Open Geospatial Consortium Inc.

Date: 2009-10-02

Reference number of this document: **OGC 09-132r1**

Version: 3 (Rev 3.1)

Category: OGC® Discussion Paper

Editor: Thomas Usländer (Ed.)

Specification of the Sensor Service Architecture (SensorSA)

Warning

This document is not an OGC Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	OGC® Discussion Paper
Document subtype:	Abstract Specification
Document stage:	Approved for Public release.
Document language:	English

Preamble to the "Specification of the Sensor Service Architecture (SensorSA)"

This document specifies the Sensor Service Architecture (SensorSA) of the European Integrated Project FP6-033564 Sensors Anywhere (SANY).
The SensorSA follows the design principles of service-oriented architectures (SOA) with a particular focus on the access, the discovery, the management and the processing of information provided by sensors and sensor networks. As such, it contains sensor-specific services and information models, however, in order to provide a higher-level, functionally and semantically richer interface to environmental information and decision-support systems it also abstracts from the peculiarities of sensors and encompasses generic information processing functionality. The SensorSA foresees mechanisms to generate events and distribute them as notifications to interested consumers. This enables spontaneous distribution of information about changing configurations in underlying sensor networks. Furthermore, the SensorSA relates the basic concepts of a resource-oriented architectural style such as resources and their representations to the SOA concepts in order to gain flexibility in discovery tasks and the mapping to underlying mainstream Web service environments.
The foundation for the conceptual architectural work for SANY is the OGC Best Practices Document 07-097 which corresponds to the Reference Model for the ORCHESTRA Architecture (RM-OA) as well as the architecture, the service and the information model specifications of the OGC Sensor Web Enablement (SWE) initiative.

For further SANY specifications see http://sany-ip.eu .

Sixth Framework Programme
Priority IST 2.5.12
Information Society Technologies

Integrated Project

Contract No.: 033564

Project Deliverable D2.3.4

Specification of the Sensor Service Architecture V3

Document Version 3.1 (29/09/2009)

OGC 09-132r1

Due date of Project Deliverable: 30/11/2009

Actual submission date: xx/xx/2009

Document Control Page

Title	Specification of the Sensor Service Architecture Version 3 (Document Version 3.1)
Creator	Fraunhofer IITB
Editor	Thomas Usländer
Description	Specification of a generic service-oriented architecture integrating the access to, the management and the processing of sensor-related information based upon the emerging standards of the Open geospatial Consortium (OGC), and resulting from the requirements analysis of diverse application domains such as maritime risk management, observation of geo-hazards and monitoring of air quality.
Publisher	SANY Consortium
Contributors	Bartha, Mihai AIT Bleier, Thomas AIT Dihé, Pascal EIG Havlik, Denis AIT Hilbring, Désirée Fraunhofer IITB Hugentobler, Marco ETH Zürich Iosifescu Enescu, Ionut ETH Zürich Kunz, Siegbert Fraunhofer IITB Puhl, Sebastian EIG Scholl, Martin EIG Jacques, Patrick Spacebel Schlobinski, Sascha EIG Simonis, Ingo OGCE Stumpp, Jörg Fraunhofer IITB Usländer, Thomas Fraunhofer IITB Watson, Kym Fraunhofer IITB
Type	Text
Format	MS Word
Language	EN-GB
Creation date	24/09/2008
Version number	3.1
Version date	29/09/2009
Last modified by	Fraunhofer IITB (Ed.) based upon input from SP2 team
Rights	Copyright “SANY Consortium”. During the drafting process, access is generally limited to the SANY Partners.
Audience	☐ internal ☒ public ☐ restricted, access granted to:

Review status	☒ Draft ☒ WP Manager accepted ☒ SP Manager accepted ☐ MB quality controlled ☐ Co-ordinator accepted	Where applicable: ☐ Accepted by the GA ☐ Accepted by the GA as public document
Action requested	☐ to be revised by Partners involved in the preparation of the Project Deliverable ☐ to be revised by all SANY Partners ☐ for approval of the WP Manager ☐ for approval of the SP Manager ☒ for approval of the Quality Manager ☐ for approval of the Project Co-ordinator ☐ for approval of the General Assembly	
Requested deadline	25/10/09	

Revision history

Revision	Date	Modified by	Comments
0.1	11/05/2007	TUS	Initial draft structure
0.2	03/07/2007	TUS	Inclusion of partners' input
0.3	20/07/2007	TUS	Result of SP2 meeting 18.-20.07.07 Sasbachwalden
0.4	07/08/2007	TUS and SP2 team	Integration of input from SP2 partners Harmonisation of text and glossary
0.5	13/09/2007	TUS	comments of partners included
0.6	15/10/2007	TUS	Results of SP2 meeting 11-12.10.07 Karlsruhe
0.7	17/10/2007	TUS	comments of partners included
0.8	05/11/2007	RDE	MB level QA review
1.0	09/11/2007	TUS	resolution of QA comments
1.1	20/03/2008	TUS	Working draft for D2.3.2
1.2	30/06/2008	TUS	partner contributions integrated, consolidated and harmonised
1.3	01/07/2008	SPF	Support of MB level QA
1.3	16/07/2008	RDE	MB level QA
1.4	25/07/2008	TUS and SP2 team	resolution of QA comments and change requests
1.41	06/08/2008	FHa	Co-ordinator accepted
2.0	24/09/2008	TUS and SP2 team	Preparation of v2 (D2.3.3)
2.1	23/01/2009	TUS and SP2 team	Input to SP2 meeting Jan. 2009 and result of discussion
2.1	06/03/2009	TUS and SP2 team	Input/update on event-driven interactions, access control, GMES/GEOSS requirements
2.1	13/03/2009	TUS and SP2 team	Result of SP2 meeting in Saarbrücken, 10-11 March 2009
2.1	02/04/2009	TUS	Inclusion of comments and contributions for the final draft
3.0	08/06/2009	TUS	Extended structure for D2.3.4
3.0	08/07/2009	TUS	Contributions of SP2 partners EIG, OGCE, SPB and Fraunhofer IITB
3.0	10/07/2009	TUS and SP2 team	Contribution of AIT and result of SP2 meeting, 9-10 July 209
3.0	11/09/2009	TUS and SP2 team	Integration of partners' contributions and resolution of comments
3.1	29/09/2009	TUS	Minor editorial corrections; publication as OGC 09-132r1

Table of Contents

Index of Figures

Index of Tables

1. Executive Summary

This document specifies the Sensor Service Architecture (SensorSA) of the European Integrated Project FP6-033564 Sensors Anywhere (SANY). In its present Version 3 it corresponds to the SANY deliverable D2.3.4.

The SensorSA belongs to the family of service-oriented architectures (SOA) but has a particular focus on the access, the management and the processing of information provided by sensors and sensor networks. As such, it contains sensor-specific services, however, in order to provide a higher-level, functionally and semantically richer interface to environmental risk management applications it also has to abstract from the peculiarities of sensors and to encompass generic information processing functionality. Thus, there is a sliding passage to the functionality of a generic service infrastructure. The SensorSA foresees mechanisms to generate events and distribute them as notifications to interested consumers. This enables spontaneous distribution of information about changing configurations in underlying sensor networks, e.g. the dynamic addition or removal of sensor devices, which is a pre-requisite for the support of the "plug-and-measure" type of operation. Furthermore, the SensorSA relates the basic concepts of a resource-oriented architecture (ROA) such as resources and their representations to the SOA concepts in order to gain flexibility in discovery tasks and the mapping to underlying mainstream Web service environments.

