19200
SetTimeOut	Sets time out for response.	time	input	long	sec	-
MeasureDistanceAngle	Causes the TPS to measure only distances and angles	progMode	input	Measure_ ProgMode	-	Track_ DistPlus_ Angles
...	…	…	…	…	…	…

Table 11: TPS Interfaces Table Example

The hardware or physical communication interfaces can also be defined in the specification, see example in the Figure 49.

```
<Interfaces id="Hardware">
  <description>Hardware link between the client (pc or laptop) and the server (TPS).</description>
  <InterfaceList groupName="SerialCommunicationLink">
    <interface name="RS_232">
      <Parameters>
        <parameter name="signalPaths" role="input" type="SignalPathsStructure">
          <description>takes a structure with paths e.g. RxD-receive, TxD-transmit, GND-ground.</description>
        </parameter>
        <parameter name="voltage" role="input" type="VoltageLevels">
          <description>Logical 0 (+3V to +25V), Logical 1 (-3V to -25V).</description>
        </parameter>
        <parameter name="pinType" role="input" type="string" default="DB9"/>
        <parameter name="baudRate" role="input" type="long"/>
        <parameter name="parity" role="input" type="string" default="None"/>
        <parameter name="dataBits" role="input" type="unsignedShort" default="8"/>
        <parameter name="stopBits" role="input" type="unsignedShort" default="1"/>
        <parameter name="terminator" role="input" type="any" default="CR/LF"/>
      </Parameters>
    </interface>
  </InterfaceList>
</Interfaces>
</InterfacesCollection>
```

Figure 49: Hardware Interface Example

The ‘Hardware’ interface has an interface group ‘SerialCommunicationLink’ which provides the necessary parameters (e.g., pin type, baudrates, databits …) for communicating with any devices capable of serial interfacing (e.g., using the RS-232 standard).

Table 12 shows the hardware interface definition in tabular form.

Interfaces: Hardware/Physical Communication					
Interface Group: SerialCommunicationLink (Connection)					
Interface Name	*Description*	*Parameters*			
		Name	*Role*	*Type*	*Default*
RS_232	Serial communication link	pinType	input	string	DB9
		baudRate	input	long	19200
		parity	input	string	None
		dataBits	input	ushort	8
		stopBits	input	ushort	1
		terminator	input	string	CR/LF

Table 12: Hardware/Physical Communication Interface Example

Note that all sensor systems that use serial communication can access the interface shown in the table. There are also other communication media groups (e.g. BluetoothSPPCommunication, WLAN, or OSI-based data link and physical layers, etc.) that can be used by sensor systems.

The interfaces discussed in this section should be wrapped into higher-level interfaces defined by the

generic sensor plug-in interfaces (see Section 6.3.2).

6.2.3.4. I/O Definitions and Parameters

These I/O data and parameter definitions should be based on the defined common data type schema. The parameters can include information on sensor calibration and other system-level configuration parameters (e.g. OS language, units of measurement, measurement mode configuration, etc.). For example, a TPS sensor system can be configured to generate point IDs automatically and the EDM to measure in reflectorless mode. Following is an example of TPS parameters definition.

```
<!--TPS Calibration Information -->
<CalibrationInfo>
  <Time>2000-12-03T11:00:00Z</Time>
  <CompensatorIndexError mode="l" unit="gon">
    <Value>0.0010</Value>
  </CompensatorIndexError>
  <CompensatorIndexError mode="t" unit="gon">
    <Value>0.0023</Value>
  </CompensatorIndexError>
  <VerticalIndexError unit="gon">
    <Value>0.0001</Value>
  </VerticalIndexError>
</CalibrationInfo>
</Parameters>
```

Figure 50: TPS Parameters Example of Calibration Information.

Table 13 illustrates the contents of the calibration in a tabular form.

TPS Calibration Information			
Time	*Calibration Parameter*	*Unit*	*Value*
2000-12-03T11:00:00Z	Compensator longitudinal (l) axis error	gon	0.0010
	Compensator transverse (t) axis error	gon	0.0023
	Vertical index (i) error	gon	0.0001
	…	…	…

Table 13: TPS Calibration Parameters Example

For sensor output and input definition, links to data storage, and dictionaries or taxonomies for *semantics* support can be provided. This can be seen as a first step in resolving semantic problems like misinterpretation of terms. Following is an example snapshot for the TPS I/O definition.

```
<!--TPS inputs definition-->
<Inputs>
  <Input name="TPS_Position" type="SimpleStationType" definition="#tpsDictionary.xml"/>
  <Input name="TPS_InitialOrientation" type="AngleType" definition="#tpsDictionary.xml"/>
  <Input name="TargetPointID" type="string" definition="#tpsDictionary.xml"/>
</Inputs>
<!--TPS outputs definition-->
<Outputs>
  <Measurements>
    <MeasurementBlock>
      <Fields>
        <Field name="PointNumber" type="word" index="1"/>
        <Field name="HzCircle" type="word" index="17"/>
        <Field name="VCircle" type="word" index="33"/>
        <Field name="SlopeDistance" type="word" index="49"/>
      </Fields>
      <DataValues dataFile="#measurementsOut1.xml"/>
    </MeasurementBlock>
  </Measurements>
</Outputs>
<!--TPS methods definition-->
<Methods xlink:href="#TPS_OperationMethods.xml"/>
</TPSSensor>
```

Figure 51: TPS I/O Definitions with Dictionary Links.

Following is a tabular presentation of the inputs and outputs definition (see Table 14).

TPS I/O Definition		
Inputs:		
Name	*Type*	*Definition (meaning)*
TPS_Position	SimpleStationType	e.g. in tpsDictionary.xml, TPS geolocation
TPS_InitialOrientation	AngleType	Initial observation direction
TargetPointID	String	ID of the observed point
Outputs:		
Name	*Type*	*Definition (meaning)*
MeasurementBlock	MeasurementASCII-BlockType	See 'Fields' collection definition under 'MeasurementBlock' element
measurementsOut.xml	XMLDocument	See 'MeasurementBlock' element for block structure definition

Table 14: TPS I/O Definitions Example

In the example, the 'tpsDictionary.xml' provides the meaning of the defined elements. This dictionary approach is also proposed in the OGC SensorML as an alternative to using complex ontologies. The "measurementsOut1.xml" contains the actual measured TPS data, whose structure is defined in the 'MeasurementBlock' element just before the 'DataValues' element. This information enables the application to know the arrangement of the data it will read from the data file or the 'DataValues'

element.

6.2.3.5. Constraints and Rules Descriptions

We propose that basic constraints and rules for the sensor model be described in conditions definition XML file. The basic constraints can be seen as the *limitations* or *conditions* that are imposed on or influence the sensor system; and these basic rules determine and guide the behaviour of that sensor system. Following is an example of TPS conditions descriptions.

```
<Conditions sensorID="TPS_Leica_TCA1800" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <ConditionList classification="Atmospheric">
    <Condition ID="1" name="severe">
      <state>Strong haze, strong sunlight, or severe heat shimmer</state>
      <visibility unit="km" mathOperation="max">5</visibility>
    </Condition>
    <Condition ID="2" name="moderate">
      <state>Light haze, moderate sunlight, or slight heat shimmer</state>
      <visibility unit="km" mathOperation="max">20</visibility>
    </Condition>
    <Condition ID="3" name="normal">
      <state>Overcast, no haze, or no heat shimmer</state>
      <visibility unit="km" mathOperation="max">40</visibility>
    </Condition>
```

Figure 52: TPS Atmospheric Conditions Example

The elements in the above conditions file are defined in an XSD schema. See the **Appendix** 12.3.2 for an example of the GPS condition file. In Figure 52, the conditions are classified into *atmospheric* and *operational*. Atmospheric conditions determine the visibility range of the TPS telescope's line of sight. Taking condition with ID '1', the atmospheric influence is considered 'severe' when the environmental state is 'strong haze, strong sunlight, or severe heat shimmer'. The result of this condition is that the TPS sensor system can have a maximum visibility of 5 kilometers and beyond this range no measurements are possible or are guaranteed to be correct.

```
<ConditionList classification="Operation">
  <Condition ID="7" name="EDM_SinglePrismObservation">
    <state>EDM observing a single prism under atmospheric influence</state>
    <Dependencies>
      <Dependency ref="Atmospheric" ID="1">
        <visibility unit="m" mathOperation="max">1500</visibility>
      </Dependency>
```

Figure 53: TPS Operation Conditions Example

The operational conditions like, for example, use of a single prism target limits the distance range measuring capability of the TPS to 1500 meters (see Figure 53). The operational conditions of a sensor system can further be constrained by other conditions (i.e. dependencies) like the atmosphere (see in the example, a reference to atmospheric condition with ID '1').

In addition to stated conditions, basic rules can also be specified in the sensor model using simple "*if-then-else*" statements. For example, if the atmospheric conditions are 'severe', then the sensor should inform the user that no reliable measurements can be guaranteed at that moment. Also if the sensor

battery power is less than a certain value, e.g. 10%, then the sensor should raise an alarm/beep else it should inform the user that it can no longer accept any user commands or queries. The remaining energy will then be used only for very critical *push* (notification) cases where, for example, a landslide movement has exceeded a pre-defined threshold.

OGC SensorML and other research works propose a more formalized way of encoding rules, e.g. using rule-based languages like Schematron, Rule Markup Language (RuleML), etc. However, these more formalized approaches will also require extending the basic XML parsers. If the rules become complex, then the parsing tools also tend to be complex. We recommend use of simple XML-based files for encoding simple basic rules. For references to this approach, see Wang's dissertation [Wang, 2008].

6.2.3.6. Metadata

Metadata is a vital part of a model that provides a *one-stop location* for sensor system information e.g. important for search and discovery. Figure 54 shows a general metadata type which can be derived and restricted or extended by specific sensor types.

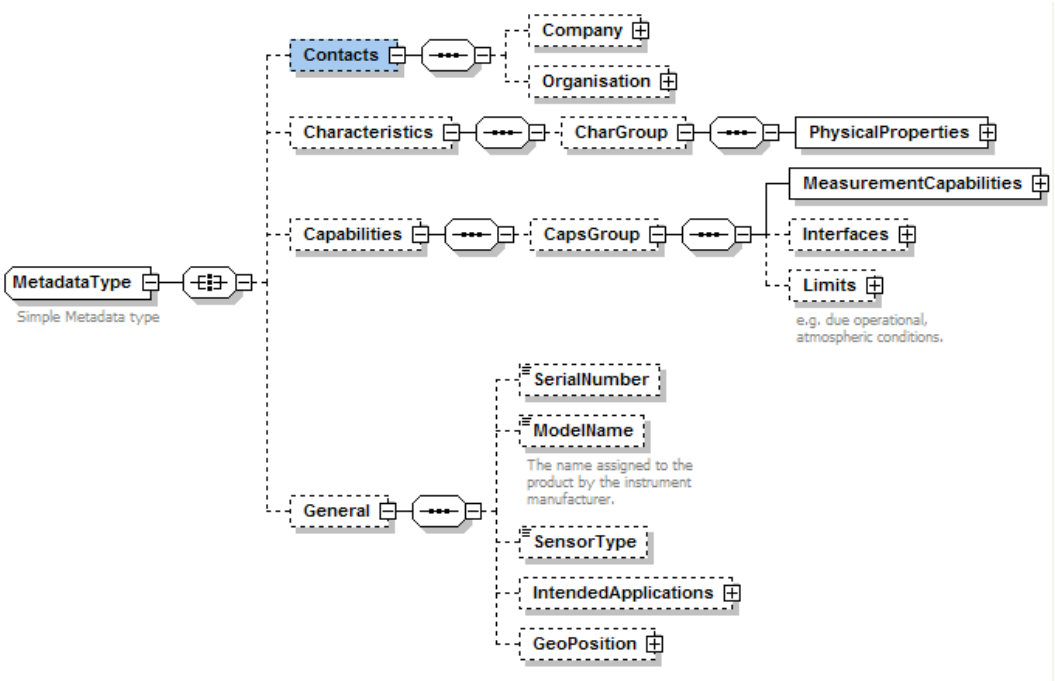

Figure 54: General Metadata Type

The following snapshots (Figure 55 and Figure 56) are extracted from metadata section of a TPS model instance. Snapshot 1 shows the TPS intended applications, its current geographical position "*GeoPosition*", which includes information about projection, coordinate system (e.g., geographic latitude and longitude, ECEF_XYZ, Gauss-Krueger (GK), UTM, etc.) and geodetic datum (e.g., World Geodetic System (WGS) 84, European 1950, DHDN (*Datum des Deutschen Hauptdreiecknetzes* – German Reference System), or can be any of the European Petroleum Survey (EPSG) codes (e.g. EPSG:31491 for DHDN/Germany GK zone1 …), etc.), as well as the quality information (e.g. accuracy expressed as standard deviations) about the position. The geographic information can be used for georeferencing the measurements made by the other non-position aware sensor systems.

```
<TPSSensor ID="TPS_Leica_TCA1800" xsi:noNamespaceSchemaLocation="tpsGenericModel.xsd"
  <!--TPS Metadata -->
  <TPSSensorMetadata>
    <General>
      <SerialNumber>618755-2</SerialNumber>
      <ModelName>TCA1800</ModelName>
      <SensorType>Terrestrial Positioning System (TPS) or Total Station</SensorType>
      <IntendedApplications>
        <Application>Engineering Surveying</Application>
        <Application>Construction projects like tunnels, highways, buildings,
        waterways, dams, roads etc.</Application>
        <Application>Terrestrial Positioning</Application>
        <Application>Structural Monitoring</Application>
        <Application>Three-Dimensional (3D) Positioning</Application>
      </IntendedApplications>
      <GeoPosition>
        <Location>
          <samplingTime>2006-12-31T12:12:01Z</samplingTime>
          <Cartesian>
            <northing_or_X>3492570.00</northing_or_X>
            <easting_or_Y>5341580.00</easting_or_Y>
            <height_or_Z>900.00</height_or_Z>
          </Cartesian>
          <CoordinateSystem>GK</CoordinateSystem>
          <Datum>DHDN</Datum>
          <Ellipsoid/>
        </Location>
        <Quality>
          <stdDevX unit="m">0.006</stdDevX>
          <stdDevY unit="m">0.006</stdDevY>
          <stdDevZ unit="m">0.01</stdDevZ>
        </Quality>
      </GeoPosition>
```

Figure 55: TPS Sensor Metadata Snapshot 1

Table 15 gives a simplified view of the contents in TPS sensor metadata snapshot 1.

TPS Sensor Metadata			
Serial Number	618755-2		
ModelName	TCA1800		
Sensor Type	TPS/ Total Station		
Sensor Classification	1. Point Acquisition Sensor 2. Distance Measuring Sensor 3. Angular & Tilt Measuring Sensor		
Intended Applications	• Engineering Surveying • Terrestrial 3D Positioning • Structural Monitoring		
GeoPosition	*Time*	2006-12-31T12:12:01Z	
	Location	*Northing*	3492570.00 m
		Easting	5341580.00 m
		Height	900.00 m
	Coordinate System	Gauss-Kruger (GK)	
	Datum	DHDN	
	Quality	*std Deviation X*	0.006 m
		std Deviation X	0.006 m
		std Deviation X	0.01 m

Table 15: TPS Metadata Example 1

Other example extracts from the metadata are shown in Figure 56 and Figure 57. The *MeasurementDataTypes* and the *MeasurementFunctionTypes* elements of the metadata provide information about the complete data types and function types of any sensor type respectively. These elements can be used for accessing and controlling the sensors. The sensor manager gets information about what data types and operations (functions) are supported by a particular sensor.

```
<CapsGroup>
  <MeasurementCapabilities>
    <MeasurementDataTypes>
      <Item name="Angles" type="AngleType" xlink:href="#AngleSensor"/>
      <Item name="Distances" type="DistanceType" xlink:href="#EDMSensor"/>
      <Item name="Coordinates" type="CoordinateType"/>
      <Item name="PointObjects" type="PointDataType"/>
      <Item name="PositionData" type="PositionType"/>
      <Item name="InclinationData" type="InclinationDataType" xlink:href="#InclinationSensor"/>
      <Item name="Time" type="DateTime" xlink:href="ISO 8601"/>
      <Item name="MeasurementBlocks" type="MeasurementASCIIBlockType"/>
      <Item name="CalibrationInfo" type="CalibrationInfoType"/>
      <Item name="AtmosphericCorrectionData" type="AtmosphericCorrDataType"/>
    </MeasurementDataTypes>
```

Figure 56: TPS Sensor Metadata Snapshot 2

The tabular form of the snapshot 2 is shown below.

Measurement Data Types		
Name	**Type**	**Responsible Component (Reference Link)**
Angles	AngleType	AngleSensor
Distances	DistanceType	EDMSensor
Coordinates	CoordinateType	
PointObjects	PointDataType	
PositionData	PositionDataType	
InclinationData	InclinationDataType	Inclination/Tilt Sensor
Time	DateTime (Definition: ISO 8601)	System Clock
MeasurementBlocks	MeasurementASCIIBlockType	
CalibrationInfo	CalibrationInfoType	
AtmosphericCorrectionData	AtmosphericCorrDataType	

Table 16: TPS Metadata Example 2

```
  <MeasurementFunctionTypes>
    <Item name="AngleMeasurement" xlink:href="#AngleSensor">
      <Resolution name="DisplayLeastCount" unit="sec">
        <Value>1</Value>
      </Resolution>
      <Accuracy standardUsed="ISO 17123-3" unit="sec">
        <Value>1.5</Value>
      </Accuracy>
    </Item>
    <Item name="DistanceMeasurement (Infared-IR)" xlink:href="#EDMSensor"/>
    <Item name="DistanceMeasurement (PinPoint Reflectorless-RL)" xlink:href="#EDMSensor"/>
    <Item name="DistanceMeasurement (Long Range-LR)" xlink:href="#EDMSensor"/>
    <Item name="Automatic Target Recognition (ATR)" xlink:href="#CCDSensor"/>
    <Item name="Power Search (PS)"/>
    <Item name="Electronic Guide Light (EGL)"/>
    <Item name="Remote Control"/>
  </MeasurementFunctionTypes>
</MeasurementCapabilities>
```

Figure 57: TPS Sensor Metadata Snapshot 3

The metadata element can also include information about the *history* or *lineage* of the model, as well as model constraints (e.g., valid time, legal, security …) as specified in the ISO 19115 metadata standard. Our element definitions of the *MeasurementCapabilities*, *Limits*, and *Interfaces* (see Figure 54 and Appendix 12.2.1) can also be used to extend other specifications (e.g., SensorML *MetadataGroup*) by creating either new elements or substituting similar existing elements. OGC SensorML proposes an *event model* concept, which documents information about the history of calibration, maintenance, or changes to algorithms. This concept can be adopted to build part of a sensor *state model* (i.e., each event object

representing the status of the sensor at that particular event time).

In summary, the workflow steps illustrated using the TPS in this subsection and above, can be applied to any other sensor system (e.g., GNSS receiver, Extensometer, etc.). The next subsection explains the mapping of the generic logical sensor models to standard-based models using standard encodings like OGC SensorML.

6.2.4. Mapping of Generic Logical Models to Standard-Based Models

The mapping should ensure that the entire generic logical model is correctly transformed to a particular standards-based format. Following are some of the techniques that can be used to map generic logical sensor models to standard-based models.

- Creating semi or fully automatic mapping (e.g., using XSLT) templates based on specific conversion rules.
- Manually profiling the SensorML elements to match the elements defined in the generic logical model
- Adapting similar existing SensorML or other standards-based sensor XML instance examples developed by other people. This approach can only be applied to simple models of the same sensor types.
- Etc.

Creating automatic mapping templates can be very demanding and sophisticated; hence in the dissertation we adopted the last two approaches. This is sufficient for explaining the mapping concept. Following are examples of the mapping results.

The root element of a SensorML document instance should reference important namespaces for standard encodings, like *sml (sensor model language)*, *gml (geography markup language)*, *ism (intelligence community information security marking)*, and *swe (SWE common encodings)*. The SensorML XSD file contains the necessary *includes* and *imports* of these various encodings (see Appendix 12.4.1).

Any constraints related to the sensor model instance, like time period for which the document can be used, whether the document is classified (non-public) or restricted for use by certain individuals or authorities, etc., can also be declared in the document (see Appendices 12.4).

The physical characteristics like device dimensions, power/battery requirements, etc., can also be documented. Information about the capabilities of the sensor, like supported measurement functions or operations and data types, are very important for sensor discovery and selection. Figure 58 illustrates some of the TPS capabilities described using OGC SensorML.

```
<sml:capabilities name="MeasurementFunctionCapabilities">
    <swe:DataRecord gml:id="measurementFunctionTypes">
<swe:field name="AngleMeasurement" xlink:href="#AngleSensor"/>
                <swe:field name="AngleMeasurementResolution">
                      <swe:Quantity gml:id="displayLeastCount">
                            <swe:uom code="sec"/>
                            <swe:quality>
```

```
<swe:Quantity referenceFrame="ISO17123-3">
                                                <swe:uom code="sec"/>
                                                <swe:value>1.5</swe:value>
                                            </swe:Quantity>
                                        </swe:quality>
                                        <swe:value>1</swe:value>
                                    </swe:Quantity>
                                </swe:field>
<swe:field name="DistanceMeasurementResolution" xlink:href="#EDMSensor"/>
<swe:field name="DistanceMeasurement (Infrared-IR)" xlink:href="#EDMSensor"/>
<swe:field name="DistanceMeasurement (PinPoint Reflectorless-RL)"
xlink:href="#EDMSensor"/>
<swe:field name="DistanceMeasurement (Long Range-LR)" xlink:href="#EDMSensor"/>
<swe:field name="Automatic Target Recognition" xlink:href="#CCDSensor"/>
<swe:field name="Electronic Guide Light" xlink:href="#EGL"/>
          <swe:field name="Power Search" xlink:href="PS"/>
          <swe:field name="Remote Controlled" xlink:href="urn:agis:def
:sensors:RemoteControl"/>
    </swe:DataRecord>
</sml:capabilities>
```

Figure 58: TPS Measurement Capabilities

For tabular view of TPS measurement capabilities, see **Table 16**.

Also of very importance is information about the limitations of a sensor due to certain physical and non-physical conditions.

Figure 59 shows a sample of the model. **Appendices** 12.3.1 and 12.3.2 describe some of these limiting conditions of a TPS and GPS in detail.

```
<sml:capabilities name="OperationalConditions" xlink:href="./ConditionsList.xml">
    <swe:DataRecord gml:id="temperatureConditions">
          <swe:field name="MeasuringTemperature">
                <swe:QuantityRange definition="urn:ogc:def:property
      :OGC:temperature">
                      <swe:uom code="degCel"></swe:uom>
                      <swe:value>-20 50</swe:value>
                </swe:QuantityRange>
          </swe:field>
          <swe:field name="StorageTemperature">
                <swe:QuantityRange definition="urn:ogc:def:property
      :OGC:temperature">
                      <swe:uom code="degCel"></swe:uom>
                      <swe:value>-40 70</swe:value>
                </swe:QuantityRange>
          </swe:field>
          <!-- Exposure to water, dust, etc.-->
    </swe:DataRecord>
</sml:capabilities>
```

Figure 59: TPS Limitations

Table 17 shows a tabular presentation of TPS limitations.

Sensor Operating Conditions (swe:DataRecord)			
Field	**Type**	**Unit**	**Values**
measuringTemperature	QuantityRange	degCel (°)	-20 to 50
storageTemperature	QuantityRange	degCel (°)	-40 to 70

Table 17: Sensor Operating Conditions

For the purposes of georeferencing the samples or measurements made by a particular sensor into a specific coordinate system of the target databases or storage system, geographic information like the GeoPosition of the sensor and its coordinate system frame can also be modelled. Figure 60 describes the TPS geographic information.

```
<sml:position name="TPSPosition">
<swe:Position localFrame="#TPS_Sensor_Frame" referenceFrame="urn:ogc:crs:EPSG:31493">
          <swe:time>
                <swe:Time referenceFrame="#SystemClockTRS">
                <!--get current time-->
                      <swe:value>now</swe:value>
                </swe:Time>
          </swe:time>
          <swe:location>
          <swe:Vector gml:id="TPSLocation"
          definition="urn:ogc:def:property:OGC:location">
                      <swe:coordinate name="x coordinate">
                            <swe:Quantity axisID="X">
                                  <swe:uom code="m"/>
                                  <swe:quality>
                                        <swe:Quantity gml:id="standardDeviationX">
                                        <swe:uom code="m"/>
                                        <swe:value>0.006</swe:value>
                                        </swe:Quantity>
                                  </swe:quality>
                                  <swe:value>3492570.00</swe:value>
                            </swe:Quantity>
                      </swe:coordinate>
                      <...>
```

Figure 60: TPS GeoPosition Information

Table 18 explains the XML instance contents.

TPS GeoPosition Information		
Local Frame (local coordinate system)	TPS_Sensor_Frame	
Spatial ReferenceFrame	e.g. EPSG:31493	
Temporal ReferenceFrame	System Clock	
Locations (vector)	*X coordinate*	3492570.00 m
	Quality (std Deviation X)	0.006 m
	…	…

Table 18: TPS GeoPosition Information Example

See **Appendices** 12.2 and 12.4 for detailed information. Note that for non-geosensors like extensometers, digital cameras, time domain reflectometers (TDRs), etc., the position information can be provided by geosensors like the TPS, GNSS receiver, etc. See examples of the digital camera and TDR model instances in the aforementioned appendices. Note also that the measurable properties can be associated with quality information.

Other necessary information resources for the system integrators or developers are the sensor system documentation. References to system manuals, technical specifications, etc., can be included in a SensorML document (see Figure 61).

```
<sml:member name="Specification Document"
xlink:arcrole="urn:ogc:def:property:OGC:specificationSheet">
       <sml:Document>
              <gml:description>System Technical Specification
              </gml:description>
              <sml:format>mime/pdf</sml:format>
              <sml:onlineResource
              xlink:href="http://www.leica-geosystems.com">
              </sml:onlineResource>
       </sml:Document>
</sml:member>
```

Figure 61: TPS Documentation Resources References

Figure 62 provides information about the *calibration event history* of the sensor. This information is very important for enhancing understanding of the sensor's state model and its respective measurements.

```
<sml:member name="calibration"
xlink:arcrole="urn:ogc:def:property:OGC:calibration">
       <sml:Event>
              <sml:date>2007-03-11T08:00:00Z</sml:date>
                     <gml:description>TPS Calibration Event
                     </gml:description>
                     <sml:contact xlink:arcrole="manufacturer"/>
                     <sml:documentation xlink:arcrole="calibrationReport">
                            <sml:Document gml:id="CalibTCA1800-2007-03-11">
                            <gml:description>Calibration report and data
                            about the sensor system deployment
```

```
                                    </gml:description>
                                    <sml:onlineResource
                                    xlink:href="http://manufDomain.com/
                                    TCACalibReport12345.pdf"/>
                                    </sml:Document>
                              </sml:documentation>
        </sml:Event>
</sml:member>
```

Figure 62: TPS Calibration Event History

Following is a tabular presentation of the calibration event object.