The foundation for the conceptual architectural work for SANY has been taken from the OGC Best Practices Document 07-097 which corresponds to the Reference Model for the ORCHESTRA Architecture (RM-OA) as specified by the European Integrated Project FP6-511678 ORCHESTRA (Open Architecture and Spatial Data Infrastructure for Risk Management). The RM-OA provides a platform-neutral abstract specification of a geospatial service-oriented architecture that responds to the requirements of environmental risk management applications. It comprises generic architecture services and information models based on and extending existing specifications of the Open Geospatial Consortium (OGC).

The objective of the SensorSA is to motivate and specify the basic design decisions derived from user requirements and generic architectural principles. It is structured according to the viewpoints of the Reference Model for Open Distributed Processing (RM-ODP) as defined in ISO/IEC 10746-1:1998 (E). The RM-ODP viewpoints are interpreted in the context of an SOA in analogy to the interpretation of the RM-ODP in the RM-OA. The SensorSA provides the basic concepts, their interrelationships (conceptual models) and abstract specifications of implementation models, services and interfaces. By abstract it is meant that the specification is independent of the specifics of a particular service platform.

The specification of the SensorSA follows an iterative design approach. Each version builds on the results of the former versions. Topics described in former versions are refined and/or new topics are taken up and specified. Four versions (V0-V3) of the SensorSA specification have been foreseen. The roadmap for the individual versions has been continuously adapted according to the analysis of the user requirements and their prioritisation as well as the technological challenges that SANY aimed to solve. The focus of the final version 3 lies on refinements and enhancements of the event-based architectural style and the use of events for cascades of Sensors Observation Service instances and catalogue harvesting.

2. Introduction

2.1. Purpose of this document

This document specifies the Sensor Service Architecture (SensorSA) of the SANY project. The SensorSA belongs to the family of service-oriented architectures (SOA) with additional support of event processing but has a particular focus on the access, management and processing of information provided by sensors and sensor networks. As such, it contains sensor-specific services. However, in order to provide a higher-level, functionally and semantically richer interface to environmental risk management applications it also has to abstract from the peculiarities of sensors and to encompass generic information processing functionality. Thus, there is a sliding passage to the functionality of a generic service infrastructure.

The service-oriented architecture specified by the European Integrated Project FP6-511678 ORCHESTRA[1] (Open Architecture and Spatial Data Infrastructure for Risk Management) in its Reference Model for the ORCHESTRA Architecture (RM-OA, 2007) has been chosen as the foundation for the conceptual architectural work in the SANY project. A major cornerstone for a SANY application environment as part of an implementation architecture is the Service Support Environment (ESA SSE, 2007). SSE is currently mainly used in the domain of earth observation sensors.

The architectural work in the SANY project distinguishes between

- a platform-neutral specification (SensorSA), and
- an implementation specification for the SANY application projects (SANY Implementation Architecture).

This approach is illustrated in Figure 2-1. The present document only specifies the conceptual SensorSA, which for simplicity is sometimes just called the SensorSA (SensorSA) or the SANY Architecture.

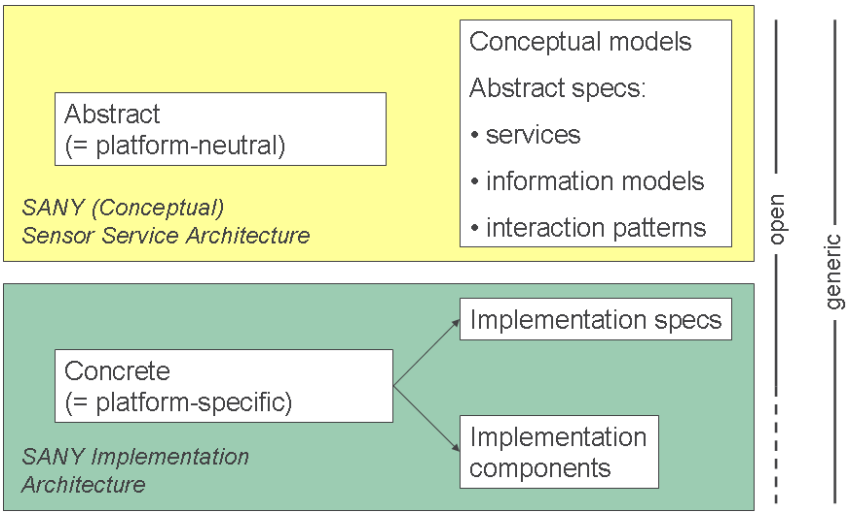


Figure 2-1: Scope of the Architectural Work in SANY

[1] http://www.orchestra.eu.org

The objective of the SensorSA is to motivate and specify the basic design decisions derived from user requirements and generic architectural principles. It provides the basic concepts and their interrelationships (conceptual models) and abstract specifications. By abstract it is meant that the specification is independent of the specifics of a particular service platform. Such an abstract specification comprises service specifications, information models and interaction patterns between the major architectural components.

The specification of the SensorSA is structured according to the viewpoints of the Reference Model for Open Distributed Processing (RM-ODP) as defined in ISO/IEC 10746-1:1998 (E). The RM-ODP viewpoints are interpreted in the context of a service-oriented architecture (SOA) in analogy to the interpretation of the RM-ODP in the Reference Model for the ORCHESTRA Architecture (RM-OA, 2007).

After the illustration of the architectural design process in section 3, the specification of the business context and related requirements are described in section 4, "Enterprise Viewpoint". Section 5 provides the specification of the "Sensor Model" used in the SensorSA. Further major concepts of the SensorSA are presented in section 6, followed by the core conceptual specification of the SensorSA, the specification of the Service and Information Viewpoints. Technological choices such as requirements and constraints about underlying Web service platforms are covered in the Technology Viewpoint in section 9. Engineering guidelines and recommendation how to use and compose the service and information models are covered in the Engineering Viewpoint in section 10.

Note 1: If concepts and models of the RM-OA are re-used without change, they are just referenced, sometimes accompanied by a short summary in order to enable a self-contained specification of the SANY Architecture. In cases where RM-OA concepts and models have been refined and even replaced for SANY purposes, they are specified in the present document, but using the style and the templates of the RM-OA. This approach ensures consistent editorial maintenance and review of both documents.

Note 2: The present version of the SensorSA continues and refines the work of the so-called ORCHESTRA Architecture Services (OA Services) with a focus on replacing the original Sensor Access Service as defined in the (RM-OA, 2007). In a subsequent version, this architecture specification intends to provide more application-oriented services, which are referred to in the RM-OA as ORCHESTRA Thematic Services. Then, the SensorSA will also define elements of an Application Architecture.

2.2. Intended Audience

The intended audience of this document includes system architects, information modellers and system developers engaged in designing sensor service networks and related applications taking into account relevant standards from ISO, OGC, W3C and OASIS.

2.3. Abbreviations and acronyms

ADC	Architecture and Data Committee (GEOSS)
AS-MI	Application Schema for Meta-information
CAFÉ	Clean Air for Europe
DG-INFSO	Directorate General for Information Society (EC)
EC	European Commission
ECMWF	European Center for Medium range Weather Forecasting
EO	Earth Observation
ESA	European Space Agency
ESDI	European Spatial Data Infrastructure
EU	European Union
FOI	Feature of Interest
FP6/7	$6^{th}/7^{th}$ Framework Programme (EC)
GEOSS	Global Earth Observation System of Systems
GFM	General Feature Model
GMES	Global Monitoring for Environment and Security
HMA	Heterogeneous Missions Accessibility
ID	Identifier
IETF	Internet Engineering Task Force
INSPIRE	Infrastructure for Spatial Information in the European Community
IdP	Identity Provider
IS	International Standard
ISO	International Standardization Organisation
IST	Information Society Technology
JRC	Joint Research Centre (EC)
MIB	Management Information Base
OASIS	1) Organization for the Advancement of Structured Information Standards 2) Open Advanced System for Disaster and Emergency Management (FP6 project)
OGC	Open Geospatial Consortium
OMG	Object Management Group
ORCHESTRA	Open Architecture and Spatial Data Infrastructure for Risk Management (FP6 project)
O&M	Observations and Measurement
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
RDF	Resource Description Framework
RM-OA	Reference Model for the ORCHESTRA Architecture
RM-ODP	Reference Model for Open Distributed Processing
SAML	Security Assertion Markup Language
SANY	Sensors Anywhere (FP6 project)
SAS	Sensor Alert Service
SDI	Spatial Data Infrastructure
SensorSA	Sensor Service Architecture
SensorML	Sensor Model Language
SIF	Standards and Interoperability Forum (GEOSS)

SLD	Styled Layer Descriptor
SN	Sensor Network
SOA	Service-oriented Architecture
SOA-RA	(OASIS) Reference Architecture for Service Oriented Architecture
SOA-RM	(OASIS) Reference Model for Service Oriented Architecture
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SSE	Service Support Environment
SSL	Secure Socket Layer
SSVN	Sensor Service Network
SWE	Sensor Web Enablement
UAA	User Management, Authentication and Authorisation
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
URN	Uniform Resource Name
UTC	Universal Coordinated Time
WADL	Web Application Description Language
W3C	World Wide Web Consortium
WIN	Wide information network for Risk Management integrated
WNS	Web Notification Service
XACML	eXtensible Access Control Markup Language

2.4. Glossary

2.4.1 General Remark

This document follows the ISO/IEC Directives, Part 2: Rules for the structure and drafting of International Standards with respect to the usage of the word “shall”**.** The word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this specification.

2.4.2 Terms and Definitions

Absolute Time (derived from ISO/IEC 18023:2006(E))
Provides 1) a means to specify an absolute time (UTC) for meta-information, and 2) a general-purpose mechanism for describing points in absolute (UTC) time.

Note: ISO/IEC 18023-1:2006 addresses the concepts, syntax and semantics for the representation and interchange of environmental data (SEDRIS). Its usage for the SensorSA as a definition and for the representation of time values is provisional. How the SEDRIS concepts go together with the ISO 19xxx series of standards will be determined. Thus, this definition may be subject to change in future versions.

Access control
Ability to enforce a policy that identifies permissible actions on a particular resource by a particular subject.

Accounting (OGC 07-097; RM-OA 2007)

Process of gathering information about the usage of resources by subjects.

Ad hoc Sensor Network
Sensor network in which communication links and/or nodes are not continually available or change dynamically. An ad hoc sensor network is often, but not necessarily, based on wireless communication between nodes with limited resources (energy supply, processing power). An ad hoc sensor network may include mobile sensors which belong to the network for a limited time or intermittently.

Alert
Synonym for notification.

Application (derived from http://www.opengeospatial.org/resources/?page=glossary)
Use of capabilities, including hardware, software and data, provided by an information system specific to the satisfaction of a set of user requirements in a given application domain.

Application Domain (OGC 07-097; RM-OA 2007)
Integrated set of problems, terms, information and tasks of a specific thematic domain that an application (e.g. an information system or a set of information systems) has to cope with.

Note: One example of an application domain is environmental risk management.

Application Schema (ISO 19109:2005)
Conceptual schema for data required by one or more applications.

Application Architecture (derived from OGC 07-097; RM-OA 2007)
Instantiation of a generic and open architecture (e.g. the ORCHESTRA Architecture) by inclusion of those thematic aspects that fulfil the purpose and objectives of a given application. The concepts for such an application stem from a particular application domain (e.g. a risk management application).

Architecture (of a system) (ISO/IEC 10746-2:1996)
Set of rules to define the structure of a system and the interrelationships between its parts.

Architecture Service (derived from OGC 07-097; RM-OA 2007)
Service that provides a generic, platform-neutral and application-domain independent functionality.

Assertion (SOA-RA**,** 2008)
An assertion is a proposition that is held to be true by a stakeholder. It is essentially a claim about the state of the world.

Note: In the context of SAML the term Assertion is used as a synonymous expression for Ticket.

Authentication (SOA-RA**,** 2008)
Concerns the identity of the participants in an exchange. Authentication refers to the means by which one participant can be assured of the identity of other participants.

Authorisation (SOA-RA**,** 2008)

Concerns the legitimacy of the interaction. Authorization refers to the means by which an owner of a resource may be assured that the information and actions that are exchanged are either explicitly or implicitly approved.

Catalogue (derived from http://www.opengeospatial.org/resources/?page=glossary)
Collection of entries, each of which describes and points to a collection of resources. Catalogues include indexed listings of resource collections, their contents, their coverages, and of meta-information. A catalogue registers the existence, location, and description of resource collections held by an Information Community. Catalogues provide the capability to add, modify and delete entries. A minimum Catalogue will include the name for the resource collection and the locational handle that specifies where these data may be found. Each catalogue is unique to its Information Community.

Component (OGC 07-097; RM-OA 2007)
Hardware component (device) or Software Component.

Conceptual model (ISO 19109:2005(E); ISO 19101)
Model that defines concepts of a universe of discourse.

Conceptual schema (ISO 19109:2005(E); ISO 19101)
Formal description of a conceptual model.

Confidentiality (SOA-RA**,** 2008)
Concerns the protection of privacy of participants in their interactions. Confidentiality refers to the assurance that unauthorized entities are not able to read messages or parts of messages that are transmitted.

Credential
Information used as proof of Identity (e.g. a password).

Note: During an Authentication process, credentials are presented to an Identity Provider to obtain related identity information (Ticket).

Discovery (derived from W3C: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#discovery)
Act of locating a machine-processable description of a resource that may have been previously unknown and that meets certain functional, informational or qualitative criteria. It involves matching a set of functional and other criteria with a set of resource descriptions.

End user (OGC 07-097; RM-OA 2007)
Members of agencies (e.g. civil or environmental protection agencies) or private companies that are involved in an application domain (e.g. risk management) and that use the applications built by the system users.

Error (of a measurement)
Difference between the measured value and the (in general unknown) 'true value' of the measured property.

Event (derived from Luckham and Schulte (Eds.) (2008) and ISO 19136)

Anything that happens or is contemplated as happening at an instant or over an interval of time.

Event Object
Record of an action that occurred at an instant or over an interval of time.

Environment (Oxford Dictionary)
1: (noun) the surroundings or conditions in which a person, animal, or plant lives or operates.
2: (the environment) the natural world, especially as affected by human activity.
3: (computing) Overall structure within which a user, computer, or program operates.

Feature (OGC 07-097; RM-OA 2007; derived from ISO 19101)
Abstraction of a real world phenomenon (ISO 19101) perceived in the context of an application.

Note: As in (RM-OA, 2007), the SANY understanding of a "real world" explicitly comprises hypothetical worlds. Features may but need not contain geospatial properties. In this general sense, a feature corresponds to an "object" in analysis and design models.

Framework (http://www.opengeospatial.org/resources/?page=glossary)
An information architecture that comprises, in terms of software design, a reusable software template, or skeleton, from which key enabling and supporting services can be selected, configured and integrated with application code.

Generic (Service, Infrastructure…) (derived from OGC 07-097; RM-OA 2007)
Independent on the organisation structure and application domain, etc. For example, a service is generic, if it is independent of the application domain. A service infrastructure is generic, if it is independent of the application domain and if it can adapt to different organisational structures at different sites, without programming (ideally).