Calibration Event	
Event Date	2007-03-11T08:00:00Z
Contact (Responsible Authority)	Manufacturer
Available Documentation (e.g. CalibrationReport)	CalibTCA1800-2007-03-11
Online Resource (URI)	http://manufDomain.com/ TCACalibReport12345.pdf

Table 19: Important Calibration Event Information

In addition to the calibration event object, SensorML also allows for encoding information about the *taskable* parameters (e.g., editable or configurable calibration data) within the 'Parameters' element.
Finally, in addition to sensor interface definitions (see model instances in **Appendix** 12.4), SensorML provides elements that can be used to model the sensor process methods (e.g., algorithms, rules set, and any implementation details), see Figure 63.

```
<sml:ProcessMethod>
       <sml:implementation
       xlink:href="http://myDomain.de/TPS_Plugin_Interfaces.xml">
              <sml:ImplementationCode language="java"
              framework="AGISSensorPluginsFramework">
              <sml:sourceRef></sml:sourceRef>
              <sml:binaryRef xlink:arcrole="GeotechSoftware">
              </sml:binaryRef>
              </sml:ImplementationCode>
       </sml:implementation>
</sml:ProcessMethod>
```

Figure 63: TPS Process Method

As an example of a process model, a TPS can describe how its raw measurements (e.g. distances, angles) , together with available input parameters (e.g. corrective quantities related to atmospheric conditions, calibration data, etc.), can be feed into a computational component (e.g. based on least squares method, etc.) and produces final measurements (e.g. coordinates). The main advantage of a process model is that the users can store their raw measurements and the associated SensorML document, which they can use at any time in the future to recalculate the coordinates. This approach avoids the current problem of measurements data 'graveyards' and allows for better integration of sensor measurements into applications.

6.2.5. Sensor System Model Tests and Validation

The developed sensor models have to be tested within a software application, for example, a sensor plug-in. These tests provide invaluable checks about the usefulness or applicability of the sensor model; adequacy and quality (e.g. model correctness) of the modelled information matched against the application (e.g., landslide monitoring) requirements or expectations. Also the modelled geographic information (e.g. sensor position) and calibration information can be tested for correct georeferencing, i.e. using a controlled sample of sensor measurements data.

Error free parsing tests of the developed logical models (GLSMs and SBSMs) can be performed, as well as results/output consistency of the queries about sensor data (e.g. metadata, I/O definitions, etc.). The computational (or mathematical) models can be tested for accurate formulation or design.

Validation of a sensor model can be proved through the aforementioned test cases. In short, the results of the model tests should help the users to assess the usefulness of the developed logical sensor model. Basic validation of the model for conformance with the XML rules can also be done using common modelling tools like Altova's XMLSpy, Microsoft Visual Studio.NET, etc.

6.3. Proposed Generic Sensor Access and Control Service (SACS)

This subsection describes a SACS for use in interoperable management of sensors in (landslide) monitoring applications.

6.3.1. General Overview of Sensor Service Framework

Figure 64 shows a simplified sensor framework architecture that can be used in a landslide monitoring application. The core of this framework is the SACS which the system developer or integrator has to integrate or implement. Through the higher-level services, for example, based on OGC SWE framework and other standards, a user can be able to search and discover available sensors in the region of interest, query the health status, capabilities, etc. of the sensors, and even visualise the sensors in a map. Alarms/alerts can be received via a sensor alert service (SAS). Common DBMS like Oracle Database or Sensor Data Repository, PostGIS database, Microsoft SQL Server, as well as common data stream management system (DSMS) can be used for managing sensor measurements and observations. A DSMS can have complex functions designed to manage different data streams (e.g., video data) and can perform pre-processing (e.g., video compression, filtering, etc.) before transmitting the data to higher-level services.

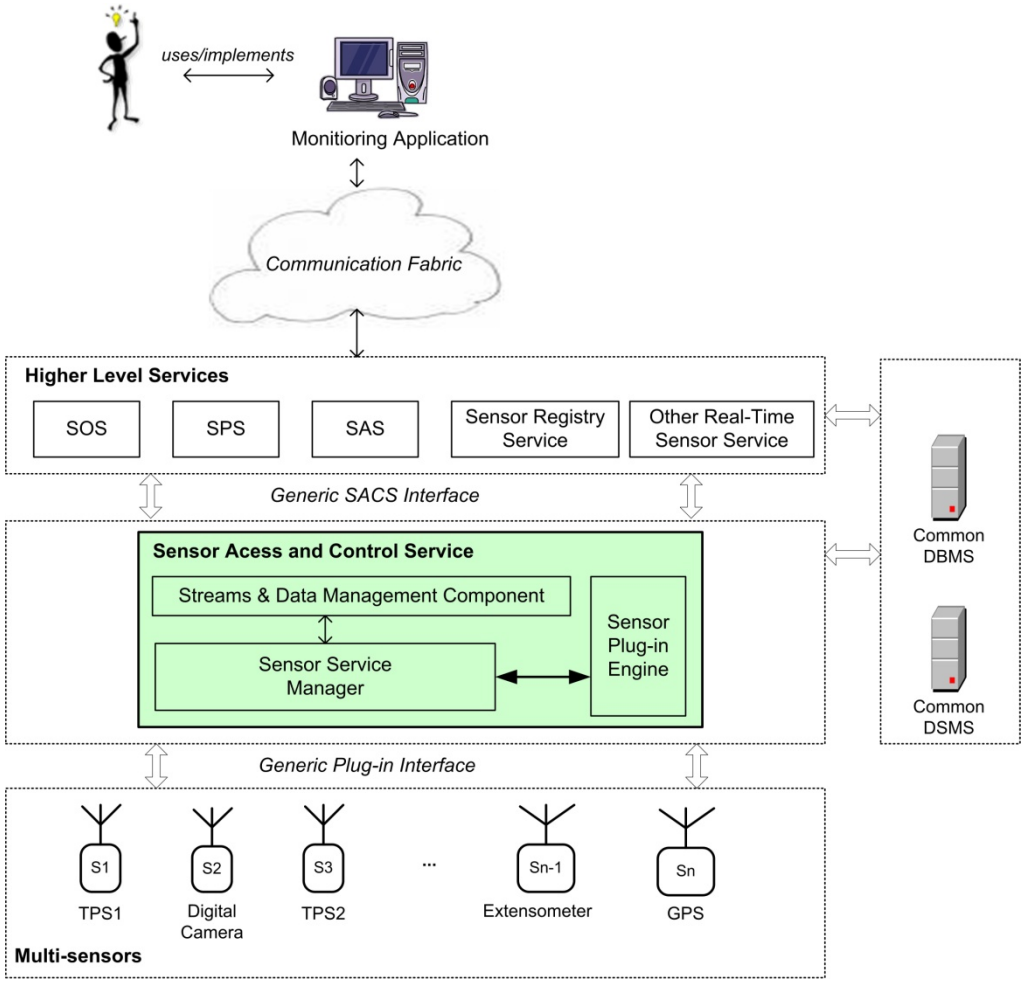

Figure 64: General Overview of a Sensor Service Framework with SACS

The communication fabric can be built so that it can alternate between synchronous mode (e.g. HTTP request/response model) and asynchronous mode (e.g., TCP-based, peer-to-peer, etc.). Interoperable communication can be supported by using any of the existing standard protocols.

Figure 65 shows an overview of the SACS main components.

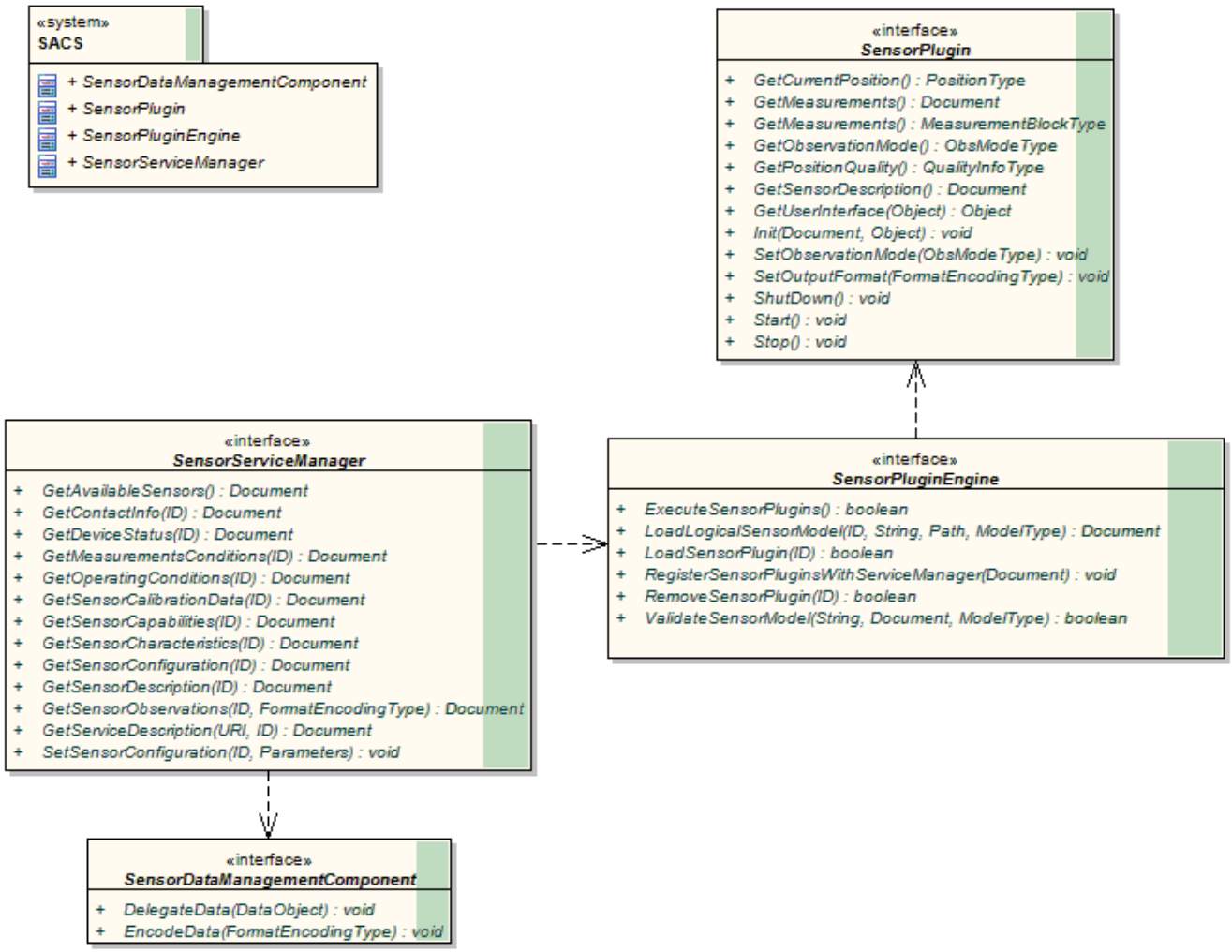

Figure 65: SACS Interfaces

The following is an explanation of the components in the above diagram.

6.3.2. Sensor Management (Plug-in) Engine

The sensor plug-in engine controls all the sensors that are used by the SACS instance.

Table 20 shows some of the important operations that build the engine. The basic requirement of the engine is that the sensor plug-in implements a generic plug-in interface (see the ExecuteSensorPlugins operation).

Table 21 lists the core interfaces of a sensor plug-in.

Operation	Description
LoadSensorPlugin (pID)	Loads a sensor plug-in with the given ID. Returns a Boolean, e.g. true if everything is ok.
RemoveSensorPlugin(pID)	Removes a plug-in with the given ID. Returns a Boolean.
ExecuteSensorPlugins()	Runs all the installed plug-ins. Each plug-in is assigned a separate thread or process. This operation invokes internally the *Init*, *Start*, and *GetUserInterface* implementations of the generic plug-in interface. Returns a Boolean.
LoadLogicalSensorModel(SensorID, ModelType,ModelName,Path) *ValidateSensorModel(ModelName,ModelType, ModelSchema)*	Loads the sensor logical model document. Its parameters can be the sensor ID, modelType (i.e. an enumeration type with GLSM, SBSM), unique model instance name, and path (file location or URI). The return type is an XML schema document of the models. The model instance can be validated against the model schema. This is usually done outside the engine for performance reasons (e.g. using external modelling software).
RegisterSensorPluginsWithServiceManager(PluginsList)	Registers all the available sensors with the SACS service manager. It needs a list of plug-ins as input.

Table 20: Sensor Plug-in Engine Operations Sample

Component	Description
Init(Object:Host,XMLDocument:PluginDescr)	Initializes a plug-in and it takes in a host object as a parameter. The plug-in uses this host object to invoke any important functions that host makes available. The document object is an XML document instance of the sensor model (e.g. SBSM or GLSM).
Start()	Starts the plug-in. Opens the communication channel between the sensor system and the software application.
GetUserInterface(Object:UIContainer)	Retrieves the GUI, if available.
GetPluginInterface()	Returns an XML interface definition file.
GetSensorDescription()	Retrieves the sensor model information as a complete XML document
Stop()	Stops the plug-in.
ShutDown()	Removes the plug-in from the attached process.
GetCurrentPosition()	Returns the geographical position of the sensor
GetPositionQuality()	Returns the quality information
GetMeasurements()	Retrieves all the measurements made by the sensor system at the time of request. The output is an XML document.
GetObservationMode()	Observation mode, as an ObsModeType enumeration element. For example, an EDM may be in tracking, fast or normal measurement mode.
SetObservationMode(ObsModeType)	Sets the observation mode using any element of the enumeration type
SetOutputFormat(FormatEncodingType)	Sets the data output format as defined in the capabilities document.

Table 21: Generic Sensor Plug-in Interface

6.3.3. Sensor Service Manager

The service manager is responsible for the management of the SACS instances. Higher-services like SOS, SPS, etc. can only interact with the SACS via the sensor service manager which defines the core SACS interfaces.

Component	Description
GetServiceDescription()	Gets description of the SACS, e.g. as a WSDL document or other formalized XML format.
GetAvailableSensors()	Gets a list of all available and useable sensors. Sensor registries can use this interface to get sensors list.
GetSensorDescription(SensorID)	Retrieves the description of a sensor as an XML document (e.g. SensorML, or Generic Model Instance). SensorID is a unique string.
GetSensorObservations(SensorID, FormatEncodingType)	Retrieves Observations/Measurements based on the FormatEncodingType e.g. SWE Common, O&M, SOAP XML, etc.
GetSensorCalibrationData(SensorID)	Gets the Sensor Calibration Information.
GetDeviceStatus(SensorID)	Gets the Health Status of the sensor.
GetSensorCapabilities(SensorID)	Retrieves the sensor capabilities document.
GetSensorCharacteristics(SensorID)	Gets the characteristics of the sensor
GetContactInfo(SensorID)	Gets the model developer contact information
GetSensorConfiguration(SensorID)	Gets the sensor configuration information
SetSensorConfiguration(SensorID, ParametersCollection)	Sets sensor configurations based on the taskable parameters specified in the parameters collection. If no parameters are given, the system uses a configuration file.
GetOperatingConditions(SensorID)	Gets the sensor system operating and atmospheric conditions.
GetMeasurementsConditions(SensorID)	Gets the measurements conditions, e.g. conditions that have to be met in order for a sensor to provide measurements. For example, a sensor may be only set to make measurements when certain pre-defined values are exceeded, or within certain levels of its available battery power.
SensorEventManager	This interface can be used to support (asynchronous) management of sensors. The sensors can be invoked as events. It defines the following interfaces: *Execute()*, *addEventListener(sensorID)*, *removeEventListener(sensorID)*.
SensorDataEventManager	This interface acts as an event handler. It retrieves the data from the sensor event. It contains a single operation called *SensorEvent(Object:data,Host:callingApp,Boolean:Status, Object: source)*

Table 22: Core SACS Interfaces

For interoperability purposes, all the above interfaces should return results that are conformant to the standard-encoding language data types or the generic logical model data types.

6.3.4. Process Executor

The service manager delegates the parsing and processing tasks to the process executor. The interfaces *LoadLogicalSensorModel* and the optional *ValidateSensorModel* are actually defined by the process executor. This component should enable the parsing of the logical models, e.g. GLSMs and SBSMs. The reading and writing of XML documents for the service manager is done by the process executor. For

advanced processing, it might be possible that this component extends the existing XML parsers in order to properly handle the defined logical models.

6.3.5. Streams and Data Management Component

This component assists the service manager by handling all the data and streams management tasks. For the dissertation, the component can be defined as a data retrieval and mapping or conversion tool. It receives the data from the sensor plug-ins and forwards it to the sensor service manager for transmission. In cases where the sensor plug-in submits raw measurements or proprietary data formats to the service manager, it is the responsibility of the data management component to convert the data into standard or common encodings like SWE Common, O&M, etc. Two interfaces can be defined respectively: *DelegateData(Data)* and *EncodeData(FormatEncodingType)*. Format encoding types can also include XML Binary format (EXI) and other binary serialization standard formats, like HDF, netCDF, GRIB, etc., for handling large volumes of data.

6.4. Summary

In this chapter, we have discussed the conceptual framework of the dissertation. A general workflow for defining and specifying generic logical sensor models, together with the associated results, has been given. The generic logical models can be used as *reference models* for comparing or analysing the applicability of different standards-based sensor modelling languages. We have discussed about the transformation or mapping of generic logical models to standard-based encodings (i.e. OGC SensorML). Examples of SensorML encodings have also been given in order to validate the applicability of SensorML.

In short, SensorML is verbose as compared to the generic logical models (see also Appendices 12.2 and 12.4). As the amount of detail about the sensor increases, the SensorML documents become less comprehensive or difficult to read. Some level of optimisation is required. Also the inclusion of GML and other standards into the SensorML is making it complex. The simplicity of our proposed generic logical models enhances better understanding of the sensors being modelled. Also the generic logical models are sensor-type based and clearly reveal associated semantics at each element level. On the other hand, SensorML is a language designed to model sensor instances and requires some level of expertise to use it. Also, the semantics is not direct but can be derived by reading the nested or child elements of a parent element. However, from our experiences, we can declare that the current version of SensorML standard can be successfully applied to model different sensors as demonstrated in this chapter.

We have also proposed an approach for integrating multi-sensors in an application and at the same time paying attention to resolving some of the integration conflicts discussed before, in order to achieve some level of interoperability. The last sections of this chapter have discussed a generic sensor access and control service, within a general sensor framework. The sensor framework serves to illustrate how SACS can generally fit into the OGC SWE concept as well as how monitoring applications can integrate or manage multi-sensors using it. However, the SACS concept we have suggested can be refined in detail. One approach is by developing several services for different application domains and the experiences gained from the results can then further be used to extend or modify the concept.

The final sensor models presented in this chapter have been adopted during usage in our landslide

monitoring application. We have modified (i.e., extended, specified, and refined) the sensor models several times during the prototype development based on the application requirements. The following Chapter 7 explains the usage of the sensor models and serves as a proof-of-concept of the dissertation.

7. Proof of Concept in a Landslide Monitoring Application

In the previous chapter, we have provided a conceptual framework in which the generic logical and standards-based sensor models play an important role in supporting interoperable management of sensors. They do provide a common query interface about sensors (i.e. common data contracts, and communication interfaces). Also discussed is a SACS concept which relies on the sensor plug-in engine and service manager for sensor management. The whole SACS framework provides a basic foundation on which interoperable management of sensors can be achieved. In addition, the concept of wrapping still plays a major role in encapsulating the proprietary differences of different vendors, and the main goal is to expose only common or standards-based interfaces.

This chapter will show the practical applicability of the discussed concepts within a landslide monitoring application. Adopted for the proof-of-concept is a sophisticated, open standards-based field-based mobile data acquisition system we have developed for landslide monitoring, under the research programme called "*Advancements of Geosciences*" [http://www.geoservices.uni-osnabrueck.de, Plan et al., 2004; Reinhardt et al., 2005; Beunig et al., 2006], sponsored by the German Ministry of Education and Research (BMBF – *Bundesministerium fuer Bildung und Forschung*). This system makes fully use of the possibilities of ubiquitous access – via wireless technologies – to various sources of information (including sensors). The system's mobile client employs sensors for the capturing of new geometries of features or for the updating geometries of existing features.

7.1. Mobile Geospatial Data Acquisition System

7.1.1. Project Background

The main motivation for the project has emanated from the need to provide a generic, standards-based data acquisition concept that enables field workers to access, analyze, acquire, quality check, and update geospatial data, whilst in the field. The system is conceptualised for use in landslide monitoring applications.

The current acquisition systems usually require experienced users with knowledge about the existing data and the underlying data models [Pundt, 2002]. Furthermore, for data acquisition systems that utilize different sensor systems, the users are expected to fully understand and correctly use these sensing systems. Before commencing any field work, the users have to examine and synchronize all the necessary data. Later on, in the field, if any unexpected circumstances occur, there is no possibility for further download of data relevant to the situation. Also, acquisition incompleteness and incorrectness of the data are often recognized when the field worker is back in the office, probably during the server update. In case, where the newly acquired data do not meet all the conditions of the data model and corresponding quality constraints, the worker might even have to revisit the exploration site. The issues related to data quality handling have been dealt with in [Wang 2008; Mäs et al, 2005] and the sensor management part is the work of this dissertation.

7.1.2. Application Scenario and Description of Test Sites

In cooperation with the local authorities (*Landesamt fuer Geologie, Rohstoffe und Bergbau, Baden-Wuerttemberg; Bayerisches Geologies Landesamt; Wasserwirtschaftsamt Rosenheim*), two test areas

have been selected for practical tests near Balingen and Rosenheim, Germany. These areas have very unstable surfaces, and it is suspected that rapid debris, rock masses, and other loose material may fall, at anytime, from the mountain slopes and may cause undesirable hazards to people using nearby roads, walking paths or to other close infrastructures. For that reason, the authorities have embarked on continuous monitoring of these areas using permanently on-site installed geotechnical and geodetic/surveying sensor systems. For example, there are a number of extensometers observing the inevitable seasonal expansions and contractions of the ditches (*ger.: Spalten*), cracks or gaps. Figure 66 shows a ditch being monitored by an extensometer.

Figure 66: Extensometer monitoring a Ditch.

Figure 67 illustrates the above image in a simple data model associating an extensometer to a ditch.

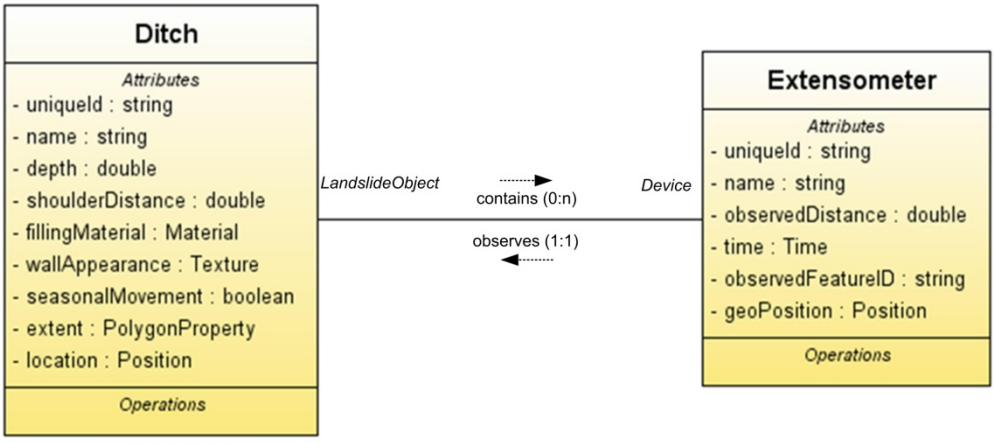

Figure 67: Ditch-Extensometer Data Model

The extensometer's daily measurements are sent regularly to a geological office for storage and evaluation. Also, automatic total stations, installed at remote locations from the observed features, register any surface movements (e.g., of embankments (*ger.: Boeschung*) close to roads or rivers). If these sensors' measurements exceed a pre-defined threshold, an alarm is generated and sent to the responsible authorities.

In case of an alarm, a responsible person goes to the field site and verifies the alarmed events. The mobile client also supports the user in decision-making process. For example, the user can download the data (i.e. map for navigation in the field) from the office server and visually compares the features (e.g. Ditches) in the database with the current physical environment. This will help the user to locate new ditches (see Figure 68) for measuring, and any other measurement points or active zones.

For existing features that have changed their geometry due to current landslide movements, the user can also download the previous measurements and information about the sensors at the time of acquisition. With this information, the user can also recreate the features from previous measurements. If any significant changes occur in the area, there may be need for updating and acquiring new features like new cracks, ditches, gaps, etc.

Figure 68: New Ditch Example

The geometry of the new features can be obtained by the measurements and the attribute information can be filled automatically or interactively. During the data acquisition process, quality checks or measures have to be performed before transacting the data on the geodatabase server.

7.1.3. System Architecture Overview

Figure 69 shows a generic architecture on which our mobile system is based on. A client can be any portable, rugged device like tablet PC, PDA or laptop with pivoting touch screen. The clients can communicate to the services via mobile communication technologies like WLAN, GPRS, UMTS and Bluetooth. Cabled communication means can be used to complement wireless means, especially when field servers are deployed. Note that the current wireless technologies like WLAN and UMTS can provide sufficient bandwidth for transporting larger amounts of data.

The clients can have access to different geoservices via standardized interfaces like OGC Web Map Services (WMS), WFS and other open service interfaces. In order to achieve full interoperability, the whole system should strictly adhere to standards.

The database layer can be built upon common databases like Oracle Spatial Database Server, OpenGIS Databases, etc. It is important that these database systems provide data that is encoded in standard formats like OGC GML, SWE Common data, etc. Existing proprietary or legacy data formats can be mapped

using wrappers into standard formats.

Since the collection of new data is done using different sensors, the implementation design of the system architecture should adhere to the concepts presented in this dissertation, in order to support interoperable interfacing to these various instruments.