Geospatial (http://www.opengeospatial.org/resources/?page=glossary)
Referring to a location relative to the Earth's surface. "Geospatial" is more precise in many geographic information system contexts than "geographic," because geospatial information is often used in ways that do not involve a graphic representation, or map, of the information.

Identity (Dictionary, 2004)
Collective aspect of the set of characteristics by which a thing is definitively recognizable or known.

Note: In the SensorSA the term Identity refers to a concept that is used to recognise a subject. A subject may have several identities

Identity Provider
Entity that issues identity information and possibly acts as authentication authority

Implementation (http://www.opengeospatial.org/resources/?page=glossary)
Software package that conforms to a standard or specification. A specific instance of a more generally defined system.

Integrity (SOA-RA**,** 2008)

Concerns the protection of information that is exchanged – either from unauthorized writing or inadvertent corruption. Integrity refers to the assurance that information that has been exchanged has not been altered.

Interface (ISO 19119:2005)
Named set of operations that characterize the behaviour of an entity.
The aggregation of operations in an interface, and the definition of interface, shall be for the purpose of software reusability. The specification of an interface shall include a static portion that includes definition of the operations. The specification of an interface shall include a dynamic portion that includes any restrictions on the order of invoking the operations.

Interoperability (ISO 19119:2005 or OGC; http://www.opengeospatial.org/resources/?page=glossary)
Capability to communicate, execute programs, or transfer data among various functional units in a manner that require the user to have little or no knowledge of the unique characteristics of those units (ISO 2382-1).

Loose coupling (W3C; http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#loosecoupling)
Coupling is the dependency between interacting systems. This dependency can be decomposed into real dependency and artificial dependency: Real dependency is the set of features or services that a system consumes from other systems. The real dependency always exists and cannot be reduced. Artificial dependency is the set of factors that a system has to comply with in order to consume the features or services provided by other systems. Typical artificial dependency factors are language dependency, platform dependency, API dependency, etc. Artificial dependency always exists, but it or its cost can be reduced. Loose coupling describes the configuration in which artificial dependency has been reduced to the minimum.

Meta-information (OGC 07-097; RM-OA 2007)
Descriptive information about resources in the universe of discourse. Its structure is given by a meta-information model depending on a particular purpose.

Note: A resource by itself does not necessarily need meta-information. The need for meta-information arises from additional tasks or a particular purpose (like catalogue organisation), where many different resources (services and data objects) must be handled by common methods and therefore have to have/get common attributes and descriptions (like a location or the classification of a book in a library).

Meta-information model (OGC 07-097; RM-OA 2007)
Implementation of a conceptual model for meta-information.

Non-repudiation (SOA-RA**,** 2008)
Concerns the accountability of participants. To foster trust in the performance of a system used to conduct shared activities it is important that the participants are not able to later deny their actions: to repudiate them. Non-repudiation refers to the means by which a participant may not, at a later time, successfully deny having participated in the interaction or having performed the actions as reported by other participants.

Notification
Message that transports one or more events. Depending on the form of the event, the notification may resemble the event that it transports.

Note: An alert is a notification. The terms notification and alert are used synonymously.

Observed Property (derived from OGC 07-022r1)
Identifier or description of the phenomenon for which the observation result provides an estimate of its value.

Observation (OGC 07-022)
Act of observing a property or phenomenon, with the goal of producing an estimate of the value of the property.

Open Architecture (OGC 07-097; RM-OA 2007)
Architecture whose specifications are published and made freely available to interested vendors and users with a view of widespread adoption of the architecture. An open architecture makes use of existing standards where appropriate and possible and otherwise contributes to the evolution of relevant new standards.

Operation (ISO 19119:2005; http://www.OpenGIS.org/docs/02-112.pdf)
Specification of a transformation or query that an object may be called to execute. An operation has a name and a list of parameters.

ORCHESTRA Architecture (OGC 07-097; RM-OA 2007)
Open architecture that comprises the combined generic and platform-neutral specification of the information and service viewpoint as part of the ORCHESTRA Reference Model.

ORCHESTRA Reference Model (http://www.eu-orchestra.org)
The ORCHESTRA Reference Model comprises a specification of all RM-ODP viewpoints for the open architecture for risk management.

Note: The ORCHESTRA Reference Model is specified in (RM-OA, 2007) and is the result of the European Integrated project ORCHESTRA. The relationship of the SANY Sensor Service Specification to the ORCHESTRA Reference Model is specified in section 6.1.

Phenomenon (OGC 07-022)
Concept that is a characteristic of one or more feature types, the value for which may be estimated by application of some procedure in an observation.

Plug-and-measure
Refers to the degree of capability to add a new sensor to a sensor network, register it in a service network and access its observations through services in all functional domains of a sensor service network without additional manual intervention.

Policy (derived from SOA-RM, 2006)
Representation of a constraint or condition on the use, deployment, or description of a resource.

Purpose (of meta-information**)** (OGC 07-097; RM-OA 2007)

Describes the goal of the usage of the resources.

(Service) Platform (OGC 07-097; RM-OA 2007)
Set of infrastructural methods, technologies and rules that describe how to specify service interfaces and related information and how to invoke services in a distributed system.
Examples for platforms are Web Services according to the W3C specifications including a GML profile for the representation of geographic information or a CORBA-based infrastructure with a UML profile according to the OMG specifications.

Principal
See Identity

Profile
Information (set of attributes) describing a subject.

Reference Model (ISO Archiving Standards; http://ssdoo.gsfc.nasa.gov/nost/isoas/us04/defn.html)
Framework for understanding significant relationships among the entities of some environment, and for the development of consistent standards or specifications supporting that environment. A reference model is based on a small number of unifying concepts and may be used as a basis for education and explaining standards to a non-specialist.

Representation (Richardson/Ruby 2007)
Comprises any useful information about the current state of a resource.

Resource (Richardson/Ruby 2007)
Anything that's important enough to be referenced as a thing itself.

Note: Applied to geospatial service-oriented architectures (derived from OGC 07-097; RM-OA 2007): Functions (possibly provided through services) or data objects (possibly modelled as features).

Security Domain
Set of resources protected in accordance with a common policy.

Sensor
Entity that provides information about an observed property at its output. A sensor uses a combination of physical, chemical or biological means in order to estimate the underlying observed property. At the end of the measuring chain electronic devices produce signals to be processed.

Note: A more detailed discussion about simple and complex forms of a sensor as well as sensor systems, also in the context of environmental models, is given in section 5. Here a sensor model is presented according to several viewpoints.

Sensor Network
Collection of sensors and optional processing nodes, in which information on properties observed by the sensors may be transferred and processed.

Note: A particular type of a sensor network is an ad hoc sensor network.

Sensor Service Architecture (SensorSA)
Open Architecture comprising a platform-neutral conceptual specification of the architectural components of a service network that includes the access to and the management of sensors, sensor networks and sensor-related information.
Sensor Service Network
Service network that is compliant to the SensorSA specification.

Sensor System
System whose components are sensors. A sensor system as a whole may itself be referred to as a sensor with an own management and sensor output interface. In addition, the components of a sensor system are individually addressable.

Service (ISO 19119:2005)
Distinct part of the functionality that is provided by an entity through interfaces.

Service Instance (derived from OGC 07-097; RM-OA 2007)
Executing manifestation of a software component that provides an external interface of a service according to an implementation specification for a given platform.

Service Network (derived from OGC 07-097; RM-OA 2007)
Set of service instances that interact in order to serve the objectives of applications. The basic unit within a service network for the provision of functions are the service instances.

Session
Temporarily valid ticket

Signal
Any internal representation (i.e. internal to the sensor) of the observed property.

Software Component (derived from component definition of http://www.opengeospatial.org/resources/?page=glossary)
Program unit that performs one or more functions and that communicates and interoperates with other components through common interfaces.

Spatial Context
Specification of a spatial location of an observed property determined by a combination of a point, a line, an area, a volume and/or a vector field.

Note: As an example for the combination of an area and a point, consider a sensor that is capable of recording an image of an area. It may deliver both a spatial context for the area (e.g. the polygon of the area) and/or for several points within that area (e.g. a grid laid upon the area).

Subject (OGC 07-097; RM-OA 2007)
Abstract representation of a user or a software component in an application.

System (ISO/IEC 10746-2:1996)

Something of interest as a whole or as comprised of parts. Therefore a system may be referred to as an entity. A component of a system may itself be a system, in which case it may be called a subsystem.

Note: For modelling purposes, the concept of system is understood in its general, system-theoretic sense. The term "system" can refer to an information processing system but can also be applied more generally.

System User (OGC 07-097; RM-OA 2007)
Provider of services that are used for an application domain as well as IT architects, system developers, integrators and administrators that conceive, develop, deploy and run applications for an application domain.

Temporal Context
Specification of the temporal reference of an observed property based on the absolute time. It can be a single point in time, a time sequence, a time period or a combination of these. In a sampling system for example several time periods and time points are needed to describe the time behaviour. However, a time point is already an abstraction which does not really exist. It means a very small time interval.

Thematic Service (derived from OGC 07-097; RM-OA 2007)
Service that provides an application domain-specific functionality built on top and by usage of architecture services and/or other thematic services.

Ticket
Information issued by an identity provider to be used as proof of identity when accessing a resource.

Uncertainty
Quantified description of the doubt about the measurement result.

Note: The error of a measurement may be small, even though the uncertainty is large.

Universe of discourse (ISO 19101)
View of the real or hypothetical world that includes everything of interest.

User (OGC 07-097; RM-OA 2007)
Human acting in the role of a system user or end user.

UTC - Coordinated Universal Time (ISO 19108:2004 (E))
Time scale maintained by the Bureau International des Poids et Mesures (International Bureau of Weights and Measures) and the International Earth Rotation Service (IERS) that forms the basis of a coordinated dissemination of standard frequencies and time signals (ITU-R Rec.TF.686-1 (1997))

Viewpoint (RM-ODP)
Subdivision of the specification of a complete system, established to bring together those particular pieces of information relevant to some particular area of concern during the design of the system.

Web Service
Self-contained, self-describing, modular service that can be published, located, and invoked across the Web. A Web service performs functions, which can be anything from simple requests to complicated business processes. Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.

W3C Web Service (W3C, http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice)
Software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

3. Architectural Framework

Continuing the pattern established by the ORCHESTRA project (RM-OA, 2007), an approach based on the ISO Reference Model for Open Distributed Processing (ISO/IEC 10746-1:1998) has been selected for the design of the SensorSA. Since a sensor service network will have the characteristic of a loosely-coupled network of systems and services instead of a "distributed processing system based on interacting objects" as presumed by RM-ODP, the RM-ODP concepts are not followed literally. The usage of RM-ODP for the design and documentation of a sensor service architecture is two-fold:

1. It is applied on a big scale to the structuring of ideas and the documentation of the SensorSA itself. A mapping of RM-ODP viewpoints to the needs of the SensorSA has been carried out and is summarised in Table 3-1:

   - The second column of Table 3-1 provides the original definitions of the viewpoints as given in the OpenGIS® Reference Model using the terms of the OpenGIS® glossary.

   - The third column of Table 3-1 indicates the mapping of the viewpoints to the SensorSA needs using the terms as defined in the SensorSA glossary (see section 2.3).

     Note: In order to highlight the fact that, as in RM-OA (2007), the SensorSA deployment will have the nature of a loosely-coupled distributed system based on networked services rather than a distributed application based on computational objects, the "computational viewpoint" is referred to as "service viewpoint" in the SensorSA.

   - The fourth column of Table 3-1 provides examples of what will be defined in the respective viewpoint.

2. It is applied on a small scale to the description of the sensor model in section 5. Here, the SensorSA understanding of a sensor is considered from the perspective of the five RM-ODP viewpoints. By this approach the multi-fold facets of the term sensor may be captured.

Note: Without further qualification, the usage of viewpoint names always refers to the viewpoint description of the SensorSA according to its interpretation in Table 3-1. In this case, the viewpoint name is always capitalised. As an example, Service Viewpoint means a reference to section 8 of this document.

View-point Name	Definition according to ISO/IEC 10746	Definition according to the OpenGIS® Reference Model	Mapping to the design process of the SensorSA	Examples
Enterprise	Concerned with the purpose, scope and policies governing the activities of the specified system within the organization of which it is a part.	Focuses on the purpose, scope and policies for that system.	Reflects the analysis phase in terms of the system and the user requirements as well as the technology assessment. Includes rules that govern actors and groups of actors, and their roles.	Business context GMES. Use case definition for a fusion service according to the needs of a pilot scenario.
Information	Concerned with the kinds of information handled by the system and constraints on the use and interpretation of that information.	Focuses on the semantics of information and information processing.	Specifies the modelling approach of all categories of information the SensorSA deals with including their thematic, spatial, and temporal characteristics as well as their meta-information.	Information objects specified in UML class diagrams and referred to by the specification of the fusion service (e.g. as parameter types).
Computational	Concerned with the functional decomposition of the system into a set of objects that interact at interfaces – enabling system distribution.	Captures component and interface details without regard to distribution.	Referred to as "Service Viewpoint" Specifies the SensorSA Interface and Service Types that aim at improving the syntactic and semantic inter-operability between services, source systems and applications.	Specification of the externally visible behaviour of a service type, e.g. UML specification of the interface types of the fusion service.
Technology	Concerned with the choice of technology to support system distribution.	Focuses on the choice of technology.	Specifies the technological choices for the platform, its characteristics and its operational issues.	Specification of the platform "Web Services" including a Sensor ML profile. Physical characteristics of sensors and sensor networks.
Engineering	Concerned with the infrastructure required to support system distribution.	Focuses on the mechanisms and functions required to support distributed interaction between objects in the system.	Specifies the mapping of the SensorSA service specifications and information models to the chosen platform. Considers the charac-teristics and principles for service networks. Specifies policies and service interaction patterns.	Provision of the service implementation specification, incl. mapping of the UML specification to WSDL. Decisions on access control and discovery policies. Communication behaviour of sensor networks.

Table 3-1: Mapping of the RM-ODP Viewpoints to the SensorSA

4. Enterprise Viewpoint

4.1. Architectural Requirements

The architectural and iterative design approach as described in (RM-OA, 2007) is taken as the basis for the specification of the SensorSA. The architectural principles that have guided the specification of the ORCHESTRA Architecture have been adapted and assessed specifically for service networks that include access to sensors and sensor-related information. The result is presented in the following sections.

4.1.1 Rigorous Definition and Use of Concepts and Standards

The SensorSA shall make rigorous use of proven concepts and standards in order to decrease dependence on vendor-specific solutions, help ensure the openness of a sensor service network and support the evolutionary development process of the SensorSA.

Note: The SensorSA relies on the standard series "OGC Sensor Web Enablement (SWE)" (Botts et al, 2006) as the starting point to refine the access and the management of sensor-related information.

4.1.2 Loosely Coupled Components

The components involved in a sensor service network shall be loosely coupled, where loose coupling implies the use of mediation to permit existing components to be interconnected without changes.