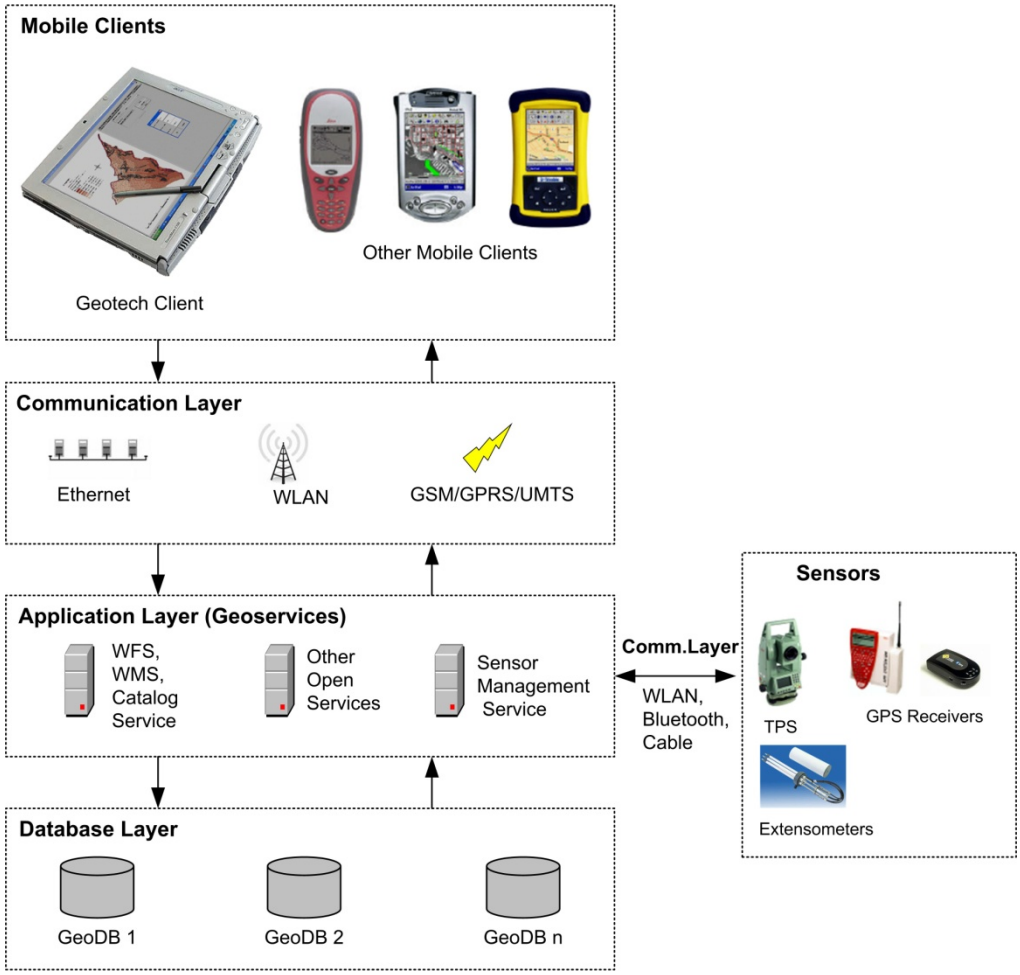

Figure 69: Generic Architecture for Mobile Geodata Acquisition System

The sensor management service is based on the SACS concept discussed in the previous chapter.

Following are the specific characteristics of our prototype data acquisition system:

Mobile Client (Geotech): A tablet PC running a Geotech software (Java application). The Geotech software is mainly composed of a GeoEditor and various plug-ins. The Geoditor is a plug-ins host with functionality like visualisation (using scalable vector graphics (SVG) format [W3C-SVG, 2003]]), graphical user interface, etc. For data acquisition, sensor plug-ins like the GPS, TPS, and extensometer, have been used.

Communication Layer: Physical communication includes WLAN, GSM, Bluetooth and serial cabling (i.e. RS-232).

Application Layer: Includes a Web Feature Service with transaction capability (WFS-T), and Sensor

Management Service. The web server consists of Apache Tomcat Servlet container and the Apache Web Server. The communication model used is the HTTP request/response.

Sensors Layer: Physical devices like total station, GPS receivers, and extensometers. Interaction with the sensors relies on the logical sensor models.

Database Layer: PostGIS Database for the application data. Currently, the sensor models are stored separately in file directories. Since these models are XML-based, they can also be stored in databases like the Oracle which support the XML format and XSQL for querying the database.

7.1.4. Sensor Wrappers

Before showing some of the system demonstrations and results, this subsection explains the sensor wrapping techniques used for the prototype. There are currently different tools that exist to support automated wrapping of sensor APIs to suit the targeted system (see previous chapter). However, we recommend use of generic wrapping tools that support automation with very minimal or no manual post-processing of the results. As an example, we have used an open source tool called SWIG [http://www.swig.org] for generating a Java TPS wrapper for the GeoCOM C/C++ library. The only preparatory work that the developer has to do is to create a *mapping interface* file, which the generator takes to perform the correct mapping of data types and structures as well as interface or method signatures. The dissertation author has extensively participated in testing the SWIG platform for correct wrapping of C/C++ to Java interfaces.

SWIG is an *interface compiler* that connects programs written in C/C++ with scripting languages (e.g. Perl, PHP, Python, Ruby, etc.) and programming languages (e.g. C#, Java, Lua, Common Lisp, etc.). In case of C/C++, it uses the header file declarations as input for wrapper generation. This platform can also be used for rapid application development – testing communication between external library and target application. Figure 70 shows a simplified workflow for integrating C/C++ libraries with different language libraries.

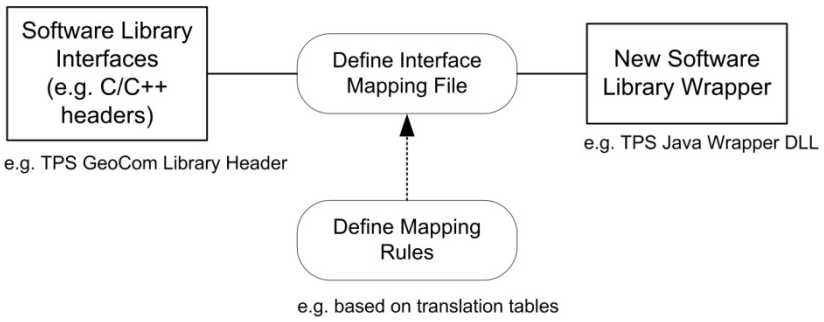

Figure 70: Simplified SWIG Workflow

Here is an example of an interface mapping rule.

GeoCOM C/C++ Interface:

```
void BAP_MeasDistanceAngle(BAP_MEASURE_PRG &DistMode, double &dHz, double &dV, double &dDist);
```

This method reads the distance measuring mode (*DistMode*) from an enumeration data structure and returns references (&) to the TPS raw measurements (*dHz = horizontal angle, dV=vertical angle, and dDist=slope distance*). However, for our Java application to know that the raw measurements are output values, the mapping interface file has to define a mapping rule as follows:

```
%exception BAP_MeasDistanceAngle(BAP_MEASURE_PRG &DistMode, double &dHz, double &dV,
double &dDist)

{

Result = (void)(BAP_MeasDistanceAngle((BAP_MEASURE_PRG&)*arg1,*arg2,*arg3,*arg4);

}

%apply double &OUTPUT {double &dHz, double &dV, double &dDist};
```

From the above snippet, the *%exception* command tells the compiler to treat this GeoCOM function as a special case and redefines the function as shown by *result* (in a syntax the JVM understands). Then use *%apply* to tell the JVM that these parameters should be treated as output.

With respect to sensor wrapping and from our experiences, we conclude that:

- Complex data types and structures always require that a developer or system integrator define some rules for correct mapping from the source system to the target system.

- Rapid deployment of sensors is very important for mission-critical applications; therefore one should avoid software wrapping tools that require much post-manual editing or those that base on trial-and error approach in order to achieve acceptable results.

7.1.5. Data Acquisition Workflow

This subsection explains a simplified workflow for acquiring new features (e.g. ditches, cracks, other measurement points, etc.). In our application, two use cases (*insert data* and *update data*) required that the geologist measures the geometry of the features by using various sensor systems (e.g. GNSS, TPS, etc.), and then update the office geodatabases from the field. The “insert” use case is for newly acquired features and the “update” use case involves editing (i.e. adding, deleting or modifying) parts of the geometry and/or attributes of the existing features.

Figure 71 shows the graphical user interface for the mobile client. An SVG display of landslide area (with features like ditches, roads …) is shown in the map control and on the right a “Feature Acquisition Plug-in” for geodata acquisition. The feature acquisition plug-ins uses different methods for acquiring measurements data.

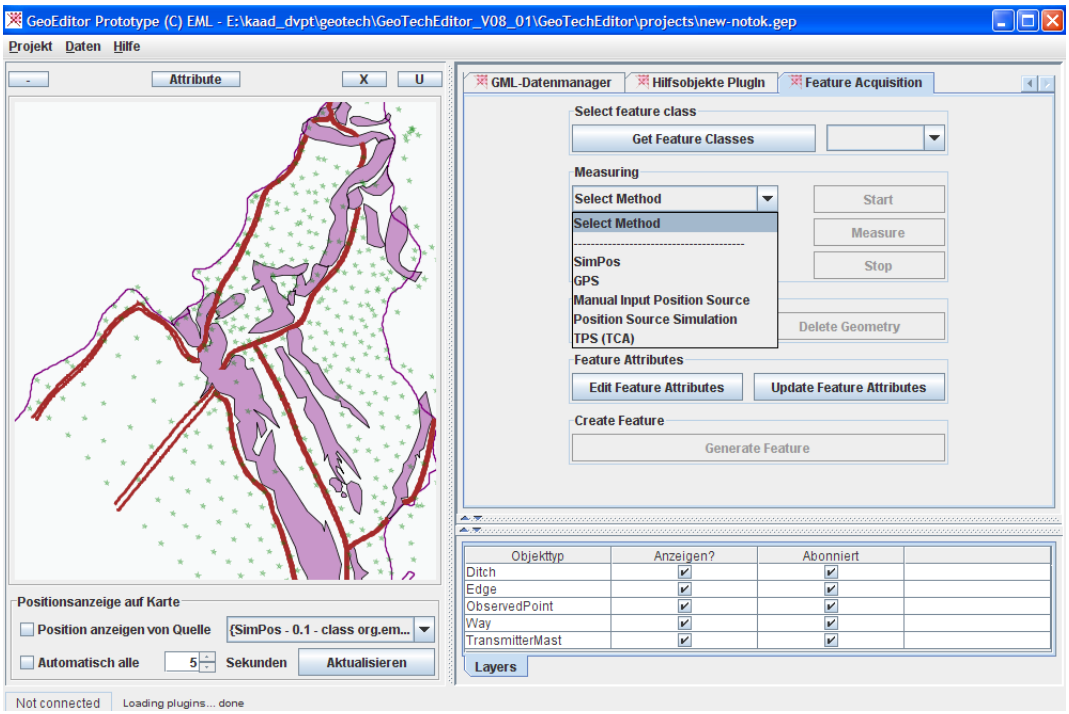

Figure 71: Overview Mobile Client Graphical User Interface (GUI)

It is during system application initialisation that an instance of SACS is created. At application start-up, the SACS instance will use its sensor service manager instance to read the sensor registry and gets all the available sensors using the 'GetAvailableSensors' interface. Currently, the sensor registry is a simple XML configuration file that contains a list of the sensor systems. Table 23 shows an example of the contents of the file.

Configuration File	
Sensor1	
SensorID	TPS_Leica_TCA_1800
SensorType	TPS
ModelType	GLSM
ModelLocation	c:/kaad/geotech/SensorModels/glsm
ValidateModel	No
Sensor2	
SensorID	ID_1
SensorType	GNSS (GPS)
ModelType	SBSM
ModelLocation	c:/kaad/geotech/SensorModels/sbsm
ValidateModel	No
…	…

Table 23: Configuration File Contents Example

The logical sensor models (GLSMs and SBSMs) are stored in respective folders as shown in the configuration file. The sensor models have been defined in chapter 6.

The Feature Acquisition plug-in gets the list of registered sensors from the sensor service manager and displays them in the graphical user interface (GUI) as shown in Figure 71. The combo box lists the

physical sensing devices (GPS, TPS) and position simulators. Note that it is at the application start-up that the sensor manager tells the sensor plug-in engine to load all the available sensors into the application space. This means that relevant software objects are reserved for storing the loaded models. The responsibilities of the sensor plug-in engine are as follows:

- Loading and initializing the sensor plug-ins (e.g. TPS Plug-in, GPS Plug-in, Extensometer Plug-in) into the application space
- Running the loaded sensor plug-ins
- If the configuration file sets the 'validateModel' parameter to 'Yes', then the respective sensor instance is validated against the model schema. Note that validation is usually done using external modelling software; hence the validation is turned off.
- Creates the plug-ins using the defined sensor models (see chapter 6)
- Registers the available sensors with the service manager instance

See chapter 6 in section 6.3.2 for the sensor plug-in engine interfaces and also for information on how the engine interacts with individual sensor plug-ins

Following is an explanation of the workflow for geodata acquisition by measurements.

1. **Access Sensor Model Information**

   Based on the logical sensor models, the user can now access and control the sensor system information via defined SACS interfaces (see chapter 6, section 6.3). Figure 72 shows a simplified interface for getting sensor information about the measurement capabilities, physical characteristics, operating conditions, and model contact information.

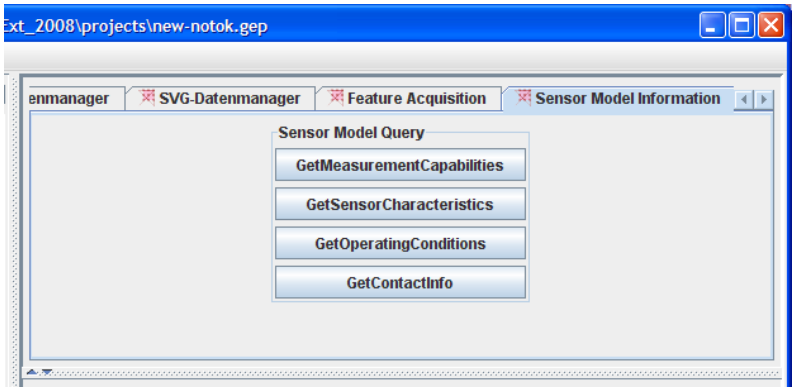

Figure 72: Sensor Model Query Interface

Depending on the selected sensor and type of the model being used (e.g. generic or standards-based), the response to the queries are in form of XML. For example, by clicking the 'GetMeasurementCapabilities' button, the Feature Acquisition plug-in sends a request to SACS instance which then retrieves an XML document using the 'GetSensorCapabilities' interface of the sensor service manager. For an example of the results, see XML instances in chapter 6, Figure 56

and Figure 58.

By clicking the 'GetOperatingConditions', the results can be as shown in Figure 53 and Figure 59. The information that the user gets through these queries can help him to better understand the sensor system and the measurements data being delivered. On the other hand, the software application is also able to know how to connect and change parameters of any specific sensor by reading the models. It can also know the data format in which the measurements are being encoded. Other information about the sensor geolocation and power levels can be obtained from the models.

Note also that the user interface for querying the models can be adjusted to have more buttons, e.g., for setting different sensor system parameters like operation mode or the data types in which the measurements should be encoded (e.g. own defined generic types, GML or SWE common data types). Equipped with the knowledge about the installed sensors and their respective data output, the user can go ahead and carry the following steps.

2. Select Measuring Method

In the Feature Acquisition plug-in (see Figure 73) the user can select a method for using during data acquisition (e.g., GPS, TPS) or any of the displayed simulators.

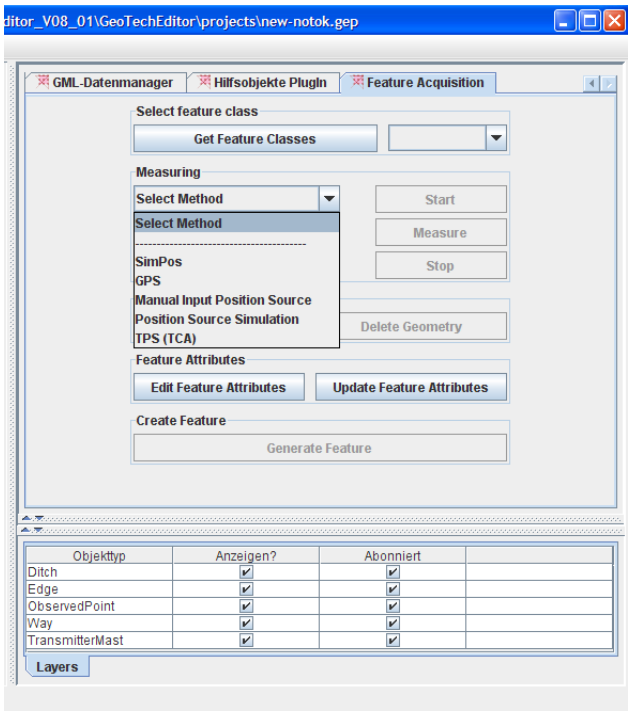

Figure 73: Select Method for Measuring (i.e. Physical Sensor System or Simulation)

If the device communication settings (read from the sensor model instance) are not correct, a message box can be displayed (see Figure 74 for the TPS).

Figure 74: GeoCOM Communication Settings

If the communication settings are correct but the sensor system is not connected, the user is notified (Figure 75).

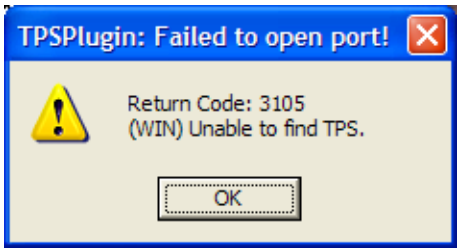

Figure 75: TPS Response message on opening port failure.

The error codes can also be documented in a separate XML file.

3. Select Feature Class

If the selected measuring method is properly configured and working, the user can get the list of all feature classes specified in the geoservice data model (e.g. from the WFS capabilities document). Figure 76 shows a combo box dynamically filled with the classes *Ditch*, *Way* and *ObservedPoint*.

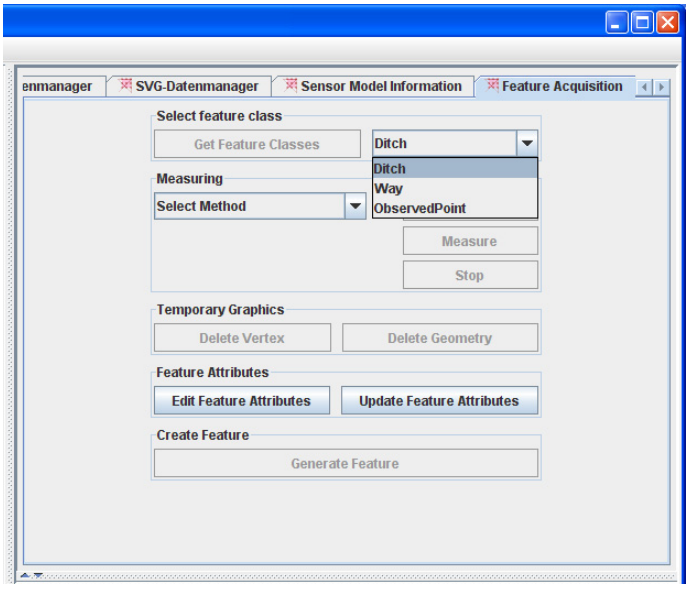

Figure 76: Get Feature Classes

The next step is to measure the feature's geometry.

4. Measure Feature Geometry

The selected feature geometry (e.g. point, line, and polygon) determines the measuring process. For example, a ditch is defined as a polygon so a series of measurements is expected and the first observed point will be identical to the last point in order to close the polygon. A line has to have at least two points, and the first and last point should not be the same.

5. Record Feature Attributes

If the geometry is completed, a form is displayed for entering the attribute information.

6. Create New Feature

The last step is to attach the attributes to the respective geometry and create a new feature. For interoperability purposes, all the features are encoded in GML and send to the map control which then renders the features into SVG format. Figure 77 shows a new ditch highlighted in the map control. The GPS can be used to acquire different observed points. Also on the right side, a GPS receiver position and quality information (HDOP and error in position) are shown. This information can also be encoded in a GPS model instance.

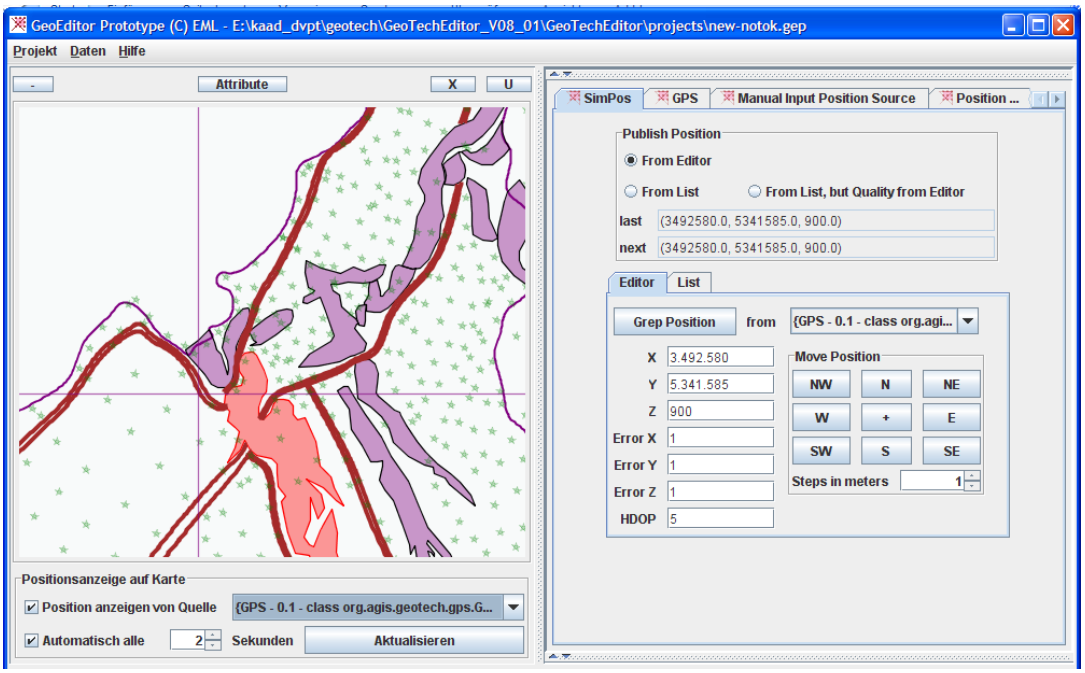

Figure 77: GPS Position and Quality Information

Finally, if *quality checks* are performed and the newly acquired features are topologically and geometrically consistent with the geodatabase model, then the user can insert the features into the office database.

Note that raw measurements data from the sensor systems can also be encoded in common formats (e.g. SWE Common, NMEA, O&M, etc.) and can be stored on the geoserver together with the respective

sensor model instances.

Following is a brief discussion about the field tests and the results.

7.1.6. Field Tests and Results

Figure 78 gives an overview of the field test configuration. The whole test area covered an area of approximately 150 by 250 m^2. The terrain varied ranging from very steep, undulated to relatively flat areas. The terrain appearance determined the number of WLAN access points to be installed. Also, in areas where access to the office geodatabases (via GSM/GPRS) has not been possible, a field server has been used.

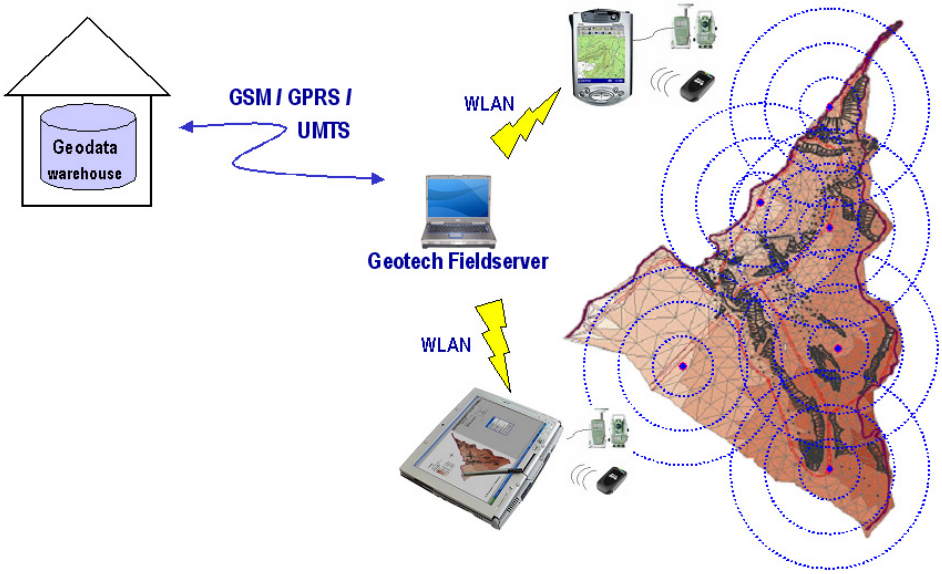

Figure 78: Balingen System Tests Configuration

The results of the tests can be summarised as follows:

- The proposed system has successfully proved that it can greatly support the user workflows by allowing him to make decision in the field about any changes in the landslide area, visualize all the data already acquired as well as previous measurements. The system can be connected to different sensors using different sensor models that are encoded in generic or standard-based formats. The pluggable sensor plug-ins made the system quickly sensor-aware. The newly acquired features have been encoded using standard-formats like GML. In short, the system provides a simplified framework for interoperable management of sensors and their respective data.

- GPS availability: With low-cost GPS receivers that are aimed for mass marketing, it has not been possible to receive signals in most parts of the wooded areas. First tests have shown that this is much better to use professional receivers. However, sensor systems like total stations (TPSs) can greatly complement GNSS devices in cases of no signal availability.

- Extensometers are very sensitive to disturbances, e.g. animals or tree falls. In order to avoid unnecessary costs of field visits in cases of false alarms, visual sensors like digital cameras can be used.

- WLAN accessibility: The WLAN connections have been very stable and reliable in the whole test area. Figure 79 shows the field geoserver connected to the WLAN router.

Figure 79: Field Geoserver and WLAN Router

- Use of sensor networks with mobile GIS applications combined with in-field quality checks can provide powerful solutions for data acquisition systems.

7.2. Results of Sensor Models Validation

The generic and standards-based models have been tested and proved adequate for the described landslide monitoring application. The models have continually been improved during the usage in the application. Appendices 12.2, 12.3, and 12.4 contain examples of the model instances for total station (TPS), GPS receiver, extensometer, digital camera and time domain reflectometer (TDR).

The information contained in the models could satisfy the case study landslide monitoring application, but these models may not be adequate for other applications. However, it is possible to extend the models since they are based on XML schema language. The rules for modification of XSD schemas also do apply to the defined models. For example, the data types like enumerations, patterns and facets can be used to define specific or limit data content and the 'any' element reserves room for adding later any data type or element. Abstract data types like MetadataType, PositionType, etc. can be modified by using *extension* or *restriction* operations of schema modelling software like Altova's XMLSpy. Refer to chapter 2.2.4.1 for more information about XML schema extensibility. Also see chapter 6, section 6.2.5 for issues to consider during sensor system model tests and validation.

However, following are some missing points and improvements to the developed sensor models.

- The generic logical models (GLSMs) could support the XML schema concept of *SubstitutionGroups* and this could facilitate better extensibility of model elements or types by substitution. This means that entities belonging to the same group or that have an "*is-a*" relationship can be substituted for each other without breaking the code for parsing the models.

- The SensorML-based models (SBSMs) are very verbose and there is sometimes unavoidable usage of deep nesting the elements in order to fully model a particular element (e.g. sensor model component) or data type (e.g. measurements output), see sample instances in **Appendix 12.4**. This can eventually lead to degradation of performance of the software parsing the model instances.

- The developed models could benefit from using standard interfaces like IEEE P1451 Smart Transducer Interfaces for defining the sensor model communication interfaces.

- Etc.