Note: In this stringency, this architectural principle is restricted to the acquisition, application and user domains of a sensor service network (see section 6.1) as its application in the sensor domain may not be possible due to the predominance of proprietary solutions. Furthermore, performance and dependability requirements may necessitate a sensor network with fixed communication relationships and tight coupling of the sensor components.

4.1.3 Technology Independence

The SensorSA shall be independent of technologies, their cycles and their changes as far as practically feasible. It must be possible to accommodate changes in technology (e.g. lifecycle of middleware technology) without changing the SensorSA itself. The SensorSA shall be independent of specific implementation technologies (e.g. middleware, programming language, operating system). In general and if possible, the SensorSA shall not be influenced by or deal with limitations of specific implementation technologies. However, the SensorSA shall be explicitly designed such that it may deal with technical limitations of specific implementation technologies in the sensor domain (see section 6.1).

Note: Limitations of existing sensor networks must be taken into account in the SensorSA. At a minimum the characteristics of the sensor-to-sensor protocols must be considered in the meta-information (e.g. dependability, quality of service, performance). This will be assessed in future versions of this document in more detail.

4.1.4 Evolutionary Development - Design for Change

The SensorSA shall be designed to evolve, i.e. it shall be possible to develop and deploy the system in an evolutionary way. The SensorSA shall be able to cope with changes of user requirements, system requirements, organisational structures, information flows and information types in the source systems.

4.1.5 Component Architecture Independence

The SensorSA shall be designed such that a service network and source systems (i.e. existing information systems, sensors and sensor networks) are architecturally decoupled. This means that the SensorSA shall not impose any architectural patterns on source systems for the purpose of having them collaborate in a service network, and no source system shall impose architectural patterns on a SensorSA.

Note: This architectural principle relies on the RM-OA definition of a source system as being a "container of unstructured, semi-structured or structured data and/or a provider of functions in terms of services. The source systems are of a very heterogeneous nature and contain information in a variety of types and formats". In the context of a sensor service network, a source system may also be a sensor or a sensor network to be integrated in a service network. Important here is that a source system is seen as a black box, i.e. no assumptions about its inner structure are made when designing a service network.

4.1.6 Generic Infrastructure

The SensorSA services shall be independent of the application domain. This means that the SensorSA services shall be designed in such a flexible and adaptable way that the SensorSA services can be used across different thematic domains and in different organisational contexts, and that the update of integrated components (e.g. sensors, applications, systems, ontologies) causes little or ideally no changes to the users of the SensorSA services.

4.2. Relationship to the ORCHESTRA Architecture

The architectural heritage of the SensorSA is the ORCHESTRA Architecture specified in (RM-OA, 2007) as an "open architecture that comprises the combined generic and platform-neutral specification of the information and service viewpoint as part of the ORCHESTRA Reference Model" (RM-OA). Consequently, the SensorSA builds upon the components of the ORCHESTRA Architecture which comprise:

- Rules and guidance about how to specify information and service models in the context of international standards. In the RM-OA, these rules are formally specified in an ORCHESTRA Meta-model[2] for information and for services.

[2] The ORCHESTRA Meta-model is an extension of the ISO/OGC-defined General Feature Model. It consists of a set of UML classes stereotyped as <MetaClass> and associated textual rules that provide guidance on how application schemas and services have to be specified in UML, i.e. on a platform-neutral abstract level. It encompasses two parts: one for the specification of information models in the form of application schemas, and one for the specification of interface and service types.

- Basic re-usable specification units for information models (e.g. pre-defined feature types[3]) and service models (e.g. re-usable interfaces).

- Textual and document templates that guide the specification of services.

- A series of textual descriptions and formal specifications of generic services (so-called ORCHESTRA Architecture Services or, in short, architecture services) that are application- and technology independent.

It is the objective of the SANY project to re-use and apply these components in the context of the SensorSA to the extent that they fulfil the requirements of building service networks[4] based on sensor information and higher-level sensor information processing.

Thus, there are two cases:

1. The SensorSA contributes to the architectural level in the sense that it specifies new architectural concepts or new architecture services, or it refines existing architecture services.

2. The SensorSA specifies new application-oriented services (in the RM-OA categorised as ORCHESTRA Thematic Services or, in short, thematic services) that typically require the call of underlying architecture service operations. In this case the SensorSA defines a so-called ORCHESTRA Application Architecture[5].

In both cases, the SensorSA uses the templates and the rules of the RM-OA in order to specify the refined and the new components.

4.3. Requirements of GMES

The GMES initiative was launched in 1998 and adopted by the ESA[6] and EU councils in 2001. It started with an initial exploratory period from 2001 to 2003. It was followed by a concrete implementation period that started in 2004 and ended in 2008. This phase is now followed by a validation phase of the first three fast-track services (Emergency Response, Land Monitoring, and Marine services).

As illustrated in Figure 4-1 GMES consists of four major components:

- The space component providing earth observation satellite data.

[3] In accordance with ISO and OGC, the RM-OA defines a feature as an "Abstraction of a real world phenomenon" (ISO 19101). Note, however, that the ORCHESTRA understanding of a "real world" explicitly comprises hypothetical worlds. Features may but need not contain geospatial properties.

[4] In analogy to (RM-OA, 2007) a service network here is defined to be set of networked hardware components and service instances that interact in order to serve the objectives of applications. Thus, the basic units within a service network for the provision of functions are the service instances, i.e. executing manifestations of software components that offer their functionality in terms of services.

[5] The RM-OA defines an ORCHESTRA Application Architecture to be an "instantiation of the ORCHESTRA Architecture by inclusion of those thematic aspects that fulfil the purpose and objectives of a given application." The concepts for such an application stem from a particular application domain (e.g. a risk management application).

[6] European Space Agency

- The in-situ component providing ground and airborne data as well as socio-economic data.

- The data integration and information management component providing access to and processing of the above space and in-situ data.

- The high level user services component allowing for monitoring of the various aspects of our environment.

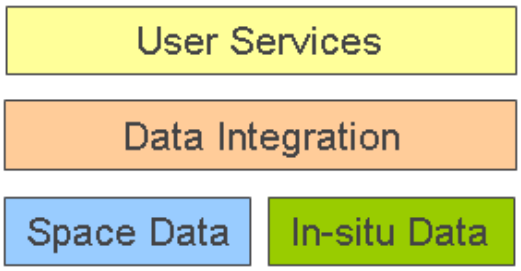


Figure 4-1: Major GMES Components

The final report for the GMES initial period (GMES FR, 2004) provides the following key architectural requirements on pages 25-26:

"For GMES to become a success, the architecture needs to facilitate the integration of standalone data and information elements. It should allow to the selection and aggregation of information from heterogeneous sources and should provide the capability to translate data and information between the various sources in real time. This applies as much to the incorporation of socio-economic data and information, as well as products derived from the space and in situ observing networks.