7.3. Summary

This chapter has shown an example of the practical application of the dissertation concept in the field of mobile data acquisition in landslide monitoring applications. The data acquisition workflow of the system has been explained and the results of field tests and sensor model validation tests have also been discussed. The developed system prototype has been successfully tested in the landslide monitoring applications, but the concept can also be applied to other fields of applications.

Following are the main advantages of the suggested concept for management of multi-sensors compared with the current industrial practices.

- Support of non-proprietary (generic and standards-based) solutions enable interoperable usage of different sensor systems in a single application space as well as allow for easy exchange of measurements data encoded in common formats, thereby eliminating the overburdening data conversion problems.

- It greatly reduces the time efforts required to integrate disparate sensor systems in new or existing legacy applications (i.e. via plug-and-play) as well as reduces application maintenance efforts.

- Usage of sensor models affords us to preserve low-level sensor measurements data and the ability to reprocess that data sometime in the future. Lack of such models usually leads to ‘data graveyards’ long time after the measurements campaign.

- Etc.

Refer to chapter 4, section 2.2.4 for more information about the importance of sensor standards and specifications, and also see section 2.2.5.1 on the advantages of interoperable management of sensors in monitoring applications.

8. Final Summary

This chapter provides an overall summary of the dissertation work. Following are the contributions matched against the objectives in chapter 1.2.

- ***Propose a conceptual framework for interoperable model-based integration of multi-sensors***: The dissertation has provided a concept for interoperable management of heterogeneous, multi-sensors within landslide monitoring applications. It has shown that interoperability can be achieved at different levels (e.g., community level and cross-community level). At community level, it can be sufficient to define and specify common, generic specifications of different sensor types that can enable easy exchange of sensor systems within that community. If there is need to ensure interoperability for broader, diverse communities at all scales (local, regional, national and international), international sensor standards and specifications should be used. Furthermore, the community defined sensor models play an important role in improving the current international standards and specifications that may be undergoing development, reengineering or refinement process.

  We have proposed a simplified architectural framework for managing different sensor systems. The core of the framework is the SACS which aims at extending the OGC SWE framework by providing a service that is only dedicated to handling sensor systems.

- ***Define and specify minimal generic logical sensor models and perform detailed analysis of needs of landslide monitoring applications***: We have analysed the detailed needs for data and sensors in landslide monitoring and early-warning applications, and we have also looked at the current situation with respect to sensor management in existing landslide monitoring applications. We have also defined and specified the logical sensor models and these can be extended in other applications, if necessary. The logical sensor models play a central role in our proposed conceptual framework. Our perspective is that these models provide a *single portal or query interface* for the respective sensor types. For users, it is through such models that they can gain knowledge about the sensors they are using as well as the provided measurements data. Equipped with detailed information about the sensors, both people and machines can make meaningful deductions about the provided measurements data. Especially for system integrators and developers, the logical models put the sensors out-of-the-box and enable them to access and use common interfaces with *semantics* attached (thereby avoiding misinterpretation), hence reducing the sensor system integration times. This is important for time-critical applications.

- ***Propose a workflow for the development of logical sensor models***: We have proposed a workflow for defining and specifying logical sensor models and have shown how these models can be mapped into standard models (e.g. using OGC SensorML, and SWE Common data types).

- ***Research on methods and technologies for achieving interoperable integration of sensors***: We have looked at the methods and techniques that can be employed in order to integrate multi-sensors in applications. We have also identified and discussed about the different types of integration conflicts that arise due to use of vendor-specific solutions (e.g. proprietary interfaces, protocols, and data formats), and have shown how some of these conflicts can be resolved at different levels of

granularity or complexity. The use of *sensor wrapping concept*, which at the moment is the only solution, has enabled us to resolve differences in connection, command and control of different sensor systems by exposing only common, generic interfaces. The measurements data have been encoded into generic and standard-based formats. The plug-in concept ensures the plug-and-play of different sensor systems into applications. The usage of SOA and Web Services approach is recommended and has been adopted for this dissertation.

- ***Investigate the applicability of sensor standards for interoperable integration of sensors***: The OGC SensorML and SWE Common data types have been used successfully to encode the already defined generic logical sensor models. However, there is need to practically test how the SACS concept fits into the OGC SWE framework of services.

- ***Validation of the concept and developed sensor models in a landslide monitoring application***: A proof of concept in form of a mobile field-based geodata acquisition system has been given. The system application has demonstrated how model-driven approach for sensor management can be realised, thereby validating the proposed concept of the dissertation.

9. Conclusions and Future Outlook

The dissertation has provided a basic conceptual framework that can be adopted and extended for interoperable management of multi-sensors in landslide monitoring applications and other application domains. The feasibility of the concept has been demonstrated in a specific landslide application. However, interoperability can be achieved at varying degrees (i.e. zero interoperability to full interoperability). Full interoperability can only be obtained if standards are 100% adopted, which may in turn enable complete self-integration of sensors into various applications.

We can conclude that through *model-driven* approaches (i.e. self-descriptive, complete, open generic or standard-based sensor models) both people and software applications can easily integrate (plug-and-use) and better understand the sensors they are working with as well as the measurements data. In addition, applications can also be able to exchange sensors (via sensor models) and integrate them with zero or minimal adjustment. In a nutshell, in order to avoid "data graveyards", applications using sensor services should store both raw sensor measurements data and the respective sensor model instances of the sensor systems used at the time of data acquisition. With respect to sensor services, SOA and Web Services (e.g., SOAP based, RESTful, etc.) play an important role is supporting loose integration of sensor systems into various applications.

Following are some research issues that can be considered for future work:

- *Modelling of Sensor Systems (Physical Devices) and Sensor Networks*

  There is need for defining and specifying other generic conceptual and logical models for other sensor types (like it has been done for the landslide monitoring applications), and also for sensor networks as a whole in other application domains. These models should be richer in semantics. For the dissertation, we have used simple XML-based dictionaries for defining sensor model terms (e.g., EDMSensor, TPS vertical index error, position, resolution, etc.) and procedures (e.g., TPS polar computation, GNNS real-time kinematic surveying, etc.), but more formalized ontology based on existing standards and specifications like Web Ontology Language (OWL) and projects like SWEET ontologies [http://sweet.jpl.nasa.gov/ontology] can enable higher levels of interoperability. Also the metadata we have defined as well as those defined by existing sensor modelling standards like OGC SensorML need to be completely harmonized with established metadata standards like ISO 19115. In addition, it is important to analyse the inherent metadata delivered by different sensor systems and how they can easily be described using the elements defined by the current metadata standards. With respect to sensor networks, the model abstractions should be generic and clearly define the various components, sensing device types, and their interfacing with the real world for various, different application scenarios. Relationships of each and every entity that builds up the network should exactly be defined, as well as any aspects related to the *quality* of network.

- *Fusion of Heterogeneous Sensor Measurements Data in a Single Application Space: Methods and Algorithms*

  There is need to investigate on methods and algorithms like extended kalman filtering (EKF), neural networks and neuro-wavelet analysis for (semi) automated fusion of various measurements data or

data streams. The data streams can be fused based on certain criteria, rules and parameters like fusion of measurements within a particular spatial extent (region of interest) that were obtained on a specific time interval or instance (temporal aspect). As sensor networks grow in the number of sensing devices, there will be a plethora of data to manage. Therefore, it is also necessary to investigate data management systems that can handle huge streams of data, perform efficient data insertion and retrieval. The data management systems can be validated in application domains that require handling of complex numerical simulations or real-time data streams like monitoring of large-scale landslides, weather, climate, and other oceanic critical events.

- *Applicability of the Sensor Access and Control Service Interfaces in Supporting Multi-Sensors*

  The SACS concept we have suggested can further be refined in detail by developing several sensor services in different application domains. However, to address a broader scope (beyond community level as currently supported by SACS in landslide monitoring applications), there is need to investigate the applicability of emerging standards like IEEE P1451 Smart Transducer Interface which define and specify vendor-neutral models and interfaces for integrating disparate sensors. Furthermore, harmonizing SACS with other ongoing research works like EU SANY Integrated Project and 52 North SWE Framework is very important. For example, part of the SANY UWEDAT (*Umweltdatenerfassungssystem*) SWE based architecture can be integrated with our SACS and the IEEE P1451 interfaces.

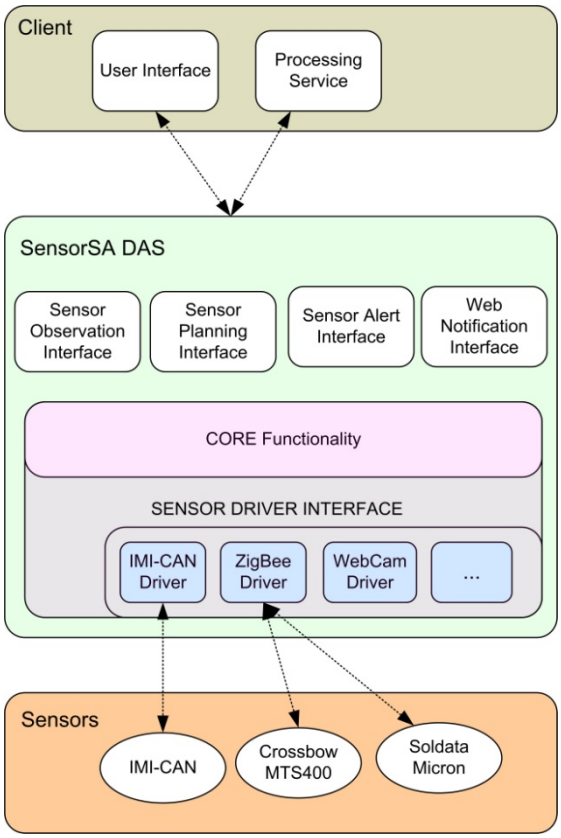

Figure 80: UWEDAT SWE Architecture [Havlik et al., 2008]

The SensorSA Data Acquisition System (DAS) sensor driver interface part (see Figure 80 above) can be included in the SACS sensor plug-in engine. The various sensor plug-ins can then wrap the respective sensor drivers. The results of the integration should enable the SACS to communicate with different sensors via open, standards-based communication interfaces.

10. References

Baarda, W. (1967): *Statistical Concepts in Geodesy*. Publications on Geodesy, Netherlands Geodetic Commission, Delft, vol. 2, no. 4.

Baarda, W. (1968): *A Testing Procedure for Use in Geodetic Networks*. Publications on Geodesy, Netherlands Geodetic Commission, Delft, vol. 2, no. 5.

Balazinska, M., Deshpande, A., Franklin, M. J., Gibbons, P. B., Gray, J., Nath, S., Hansen, M., Liebhold, M., Szalay, A. and Tao, V. (2007): *Data Management in the Worldwide Sensor Web.* IEEE Computer Society. Pervasive Computing, 1536-1268, 07, p. 10-20.

Barkmeyer, E. J., Feeney, A. B., Denno, P., Flater, D. W., Libes, D. E., Steves, M. P. and Wallace, E. K. (2003): *Concepts for Automating Systems Integration*. National Institute of Standards and Technology; Technology Administration, U.S. Department of Commerce, NISTIR 6928.

Beck, T. J. and Kane, W. F. (1996): *Current and potential uses of time domain reflectometry for geotechnical monitoring*. Proceedings of 47th Highway Geology Symposium, Cody, WY., Wyoming Department of Transportation.

Becker, J.-M., Heister, H. and Slaboch, V. (2000): *New Technical Standards Improving the Quality in Positioning and Measurement.* FIG Proceedings:Quo Vadis International Conference, Prague.

Bernard, L., Kanellopoulos, I., Annoni, A., Millot, M., Iso, J. N., Nowak, J., and Toth, K. (2005a): *What Technology does INSPIRE need? – Development and research requirements.* Joint Research Centre; http://inspire.jrc.it.

Bernard, L., Kanellopoulos, I., Annoni, A., and Smits, P. (2005b): *The European geoportal – one step towards the establishment of a European Spatial Data Infrastructure.* ScienceDirect; Computers, Environment and Urban Systems, 29, p. 15-31.

Bill, R. (2008): *Precise Positioning in ad hoc Geosensor Networks*. EuroSDR and ISPRS GeoSensor Workshop, Hannover, Germany, 20-22 February, 2008.

Bliudze, S. and Sifakis, J. (2007): *The Algebra of Connectors – Structuring Interaction in BIP*. EMSOFT'07, ACM.

Breunig, M., Bär, W., Thomsen, A., Häussler, J., Kipfer, A., Kandawasvika, A., Mäs, S., Reinhardt, W., Wang, F., Brandt, S., Staub, G. and Wiesel, J. (2006): *Advancement of Mobile Geoservices: Potential and Experiences*. In: GEOTECHNOLOGIEN "Science Report" No. 8; Information Systems in Earth Management - From Science to Application, Results from the First Funding Period (2002-2005), ISSN 1619-7399.

Caspary, W. (1992): *Qualitätsmerkmale von Geodaten.* Zeitschrift für Vermessungswesen (ZfV), 7, p. 360-367.

Caspary, W. (1993): *Qualitätsaspekte bei Geo-Informationssystemen.* Zeitschrift für Vermessungswesen (ZfV), 8/9, p. 444-449.

Chiang, K. W. and EL-Sheimy, N. (2004): *Perfomance analysis of a neural network based INS/GPS integration architecture for land vehicle navigation*. Proceedings of the 4th International Symposium on Mobile Mapping Technology, Kunming, China.

Cui, C. K. and Chen, G. (1999): *Kalman Filtering*. 3rd Edition, Springer.

Dantu, R., Abbas, K., O'Neill, M. and Mikler, A. (2004): *Data Centric Modeling of Environmental Sensor Networks*. IEEE Communications Society. Globecom 2004 Workshops.

Dasarathy, B. V. (1997): *Sensor fusion potential exploitation-innovative architectures and illustrative applications*. Proc. IEEE, vol. 85, p. 24–38, January 1997.

Di, L., Krasse, W. and Kobler, B. (2001): *The New ISO TC 211 Standard Project on Sensor and Data Models for Imagery and Gridded Data.* IEEE.

Dict.-US-DoD (2005): *Department of Defense Dictionary of Military and Associated Terms: Spadats - stereographic coverage.*

Dunnicliff, J. (1993): *Geotechnical instrumentation for monitoring field performance*. John Wiley & Sons, Inc. , New York, p. 577.

Durrant-Whyte, H. F. (1998): *Sensor Models and Multisensor Integration.* The International Journal of Robotics Research, vol. 7, no. 6, p. 97-113.

ECE (2005): *Sensor Models: Lecture Notes – ECE /OPTI 531*. Department of Electrical and Computer Engineering (ECE), University of Arizona, Arizona, Fall 2005.

Feldman, M. (2003): *Enterprise Wrappers for Information Assurance*. IEEE Proceedings of the DARPA Information Survivability Conference and Explosition.

Ferretti, A., Prati, C. and Rocca, F. (2001): *Permanent Scatteres in SAR Interferometry*. IEEE Trans. On Geoscience and Remote Sensing, vol. 39, 1.

Fielding, R. (2000): *Architectural Styles and the Design of Network-based Software Architectures*. University of California. Dissertation.

Gebre-Egziabher, D. (2007): *"What is the difference between 'loose', 'tight', 'ultra-tight' and 'deep' integration strategies for INS and GNSS?"* InsideGNSS, p. 28-33, January/February, 2007.

GeoSens-FeMon (2003): *Felsmonitor Winkelgrat: Alarm-und Überwachungssystem zur Sicherung der Kreisstrasse 7145 gegen Felssturz*.

Gili, J. A., Corominas, J. and Rius, J. (2000): *Using Global Positioning System Techniques In Landslide Monitoring*. Engineering Geology 55 (2000) Barcelona, Espanola, 167-192.

Goodall, C., El-Sheimy, N. and Chiang, K. W. (2005): *The development of a GPS/MEMS INS integrated system utilizing a hybrid processing architecture*. ION GNSS 18th International Technical Meeting of the Satellite Division, Long Beach, CA.

Hall, D. L. and Llinas, J. (1997): *An introduction to multi-sensor data fusion*. Proc. IEEE, vol. 85, p. 6–23.

Havlik, D., Bartha, M., and Schimak, G. (2008): *SensorSA Data acquisition System*. Sensing a changing World, Wageningen University and Research Centre, Volume CGI-08-005, Wageningen, p. 14, 2008.

ICSM (1996): *Standards and Practices for Control Surveys.* Inter-Governmental Committee Surveying and Mapping.

ICSM (1996): *Intergovernmental Committee on Surveying & Mapping (ICSM)*. http://www.icsm.gov.au/.

IEEE-1451: *Family of Transducer Interface Standards*. http://ieee1451.nist.gov/.

IETF-RFC-959 (1985): *File Transfer Protocol (FTP)*. http://www.ietf.org/rfc/rfc959.txt.

IETF-RFC-2396 (1998): *Uniform Resource Identifiers (URI): Generic Syntax*. http://www.ietf.org/rfc/rfc2396.txt.

IETF-RFC-2821 (2001): *Simple Mail Transfer Protocol*. http://www.ietf.org/rfc/rfc2821.txt.

ISO-3534-1 (1993): *Statistics -- Vocabulary and symbols -- Part 1: Probability and general statistical terms*.

ISO-8879 (1986): *Information processing - Text and office systems - Standard Generalized Markup Language (SGML)*.

ISO-10303-11 (2004): *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*.

ISO-19101 (2002): *Geographic Information - Reference model*.

ISO-19109 (2005): *Geographic information -- Rules for application schema*.

ISO-19113 (2002): *Geographic information -- Quality principles*.

ISO-19115 (2003): *Geographic information -- Metadata*.

ISO-19136 (2007): *Geographic Information – Geography Markup Language.*

ISO-19757-3 (2006): *Information technology -- Document Schema Definition Languages (DSDL) -- Part 3: Rule-based validation -- Schematron*.

ISO/IEC-7498-1 (1994): *Information technology – Open Systems Interconnection — Basic Reference Model: The Basic Model*. International Organization for Standardization (ISO), Geneva, Switzerland,

ISO/IEC-9075-1 (2008): *Information Technology - Database languages - SQL - Part1: Framework (SQL/Framework)*. July 2008.

ISO/IEC-10746-1 (1998): *Information technology – Open Distributed Processing: Reference Model – Part 1: Overview*.

ISO/IEC-Guide-98-3 (2008): *Uncertainty of measurement -- Part 3: Guide to the expression of uncertainty in measurement (GUM:1995)*.

ISO/IS-10303-25 (2005): *Industrial automation systems and integration -- Product data representation and exchange -- Part 25: Implementation methods: EXPRESS to XMI binding*.

ISO/TC211-19130 *Sensor and data model for imagery and gridded data*. New Work Item Proposal (NWIP).

Jäger, R. (2008): *Deformation Integrity Monitoring for GNSS-Positioning Services by the Karlsruhe Approach (MONIKA) - Concept, realisation and present results.* Geomatika (2). Scientific Proceedings of Riga Technical University,

Jäger, R., Kälber, S. and Oswald, M. (2006): *GNSS/GPS/LPS based Online Control and Alarm System (GOCA) Mathematical Models and Technical Realisation of a System for Natural and Geotechnical Deformation Monitoring and Analysis*. Proceedings to GeoSiberia 2006, Novsibirsk, Russland., 1, p. 32-43,

Jellema, J. (2008): *SWE-connected sensors: Datasource for geo-information*. EuroSDR and ISPRS GeoSensor Workshop, Hannover, Germany, 20-22 February, 2008.

Joos, G. (1994): *Quality Aspects of Geo-Informations*. The fifth European conference on geographical information systems, EGIS, Paris, France.

Joos, G. (2000): *Zur Qualität von objektstrukturierten Geodaten*. Fakultät für Bauingenieur- und Vermessungswesen, Universität der Bundeswehr München. Dissertation. p. 141.

Julier, S. and Uhlmann, K. (2004): *Unscented Filtering and Nonlinear Estimation*. Proceedings of the IEEE, vol. 92, no. 3, p. 401-422.

Kandawasvika, A. and Reinhardt, W. (2005a): *Concept for interoperable usage of multi-sensors within a landslide monitoring application scenario*. In Proceedings of 8th AGILE Conference on Geographic Information Science Estoril, Portugal.

Kandawasvika, A. and Reinhardt, W. (2005b): *Investigation on multi-GeoSensors integration using OGC SensorWeb in a landslide monitoring system*. GI-Days 2005, Muenster, Germany.

Kane, W. F. and Beck, T. J. (1999): *Advances in Slope Instrumentation: TDR and Remote Data Acquisition Systems*. Field Measurements in Geomechanics, 5th International Symposium on Field Measurements in Geomechanics, Singapore, p. 101-105.

Kane, W. F. and Beck, T. J. (2000): *Instrumentation Practice for slope monitoring*. Association of Engineering Geologists.

Kane, W. F., Novotny, C. W., Owen, J. L. T. and Anbessaw, A. (2007): *Automated Slope Stability Monitoring of a Large Unstable Slope*. Association of Environmental & Engineering Geologists, Los Angeles, California.

Kane, W. F., Beck, T. J., Anderson, N. O. and Perez, H. (1996): *Remote Monitoring of Unstable Slopes Using Time Domain Reflectometry*. Proceedings, Eleventh Thematic Conference and Workshops on Applied Geologic Remote Sensing, ERIM, Ann Arbor, MI, 2, p. 431-440.

Krasnjuk, V., Arandjelovic, D., Petrovic, M. and Hribsek, M. (1994): *A CCD Model Applied to Colour Camera Characteristics Measurement*. International Broadcasting Convention, Conference Publication No. 397.

Kresse, W. (2006): *Standardization of Sensor and Data Models*. XXIII FIG Congress, Munich, Germany, October, 2006.

Langegger, A., Wöß, W. and Blöchl, M. (2008): *A Semantic Web Middleware for Virtual Data Integration on the Web*. ISSN 0302-9743. Springler Berlin/Heidelberg.

Leica-GeoMos (2006): *Leica Geosystems Monitoringsysteme: Zuverlässige Lösungen für die Bauwerksüberwachung.*

Leica-GeoMos (2007): *Leica GeoMos v3.0 Release Notes*.

Leica-GeoMos (2008): *Leica GeoMoS: Automatic Monitoring System*.

Lenzerini, M. (2002): *Data Integration: A Theoretical Perspective*. PODS 2002, p. 233-246.

Li, D. and Wang, J. (2006): *Kalman filter design strategies for code tracking loop in Ultra-Tight GPS/INS/PL integration*. ION National Technical Meeting, Monterey, California, 18-20 January, 2006.

Li, Y. and Wang, J. (2006): *Low-cost tightly coupled GPS/INS integration based on a nonlinear Kalman filtering design*. ION National Technical Meeting, Monterey, California, 18-20 January, 2006.

Luo, R. C., Yih, C. and Su, K. L. (2002): *Multisensor Fusion and Integration: Approaches, Applications, and Future Research Directions*. IEEE Sensors Journal, vol. 2, no. 2.

Luo, R. C. and Kay, M. G. (1990): *A tutorial on multisensor integration and fusion.* IEEE Ind. Electron., 1, p. 707-722.

Mantovani, F., Soeters, R. and Van Westen, C. J. (1996): *Remote sensing techniques for landslide studies and hazard zonation in Europe*. Geomorphology, 15(3-4), p. 213-225.

Mäs, S., Reinhardt, W., Kandawasvika, A. and Wang, F. (2005): *Concepts for quality assurance during mobile online data acquisition*. In Proceedings of 8th AGILE Conference on Geographic Information Science Estoril, Portugal.

McCann, R., Kramnik, A., Shen, W., Varadarajan, V. and Sobulo, O. (2005): *Integrating data from disparate sources: a mass collaboration approach*. ICDE 2005. Proceedings of 21st International Conference, p. 487 - 488 5-8 April 2005.

Mikkelsen, P. E. (1996): *Field instrumentation*. In Turner, A. K., and Schuster, R. L. (eds.), Landslides. Investigation and Mitigation, Transportation Research Board, National Academy Press, Washington, DC, Special Report 247, p. 278-316.

Nittel, W. and Stefanidis, A. (2005): *GeoSensor Networks and Virtual GeoReality*. GeoSensor Networks, p. 1-9.

Nittel, W. and Stefanidis, A. (2005): *GeoSensor Networks*. CRC Press.

OGC-O&M-Part1 (2007): *Observations and Measurements - Part 1- Observation Schema.* Open Geospatial Consortium, Inc., OGC 07-022r1.

OGC-O&M-Part2 (2007): *Observations and Measurements - Part 2- Sampling Features.* Open Geospatial Consortium, Inc., OGC 07-002r3.

OGC-SensorML (2007): *OpenGIS® Sensor Model Language (SensorML) Implementation Specification.* Open Geospatial Consortium, Inc., OGC 07-000.

OGC-SWE-ARCH (2006): *OGC Sensor Web Enablement: Overview and High Level Architecture*. OpenGIS White Paper.

Plan, O., Mäs, S., Reinhardt, W., Kandawasvika, A. and Wang, F. (2004): *Konzepte für die mobile Erfassung von Geodaten*. IfGI prints. Geoinformation und Mobilität von der Forschung zur praktischen Anwendung, Münsteraner GI-Tagen p. 97-105,

Pundt, H. (2002): *Field Data Acquisition with Mobile GIS: Dependencies Between Data Quality and Semantics.* GeoInformatica, 6, 4, p. 363-380.

Reinhardt, W., Kandawasvika, A., Mäs, S. and Wang, F. (2005): *Mobile Geospatial Data Collection - concepts and first experiences*. In: Proceedings of the International Cartographic Conference, A Coruna, Spain.

Roddick, J. F., Egenhofer, M. J., Hoel, E., Papadias, D. and Salzberg, B. (2004): *Spatial, Temporal and Spatio-Temporal Databases – Hot Issues and Directions for PhD Research*. SIGMOD Record, vol. 33, no. 2, June 2004.

Rowe, M. (2008): *Test and Measurement World: Software Wrapper Links Matlab to Instruments*. http://www.edn-europe.com. Last Accessed on: 10 July 2008.

Rüeger, J. M. (1996): *Electronic Distance Measurement: An Introduction*. Fourth Edition, Springler-Verlag.

Sailor, M. J. and Link, J. R. (2005): *Smart dust: Nanostructured devices in a grain of sand*. Chemical Communications, vol. 11, p. 1375.

Scaioni, M., Giussani, A., Roncoroni, F., Sgrenzaroli, M. and Vassena, G. (2004): *Monitoring of Geological sites by Laser Scanning Techniques*.

Schimak, G. and Havlik, D. (2009): *Sensors Anywhere – Sensor Web Enablement in Risk Management Applications*. In European Research Centre of Informatics and Mathematics (ERCIM) magazine, no. 76, June, 2009.

Schouppe, M. (2008): *Geosensor networks for Disaster Management - Ongoing Community Research*. EuroSDR and ISPRS GeoSensor Workshop, Hannover, Germany.

Simonis, I. and Echterhoff, J. (2008): *GEOSS AND THE SENSOR WEB*. GEOSS Sensor Web Workshop Report, Geneva, Switzerland.