GMES must therefore provide a structured framework for data integration and information management, i.e., a European shared information capacity. The following key architectural and user-oriented requirements will therefore drive the implementation of GMES:

- *Openness, based on agreed open standards, facilitating seamless communication and interoperability, i.e. the ability of different devices or systems (usually from different vendors) to work together, as well as enabling user service autonomy;*

- *Federated architecture, enabling systems to grow and evolve;*

- *Simplicity of architecture (e.g. modularity of components), to break the complexity barrier, systems must be made easier to design, administer and use;*

- *Self-configuration, programmability, scalability (e.g. to handle various levels of operational load and external conditions);*

- *Dependability, i.e. the system's resilience to security threats or breakdown;*

- *User-friendliness of services and interfaces, e.g. in the handling of user request services, access control, workflow management, delivery management, visualisation, data extraction (e.g. "multilinguality"), multiuser sessions, administration;*

- *Data security, protection of provider and user data against alteration, theft and misuse;*
- *Quality of service;*
- *Ubiquity of access, including global reach."*

In a reflection paper on Data Integration and Information Management Capacity (GMES, 2005), prepared by DG-INFSO in close collaboration with ESA and JRC and IST Integrated Projects of the 6th Framework Programme, the following provisional and non-exhaustive list of functional requirements is given:

"1. Selection of information from heterogeneous sources

a. *Publishing*
b. *Discovery (data/information and services)*
c. *Catalogue (search, browse, etc.)*
d. *Tasking*
e. *Ordering*
f. *Access to data*
g. *Data/information Mining*

2. Aggregation of information from heterogeneous sources

a. *Data/information fusion*
b. *Map overlay*

3. Translation of data and information between various sources in real time

a. *Geo-referencing and re-projection*
b. *Feature translation*
c. *Semantic interoperability*
d. *Multilingual*
e. *Data formatting*
f. *Data generalisation*

4. Others

a. *User management and Security (including Terms and Conditions to access GMES Services)*
b. *Service level agreement*
c. *Quality (for metadata[7], data and services)"*

[7] As this text is cited from (GMES, 2005), the term "metadata" is used here whereas in the SANY Service Architecture the term meta-information is used in order to indicate that it is understood as interpreted data for a specific purpose (e.g. discovery).

The same reflection paper suggests that the approach needed to create an efficient data integration and information management component is to use a "system of systems" design as explained below:

"The GMES information and services infrastructure goes well beyond a simple exchange of data. It addresses the need to integrate business and workflow processes which span across boundaries, be they political, administrative or thematic. It addresses the main issues of heterogeneity and fragmentation of the information: "heterogeneity" meaning that the same information is often represented differently, "fragmentation" meaning that needed information is spread over multiple locations.

The approach is to seamlessly integrate existing systems into a "system of systems" perspective. System of systems is an emerging design and development method of complex systems build from large scale component systems. The subsystems that comprise the system of systems are generally built by different organisations, having different goals, are very often built to different quality standards, and are managed independently.

Reusing existing legacy systems in a dependable fashion without the need for extensive re-engineering is a key problem currently faced by industry. System of systems can be seen as new systems linking data, services and workflows to produce new data, new services together with metadata (information about the information products generated[8]).

This approach, which was a research topic in the last decade, is now becoming sufficiently mature. In the short term, robust system architectures will be developed and tested allowing the exchange of data and services that are well identified: This is the syntactic interoperability phase. In the longer term, we expect to achieve significant semantic interoperability, which will allow cross-system search for data and services. It could be dubbed the "GMES Google" since it will work for data and services in the way web search engine works for web pages.

A number of non technical obstacles should however be addressed to eventually reduce the complexity of the implementation, such as trusted electronic billing principles between data providers and services providers to partly overcome the hurdles created by different data policies. Mechanism should also be agreed upon to manage access privilege across institutional borders in a practical, transparent and secure way.

This approach will allow sharing and efficient management of information that is consistent across organisations, borders and thematic domains such as from land use and mapping to risk management and security."

The suggested system of systems approach is still regarded today as the best approach as indicated in the following excerpt of the report of Joint Operability Workshop held in April 2007 (JOW, 2007):

"There was broad agreement that any proposed solution for a single information space for the environment must allow for multiple architectures and hence be a "system of systems". Further comments on this topic included:

[8] In the SANY Service Architecture called "meta-information".

- *Hierarchical architecture is required to cope with services at various levels: EO ground segment services, GMES core services, GMES downstream services etc.*

- *SOA is well suited for deployment of system of systems.*

- *Significant investments have been made already by ESA, ECMWF[9] etc. Therefore it is unrealistic that everybody will adopt a single architecture. This leads to a system of interoperable systems which may be based (internally) on heterogeneous technologies.*

- *An exhaustive list of use cases is unfeasible – the number of possibilities is too large and is changing rapidly. Thus, a flexible architecture is needed which also leads to the concept of a system of systems. The fully top-down approach and pan-European architecture that was anticipated as recently as five years ago appear now to not be feasible. The concept is being replaced by pan-European interoperability in a system of systems-like architecture.*

- *Many systems are legacy systems in this field as satellites have a relatively short life and scientific advances in EO are rapid. It is totally unfeasible to convert all existing data so they all have the same data and metadata formats. Any viable solution must be based on fully utilising legacy systems. Harmonised, "on the fly" access to legacy datasets is required.*

- *Access to data from heterogeneous missions must be seamless and transparent."*

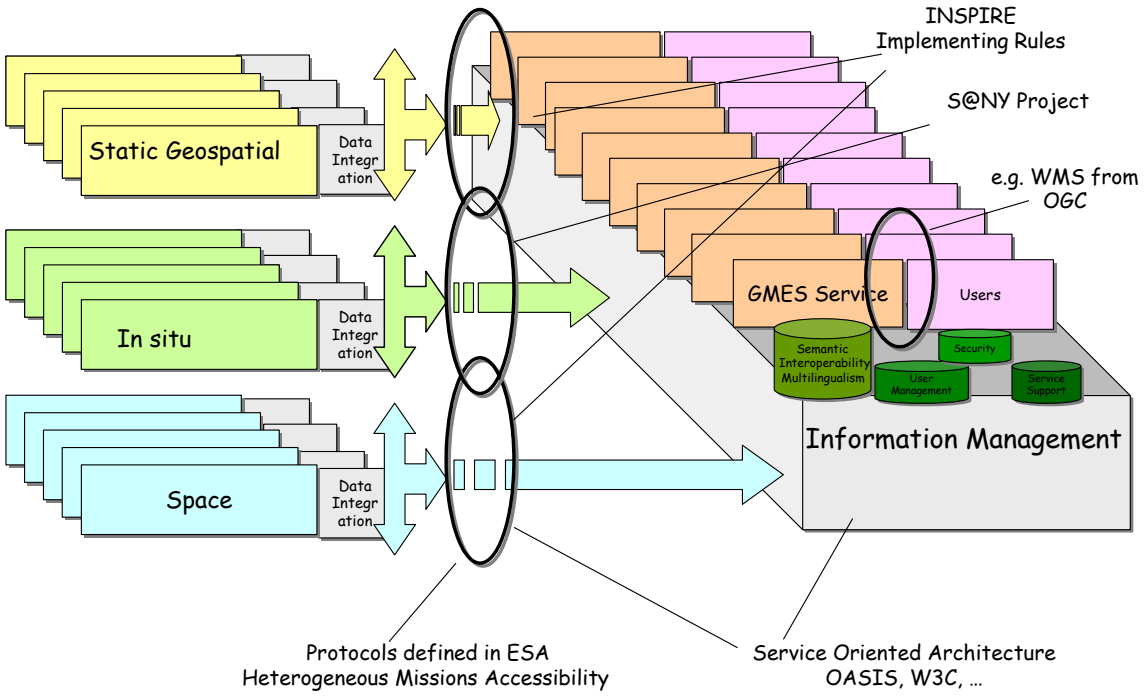


Figure 4-2: Basic Conceptual Representation of GMES

The reflection paper mentioned above on Data Integration and Information Management Capacity (GMES, 2005) provides the following basic conceptual representation of the four components of GMES (Figure 4.3, which was slightly enhanced to show projects and standards related to interfaces).

[9] European Centre for Medium range Weather Forecasting

The role of the SANY project is to define an open architecture that addresses all types of in-situ sensor networks so that in-situ data can be integrated with space data to form a Data Integration and Information Management component/layer that will enable user access and sharing of environmental information.

Figure 4-3 depicts the vision of the ESA Earth Observation Ground Segment Program Board showing the GMES architectural aspects related to the space ground segments.