Sippel, K. (2001): *MODERN MONITORING SYSTEM SOFTWARE DEVELOPMENT*. 10th FIG International Symposium on Deformation Measurements, Orange, California, USA.

Sippel, K. D. and Hill, C. D. (2002): *Modern Deformation Monitoring: A Multi Sensor Approach*. FIG XXII International Congress, Washington, D.C. USA, April 19-26 2002.

Smuda, B. (2003): *Software Wrappers for Rapid Prototyping JAUS-Based Systems*. IEEE Proceedings of the DARPA Information Survivability Conference and Explosition.

Spitznagel, B. and Garlan, D. (2001): *A Compositional Approach for Constructing Connectors*. IEEE.

Stallings, W. R. (1998): *The Origins of OSI*. http://williamstallings.com/Extras/OSI.html. Last Accessed on: 11 January 2008.

Staudinger, M. (2000): *A Cost Oriented Approach to Geodetic Network Optimization*. Institute for Geoinformation, Vienna University of Technology.

Stefanidis, A. (2005): *Geosensor Networks and Spatial Web*. Position paper, National Center for Geographic Information and Analysis, University of Maine.

Tao, V., Liang, S., Croitoru, A., Haider, Z. M. and Wang, C. (2003): *GeoSWIFT: an Open Geospatial Sensing Services for Sensor Web*.

Vashney, P. K. (1997): *Multisensor data fusion.* Electron. Comm. Eng. J.,, vol. 9, 6, p. 245–253.

W3C-SOAP (2007): *SOAP Version 1.2 Part1: Messaging Framework (Second Edition)*. http://www.w3.org/TR/soap12-part1.

W3C-SOAP (2000): *Simple Object Access Protocol (SOAP) 1.1*. http://www.w3.org/TR/soap.

W3C-SVG (2003): *Scalable Vector Graphics (SVG) 1.1 Specification*. http://www.w3.org/TR/SVG/.

Walter, K., Niemeyer, F. and Bill, R. (2008): *Geosensor Web Enablement in Early Warning Systems for Landslides*. EuroSDR and ISPRS GeoSensor Workshop, Hannover, Germany, 20-22 February, 2008.

Wang, F. (2008): *Handling Data Consistency through Spatial Data Integrity Rules in Constraint Decision Tables*. Fakultät für Bauingenieur- und Vermessungswesen, Universität der Bundeswehr München. Dissertation. p. 125.

Wang, J. J., Wang, J., Sinclair, D. and Watts, L. (2006): *Designing a Neural Network for GPS/INS/PL Integration*. IGNSS Symposium, Australia.

Warneke, B., Last, M., Liebowitz, B. and Pister, K. (2001): *Smart Dust. Communicating with a Cubic-Millimeter*. Computer, vol. 34, p. 44-51.

Wendel, J. and Metzger, J. (2005): *A performance comparison of tightly coupled GPS/INS navigation systems based on extended and sigma point Kalman filters*. ION GNSS 18th International Technical Meeting of the Satellite Division, Long Beach, CA, 13-16 September 2005.

Wolf, P. R. and Ghiliani, C. D. (1997): *Adjustment Computations: Statistics and Least Squares in Surveying and GIS.*

Internet Resources:

Altova: http://www.altova.com.

EAA (2004): *European Environment Agency*. http://www.eea.eu.int/main_html.

EU-GMES (1998): *Global Monitoring for Environment and Security*. http://ec.europa.eu/gmes/index_en.htm. Last accessed on 11.11.2008.

Garmin: *Garmin Ltd*. www.garmin.com.

JNI-Specification: *Java Native Interface (JNI) Specification*. http://java.sun.com/j2se/1.4.2/docs/guide/jni. Last accessed on 24.10.2008.

Leica *Leica Geosystems*. http://www.leica-geosystems.com.

LGRB (2002): *LGRB-Nachrichten Nr.8* http://www.lgrb.uni-freiburg.de. Last accessed on 24.07.2007.

Maxim *Maxim Integrated Products* http://www.maxim-ic.com. Last accessed on: 24.01.2008

Microsoft-DCOM *Microsoft's Distributed COM*. http://msdn.microsoft.com/en-us/library/ms809340.aspx. Last accessed on 24.10.2008.

NI-LabVIEW *National Instruments LabVIEW Software*. http://www.ni.com/labview/.

NASA-GSFC-ISC: *NASA Goddard Space Flight Center – Information Systems Center*. http://aaaprod.gsfc.nasa.gov/sensorweb. Last accessed on 09.11.2007.

OMG: *Object Management Group*. http://www.omg.org.

Topcon: *Topcon Europe Positioning*. http://www.topconeurope.com.

US-DOI: *United States of America: Department of Interior*. http://www.doi.gov/nathaz/index.html. Last accessed on 09.11.2007.

W3C-HTTP: *Hypertext Transfer Protocol*. http://www.w3.org/Protocols/.

Wiki *Wikipedia*. http://en.wikipedia.org/wiki/.

11. List of Abbreviations and Acronyms

ANSI	American National Standards Institute
ATIS	Alliance for Telecommunications Industry Solutions. http://www.atis.org.
BPEL	Business Process Execution Language
DCOM	Distributed Component Object Model
EAA	European Environment Agency
EU	European Union
FP	Framework Programme
GEOSS	Global Earth Observation Systems
GLSM	Generic Logical Sensor Model
GMES	Global Monitoring for Environment and Security
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
INS	Inertial Navigation System
INSPIRE	Infrastructure for Spatial Information in Europe
ISC	Information Systems Center
ISO	International Organization for Standardization
IT	Information Technology
JNI	Java Native Interface
JRC	Joint Research Centre
MEMS	Micro-Electro-Mechanical Systems
OASIS	Organization for the Advancement of Structured Information Standards http://www.oasis-open.org/.
OGC	Open Geospatial Consortium, Inc.® http://www.opengeospatial.org.
OGC-SWE	OGC Sensor Web Enablement. http://www.opengeospatial.org/projects/groups/sensorweb.
OMG	Object Management Group. http://www.omg.org/.
OMG-CORBA	OMG's Common Object Request Broker Architecture. http://www.corba.org/.
ORCHESTRA	Open Architecture and Spatial Data Infrastructure for Risk Management
ORNL	Oak Ridge National Laboratory. http://search1.ornl.gov/.
ORNL-SN	Oak Ridge National Laboratory - Sensor Net. http://www.sensornet.gov/.
OSI	Open Systems Interconnection
RMODP	Reference Model for Open Distributed Processing
SACS	Sensor access and control service
SBSM	Standards-based Sensor Model
SOA	Service Oriented Architecture
SQL	Structured Query Language
TC	Technical Committee
TCP	Transmission Control Protocol
TPS	Total Station/Terrestrial Positioning System
W3C	World Wide Web Consortium. http://www.w3.org/.
WLAN	Wireless Local Area Network

12. Appendices

12.1. Open Systems Interconnection (OSI) Reference Model

The model specifies seven layers, **from higher-level to lower-level**, which are Application, Presentation, Session, Transport, Network, Data Link, and Physical. If systems adopt the OSI model, they become interoperable at the following layers.

Layer 7: Application: Provides services to user-defined application processes, but not to the end user. Examples are file transfer applications using File Transfer Protocol (FTP) or web services or applications using Hypertext Transfer Protocol (HTTP) of W3C. The HTTP protocol relies on the Transmission Control Protocol/Internet Protocol (TCP/IP) maintained by the Internet Engineering Task Force (IETF) [http://www.ietf.org/].

Layer 6: Presentation: Establishes a context (based on syntax and semantics, etc.) between application layer entities and also performs mapping between data units. The original presentation layer used the Basic Encoding Rules of Abstract Syntax Notation One (ANS.1) [ISO/IEC and ITU-T, 1984] for e.g. data encryption or serialization of objects and other data structures into and out of eXtensible Markup Language (XML) [W3C] format using the defined XML encoding rules.

Layer 5: Session: Controls the dialogues, connections or sessions between computers. It is commonly used in application environments that use remote procedure calls (RPCs).

Layer 4: Transport: Used for the transmission of data between end users. Examples are the TCP and the User Datagram Protocol (UDP).

Layer 3: Network: Provides services for transferring variable length data sequences from one node (source) to another (destination). A logical address scheme is used to identify the network nodes uniquely. Example is the common Internet Protocol (IP).

Layer 2: Data Link: Provides the functional and procedural means to transfer data between network entities. It is checks the service quality (i.e. presents of errors) of the physical layer. Examples of Data Links are IEEE 802.3 for the wired Local Area Network (LAN) protocol and the IEEE 802.11 for the wireless LAN.

Layer 1: Physical: Defines the electrical and physical specifications for the devices, and also specifies the relationship between a device and a physical medium (e.g. network interface card, cables, radio link, etc.) as well as the network functions. The specifications include information about: layout of pins, voltages, cable specifications (e.g. copper, fiber,…), network adapters, etc. This information should enable the devices to connect with the medium. Examples are the serial protocols (e.g. RS-232, RS-422, RS-485,…), Ethernet 100 Base TX, WLAN 802.11x, Digital Line Subscriber (DSL), etc. It is through the physical layer that the data is moved in and out of the network interface.

12.2. Examples: Generic Logical Sensor Model Instances

12.2.1. TPS Sensor Instance

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mit XMLSpy v2007 sp1 -->
<!--TPS Instance document. Developed by Admire Kandawasvika-->
<!--Date of completion: 31.12.2006-->
<!--Last revision: 24.04.2008-->
<TPSSensor ID="TPS_Leica_TCA1800" xsi:noNamespaceSchemaLocation="tpsGenericModel.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink">
       <!--TPS Metadata -->
       <TPSSensorMetadata>
              <General>
                     <SerialNumber>618755-2</SerialNumber>
                     <ModelName>TCA1800</ModelName>
                     <SensorType>Terrestrial Positioning System (TPS) or Total Station
                     </SensorType>
                     <IntendedApplications>
                            <Application>Engineering Surveying</Application>
                            <Application>Construction projects like tunnels, highways,
                            buildings, waterways, dams, roads etc.
                     </Application>
                            <Application>Terrestrial Positioning</Application>
                            <Application>Structural Monitoring</Application>
                            <Application>Three-Dimensional (3D) Positioning
                            </Application>
                     </IntendedApplications>
                     <GeoPosition>
                            <Location>
                                   <samplingTime>2006-12-31T12:12:01Z
                                   </samplingTime>
                                   <Cartesian>
                                          <northing_or_X>3492570.00</northing_or_X>
                                          <easting_or_Y>5341580.00</easting_or_Y>
                                          <height_or_Z>900.00</height_or_Z>
                                   </Cartesian>
                                   <CoordinateSystem>GK</CoordinateSystem>
                                   <Datum>DHDN</Datum>
                                   <Ellipsoid/>
                            </Location>
                            <Quality>
                                   <stdDevX unit="m">0.006</stdDevX>
                                   <stdDevY unit="m">0.006</stdDevY>
                                   <stdDevZ unit="m">0.01</stdDevZ>
                            </Quality>
                     </GeoPosition>
              </General>
              <Capabilities>
                     <CapsGroup>
                            <MeasurementCapabilities>
                                   <MeasurementDataTypes>
                                          <Item name="Angles" type="AngleType"
                                          xlink:href="#AngleSensor"/>
                                          <Item name="Distances" type="DistanceType"
                                          xlink:href="#EDMSensor"/>
                                          <Item name="Coordinates" type="CoordinateType"/>
                                          <Item name="PointObjects" type="PointDataType"/>
                                          <Item name="PositionData" type="PositionType"/>
                                          <Item name="InclinationData"
                                          type="InclinationDataType"
                                          xlink:href="#InclinationSensor"/>
                                          <Item name="Time" type="DateTime" xlink:href="ISO
                                          8601"/>
                                          <Item name="MeasurementBlocks"
                                          type="MeasurementASCIIBlockType"/>
                                          <Item name="CalibrationInfo"
                                          type="CalibrationInfoType"/>
```

```
                        <Item name="AtmosphericCorrectionData"
                        type="AtmosphericCorrDataType"/>
                    </MeasurementDataTypes>
                    <MeasurementFunctionTypes>
                        <Item name="AngleMeasurement"
                        xlink:href="#AngleSensor">
                            <Resolution name="DisplayLeastCount"
                            unit="sec">
                                <Value>1</Value>
                            </Resolution>
                            <Accuracy standardUsed="ISO 17123-3"
                            unit="sec">
                                <Value>1.5</Value>
                            </Accuracy>
                        </Item>
                        <Item name="DistanceMeasurement (Infared-IR)"
                        xlink:href="#EDMSensor"/>
                        <Item name="DistanceMeasurement (PinPoint
                        Reflectorless-RL)" xlink:href="#EDMSensor"/>
                        <Item name="DistanceMeasurement (Long Range-LR)"
                        xlink:href="#EDMSensor"/>
                        <Item name="Automatic Target Recognition (ATR)"
                        xlink:href="#CCDSensor"/>
                        <Item name="Power Search (PS)"/>
                        <Item name="Electronic Guide Light (EGL)"/>
                        <Item name="Remote Control"/>
                    </MeasurementFunctionTypes>
                </MeasurementCapabilities>
                <Interfaces definition="#TPS_Plugin_Interfaces.xml"/>
                <!--TPS Limitations based on different conditions. -->
                <Limits definition="#ConditionsList.xml">
                    <Item name="Durability">
                        <Condition name="MeasuringTemperature">
                            <Range unit="°C">
                                <Values>-20 +50</Values>
                            </Range>
                        </Condition>
                        <Condition name="StorageTemperature">
                            <Range unit="°C">
                                <Values>-40 +70</Values>
                            </Range>
                        </Condition>
                    <!-- Exposure to water or dust etc.-->
                    </Item>
                </Limits>
        </CapsGroup>
</Capabilities>
<Characteristics>
        <CharGroup>
                <PhysicalProperties>
                    <Property name="width" unit="mm">
                        <Value>245</Value>
                    </Property>
                    <Property name="length" unit="mm">
                        <Value>219</Value>
                    </Property>
                    <Property name="height" unit="mm">
                        <Value>343</Value>
                    </Property>
                    <Property name="weight" unit="kg">
                        <Value>6.6</Value>
                    </Property>
                </PhysicalProperties>
        </CharGroup>
</Characteristics>
<Contacts>
        <Organisation>
                <Name>AGIS, UniBW</Name>
                <Address>
                    <Street>Werner-Heisenberg-Weg 37</Street>
```

```
                                <ZipCode>87755</ZipCode>
                                <City>Munich</City>
                                <State>Bavaria</State>
                                <Country>Germany</Country>
                        </Address>
                        <Person>
                                <FirstName>Admire</FirstName>
                                <LastName>Kandawasvika</LastName>
                                <Position>Research Associate</Position>
                                <Telephone>+49-6004-3873</Telephone>
                                <Email>admire.kandawasvika@unibw.de</Email>
                        </Person>
                        <Homepage>http://www.agis.unibw.de</Homepage>
                </Organisation>
        </Contacts>
</TPSSensorMetadata>
<!--TPS sensors composition -->
<Sensors>
        <EDMSensor ID="EDM_001">
                <EDMMetadata>
                        <General>
                                <SerialNumber>12345-678</SerialNumber>
                                <ModelName>EDM</ModelName>
                                <SensorType>Electronic Distance Measurement</SensorType>
                                <IntendedApplications>
                                        <Application>Distance measurements</Application>
                                </IntendedApplications>
                        </General>
                        <Capabilities>
                                <CapsGroup>
                                        <MeasurementCapabilities>
                                                <MeasurementDataTypes>
                                                        <Item name="Distances"
                                                        type="DistanceType"/>
                                                </MeasurementDataTypes>
                                                <MeasurementFunctionTypes>
                                                        <Item name="DistanceMeasurement">
                                                                <Resolution
name="DisplayLeastCount" measuringMode="EDM_MODE.EDM_SINGLE_STANDARD" unit="mm">
                                                                        <Value>1</Value>
                                                                </Resolution>
                                                                <Accuracy
measuringMode="EDM_MODE.EDM_SINGLE_STANDARD" unit="mm">
                                                                        <Value>2</Value>
                                                                </Accuracy>
                                                        </Item>
                                                </MeasurementFunctionTypes>
                                        </MeasurementCapabilities>
                                </CapsGroup>
                        </Capabilities>
                </EDMMetadata>
                <Parameters>
                        <MeasurementConfig>
                                <Mode>
                                        <MeasureMode type="EDM_SINGLE_STANDARD">
                                                <Frequency unit="Hz">
                                                        <Value>100</Value>
                                                </Frequency>
                                        </MeasureMode>
                                </Mode>
                        </MeasurementConfig>
                </Parameters>
                <Inputs>
                        <Input name="EM_Signal" type="EDMSignalType"/>
                </Inputs>
                <Outputs>
                        <Distances>
                                <SlopeDistance xlink:href="#output_SD.xml"/>
                                <HzDistance xlink:href="#output_HzD.xml"/>
                        </Distances>
```

```
                </Outputs>
        </EDMSensor>
        <AngleSensor>
        <!--Omitted-->
        </AngleSensor>
</Sensors>
<!--TPS components -->
<Components>
        <Telescope>
                <ShortestDistance unit="m">
                        <Value>1.7</Value>
                </ShortestDistance>
                <Magnification>30x</Magnification>
                <ImageSetup>upright</ImageSetup>
                <ObjectiveDiameter unit="mm">
                        <Value>40</Value>
                </ObjectiveDiameter>
                <FOV unit="deg">
                        <Value>1.5</Value>
                </FOV>
                <Transit>Full</Transit>
                <Focusing>coarse</Focusing>
        </Telescope>
        <Compensator ID="Model_1101">
                <Type>Liquid</Type>
                <NumOfAxes>Dual</NumOfAxes>
                <Range unit="gon">
                        <Value>0.07</Value>
                </Range>
                <Accuracy unit="mgon">
                        <Value>0.2</Value>
                </Accuracy>
        </Compensator>
        <Plummet>
                <Magnification/>
                <Position>
                        <Location>
                                <samplingTime>2006-12-31T12:12:02Z</samplingTime>
                                <Cartesian>
                                        <northing_or_X>3492570.00</northing_or_X>
                                        <easting_or_Y>5341580.00</easting_or_Y>
                                        <height_or_Z>901.00</height_or_Z>
                                </Cartesian>
                        </Location>
                </Position>
                <Accuracy unit="mm" type="2sigma">
                        <Value>1.5</Value>
                </Accuracy>
        </Plummet>
        <Battery>
                <Type>Nickel Metal Hydride (NiMH)</Type>
                <Voltage unit="V">
                        <Value>6</Value>
                </Voltage>
                <Capacitance type="GEB121" unit="Ah">
                        <Value>3.6</Value>
                </Capacitance>
                <MountLocation>Internal</MountLocation>
                <PowerSupply unit="V">
                        <Value>12</Value>
                </PowerSupply>
                <EnergyLevel unit="%">
                        <Value>99</Value>
                </EnergyLevel>
        </Battery>
        <ATR/>
        <PowerSearch/>
        <Clock/>
</Components>
```

```
	<!--TPS Parameters e.g. general, measurement specific and calibration information.
	-->
	<Parameters>
		<GeneralConfig>
			<Date>2006-12-31</Date>
			<SysTime>12:12:02Z</SysTime>
			<Beep>HIGH</Beep>
			<Languages>
				<Lang>German</Lang>
				<Lang>English</Lang>
				<Lang>French</Lang>
			</Languages>
			<Units>
				<DistanceUnit>m</DistanceUnit>
				<AngleUnit>gon</AngleUnit>
			</Units>
			<CoordinateSequence Value="East_North_Height"/>
		</GeneralConfig>
		<!--TPS Measurement configuration -->
		<MeasurementConfig>
			<Mode>
				<PointIDMode value="Auto"/>
				<OffsetsMode value="permanent"/>
				<MeasureMode type="EDM_CONT_REFLESS">
				</MeasureMode>
			</Mode>
		</MeasurementConfig>
		<!--TPS Calibration Information -->
		<CalibrationInfo>
			<Time>2000-12-03T11:00:00Z</Time>
			<CompensatorIndexError mode="l" unit="gon">
				<Value>0.0010</Value>
			</CompensatorIndexError>
			<CompensatorIndexError mode="t" unit="gon">
				<Value>0.0023</Value>
			</CompensatorIndexError>
			<VerticalIndexError unit="gon">
				<Value>0.0001</Value>
			</VerticalIndexError>
		</CalibrationInfo>
	</Parameters>
	<!--TPS inputs definition-->
	<Inputs>
		<Input name="TPS_Position" type="SimpleStationType"
definition="#tpsDictionary.xml"/>
		<Input name="TPS_InitialOrientation" type="AngleType"
definition="#tpsDictionary.xml"/>
		<Input name="TargetPointID" type="string" definition="#tpsDictionary.xml"/>
	</Inputs>
	<!--TPS outputs definition-->
	<Outputs>
		<Measurements>
			<MeasurementBlock>
				<Fields>
					<Field name="PointNumber" type="word" index="1"/>
					<Field name="HzCircle" type="word" index="17"/>
					<Field name="VCircle" type="word" index="33"/>
					<Field name="SlopeDistance" type="word" index="49"/>
				</Fields>
				<DataValues dataFile="#measurementsOut1.xml"/>
			</MeasurementBlock>
		</Measurements>
	</Outputs>
	<!--TPS methods definition-->
	<Methods xlink:href="#TPS_OperationMethods.xml"/>
</TPSSensor>
```

12.2.2. GPS Sensor Instance

```
<?xml version="1.0" encoding="UTF-8"?>
<!--GPS Example Instance. -->
<!--Developed by Admire Kandawasvika. -->
<!--Date:23.02.2007 -->
<!--Last Revision:30.08.2008 -->
<GPSReceiver ID="ID_1" xsi:noNamespaceSchemaLocation="gpsGenericModel.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink">
	<!--GPS Metadata section -->
	<gpsSensorMetadata>
		<Contacts>
			<Organisation>
				<Name>AGIS, UniBW</Name>
				<Address>
					<Street>Werner-Heisenberg-Weg 37</Street>
					<ZipCode>87755</ZipCode>
					<City>Munich</City>
					<State>Bavaria</State>
					<Country>Germany</Country>
				</Address>
				<Person>
					<FirstName>Admire</FirstName>
					<LastName>Kandawasvika</LastName>
					<Position>Research Associate</Position>
					<Telephone>+49-6004-3873</Telephone>
					<Email>admire.kandawasvika@unibw.de</Email>
				</Person>
				<Homepage>http://www.agis.unibw.de</Homepage>
			</Organisation>
		</Contacts>
		<General>
			<SerialNumber>CS123-4563</SerialNumber>
			<Chipset>SiRF Star IIe/LP</Chipset>
			<Channels>12</Channels>
			<ModelName>GP600</ModelName>
			<SensorType>Global Positioning System (GPS) Receiver</SensorType>
			<IntendedApplications>
				<Application>Navigation and positioning</Application>
				<Application>Time and Frequency dissemination</Application>
				<Application>Leisure activities</Application>
				<Application>Road and route determination</Application>
				<Application>journey-route planning</Application>
				<Application>Safety applications</Application>
				<Application>Fleet management</Application>
				<Application>Logistics applications</Application>
			</IntendedApplications>
			<!-- Last recorded GPS Position. -->
			<GeoPosition>
				<Location>
					<samplingTime>2005-12-17T09:30:47.0Z</samplingTime>
					<Cartesian>
						<northing_or_X>3492570.00</northing_or_X>
						<easting_or_Y>5341580.00</easting_or_Y>
						<height_or_Z>900.00</height_or_Z>
					</Cartesian>
					<CoordinateSystem>GK</CoordinateSystem>
					<Datum>WGS84</Datum>
					<Ellipsoid>WGS84</Ellipsoid>
				</Location>
				<Quality>
					<EPE unit="m">
						<Value>1</Value>
					</EPE>
					<PDOP>6.2</PDOP>
					<TDOP>2.0</TDOP>
				</Quality>
			</GeoPosition>
```

```
        </General>
        <Characteristics>
                <CharGroup>
                        <PhysicalProperties>
                                <Property name="width" unit="mm">
                                        <Value>54.5</Value>
                                </Property>
                                <Property name="length" unit="mm">
                                        <Value>92.0</Value>
                                </Property>
                                <Property name="height" unit="mm">
                                        <Value>22.2</Value>
                                </Property>
                                <Property name="weight" unit="g">
                                        <Value>180</Value>
                                </Property>
                        </PhysicalProperties>
                </CharGroup>
        </Characteristics>
<!-- GPS Capabilities section. -->
<Capabilities>
                <CapsGroup>
                        <MeasurementCapabilities>
                                <MeasurementDataTypes>
                                        <Item name="Coordinates" type="CoordinateType"/>
                                        <Item name="PointObjects"
                                        type="WayPointDataType"/>
                                        <Item name="ReceiverPositionData"
                                        type="PositionType"/>
                                        <Item name="SVPositionData" type="PositionType"/>
                                        <Item name="Time" type="GPSTime"/>
                                        <Item name="MeasurementBlocks"
                                        type="MeasurementASCIIBlockType"/>
                                        <Item name="CalibrationInfo"
                                        type="CalibrationInfoType"/>
                                        <Item name="OutputMessages" type="NMEA0183Type"/>
                                </MeasurementDataTypes>
                        </MeasurementCapabilities>
                        <Interfaces definition="./GPS_Plugin_Interfaces.xml"/>
                        <!--GPS Limitations based on different conditions like
                environmental or operational {e.g. frequency}-->
                        <Limits definition="./GPS_ConditionsList.xml">
                                <Item name="Durability">
                                        <Condition name="MeasuringTemperature">
                                                <Range unit="°C">
                                                        <Values>-10 +50</Values>
                                                </Range>
                                        </Condition>
                                        <Condition name="Humidity">
                                                <Range unit="%">
                                                        <Values>0 95</Values>
                                                </Range>
                                        </Condition>
                                <!-- Maximum altitude, maximum platform speed, etc.-->
                                </Item>
                        </Limits>
                </CapsGroup>
        </Capabilities>
</gpsSensorMetadata>
<!-- GPS internal components. NB. Not all definitions are included here !!. -->
<Components>
        <Antenna/>
        <LNA/>
        <Mixer/>
        <SAW/>
        <AGC/>
        <DSP/>
        <Clock/>
        <Battery>
                <Type>Rechargeable Li-polymer</Type>
```

```
                    <Voltage unit="V">
                        <Value>3.3</Value>
                    </Voltage>
                    <Capacitance unit="mA">
                        <Value>110</Value>
                    </Capacitance>
                    <MountLocation>Internal</MountLocation>
                    <PowerSupply unit="V">
                        <Value>220</Value>
                    </PowerSupply>
                </Battery>
        </Components>
        <!-- GPS Parameters. -->
        <Parameters>
                <GeneralConfig>
                    <SysTime>14:20:00.0Z</SysTime>
                    <Languages>
                        <Lang>English</Lang>
                        <Lang>German</Lang>
                        <Lang>Chinese</Lang>
                    </Languages>
                    <AntennaHeight unit="m">
                        <Value>3.14</Value>
                    </AntennaHeight>
                </GeneralConfig>
                <!-- Current GPS measurement operating mode. -->
                <MeasurementConfig>
                    <Mode>
                        <Surveying>
                            <PostProcessedStatic definition="./gpsDictionary.xml"/>
                        </Surveying>
                    </Mode>
                    <MessageTypes>
                        <nmeaMessage>GPBOD</nmeaMessage>
                        <nmeaMessage>GPGGA</nmeaMessage>
                        <nmeaMessage>GPGLL</nmeaMessage>
                        <nmeaMessage>GPGSV</nmeaMessage>
                    </MessageTypes>
                </MeasurementConfig>
                <CalibrationInfo>
                    <Time>2007-12-15T15:11:00Z</Time>
                    <ClockData dataFile="{url_location}.clockData.xml"/>
                    <SV_Ephemeris dataFile="{url_location}.ephmerisData.xml"/>
                    <Ionospheric dataFile="{url_location}.ionosphericData.xml"/>
                </CalibrationInfo>
        </Parameters>
        <!-- GPS output definition. -->
        <Outputs>
                <Measurements>
                    <MeasurementBlock blockSeparator="$" fieldSeparator=","
                    decimalPoint=".">
                        <DataValues dataFile="./gpsOut.xml"/>
                    </MeasurementBlock>
                </Measurements>
        </Outputs>
        <!--GPS methods definition-->
        <Methods xlink:href="./GPS_OperationMethods.xml"/>
</GPSReceiver>
```

12.3. Other Sensor XML Documents

12.3.1. TPS Conditions XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Instance document describing the conditions that influences the use or operation of
the total station e.g. TPS TCA 1800. The information provided is only for explanatory
purposes.-->
```

```
<!--- developed by Admire Kandawasvika -->
<!-- last revision: 08.05.2008 -->
<Conditions sensorID="TPS_Leica_TCA1800" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="E:\phardmasy\dvpt\case_study_poc\sensors_msmts_modeling\Se
nsorsGenericModels\tps_24.05.08\tpsConditionsDefinition.xsd">
	<ConditionList classification="Atmospheric">
		<Condition ID="1" name="severe">
			<state>Strong haze, strong sunlight, or severe heat shimmer</state>
			<visibility unit="km" mathOperation="max">5</visibility>
		</Condition>
		<Condition ID="2" name="moderate">
			<state>Light haze, moderate sunlight, or slight heat shimmer</state>
			<visibility unit="km" mathOperation="max">20</visibility>
		</Condition>
		<Condition ID="3" name="normal">
			<state>Overcast, no haze, or no heat shimmer</state>
			<visibility unit="km" mathOperation="max">40</visibility>
		</Condition>
		<Condition ID="4">
			<state>Objects in strong sunlight, severe heat shimmer</state>
		</Condition>
		<Condition ID="5">
			<state>Object in shade, or sky overcast</state>
		</Condition>
		<Condition ID="6">
			<state>Underground, night and twilight</state>
		</Condition>
	</ConditionList>
	<ConditionList classification="Operation">
		<Condition ID="7" name="EDM_SinglePrismObservation">
			<state>EDM observing a single prism under atmospheric influence
			</state>
			<Dependencies>
				<Dependency ref="Atmospheric" ID="1">
					<visibility unit="m" mathOperation="max">1500
					</visibility>
				</Dependency>
				<Dependency ref="Atmospheric" ID="2">
					<visibility unit="m" mathOperation="max">5000
					</visibility>
				</Dependency>
				<Dependency ref="Atmospheric" ID="3">
					<visibility unit="m" mathOperation="greaterThan">5000
					</visibility>
				</Dependency>
			</Dependencies>
		</Condition>
		<Condition ID="8" name="EDM_ThreePrismsObservation">
			<state>EDM observing three prisms under atmospheric influence
			</state>
			<Dependencies>
				<Dependency ref="Atmospheric" ID="1">
					<visibility unit="m" mathOperation="max">2000
					</visibility>
				</Dependency>
				<Dependency ref="Atmospheric" ID="2">
					<visibility unit="m" mathOperation="max">7000
					</visibility>
				</Dependency>
				<Dependency ref="Atmospheric" ID="3">
					<visibility unit="m" mathOperation="greaterThan">9000
					</visibility>
				</Dependency>
			</Dependencies>
		</Condition>
		<Condition ID="9" name="EDM_ReflectorlessWhiteTargetObservation">
			<state>EDM observing no reflector {white target} under atmospheric
			influence
			</state>
```

```
                        <Dependencies>
                                <Dependency ref="Atmospheric" ID="4">
                                        <visibility unit="m" mathOperation="max">140
                                        </visibility>
                                </Dependency>
                                <Dependency ref="Atmospheric" ID="5">
                                        <visibility unit="m" mathOperation="max">170
                                        </visibility>
                                </Dependency>
                                <Dependency ref="Atmospheric" ID="6">
                                        <visibility unit="m" mathOperation="greaterThan">170
                                        </visibility>
                                </Dependency>
                        </Dependencies>
                </Condition>
                <Condition ID="9" name="EDM_ReflectorlessGreyAlbedoObservation">
                        <state>EDM observing no reflector {grey albedo 0.25} under
                        atmospheric influence
                        </state>
                        <Dependencies>
                                <Dependency ref="Atmospheric" ID="4">
                                        <visibility unit="m" mathOperation="max">70
                                        </visibility>
                                </Dependency>
                                <Dependency ref="Atmospheric" ID="5">
                                        <visibility unit="m" mathOperation="max">100
                                        </visibility>
                                </Dependency>
                                <Dependency ref="Atmospheric" ID="6">
                                        <visibility unit="m" mathOperation="greaterThan">100
                                        </visibility>
                                </Dependency>
                        </Dependencies>
                </Condition>
        </ConditionList>
</Conditions>
```

12.3.2. GPS Conditions XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Instance document describing the conditions that influences the use or operation of
the GPS receiver. The information provided is only for explanatory purposes.-->
<!--- developed by Admire Kandawasvika -->
<!-- last revision: 08.05.2008 -->
<Conditions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="E:\phardmasy\dvpt\case_study_poc\sensors_msmts_modeling\Se
nsorsGenericModels\gps_29.05.08\gpsConditionsDefinition.xsd" sensorID="ID_1">
        <ConditionList classification="Environment">
                <Condition ID="1" name="severe">
                        <state>Very dense tall buildings and trees, within very high
                        mountains, intensive multipath
                        </state>
                </Condition>
                <Condition ID="2" name="moderate">
                        <state>Widely spaced buildings and trees, within very low mountains
                        or minimal multipath
                        </state>
                </Condition>
                <Condition ID="3" name="normal">
                        <state>Open or clear surroundings, no signal obstraction, or no or
                        negligible multipath
                        </state>
                </Condition>
        </ConditionList>
</Conditions>
```

12.4. Examples: OGC SensorML Based Sensor Instances

12.4.1. TPS Sensor Instance

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Developed By Admire Kandawasvika-->
<!--Creation Date: 23.08.2007. Based on SensorML Spec. OGC 07-000 released on 17.07.2007.
-->
<!--Last Update: 02.10.2008-->
<sml:SensorML version="1.0.1" xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:swe="http://www.opengis.net/swe/1.0.1" xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:ism="urn:us:gov:ic:ism:v2"
xsi:schemaLocation="http://www.opengis.net/sensorML/1.0.1
E:\kaad_dvpt\ogc_schemas\sensorML\1.0.1\sensorML.xsd">
<sml:member>
<!-- ******************TPS Sensor System************************************** -->
<sml:System gml:id="LeicaTCA1800">
<gml:description>TCA1800 Total Positioning Station of Leica Geosystems
</gml:description>
<gml:name>Leica TCA1800</gml:name>
<!--**************************Keywords*********************************************-->
<sml:keywords>
       <sml:KeywordList codeSpace="urn:agis:def:keyword:AGIS:keywords">
              <sml:keyword>Total Station</sml:keyword>
              <sml:keyword>Terrestrial Surveying Instrument</sml:keyword>
              <sml:keyword>TPS</sml:keyword>
       </sml:KeywordList>
</sml:keywords>
<!-- ***********System Identification Information********************************* -->
<sml:identification>
       <sml:IdentifierList>
              <sml:identifier name="ShortName">
                     <sml:Term definition="urn:ogc:def:identifier:OGC:shortName">
                            <sml:value>Leica TCA1800</sml:value>
                     </sml:Term>
              </sml:identifier>
              <sml:identifier name="ModelName">
                     <sml:Term definition="urn:ogc:def:identifier:OGC:modelName">
                            <sml:value>TCA1800</sml:value>
                     </sml:Term>
              </sml:identifier>
              <sml:identifier name="ManufacturerName">
                     <sml:Term definition="urn:ogc:def:identifier:OGC:manufacturerName">
                            <sml:value>Leica Geosystems</sml:value>
                     </sml:Term>
              </sml:identifier>
              <sml:identifier name="SerialNumber">
                     <sml:Term definition="urn:ogc:def:identifier:OGC:serialNumber">
                            <sml:value>618755-2</sml:value>
                     </sml:Term>
              </sml:identifier>
       </sml:IdentifierList>
</sml:identification>
<!--****************System Classification and Applications*********************-->
<sml:classification>
       <sml:ClassifierList>
              <sml:classifier name="SensorType">
                     <sml:Term definition="urn:agis:def:identifier:AGIS:sensorType">
                            <sml:codeSpace xlink:href="urn:agis:classifier:
                            AGIS:sensorTypes"/>
                            <sml:value>Total Station or Terrestrial Positioning System
                            (TPS)
                            </sml:value>
                     </sml:Term>
              </sml:classifier>
              <sml:classifier name="SensorType">
                     <sml:Term definition="urn:agis:def:identifier:AGIS:sensorType">
                            <sml:codeSpace xlink:href="urn:agis:classifier:
```

```
                        AGIS:sensorTypes"/>
                        <sml:value>Point Acquisition Sensor</sml:value>
                    </sml:Term>
                </sml:classifier>
                <sml:classifier name="Application">
                    <sml:Term definition="urn:ogc:classifier:OGC:application">
                        <sml:value>Engineering Surveying</sml:value>
                    </sml:Term>
                </sml:classifier>
                    <sml:classifier name="Application">
                    <sml:Term definition="urn:ogc:classifier:OGC:application">
                        <sml:value>Three-Dimensional (3D) Positioning</sml:value>
                    </sml:Term>
                </sml:classifier>
                <sml:classifier name="Application">
                    <sml:Term definition="urn:ogc:classifier:OGC:application">
                        <sml:value>Structural Monitoring</sml:value>
                    </sml:Term>
                </sml:classifier>
                <sml:classifier name="Application">
                    <sml:Term definition="urn:ogc:classifier:OGC:application">
                        <sml:value>Construction projects like tunnels, highways,
                        buildings, waterways,dams, roads etc.
                        </sml:value>
                    </sml:Term>
                </sml:classifier>
        </sml:ClassifierList>
</sml:classification>
<!--********************Document Constraints(operational, security,rights)**********-->
<sml:validTime>
        <gml:TimePeriod gml:id="ValidTime">
            <gml:beginPosition>2006-12-15T10:00:00Z</gml:beginPosition>
            <gml:endPosition indeterminatePosition="after"/>
        </gml:TimePeriod>
</sml:validTime>
<sml:securityConstraint>
        <sml:Security ism:classification="NU"/>
</sml:securityConstraint>
<sml:legalConstraint>
        <sml:Rights>
            <sml:documentation xlink:arcrole="urn:ogc:def:role:OGC:liabilities">
                <sml:Document>
                    <gml:description>The author of this document is not liable for
                    any accuracy of the information contained herein.
                    </gml:description>
                </sml:Document>
            </sml:documentation>
        </sml:Rights>
</sml:legalConstraint>
<!--****************** System Physical Characteristics*****************************-->
<sml:characteristics name="Physical Properties">
        <swe:DataRecord definition="urn:ogc:def:property:OGC:physicalProperties">
            <swe:field name="physicalProperties">
                <swe:DataRecord>
                    <swe:field name="width">
                        <swe:Quantity definition="urn:agis:def:property
                        :Manuf:width">
                            <swe:uom code="mm"/>
                            <swe:value>245</swe:value>
                        </swe:Quantity>
                    </swe:field>
                    <swe:field name="length">
                        <swe:Quantitydefinition="urn:agis:def:property:
                        Manuf:length">
                            <swe:uom code="mm"/>
                            <swe:value>219</swe:value>
                        </swe:Quantity>
                    </swe:field>
                    <swe:field name="height">
                        <swe:Quantity definition="urn:manuf:def:property:
```

```
                                    Manuf:height">
                                          <swe:uom code="mm"/>
                                          <swe:value>343</swe:value>
                                    </swe:Quantity>
                              </swe:field>
                              <swe:field name="weight">
                                    <swe:Quantity definition="urn:ogc:def:property
                                    :OGC:weight">
                                          <swe:uom code="kg"/>
                                          <swe:value>6.6</swe:value>
                                    </swe:Quantity>
                              </swe:field>
                        </swe:DataRecord>
                  </swe:field>
            </swe:DataRecord>
      </sml:characteristics>
      <sml:characteristics name="Electrical Requirements">
            <swe:DataRecord gml:id="electricalPower" definition="urn:ogc:def:property
            :OGC:powerRequirement">
                  <swe:field name="Voltage">
                        <swe:QuantityRange definition="urn:ogc:def:property:OGC:voltage">
                              <swe:uom code="V"/>
                              <swe:value>6 12</swe:value>
                        </swe:QuantityRange>
                  </swe:field>
                  <swe:field name="CurrentType">
                        <swe:Category definition="urn:ogc:def:property
                        :OGC:electricalCurrentType">
                              <swe:value>DC</swe:value>
                        </swe:Category>
                  </swe:field>
                  <swe:field name="Amp">
                        <swe:QuantityRange definition="urn:ogc:def:property
                        :OGC:electricalCurrent">
                              <swe:uom code="mA"></swe:uom>
                              <swe:value>20 40</swe:value>
                        </swe:QuantityRange >
                  </swe:field>
            </swe:DataRecord>
      </sml:characteristics>
      <!--********************System Capabilities*****************************************-->
      <sml:capabilities name="MeasurementDataCapabilities">
            <swe:DataRecord gml:id="measurementDataTypes">
                  <swe:field name="Angle">
                        <swe:Quantity/>
                  </swe:field>
                  <swe:field name="Distance">
                        <swe:Quantity/>
                  </swe:field>
                  <swe:field name="Coordinates">
                        <swe:Vector definition="urn:agis:def:dataTypes:CoordinateType1">
                              <swe:coordinate name="time">
                                    <swe:Time/>
                              </swe:coordinate>
                              <swe:coordinate name="X">
                                    <swe:Quantity/>
                              </swe:coordinate>
                              <swe:coordinate name="Y">
                                    <swe:Quantity/>
                              </swe:coordinate>
                              <swe:coordinate name="Z">
                                    <swe:Quantity/>
                              </swe:coordinate>
                        </swe:Vector>
```

```
            </swe:field>
            <swe:field name="PointObject" xlink:href="urn:agis:def:dataTypes
            :PointObject"/>
            <swe:field name="PositionData">
                  <swe:Position/>
            </swe:field>
            <swe:field name="InclinationData" xlink:href="urn:agis:def:dataTypes
            :InclinationDataType"/>
            <swe:field name="Time">
                  <swe:Time/>
            </swe:field>
            <swe:field name="MeasurementsData" xlink:href="urn:agis:def:dataTypes
            :MeasurementsData"/>
            <swe:field name="CalibrationInfo" xlink:href="urn:agis:def:dataTypes
            :CalibrationInfoType"/>
            <swe:field name="AtmosphericCorrectionData"
            xlink:href="urn:agis:def:dataTypes:AGIS:AtmosphericCorrDataType"/>
      </swe:DataRecord>
</sml:capabilities>
<sml:capabilities name="MeasurementFunctionCapabilities">
      <swe:DataRecord gml:id="measurementFunctionTypes">
            <swe:field name="AngleMeasurement" xlink:href="#AngleSensor"/>
                  <swe:field name="AngleMeasurementResolution">
                        <swe:Quantity gml:id="displayLeastCount">
                              <swe:uom code="sec"/>
                              <swe:quality>
                                    <swe:Quantity referenceFrame="ISO17123-3">
                                          <swe:uom code="sec"/>
                                          <swe:value>1.5</swe:value>
                                    </swe:Quantity>
                              </swe:quality>
                              <swe:value>1</swe:value>
                        </swe:Quantity>
                  </swe:field>
            <swe:field name="DistanceMeasurementResolution" xlink:href="#EDMSensor"/>
            <swe:field name="DistanceMeasurement (Infrared-IR)"
            xlink:href="#EDMSensor"/>
            <swe:field name="DistanceMeasurement (PinPoint Reflectorless-RL)"
            xlink:href="#EDMSensor"/>
            <swe:field name="DistanceMeasurement (Long Range-LR)"
            xlink:href="#EDMSensor"/>
            <swe:field name="Automatic Target Recognition" xlink:href="#CCDSensor"/>
            <swe:field name="Electronic Guide Light" xlink:href="#EGL"/>
            <swe:field name="Power Search" xlink:href="PS"/>
            <swe:field name="Remote Controlled" xlink:href="urn:agis:def
            :sensors:RemoteControl"/>
      </swe:DataRecord>
</sml:capabilities>
<sml:capabilities name="OperationalConditions" xlink:href="./ConditionsList.xml">
      <swe:DataRecord gml:id="temperatureConditions">
            <swe:field name="MeasuringTemperature">
                  <swe:QuantityRange definition="urn:ogc:def:property
                  :OGC:temperature">
                        <swe:uom code="degCel"></swe:uom>
                        <swe:value>-20 50</swe:value>
                  </swe:QuantityRange>
            </swe:field>
            <swe:field name="StorageTemperature">
                  <swe:QuantityRange definition="urn:ogc:def:property
                  :OGC:temperature">
                        <swe:uom code="degCel"></swe:uom>
                        <swe:value>-40 70</swe:value>
                  </swe:QuantityRange>
            </swe:field>
            <!-- Exposure to water, dust, etc.-->
      </swe:DataRecord>
</sml:capabilities>
<!--*******Contact Information plus ************************************-->
<sml:contact>
<sml:ContactList gml:id="TCA1800ModelContacts">
```

```
      <gml:description>Contacts information about the TPS Instrument resources
      </gml:description>
      <sml:member xlink:arcrole="urn:ogc:def:property:OGC:author">
            <sml:ResponsibleParty>
                  <sml:individualName>Admire Kandawasvika</sml:individualName>
                  <sml:contactInfo>
                        <sml:phone>
                              <sml:voice>+49 89 6004 3873</sml:voice>
                              <sml:facsimile>+49 89 6004 3906</sml:facsimile>
                        </sml:phone>
                        <sml:address>
                              <sml:deliveryPoint>Werner Heisenberg Weg 39
                              </sml:deliveryPoint>
                              <sml:city>Munich</sml:city>
                              <sml:postalCode>87755</sml:postalCode>
                              <sml:country>Germany</sml:country>
                              <sml:electronicMailAddress>admire.kandawasvika@unibw.de
                              </sml:electronicMailAddress>
                        </sml:address>
                  </sml:contactInfo>
            </sml:ResponsibleParty>
      </sml:member>
</sml:ContactList>
</sml:contact>
<!--**************Other System Resources*********** ******************************-->
<sml:documentation>
      <sml:DocumentList>
            <sml:member name="User Manual"
            xlink:arcrole="urn:ogc:def:property:OGC:userManual">
                  <sml:Document>
                        <gml:description>System User Manual</gml:description>
                              <sml:format>mime/pdf</sml:format>
                                    <sml:onlineResource xlink:href="http://
                                    www.leica-geosystems.com">
                              </sml:onlineResource>
                  </sml:Document>
            </sml:member>
            <sml:member name="Specification Document"
            xlink:arcrole="urn:ogc:def:property:OGC:specificationSheet">
                  <sml:Document>
                        <gml:description>System Technical Specification
                        </gml:description>
                        <sml:format>mime/pdf</sml:format>
                        <sml:onlineResource
                        xlink:href="http://www.leica-geosystems.com">
                        </sml:onlineResource>
                  </sml:Document>
            </sml:member>
            <sml:member name="Product Image"
            xlink:arcrole="urn:ogc:def:property:OGC:objectImage">
                  sml:Document>
                        <gml:description>System Product Image</gml:description>
                              <sml:format>mime/image/jpeg</sml:format>
                              <sml:onlineResource
                               xlink:href="http://www.leica-geosystems.com">
                              </sml:onlineResource>
                  </sml:Document>
            </sml:member>
      </sml:DocumentList>
</sml:documentation>
<sml:history>
      <sml:EventList>
            <sml:member name="deployment"
            xlink:arcrole="urn:ogc:def:property:OGC:deployment">
                  <sml:Event>
                        <sml:date>2007-05-23T09:00:00Z</sml:date>
                        <gml:description>TPS Deployment Event</gml:description>
                              <sml:contact xlink:arcrole="installer"
                              xlink:href="http://myDomain.de/installer.xml">
                        </sml:contact>
```

```
                        <sml:documentation xlink:arcrole="deploymentReport">
                                <sml:Document>
                                        <gml:description>A report about the sensor system
                                        deployment
                                        </gml:description>
                                        <sml:onlineResource
                                        xlink:href="http://myDomain.de/
                                        tpsTCADeployReport.pdf">
                                        </sml:onlineResource>
                                </sml:Document>
                        </sml:documentation>
                </sml:Event>
        </sml:member>
        <sml:member name="calibration"
        xlink:arcrole="urn:ogc:def:property:OGC:calibration">
                <sml:Event>
                        <sml:date>2007-03-11T08:00:00Z</sml:date>
                                <gml:description>TPS Calibration Event
                                </gml:description>
                                <sml:contact xlink:arcrole="manufacturer"/>
                                <sml:documentation xlink:arcrole="calibrationReport">
                                        <sml:Document gml:id="CalibTCA1800-2007-03-11">
                                        <gml:description>Calibration report and data
                                        about the sensor system deployment
                                        </gml:description>
                                        <sml:onlineResource
                                        xlink:href="http://manufDomain.com/
                                        TCACalibReport12345.pdf"/>
                                        </sml:Document>
                                </sml:documentation>
                </sml:Event>
        </sml:member>
    </sml:EventList>
</sml:history>
<!--******************Geospatial Information*************************************-->
<!--******************1. Spatial Reference Frame*********************************-->
<sml:spatialReferenceFrame>
    <gml:EngineeringCRS gml:id="TPS_Sensor_Frame">
        <gml:srsName>TPS Coordinate System Frame</gml:srsName>
        <gml:usesCS>
                <gml:CartesianCS gml:id="TPS_CS">
                        <gml:csName>TPS Cartesian System</gml:csName>
                        <gml:usesAxis>
                                <gml:CoordinateSystemAxis gml:id="X" gml:uom="m">
                                <!--<gml:csName>x coordinate</gml:csName>-->
                                        <gml:datumName>DHDN</gml:datumName>
                                        <gml:axisID>
                                                <gml:csName>X coordinate</gml:csName>
                                        </gml:axisID>
                                        <gml:axisID>
                                                <gml:csName>Northing</gml:csName>
                                        </gml:axisID>
                                        <gml:axisAbbrev>X</gml:axisAbbrev>
                                        <gml:axisDirection>North</gml:axisDirection>
                                </gml:CoordinateSystemAxis>
                        </gml:usesAxis>
                        <gml:usesAxis>
                                <gml:CoordinateSystemAxis gml:id="Y" gml:uom="m">
                                <!--<gml:csName>y coordinate</gml:csName>-->
                                        <gml:datumName>DHDN</gml:datumName>
                                        <gml:axisID>
                                                <gml:csName>Y coordinate</gml:csName>
                                        </gml:axisID>
                                        <gml:axisID>
                                                <gml:csName>Easting</gml:csName>
                                        </gml:axisID>
                                        <gml:axisAbbrev>Y</gml:axisAbbrev>
                                        <gml:axisDirection>East</gml:axisDirection>
                                </gml:CoordinateSystemAxis>
                        </gml:usesAxis>
```

```
                        <gml:usesAxis>
                              <gml:CoordinateSystemAxis gml:id="Z" gml:uom="m">
                              <!--<gml:csName>z coordinate</gml:csName>-->
                                    <gml:datumName>DHDN</gml:datumName>
                                    <gml:axisID>
                                          <gml:csName>Z coordinate</gml:csName>
                                    </gml:axisID>
                                    <gml:axisID>
                                          <gml:csName>Height</gml:csName>
                                    </gml:axisID>
                                    <gml:axisAbbrev>Z</gml:axisAbbrev>
                                    <gml:axisDirection>Up</gml:axisDirection>
                              </gml:CoordinateSystemAxis>
                        </gml:usesAxis>
                  </gml:CartesianCS>
            </gml:usesCS>
            <gml:usesEngineeringDatum/>
      </gml:EngineeringCRS>
</sml:spatialReferenceFrame>
<!--******************2. Current GeoPosition************************************-->
<sml:position name="TPSPosition">
      <swe:Position localFrame="#TPS_Sensor_Frame"
      referenceFrame="urn:ogc:crs:EPSG:31493">
            <swe:time>
                  <swe:Time referenceFrame="#SystemClockTRS">
                  <!--get current time-->
                        <swe:value>now</swe:value>
                  </swe:Time>
            </swe:time>
            <swe:location>
                  <swe:Vector gml:id="TPSLocation"
                  definition="urn:ogc:def:property:OGC:location">
                        <swe:coordinate name="x coordinate">
                              <swe:Quantity axisID="X">
                                    <swe:uom code="m"/>
                                    <swe:quality>
                                          <swe:Quantity gml:id="standardDeviationX">
                                                <swe:uom code="m"/>
                                                <swe:value>0.006</swe:value>
                                          </swe:Quantity>
                                    </swe:quality>
                                    <swe:value>3492570.00</swe:value>
                              </swe:Quantity>
                        </swe:coordinate>
                        <swe:coordinate name="y coordinate">
                              <swe:Quantity axisID="Y">
                                    <swe:uom code="m"/>
                                    <swe:quality>
                                          <swe:Quantity gml:id="standardDeviationY">
                                                <swe:uom code="m"/>
                                                <swe:value>0.006</swe:value>
                                          </swe:Quantity>
                                    </swe:quality>
                                    <swe:value>5341580.00</swe:value>
                              </swe:Quantity>
                        </swe:coordinate>
                        <swe:coordinate name="z coordinate">
                              <swe:Quantity axisID="Z">
                                    <swe:uom code="m"/>
                                    <swe:quality>
                                          <swe:Quantity gml:id="standardDeviationZ">
                                          <swe:uom code="m"/>
                                          <swe:value>0.001</swe:value>
                                          </swe:Quantity>
                                    </swe:quality>
                                    <swe:value>900.00</swe:value>
                              </swe:Quantity>
                        </swe:coordinate>
                  </swe:Vector>
            </swe:location>
```

```
        </swe:Position>
</sml:position>
<!--**********************System Interface Definition*******************************-->
<sml:interfaces>
        <sml:InterfaceList>
                <sml:interface name="TPS_PlugIn_Interfaces"
                xlink:href="http://myDomain.de/TPS_Plugin_Interfaces.xml">
                </sml:interface>
                <sml:interface name="RS-232">
                        <sml:InterfaceDefinition>
                        <!-- OSI based Interface Definition-->
                                <sml:physicalLayer>
                                        <swe:DataRecord definition="urn:ogc:def:property
                                        :OGC:RS232">
                                                <swe:field name="dataBits">
                                                        <swe:Count
                                                        definition="urn:ogc:def:property:OGC
                                                        :numberOfDataBits">
                                                                <swe:value>8</swe:value>
                                                        </swe:Count>
                                                </swe:field>
                                                <swe:field name="stopBits">
                                                        <swe:Count
                                                        definition="urn:ogc:def:property:OGC
                                                        :numberOfStopBits">
                                                                <swe:value>1</swe:value>
                                                        </swe:Count>
                                                </swe:field>
                                                <swe:field name="parity">
                                                        <swe:Boolean
                                                        definition="urn:ogc:def:property:OGC
                                                        :parity">
                                                                <swe:value>false</swe:value>
                                                        </swe:Boolean>
                                                </swe:field>
                                        </swe:DataRecord>
                                </sml:physicalLayer>
                                <sml:mechanicalLayer>
                                        <swe:DataRecord definition="urn:ogc:def:property:OGC
                                        :DB9">
                                                <swe:field name="pin Layout">
                                                        <swe:Category definition="urn:ogc:def:
                                                        property:OGC:pinout">
                                                                <swe:value>EIA574</swe:value>
                                                        </swe:Category>
                                                </swe:field>
                                                <swe:field name="signalPaths">
                                                        <swe:Text>
                                                                <swe:value>RxD TxD GND</swe:value>
                                                        </swe:Text>
                                                </swe:field>
                                        </swe:DataRecord>
                                </sml:mechanicalLayer>
                        </sml:InterfaceDefinition>
                </sml:interface>
        </sml:InterfaceList>
</sml:interfaces>
<!--**********************System Inputs*****************************************-->
<sml:inputs>
        <!--Can define data types or observable properties here.-->
</sml:inputs>
<!--**********************System Outputs****************************************-->
<sml:outputs>
        <sml:OutputList>
                <sml:output name="MeasurementsData" xlink:href="./measurementsOut01.xml">
                        <swe:DataArray>
                                <swe:elementCount>
                                        <swe:Count gml:id="numOfDataRecords" fixed="false">
                                                <swe:value>10000</swe:value>
                                        </swe:Count>
```

```
                        </swe:elementCount>
                        <swe:elementType name="dataBlock">
                              <swe:DataRecord>
                                    <gml:description>Following is the data definition
                                    of the block content.
                                    </gml:description>
                                    <swe:field name="Time">
                                          <swe:Time/>
                                    </swe:field>
                                    <swe:field name="PointNumber">
                                          <swe:Count/>
                                    </swe:field>
                                    <swe:field name="Hz">
                                          <swe:Quantity gml:id="horizontalAngle">
                                          <swe:uom code="deg"></swe:uom>
                                          </swe:Quantity>
                                    </swe:field>
                                    <swe:field name="V">
                                          <swe:Quantity gml:id="verticalAngle">
                                          <swe:uom code="deg"></swe:uom>
                                          </swe:Quantity>
                                    </swe:field>
                                    <swe:field name="Dist">
                                          <swe:Quantity gml:id="slopeDistance">
                                          <swe:uom code="m"></swe:uom>
                                          </swe:Quantity>
                                    </swe:field>
                                    <swe:field name="X">
                                          <swe:Quantity gml:id="x">
                                          <swe:uom code="m"></swe:uom>
                                          </swe:Quantity>
                                    </swe:field>
                                    <swe:field name="Y">
                                          <swe:Quantity gml:id="y">
                                          <swe:uom code="m"></swe:uom>
                                          </swe:Quantity>
                                    </swe:field>
                                    <swe:field name="Z">
                                          <swe:Quantity gml:id="z">
                                          <swe:uom code="m"></swe:uom>
                                          </swe:Quantity>
                                    </swe:field>
                              </swe:DataRecord>
                        </swe:elementType>
                  <swe:encoding>
                        <swe:TextBlock decimalSeparator="." blockSeparator="$"
                        tokenSeparator=","></swe:TextBlock>
                  </swe:encoding>
                  </swe:DataArray>
            </sml:output>
      </sml:OutputList>
</sml:outputs>
<!--***************TPS Parameters (Taskable)**********************************-->
<sml:parameters>
      <sml:ParameterList>
      <!--sets system communication language-->
            <sml:parameter name="SystemLanguage">
                  <swe:SimpleDataRecord>
                        <swe:field name="defaultLanguage">
                              <swe:Text><swe:value>English</swe:value></swe:Text>
                        </swe:field>
                        <swe:field name="Language_2">
                              <swe:Text><swe:value>German</swe:value></swe:Text>
                        </swe:field>
                        <swe:field name="Language_3">
                              <swe:Text><swe:value>French</swe:value></swe:Text>
                        </swe:field>
                  </swe:SimpleDataRecord>
            </sml:parameter>
<!--Configures units of measure-->
```

```
            <sml:parameter name="UnitsOfMeasure">
                  <swe:SimpleDataRecord>
                        <swe:field name="DistanceUnit">
                              <swe:Text/>
                        </swe:field>
                        <swe:field name="AngleUnit">
                              <swe:Text/>
                        </swe:field>
                  </swe:SimpleDataRecord>
            </sml:parameter>
<!--Sets system time based on user's local time-->
            <sml:parameter name="sysTime">
                  <swe:Time/>
            </sml:parameter>
<!--Sets TPS Point Coding Mode-->
            <sml:parameter name="pointIDLogModeAuto">
                  <swe:Boolean>
                        <gml:description>PointID logging mode. Automatic or Manual.
                        </gml:description>
                        <swe:value>true</swe:value>
                  </swe:Boolean>
            </sml:parameter>
<!--Configures EDM distance measuring mode-->
            <sml:parameter name="EDM_Measure_Mode">
                  <swe:Category>
                        <swe:constraint>
                              <swe:AllowedTokens>
                                    <swe:valueList>EDM_SINGLE_STANDARD
EDM_SINGLE_FAST EDM_CONT_STANDARD EDM_CONT_TRACKING EDM_CONT_REFLESS EDM_AVERAGE_IR
EDM_AVERAGE_SR EDM_AVERAGE_LR
                                    </swe:valueList>
                              </swe:AllowedTokens>
                        </swe:constraint>
                  </swe:Category>
            </sml:parameter>
<!--Sets the inclination sensor running mode-->
            <sml:parameter name="InclinationProgram">
                  <swe:Category>
                        <swe:constraint>
                              <swe:AllowedTokens>
                                    <swe:valueList>MeasureInclination
                                    ComputeInclination Auto
                                    </swe:valueList>
                              </swe:AllowedTokens>
                        </swe:constraint>
                  </swe:Category>
            </sml:parameter>
<!--Configures system's measure program mode-->
            <sml:parameter name="MeasureProgMode">
                  <swe:Category>
                        <swe:constraint>
                              <swe:AllowedTokens>
                                    <swe:valueList>NoMeasurements AnglesOnly
DefaultDistMeasurement TrackDistPlusAngles ClearDistances StopTracking
TrackDistWithRedLaserOn
                                    </swe:valueList>
                              </swe:AllowedTokens>
                        </swe:constraint>
                  </swe:Category>
            </sml:parameter>
<!--Sets system's calibration data (user or manufacturer)-->
            <sml:parameter name="CalibrationData">
                  <swe:SimpleDataRecord>
                        <swe:field name="CompensatorIndexError_L">
                              <swe:Quantity
                              definition="urn:manuf:def:calibration:IndexError">
                              <gml:description>longitudinal axis error. Important for
                              tilt determination.
                              </gml:description>
                                    <swe:uom code="gon"></swe:uom>
```

```
                                        <swe:value>0.0010</swe:value>
                                </swe:Quantity>
                        </swe:field>
                        <swe:field name="CompensatorIndexError_T">
                                <swe:Quantity
                                definition="urn:manuf:def:calibration:IndexError">
                                <gml:description>Transverse axis error. Important for
                                tilt determination.
                                </gml:description>
                                        <swe:uom code="gon"></swe:uom>
                                        <swe:value>0.0023</swe:value>
                                </swe:Quantity>
                        </swe:field>
                        <swe:field name="VerticalIndexError_T">
                                <swe:Quantity
                                definition="urn:manuf:def:calibration:IndexError">
                                <gml:description>Index error of the vertical circle.
                                Vertical angles should be corrected for this.
                                </gml:description>
                                        <swe:uom code="gon"></swe:uom>
                                        <swe:value>0.0001</swe:value>
                                </swe:Quantity>
                        </swe:field>
<!--Horizontal Collimation Error-->
<!--Tilting-axis error-->
                        </swe:SimpleDataRecord>
                </sml:parameter>
        </sml:ParameterList>
</sml:parameters>
<!--**********************TPS Internal Components**********************************-->
<sml:components>
        <sml:ComponentList>
                <sml:component name="EDMSensor"
                xlink:arcrole="urn:ogc:def:process:OGC:sensor"
                xlink:href="urn:agis:def:sensors:AGIS:EDMSensor">
                </sml:component>
                <sml:component name="AngleSensor"
                xlink:arcrole="urn:ogc:def:process:OGC:sensor"
                xlink:href="urn:agis:def:sensors:AGIS:AngleSensor">
                </sml:component>
                <sml:component name="Telescope"
                xlink:arcrole="urn:ogc:def:process:OGC:detector"
                xlink:href="urn:agis:def:sensors:AGIS:Telescope">
                </sml:component>
                <sml:component name="LiquidCompensator"
                xlink:arcrole="urn:ogc:def:process:OGC:detector"
                xlink:href="urn:agis:def:sensors:AGIS:LiquidCompensator">
                </sml:component>
                <sml:component name="OpticalPlummet"
                xlink:arcrole="urn:ogc:def:process:OGC:detector"
                xlink:href="urn:agis:def:sensors:AGIS:OpticalPlummet">
                </sml:component>
                <sml:component name="CCDChip"
                xlink:arcrole="urn:ogc:def:process:OGC:detector"
                xlink:href="urn:agis:def:sensors:AGIS:CCDCamera">
                </sml:component>
                <sml:component name="Clock" xlink:arcrole="urn:ogc:def:process:OGC:detector"
                xlink:href="urn:agis:def:sensors:AGIS:TPSClock">
                </sml:component>
                <sml:component name="PowerSearch"
                xlink:arcrole="urn:ogc:def:process:OGC:detector"
                xlink:href="urn:agis:def:sensors:AGIS:TPSPowerSearch">
                </sml:component>
                <sml:component name="Battery"
                xlink:arcrole="urn:agis:def:process:AGIS:component"
                xlink:href="urn:agis:def:sensors:AGIS:TPSBattery">
                </sml:component>
        </sml:ComponentList>
</sml:components>
</sml:System>
```

```
</sml:member>
<!--**********System Process Model************************************-->
<sml:member>
       <sml:ProcessModel>
              <gml:description>Example Process Model of TPS.</gml:description>
                     <sml:method>
                            <sml:ProcessMethod>
                                   <!--Rules Set-->
                                   <sml:rules>
                                          <sml:RulesDefinition></sml:RulesDefinition>
                                   </sml:rules>
                                   <!--Implementations-->
                                   <sml:implementation
                            xlink:href="http://myDomain.de/TPS_Plugin_Interfaces.xml">
                                   <sml:ImplementationCode language="java"
                                   framework="AGISSensorPluginsFramework">
                                                 <sml:sourceRef></sml:sourceRef>
                                                 <sml:binaryRef
                                          xlink:arcrole="GeotechSoftware">
                                                 </sml:binaryRef>
                                          </sml:ImplementationCode>
                                   </sml:implementation>
                            </sml:ProcessMethod>
                     </sml:method>
              </sml:ProcessModel>
       </sml:member>
</sml:SensorML>
<!--*******************************END*****************************************-->
```

12.4.2. TDR Sensor Instance

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Developed By Admire Kandawasvika-->
<!--Creation Date: 18.08.2008. Based on SensorML Spec. OGC 07-000 released on 17.07.2007.
-->
<!--Last Update: 03.10.2008-->
<sml:SensorML version="1.0.1" xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:swe="http://www.opengis.net/swe/1.0.1" xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:ism="urn:us:gov:ic:ism:v2"
xsi:schemaLocation="http://www.opengis.net/sensorML/1.0.1
E:\kaad_dvpt\ogc_schemas\sensorML\1.0.1\sensorML.xsd">
<sml:member>
<!-- ******************Time Domain Interferometer****************************** -->
<sml:System gml:id="WavetelTDR-44">
       <gml:description>Handheld Time Domain Reflectometer (TDR) 44. The device can be
       used to locate faults and other cable impedances.
       </gml:description>
       <gml:name>Wavetel TDR 44</gml:name>
<!--*******************************Keywords*****************************************-->
<sml:keywords>
       <sml:KeywordList codeSpace="urn:agis:def:keyword:AGIS:keywords">
              <sml:keyword>Extensometer</sml:keyword>
              <sml:keyword>Movement Detector</sml:keyword>
              <sml:keyword>Time Domain Reflectometer</sml:keyword>
       </sml:KeywordList>
</sml:keywords>
<!-- ****************System Identification Information************************** -->
<sml:identification>
       <sml:IdentifierList>
              <sml:identifier name="ShortName">
                     <sml:Term definition="urn:ogc:def:identifier:OGC:shortName">
                            <sml:value>Wavetel TDR-44</sml:value>
                     </sml:Term>
              </sml:identifier>
              <sml:identifier name="ModelName">
                     <sml:Term definition="urn:ogc:def:identifier:OGC:modelName">
                            <sml:value>44</sml:value>
```

```
                </sml:Term>
            </sml:identifier>
            <sml:identifier name="ManufacturerName">
                <sml:Term definition="urn:ogc:def:identifier:OGC:manufacturerName">
                    <sml:value>Wavetel</sml:value>
                </sml:Term>
            </sml:identifier>
        </sml:IdentifierList>
</sml:identification>
<!--******************System Classification*************************************-->
<sml:classification>
        <sml:ClassifierList>
            <sml:classifier name="SensorType">
                <sml:Term definition="urn:agis:def:identifier:AGIS:sensorType">
                    <sml:codeSpace
                    xlink:href="urn:agis:classifier:AGIS:sensorTypes"/>
                    <sml:value>Extensometer</sml:value>
                </sml:Term>
            </sml:classifier>
            <sml:classifier name="SensorType">
                <sml:Term definition="urn:agis:def:identifier:AGIS:sensorType">
                    <sml:codeSpace
                    xlink:href="urn:agis:classifier:AGIS:sensorTypes"/>
                    <sml:value>Distance Measurement Sensor</sml:value>
                </sml:Term>
            </sml:classifier>
            <sml:classifier name="Application">
                <sml:Term definition="urn:ogc:classifier:OGC:application">
                    <sml:value>Impedence or Resistance Measurements</sml:value>
                </sml:Term>
            </sml:classifier>
            <sml:classifier name="Application">
                <sml:Term definition="urn:ogc:classifier:OGC:application">
                    <sml:value>Cable fault, breaks detections</sml:value>
                </sml:Term>
            </sml:classifier>
            <sml:classifier name="Application">
                <sml:Term definition="urn:ogc:classifier:OGC:application">
                    <sml:value>Monitoring of Boreholes</sml:value>
                </sml:Term>
            </sml:classifier>
            <sml:classifier name="Application">
                <sml:Term definition="urn:ogc:classifier:OGC:application">
                    <sml:value>Displacements Detection</sml:value>
                </sml:Term>
            </sml:classifier>
        </sml:ClassifierList>
</sml:classification>
<!--*****************Document Constraints(operational, security,rights)*************-->
<sml:validTime>
        <gml:TimePeriod gml:id="ValidTime">
            <gml:beginPosition>2008-18-17T10:00:00Z</gml:beginPosition>
            <gml:endPosition indeterminatePosition="after"/>
        </gml:TimePeriod>
</sml:validTime>
<sml:securityConstraint>
        <...>
</sml:securityConstraint>
<!--***********System Physical Characteristics***********************************-->
<sml:characteristics name="Physical Properties">
        <swe:DataRecord definition="urn:ogc:def:property:OGC:physicalProperties">
            <swe:field name="physicalProperties">
                <swe:DataRecord>
                    <swe:field name="caseWidth">
                        <swe:Quantity
                        definition="urn:agis:def:property:Manuf:width">
                            <swe:uom code="mm"/>
                            <swe:value>50</swe:value>
                        </swe:Quantity>
                    </swe:field>
```

```
                        <swe:field name="caseLength">
                                <swe:Quantity
                                definition="urn:agis:def:property:Manuf:length">
                                        <swe:uom code="mm"/>
                                        <swe:value>100</swe:value>
                                </swe:Quantity>
                        </swe:field>
                        <swe:field name="caseHeight">
                                <swe:Quantity
                                definition="urn:manuf:def:property:Manuf:height">
                                        <swe:uom code="mm"/>
                                        <swe:value>210</swe:value>
                                </swe:Quantity>
                        </swe:field>
                        <swe:field name="weight">
                                <swe:Quantity
                                definition="urn:ogc:def:property:OGC:weight">
                                        <swe:uom code="g"/>
                                        <swe:value>550</swe:value>
                                </swe:Quantity>
                        </swe:field>
                        <swe:field name="caseMaterial">
                                <swe:Text
                                definition="urn:ogc:def:property:OGC:material">
                                        <swe:value>Fiber Glass</swe:value>
                                </swe:Text>
                        </swe:field>
                        <swe:field name="Leads">
                                <swe:Text
                                definition="urn:manuf:def:property:Manuf:leads">
                                        <swe:value>2m safety plug-clip</swe:value>
                                </swe:Text>
                        </swe:field>
                </swe:DataRecord>
        </swe:field>
    </swe:DataRecord>
</sml:characteristics>
<sml:characteristics name="Electrical Requirements">
    <swe:DataRecord gml:id="electricalPower"
    definition="urn:ogc:def:property:OGC:powerRequirement">
        <swe:field name="Voltage">
            <swe:Quantity definition="urn:ogc:def:property:OGC:voltage">
                <swe:uom code="V"/>
                <swe:value>6.5</swe:value>
            </swe:Quantity>
        </swe:field>
        <swe:field name="CurrentType">
            <swe:Category
            definition="urn:ogc:def:property:OGC:electricalCurrentType">
                <swe:value>DC</swe:value>
            </swe:Category>
        </swe:field>
    </swe:DataRecord>
</sml:characteristics>
<!--********************System Capabilities*****************************************-->
<sml:capabilities name="MeasurementCapabilities">
    <swe:DataRecord gml:id="measurementCapabilities">
        <swe:field name="PVF">
            <swe:QuantityRange>
                <gml:description>Variable Propagation Velocity Factor (PVF).
                </gml:description>
                <swe:value>0.01 0.99</swe:value>
            </swe:QuantityRange>
        </swe:field>
        <swe:field name="MeasureRange">
            <swe:QuantityRange>
                <gml:description>Distance measurement range.
                </gml:description>
                <swe:value>0 3000</swe:value>
            </swe:QuantityRange>
```

```
            </swe:field>
            <swe:field name="DistanceAccuracy">
                  <swe:ConditionalValue>
                        <swe:condition name="MeasureRange">
                              <swe:QuantityRange>
                                    <swe:uom code="m"></swe:uom>
                                    <swe:value>0 3000</swe:value>
                              </swe:QuantityRange>
                        </swe:condition>
                        <swe:data>
                              <swe:Quantity>
                                    <gml:description>Distance accuracy is 0.9 % of
                                    the measured range.
                                    </gml:description>
                                    <swe:uom code="%"></swe:uom>
                                    <swe:value>0.9</swe:value>
                              </swe:Quantity>
                        </swe:data>
                  </swe:ConditionalValue>
            </swe:field>
            <swe:field name="MeasureResolution">
                  <swe:ConditionalValue>
                        <swe:condition name="MeasureRange">
                              <swe:QuantityRange>
                                    <swe:uom code="m"></swe:uom>
                                    <swe:value>0 3000</swe:value>
                              </swe:QuantityRange>
                        </swe:condition>
                        <swe:data>
                              <swe:Quantity>
                                    <gml:description>Measure resolution is 1 % of the
                                    measured range.
                                    </gml:description>
                                    <swe:uom code="%"></swe:uom>
                                    <swe:value>1</swe:value>
                              </swe:Quantity>
                        </swe:data>
                  </swe:ConditionalValue>
            </swe:field>
            <swe:field name="Gain">
                  <swe:Category>
                        <swe:value>AutoSet</swe:value>
                  </swe:Category>
            </swe:field>
            <swe:field name="OutputImpedance">
                  <swe:Quantity>
                  <gml:description>maximum output resistence or impendance
                  </gml:description>
                        <swe:uom code="ohm"></swe:uom>
                        <swe:value>120</swe:value>
                  </swe:Quantity>
            </swe:field>
            <swe:field name="OutputUpdateRate">
                  <swe:Quantity>
                        <swe:uom code="{measure}/s"></swe:uom>
                        <swe:value>1</swe:value>
                  </swe:Quantity>
            </swe:field>
      </swe:DataRecord>
</sml:capabilities>
<sml:capabilities name="OperationalConditions" xlink:href="./ConditionsList.xml">
      <swe:DataRecord gml:id="Environmental">
            <swe:field name="MeasuringTemperature">
                  <swe:QuantityRange definition="urn:ogc:def:property:OGC:temperature">
                        <swe:uom code="degCel"></swe:uom>
                        <swe:value>-20 60</swe:value>
                  </swe:QuantityRange>
            </swe:field>
            <swe:field name="StorageTemperature">
                  <swe:QuantityRange definition="urn:ogc:def:property:OGC:temperature">
```

```
                        <swe:uom code="degCel"></swe:uom>
                        <swe:value>-30 70</swe:value>
                    </swe:QuantityRange>
                </swe:field>
                <swe:field name="Humidity">
                    <swe:Quantity definition="urn:ogc:def:property:OGC:temperature">
                        <swe:uom code="%"></swe:uom>
                        <swe:value>93</swe:value>
                    </swe:Quantity>
                </swe:field>
        </swe:DataRecord>
</sml:capabilities>
<!--************************Contact Information***********************************-->
<sml:contact>
        <sml:ContactList gml:id="TDR-44Vendor">
                <gml:description>Contacts information about the Instrument resources
                </gml:description>
                <sml:member xlink:arcrole="urn:ogc:def:property:OGC:vendor">
                    <sml:ResponsibleParty>
                        <sml:organizationName>Wavetel</sml:organizationName>
                        <sml:contactInfo>
                            <sml:phone>
                            <sml:voice>+33 297 35 36 12</sml:voice>
                            <sml:facsimile>+33 297 35 36 13</sml:facsimile>
                            </sml:phone>
                            <sml:address>
                                <sml:deliveryPoint>Espace du Ter - 13 Boulevard
                                Jean MONNET
                                </sml:deliveryPoint>
                                <sml:city>LARMOR PLAGE</sml:city>
                                <sml:postalCode>56260</sml:postalCode>
                                <sml:country>France</sml:country>
                                <sml:electronicMailAddress>info@wavetel.fr
                                </sml:electronicMailAddress>
                            </sml:address>
                        </sml:contactInfo>
                    </sml:ResponsibleParty>
                </sml:member>
        </sml:ContactList>
</sml:contact>
<!--******************Other System Resources*************************************-->
<sml:documentation>
        <sml:DocumentList>
                <sml:member name="User Manual"
                xlink:arcrole="urn:ogc:def:property:OGC:userManual">
                    <sml:Document>
                        <gml:description>System User Manual</gml:description>
                        <sml:format>mime/pdf</sml:format>
                    </sml:Document>
                </sml:member>
                <sml:member name="Specification Document"
                xlink:arcrole="urn:ogc:def:property:OGC:specificationSheet">
                    <sml:Document>
                    <gml:description>System Technical Specification</gml:description>
                        <sml:format>mime/pdf</sml:format>
                    </sml:Document>
                </sml:member>
        </sml:DocumentList>
</sml:documentation>
<sml:history>
        <sml:EventList>
                <sml:member name="deployment"
                xlink:arcrole="urn:ogc:def:property:OGC:deployment">
                    <sml:Event>
                        <sml:date>unknown</sml:date>
                            <gml:description>TDR Deployment Event</gml:description>
                            <sml:contact xlink:arcrole="installer"
                            xlink:href="http://myDomain.de/installer.xml">
                            </sml:contact>
                            <sml:documentation xlink:arcrole="deploymentReport">
```

```
                                        <sml:Document>
                                        <gml:description>A report about the sensor system
                                        deployment
                                        </gml:description>
                                        <sml:onlineResource
                                  xlink:href="http://myDomain.de/TDR44DeployReport.pdf">
                                        </sml:onlineResource>
                                        </sml:Document>
                                  </sml:documentation>
                        </sml:Event>
                  </sml:member>
                  <sml:member name="calibration"
                  xlink:arcrole="urn:ogc:def:property:OGC:calibration">
                        <sml:Event>
                              <sml:date>unknown</sml:date>
                              <gml:description>TPS Calibration Event</gml:description>
                                    <sml:contact xlink:arcrole="manufacturer"/>
                                          <sml:documentation
                                          xlink:arcrole="calibrationReport">
                                                <sml:Document gml:id="CalibTDR_unkown">
                                                <gml:description>Calibration report and
                                                data about the sensor system deployment
                                                </gml:description>
                                                <sml:onlineResource
                                    xlink:href="http://manufDomain.com/TDRCalibReport12345.pdf"/>
                                                </sml:Document>
                                          </sml:documentation>
                        </sml:Event>
                  </sml:member>
            </sml:EventList>
      </sml:history>
      <!--******************Geospatial Information*************************************-->
      <!--**************************1. Spatial Reference Frame**************************-->
      <sml:spatialReferenceFrame>
            <...>
      </sml:spatialReferenceFrame>
      <!--*********************************2. Current GeoPosition************************-->
      <sml:position name="TDRPosition">
            <swe:Position localFrame="#TDR_Sensor_Frame" referenceFrame="TPS_Sensor_Frame">
                  <swe:time>
                        <swe:Time referenceFrame="#DataLoggerClockTRS">
                              <!--get current time-->
                              <swe:value>now</swe:value>
                        </swe:Time>
                  </swe:time>
                  <swe:location>
                        <swe:Vector gml:id="TDRLocation"
                        definition="urn:ogc:def:property:OGC:location">
                              <...>
                        </swe:Vector>
                  </swe:location>
            </swe:Position>
      </sml:position>
      <!--********************System Interface Definition*******************************-->
      <sml:interfaces>
            <sml:InterfaceList>
                  <sml:interface name="TDR_PlugIn_Interfaces"
                  xlink:href="http://myDomain.de/TDR_Plugin_Interfaces.xml">
                  </sml:interface>
                  <sml:interface name="RS-232">
                        <sml:InterfaceDefinition>
                        <!-- OSI based Interface Definition-->
                              <...>
                        </sml:InterfaceDefinition>
                  </sml:interface>
            </sml:InterfaceList>
      </sml:interfaces>
      <!--**************************System Inputs*****************************************-->
      <sml:inputs>
            <sml:InputList>
```

```
            <sml:input name="signalPulse"></sml:input>
      </sml:InputList>
</sml:inputs>
<!--******************System Outputs*************************************************-->
<sml:outputs>
      <sml:OutputList>
      <!--waveform is form of a data curve-->
            <sml:output name="waveForm">
                  <swe:Curve>
                        <swe:elementCount>
                              <swe:Count/>
                        </swe:elementCount>
                        <swe:elementType>
                              <swe:SimpleDataRecord>
                                    <swe:field name="Distance">
                                          <swe:Quantity>
                                                <swe:uom code="m"></swe:uom>
                                          </swe:Quantity>
                                    </swe:field>
                                    <swe:field name="reflectionCoefficient">
                                          <swe:Quantity/>
                                    </swe:field>
                              </swe:SimpleDataRecord>
                        </swe:elementType>
                        <swe:encoding>
                              <swe:TextBlock decimalSeparator="." blockSeparator="$"
                              tokenSeparator=";">
                              </swe:TextBlock>
                        </swe:encoding>
                        <swe:values xlink:href="./tdrOutput01.xml"/>
                  </swe:Curve>
            </sml:output>
<!--single values-->
            <sml:output name="outputImpedance">
                  <swe:Quantity/>
            </sml:output>
      </sml:OutputList>
</sml:outputs>
<!--*********************************Parameters (Taskable)**************************-->
<sml:parameters>
      <sml:ParameterList>
      <!--sets system communication language-->
            <sml:parameter name="SystemLanguage">
                  <swe:SimpleDataRecord>
                        <swe:field name="defaultLanguage">
                              <swe:Text><swe:value>English</swe:value></swe:Text>
                        </swe:field>
                        <swe:field name="Language_2">
                              <swe:Text><swe:value>German</swe:value></swe:Text>
                        </swe:field>
                        <swe:field name="Language_3">
                              <swe:Text><swe:value>French</swe:value></swe:Text>
                        </swe:field>
                  </swe:SimpleDataRecord>
            </sml:parameter>
            <!--Sets update rate-->
            <sml:parameter name="measureUpdateRate" xlink:href="#OutputUpdateRate"/>
            <!--Sets TDR velocity factor-->
            <sml:parameter name="velocityFactor" xlink:href="#PVF"/>
            <!--calibration etc., if available-->
      </sml:ParameterList>
</sml:parameters>

<!--***************Internal Components*********************************************-->
<sml:components>
      <sml:ComponentList>
            <sml:component name="Pulser"
            xlink:arcrole="urn:ogc:def:process:OGC:transducer"
            xlink:href="urn:agis:def:sensors:AGIS:TDRPulser">
            </sml:component>
```

```
            <sml:component name="Sampler"
            xlink:arcrole="urn:ogc:def:process:OGC:detector"
            xlink:href="urn:agis:def:sensors:AGIS:TDRSampler">
            </sml:component>
            <sml:component name="CoaxialCable"
            xlink:arcrole="urn:ogc:def:process:OGC:detector"
            xlink:href="urn:agis:def:sensors:AGIS:TDRCoaxialCable">
            </sml:component>
            <sml:component name="Battery"
            xlink:arcrole="urn:agis:def:process:AGIS:component"
            xlink:href="urn:agis:def:sensors:AGIS:TDRBattery">
            </sml:component>
      </sml:ComponentList>
</sml:components>
</sml:System>
</sml:member>
<!--******************System Process Model*****************************************-->
<sml:member>
      <sml:ProcessModel>
            <gml:description>process model</gml:description>
                  <sml:method>
                        <sml:ProcessMethod>
                              <!--Rules Set-->
                              <sml:rules>
                                    <sml:RulesDefinition></sml:RulesDefinition>
                              </sml:rules>
                              <!--Implementations-->
                              <sml:implementation
                        xlink:href="http://myDomain.de/TDR_Plugin_Interfaces.xml">
                                    <sml:ImplementationCode language="java"
                                    framework="AGISSensorPluginsFramework">
                                          <sml:sourceRef></sml:sourceRef>
                                          <sml:binaryRef
                              xlink:arcrole="GeotechSoftware"></sml:binaryRef>
                                    </sml:ImplementationCode>
                              </sml:implementation>
                        </sml:ProcessMethod>
                  </sml:method>
            </sml:ProcessModel>
      </sml:member>
</sml:SensorML>
<!--***************************END*************************************************-->
```

12.4.3. DigiCam Sensor Instance

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Developed By Admire Kandawasvika-->
<!--Creation Date: 21.09.2007. Based on SensorML Spec. OGC 07-000 released on 17.07.2007.
-->
<!--Last Update: 03.10.2008-->
<sml:SensorML version="1.0.1" xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:swe="http://www.opengis.net/swe/1.0.1" xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:ism="urn:us:gov:ic:ism:v2"
xsi:schemaLocation="http://www.opengis.net/sensorML/1.0.1
E:\kaad_dvpt\ogc_schemas\sensorML\1.0.1\sensorML.xsd">
<sml:member>
<!-- ******************Sensor System****************************** -->
<sml:System gml:id="TravellerDCXZ6Camera">
      <gml:description>Traveller DCXZ6 Digital Camera with video capabilities.
      </gml:description>
      <gml:name>Traveller DCXZ6 DigiCam</gml:name>
<!--
*************************************Keywords*****************************************
*****************************-->
<sml:keywords>
      <sml:KeywordList codeSpace="urn:agis:def:keyword:AGIS:keywords">
            <sml:keyword>Digital Camera</sml:keyword>
```

```
                <sml:keyword>DigiCam</sml:keyword>
        </sml:KeywordList>
</sml:keywords>
<!-- ********************System Identification Information*********************** -->
<sml:identification>
        <sml:IdentifierList>
                <sml:identifier name="ShortName">
                        <sml:Term definition="urn:ogc:def:identifier:OGC:shortName">
                                <sml:value>TravDCXZ6DigiCam</sml:value>
                        </sml:Term>
                </sml:identifier>
                <sml:identifier name="ModelName">
                        <sml:Term definition="urn:ogc:def:identifier:OGC:modelName">
                                <sml:value>23179004</sml:value>
                        </sml:Term>
                </sml:identifier>
                <sml:identifier name="VendorName">
                        <sml:Term definition="urn:ogc:def:identifier:OGC:vendorName">
                                <sml:value>SUPRA Foto-Elektronik-Vertreibs GmbH</sml:value>
                        </sml:Term>
                </sml:identifier>
                <sml:identifier name="SerialNumber">
                        <sml:Term definition="urn:ogc:def:identifier:OGC:serialNumber">
                                <sml:value>9katrad9da003ql</sml:value>
                        </sml:Term>
                </sml:identifier>
        </sml:IdentifierList>
</sml:identification>
<!--**********************System Classification***********************************-->
<sml:classification>
        <sml:ClassifierList>
                <sml:classifier name="SensorType">
                        <sml:Term definition="urn:agis:def:identifier:AGIS:sensorType">
                        <sml:codeSpace xlink:href="urn:agis:classifier:AGIS:sensorTypes"/>
                                <sml:value>Digital Camera</sml:value>
                        </sml:Term>
                </sml:classifier>
                <sml:classifier name="SensorType">
                        <sml:Term definition="urn:agis:def:identifier:AGIS:sensorType">
                        <sml:codeSpace xlink:href="urn:agis:classifier:AGIS:sensorTypes"/>
                                <sml:value>Image Acquisition Sensor</sml:value>
                        </sml:Term>
                </sml:classifier>
                <sml:classifier name="Application">
                        <sml:Term definition="urn:ogc:classifier:OGC:application">
                                <sml:value>Digital Imaging</sml:value>
                        </sml:Term>
                </sml:classifier>
                <sml:classifier name="Application">
                        <sml:Term definition="urn:ogc:classifier:OGC:application">
                                <sml:value>Photos and Videos Capture</sml:value>
                        </sml:Term>
                </sml:classifier>
                <sml:classifier name="Application">
                        <sml:Term definition="urn:ogc:classifier:OGC:application">
                                <sml:value>Surveillance</sml:value>
                        </sml:Term>
                </sml:classifier>
        </sml:ClassifierList>
</sml:classification>
<!--************Document Constraints(operational, security,rights)*****************-->
<sml:validTime>
        <gml:TimePeriod gml:id="ValidTime">
                <gml:beginPosition>2007-09-21T09:00:00Z</gml:beginPosition>
                <gml:endPosition indeterminatePosition="now"/>
        </gml:TimePeriod>
</sml:validTime>
<!--***********System Physical Characteristics*************************************-->
<sml:characteristics name="Physical Properties">
        <swe:DataRecord definition="urn:ogc:def:property:OGC:physicalProperties">
```

```
            <swe:field name="physicalProperties">
                  <swe:DataRecord>
                        <swe:field name="width">
                              <swe:Quantity
                              definition="urn:agis:def:property:Manuf:width">
                                    <swe:uom code="mm"/>
                                    <swe:value>25.5</swe:value>
                              </swe:Quantity>
                        </swe:field>
                        <swe:field name="length">
                              <swe:Quantity
                              definition="urn:agis:def:property:Manuf:length">
                                    <swe:uom code="mm"/>
                                    <swe:value>93</swe:value>
                              </swe:Quantity>
                        </swe:field>
                        <swe:field name="height">
                              <swe:Quantity
                              definition="urn:manuf:def:property:Manuf:height">
                                    <swe:uom code="mm"/>
                                    <swe:value>57</swe:value>
                              </swe:Quantity>
                        </swe:field>
                        <swe:field name="weight">
                              <swe:Quantity
                              definition="urn:ogc:def:property:OGC:weight">
                                    <swe:uom code="g"/>
                                    <swe:value>164</swe:value>
                              </swe:Quantity>
                        </swe:field>
                  </swe:DataRecord>
            </swe:field>
      </swe:DataRecord>
</sml:characteristics>
<sml:characteristics name="Electrical Requirements">
      <swe:DataRecord gml:id="electricalPower"
      definition="urn:ogc:def:property:OGC:powerRequirement">
            <swe:field name="Voltage">
                  <swe:Quantity definition="urn:ogc:def:property:OGC:voltage">
                        <swe:uom code="V"/>
                        <swe:value>5</swe:value>
                  </swe:Quantity>
            </swe:field>
            <swe:field name="CurrentType">
                  <swe:Category
                  definition="urn:ogc:def:property:OGC:electricalCurrentType">
                        <swe:value>DC</swe:value>
                  </swe:Category>
            </swe:field>
            <swe:field name="Amp">
                  <swe:Quantity
                  definition="urn:ogc:def:property:OGC:electricalCurrent">
                        <swe:uom code="A"></swe:uom>
                        <swe:value>2</swe:value>
                  </swe:Quantity>
            </swe:field>
      </swe:DataRecord>
</sml:characteristics>
<!--******************System Capabilities****************************************-->
<sml:capabilities name="Optical Capabilities"
xlink:href="./TravDCXZ6DigiCam.xml#OpticalCapabilities">
      <swe:DataRecord gml:id="opticalCapabilities">
            <swe:field name="FocalLength">
                  <swe:QuantityRange
                  definition="urn:ogc:def:property:OGC:1.0:FocalLength" >
                        <swe:uom code="mm"></swe:uom>
                        <swe:value>7.8 46.8</swe:value>
                  </swe:QuantityRange>
            </swe:field>
            <swe:field name="OpticalZoom">
```

```
                        <swe:Quantity>
                                <gml:description>optical zoom max value</gml:description>
                                <swe:uom code="1x"></swe:uom>
                                <swe:value>6</swe:value>
                        </swe:Quantity>
                </swe:field>
                <swe:field name="DigitalZoom">
                        <swe:QuantityRange>
                                <gml:description>digital zoom range value</gml:description>
                                <swe:uom code="1x"></swe:uom>
                                <swe:value>2 10</swe:value>
                        </swe:QuantityRange>
                </swe:field>
        </swe:DataRecord>
</sml:capabilities>
<sml:capabilities name="Imaging Capabilities"
xlink:href="./TravDCXZ6DigiCam.xml#VideoCapabilities">
        <swe:DataRecord gml:id="imagingCapabilities">
                <!--Image Resolution-->
                <swe:field name="ImageResolution">
                        <swe:DataRecord>
                                <swe:field name="LowResolution">
                                        <swe:SimpleDataRecord >
                                                <swe:field name="x">
                                                        <swe:Count>
                                                                <swe:value>2048</swe:value>
                                                        </swe:Count>
                                                </swe:field>
                                                <swe:field name="y">
                                                        <swe:Count>
                                                                <swe:value>1536</swe:value>
                                                        </swe:Count>
                                                </swe:field>
                                        </swe:SimpleDataRecord >
                                </swe:field>
                                <swe:field name="MediumResolution">
                                        <swe:SimpleDataRecord >
                                                <swe:field name="x">
                                                        <swe:Count>
                                                                <swe:value>2816</swe:value>
                                                        </swe:Count>
                                                </swe:field>
                                                <swe:field name="y">
                                                        <swe:Count>
                                                                <swe:value>2112</swe:value>
                                                        </swe:Count>
                                                </swe:field>
                                        </swe:SimpleDataRecord>
                                </swe:field>
                                <swe:field name="HighResolution">
                                        <swe:SimpleDataRecord >
                                                <swe:field name="x">
                                                        <swe:Count>
                                                                <swe:value>3648</swe:value>
                                                        </swe:Count>
                                                </swe:field>
                                                <swe:field name="y">
                                                        <swe:Count>
                                                                <swe:value>2736</swe:value>
                                                        </swe:Count>
                                                </swe:field>
                                        </swe:SimpleDataRecord>
                                </swe:field>
                        </swe:DataRecord>
                </swe:field>
                <!--Video Resolution-->
                <swe:field name="VideoFrameResolution">
                        <swe:SimpleDataRecord >
                                <swe:field name="x">
                                        <swe:Count>
```

```
                                        <swe:value>640</swe:value>
                                    </swe:Count>
                                </swe:field>
                                <swe:field name="y">
                                    <swe:Count>
                                        <swe:value>480</swe:value>
                                    </swe:Count>
                                </swe:field>
                            </swe:SimpleDataRecord>
                    </swe:field>
                    <!--Image Formats-->
                    <swe:field name="ImageFormats">
                            <swe:Text><swe:value>jpeg</swe:value></swe:Text>
                    </swe:field>
                    <!--Video Formats-->
                    <swe:field name="VideoFormats">
                            <swe:Category><swe:value>avi mpge4</swe:value></swe:Category>
                    </swe:field>
                    <!--other-->
                    <swe:field name="AutoFocus">
                            <swe:Boolean definition="urn:ogc:def:property:OGC:1.0:autoFocus">
                                    <swe:value>true</swe:value>
                            </swe:Boolean>
                    </swe:field>
            </swe:DataRecord>
    </sml:capabilities>
    <!--***************Contact Information*********************************************-->
    <sml:contact>
            <sml:ContactList gml:id="Vendor">
            <gml:description>Contacts information about the  camera.</gml:description>
                    <sml:member xlink:arcrole="urn:ogc:def:property:OGC:author">
                            <sml:ResponsibleParty>
                                    <sml:organizationName>SUPRA Foto-Elektronik-Vertreibs GmbH
                                    </sml:organizationName>
                                    <sml:contactInfo>
                                            <sml:phone>
                                                    <sml:voice>+49 180 500 5723</sml:voice>
                                                    <sml:facsimile>+49 631 351 9949</sml:facsimile>
                                            </sml:phone>
                                            <sml:address>
                                                    <sml:deliveryPoint>Denisstraße 28A
                                                    </sml:deliveryPoint>
                                                    <sml:city>Kaiserslautern</sml:city>
                                                    <sml:postalCode>67663</sml:postalCode>
                                                    <sml:country>Germany</sml:country>
                                            </sml:address>
                                    </sml:contactInfo>
                            </sml:ResponsibleParty>
                    </sml:member>
            </sml:ContactList>
    </sml:contact>
    <!--*************Other System Resources*****************************************-->
    <sml:documentation>
            <sml:DocumentList>
                    <... >
            </sml:DocumentList>
    </sml:documentation>
    <sml:history>
            <sml:EventList>
                    <sml:member name="deployment"
                    xlink:arcrole="urn:ogc:def:property:OGC:deployment">
                                    <sml:Event>
                                            <sml:date>2007-09-21T09:00:00Z</sml:date>
                                            <gml:description>DigiCam Deployment Event
                                            </gml:description>
                                            <sml:contact xlink:arcrole="installer"
                                            xlink:href="http://myDomain.de/installer.xml">
                                            </sml:contact>
                                            <sml:documentation xlink:arcrole="deploymentReport">
                                                    <sml:Document>
```

```
                                        <gml:description>A report about the sensor system
                                        deployment
                                        </gml:description>
                                        <sml:onlineResource
xlink:href="http://myDomain.de/TravellerDCXZ6CameraDeployReport.pdf">
                                        </sml:onlineResource>
                                        </sml:Document>
                                    </sml:documentation>
                                </sml:Event>
                        </sml:member>
                </sml:EventList>
        </sml:history>
        <!--**********Geospatial Information************************************************-->
        <!--**********1. Spatial Reference Frame*******************************************-->
        <sml:spatialReferenceFrame>
                <gml:EngineeringCRS gml:id="TPS_Sensor_Frame">
                        <...>
                </gml:EngineeringCRS>
        </sml:spatialReferenceFrame>
        <!--******************2. Current GeoPosition*****************************************-->
        <sml:position name="CamPosition">
                <swe:Position localFrame="#Cam_Sensor_Frame" referenceFrame="TPS_Sensor_Frame">
                        <...>
                </swe:Position>
        </sml:position>
        <!--****************System Interface Definition*************************************-->
        <sml:interfaces>
                <sml:InterfaceList>
                        <sml:interface name="Cam_PlugIn_Interfaces"
                        xlink:href="http://myDomain.de/Cam_Plugin_Interfaces.xml">
                        </sml:interface>
                        <sml:interface name="USB 2.0 High Speed">
                                <sml:InterfaceDefinition>
                                <!-- OSI based Interface Definition-->
                                <sml:physicalLayer xlink:href="urn:ogc:def:property:OGC:USB"/>
                                <sml:mechanicalLayer xlink:href="urn:ogc:def:property:OGC:USBCabel"/>
                                </sml:InterfaceDefinition>
                        </sml:interface>
                </sml:InterfaceList>
        </sml:interfaces>
        <!--********************System Inputs*************************************************-->
        <sml:inputs>
                <sml:InputList>
                        <sml:input name="Radiance">
                                <swe:ObservableProperty
                                definition="urn:ogc:def:property:OGC:radiance"/>
                        </sml:input>
                        <sml:input name="SceneCoverage">
                                <swe:GeoLocationArea>
                                        <swe:member>
                                                <swe:Envelope>
                                                        <swe:lowerCorner/>
                                                        <swe:upperCorner/>
                                                </swe:Envelope>
                                        </swe:member>
                                </swe:GeoLocationArea>
                        </sml:input>
                </sml:InputList>
        </sml:inputs>
        <!--***************System Outputs*************************************************-->
        <sml:outputs>
                <sml:OutputList>
                        <sml:output name="Image">
                                <swe:DataRecord>
                                        <swe:field name="timeOfCapture"></swe:field>
                                        <swe:field name="format"></swe:field>
                                        <swe:field name="resolution"></swe:field>
                                </swe:DataRecord>
                        </sml:output>
                        <sml:output name="Video">
```

```
                    <swe:DataRecord>
                          <swe:field name="timeOfCapture"></swe:field>
                          <swe:field name="format"></swe:field>
                          <swe:field name="resolution"></swe:field>
                    </swe:DataRecord>
             </sml:output>
       </sml:OutputList>
</sml:outputs>
<!--**************Parameters (Taskable)*****************************************-->
<sml:parameters>
       <sml:ParameterList>
             <!--sets system communication language-->
             <sml:parameter name="SystemLanguage">
                    <swe:SimpleDataRecord>
                          <swe:field name="defaultLanguage">
                                <swe:Text><swe:value>English</swe:value></swe:Text>
                          </swe:field>
                          <swe:field name="Language_2">
                                <swe:Text><swe:value>German</swe:value></swe:Text>
                          </swe:field>
                          <swe:field name="Language_3">
                                <swe:Text><swe:value>French</swe:value></swe:Text>
                          </swe:field>
                    </swe:SimpleDataRecord>
             </sml:parameter>
             <!--Sets system time based on user's local time-->
             <sml:parameter name="sysTime">
                    <swe:Time/>
             </sml:parameter>
             <!--Sets Point Coding Mode-->
             <sml:parameter name="pointIDLogModeAuto">
                    <swe:Boolean>
                    <gml:description>PointID logging mode. Automatic or Manual.
                    </gml:description>
                          <swe:value>true</swe:value>
                    </swe:Boolean>
             </sml:parameter>
             <!—Other parameters-->
             <sml:parameter name="ZoomDigital" xlink:href="#DigitalZoom"/>
             <sml:parameter name="ZoomOptical" xlink:href="#OpticalZoom"/>
             <sml:parameter name="ImageResolution" xlink:href="#ImageFormats"/>
             <sml:parameter name="VideoFormat" xlink:href="#VideoFormats"/>
             <sml:parameter name="AutoFocus" xlink:href="#AutoFocus"/>
       </sml:ParameterList>
</sml:parameters>

<!--******************Internal Components*****************************************-->
<sml:components>
       <sml:ComponentList>
             <sml:component name="ImageSensor"
             xlink:arcrole="urn:ogc:def:process:OGC:sensor"
             xlink:href="urn:agis:def:sensors:AGIS:CCDSensor">
             </sml:component>
             <sml:component name="OpticalBlock1"
             xlink:arcrole="urn:ogc:def:process:OGC:process"
             xlink:href="urn:agis:def:process:AGIS:OpticalBlock1">
             </sml:component>
             <sml:component name="LPF" xlink:arcrole="urn:ogc:def:process:OGC:process"
             xlink:href="urn:agis:def:process:AGIS:OpticalLPF">
             </sml:component>
             <sml:component name="Clock" xlink:arcrole="urn:ogc:def:process:OGC:detector"
             xlink:href="urn:agis:def:sensors:AGIS:CamClock">
             </sml:component>
             <sml:component name="Battery"
             xlink:arcrole="urn:agis:def:process:AGIS:component"
             xlink:href="urn:agis:def:sensors:AGIS:CamBattery">
             </sml:component>
             <!--etc. other components-->
       </sml:ComponentList>
</sml:components>
```

```
</sml:System>
</sml:member>
<!--********************System Process Model*************************************-->
<sml:member>
       <sml:ProcessModel>
              <sml:method>
                     <sml:ProcessMethod>
                            <!--Rules Set-->
                            <sml:rules>
                                   <sml:RulesDefinition></sml:RulesDefinition>
                            </sml:rules>
                            <!--Implementations-->
                            <sml:implementation
                            xlink:href="http://myDomain.de/CAM_Plugin_Interfaces.xml">
                            <sml:ImplementationCode language="java"
                            framework="AGISSensorPluginsFramework">
                                   <sml:sourceRef></sml:sourceRef>
                                   <sml:binaryRef xlink:arcrole="GeotechSoftware">
                                   </sml:binaryRef>
                            </sml:ImplementationCode>
                            /sml:implementation>
                     </sml:ProcessMethod>
              </sml:method>
       </sml:ProcessModel>
</sml:member>
</sml:SensorML>
<!--***********************END*****************************************************-->
```

Acknowledgements

My deepest gratitude goes to my mentor and advisor Univ.-Prof. Dr.-Ing. Wolfgang Reinhardt, whose encouragement, guidance, and mentoring have lead to the completion of this dissertation. The fruitful discussions I have had with him have taught me different perspectives of reasoning, questioning thoughts, and expressing ideas. I have enjoyed exploring this dissertation under his supervision, patience, kindness, and guidance. I am also very grateful to Univ.-Prof. Dr.rer.nat Lars Bernard for the valuable advice, and insightful discussions. Many thanks to Univ.-Prof. i.R. Dr.-Ing Wilhelm Caspary who has also encouraged me to endure the research work.

On this long journey, I have also travelled with many past and current work colleagues in AGIS Research Group at the University of Bundeswehr Munich to whom I would like to give my sincere appreciation. Thanks to Iris Wiebrock, Dr.-Ing. Fei Wang, Dr.-Ing. Simone Stürmer, Dr.-Ing. Oliver Plan, Stephan Mäs, Stephan Ströbel, Eva Nuhn and others. The dissertation colloquiums have always been productive and the valuable criticisms have greatly helped me to understand and shape my dissertation work.

I am also very grateful to TOPCON and Leica Geosystems companies for providing me with access to important materials that I have used for my dissertation. Thanks also to Dr. Ruch and Dr. Möbus for the valuable information on the landslide monitoring theme.

Heartily thanks to my parents, brothers and sisters who have always supported and encouraged me in pursuing my goals.

This work will not have been possible without the love, care, sacrifice, and determined support from my wife Katharina. To my sons, Jacob and Tinotenda, thanks for being there and for giving me the reasons to work hard and pursue my life goals.

To friends and those whom I have not mentioned, but inspired and encouraged me in one way or another, many thanks to you all!

Curriculum Vitae

Name	**Admire Masiyiwa Kandawasvika**
Date of Birth	23rd of February 1974
Place of Birth	Harare, Zimbabwe
Languages	English, Shona, German
1994 – 1997	B.Sc. Engineering (Honours) Degree in Surveying at the University of Zimbabwe.
1998 – 05/1999	Project Engineer for "Geomatics Engineering Pvt/Ltd" in Harare, Zimbabwe.
06/1999 – 08/1999	GIS/Database Officer for an Aquatic Resources Management Project "ALCOM" of the Food and Agricultural Organisation (FAO)/United Nations (UN), Harare, Zimbabwe.
09/1999 – 04/2001	M.Sc. in Photogrammetry and Geoinformatics at Stuttgart University of Applied Sciences (*Hochschule für Technik*), Stuttgart, Germany.
04/2001 – 09/2001	Research Assistent (*wissenschaftliche Hilfskraft*) at Stuttgart University of Applied Sciences (*Hochschule für Technik*), Stuttgart, Germany.
10/2001 – 2009	Research Associate (*wissenschaftlicher Mitarbeiter*) in AGIS Research Group at the University of Bundeswehr Munich (*UniBw München*), in Neubiberg, Germany. 2001 – 2003: **PARAMOUNT** (Public Safety & Commercial Info-Mobility Applications & Services in the Mountains) - EU/IST Project. 2003 – 2005: **Advancement of Geoservices** (*Verbundprojekt Weiterentwicklung von Geodiensten*) – BMBF (German Federal Ministry of Education and Research) and DFG (German Research Council) Project. 2005 – 2009: **ASYS** – A real-time system for the documentation of private drainage/sewer systems (*ein System zur Dokumentation des Verlaufes von Grundstücksentwässerungsanlagen*).

Contact: admirek@yahoo.com