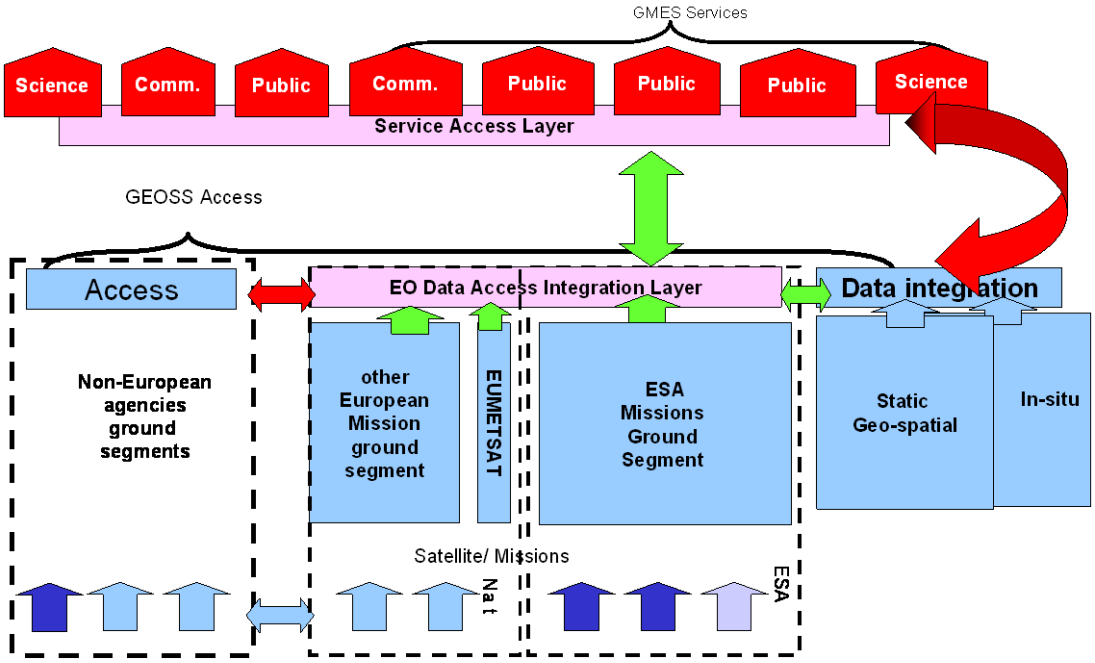


Figure 4-3: Ground Segment Program Board Vision GMES

As suggested in this figure, the Data Access Integration Layer could be extended to include the access to in-situ data which would provide a common infrastructure on which GMES services could be built.

As recommended in the following extract of the reflection paper on Data Integration and Information Management Capacity (GMES, 2005), the technology of choice to be used to implement the GMES architecture is SOA:

"*The underline architecture for ensuring interoperable GMES services is the Service Oriented Architecture (SOA).*

Modern architectural designs are based on a less tightly coupled collaboration of distributed services. For this, the term SOA is becoming widely used, but there is not a lot of precision in the way that it is used. The World Wide Web Consortium (W3C) for example refers to SOA as 'A set of components which can be invoked, and whose interface descriptions can be published and discovered'. This is broad definition highlights a key aspect of an SOA: components (e.g. functionalities) can be discovered and invoked dynamically. Thus this type of architectures is a motion away from "hard wiring" of business processes, which makes change to new circumstances very difficult. SOAs are thus inherently more flexible and adaptable than most approaches.

A service-oriented architecture is also proposed for the following reasons:

- *A SOA represents business functionality as implementation- and vendor neutral and standards-based shared services.*

- *Existing legacy systems can be extended with service interfaces, and in that way become part of a SOA.*

- *By means of the inherent service discovery and access flexibility (see above), a SOA enables GMES/INSPIRE service providers to be more agile and to respond more quickly to changing business needs and evolving requirements.*

- *A set of common generic services can provide standard, domain-independent functionality (for discovery, search, navigation, data access, authorisation etc.) which only needs to be implemented once.*

- *Sharing of services — no need to "re-invent the wheel"*

- *Loose coupling — ability to update applications with minimal effect on services that invoke them*

- *Location transparency — ability to re-host applications with minimal effect on services that invoke them*

GMES applications built on top of a joint, adopted infrastructure will be interoperable and much easier to integrate into multi-purpose, cross-application operations."

As a consequence, the SensorSA shall adopt a service-oriented architecture based on open standards in order to contribute to an emerging GMES infrastructure which demands flexibility, stability, and durability while preventing vendor lock-in.

There are a number of projects, both concluded and ongoing, that are providing major contributions in terms of architectures suitable for the GMES infrastructure, for example the FP6 Integrated Projects OASIS, ORCHESTRA, and WIN on the EC side and HMA on the ESA side. All of these projects use the ISO RM-ODP methodology which is also adopted by the OGC Reference Model (OGC 03-040).

4.4. Requirements of GEOSS

The purpose of the Global Earth Observation System of Systems (GEOSS) is to build on and add value to existing earth observation systems to achieve comprehensive, coordinated, and sustained observations of the whole planet. It is a "system of systems" that will facilitate the sharing of observations and derived products obtained by the cooperating earth observation systems for the benefit of a broad range of user communities around the globe.

GEOSS is being developed by the Group on Earth Observations (GEO) which includes 73 Governments and the European Commission. In addition, 51 intergovernmental, international, and regional organizations with a mandate in Earth observation or related issues have been recognized as participating organizations (status: June 2008, source: http://earthobservations.org/about_geo.shtml). GEO has established a GEOSS 10-Year Implementation Plan (GEOSS 1000R, 2005) that was adopted at the 3rd Earth Observation Summit in Brussels in February 2005.

GEOSS addresses the requirements of users in the following nine societal benefit areas as briefly summarised below:

- Disasters: Reducing loss of life and property from natural and human-induced disasters
- Health: Understanding environmental factors affecting human health and well-being
- Energy: Improving management of energy resources
- Climate: Understanding, assessing, predicting, mitigating, and adapting to climate variability and change.
- Water: Improving water resource management through better understanding of the water cycle
- Weather: Improving weather information, forecasting, and warning
- Ecosystems: Improving the management and protection of terrestrial, coastal, and marine ecosystems
- Agriculture: Supporting sustainable agriculture and combating desertification
- Biodiversity: Understanding, monitoring, and conserving biodiversity

GEOSS is a federated system which is assembled from components contributed by GEO members and participating organisations. These components are basically used to acquire observations, to process observation data into products, and to exchange/disseminate these observations and products.

One of the key aspects of GEOSS is therefore the interoperability between the various and numerous components which should be based on open international standards.

The architecture of GEOSS is being defined under the supervision of the Architecture and Data Committee (ADC) which has provided high level strategic and tactical guidelines for the implementation of GEOSS. One of the tasks on the first two-year work plan of the ADC was to conduct a GEOSS Architecture Implementation Pilot to test and validate architectural aspects of GEOSS. The first phase of the AIP has been successfully conducted in 2007 with 75 registered services. A second phase is currently in progress and should be completed by end of May 2009.

The Architecture Implementation Pilot specification adopts the structure of the RM-ODP viewpoints. The computational viewpoint proposes a service-oriented architecture (SOA) approach in which components/services interact through well defined interfaces based on open standards. The services are further organized into three tiers: a lower tier providing access to various types of data, a middle tier providing business processes acting on these data, and a top tier providing user interfaces for users consuming the data.

The engineering viewpoint further identifies component types consistent with the enterprise and computational viewpoints as shown in Figure 4-4.

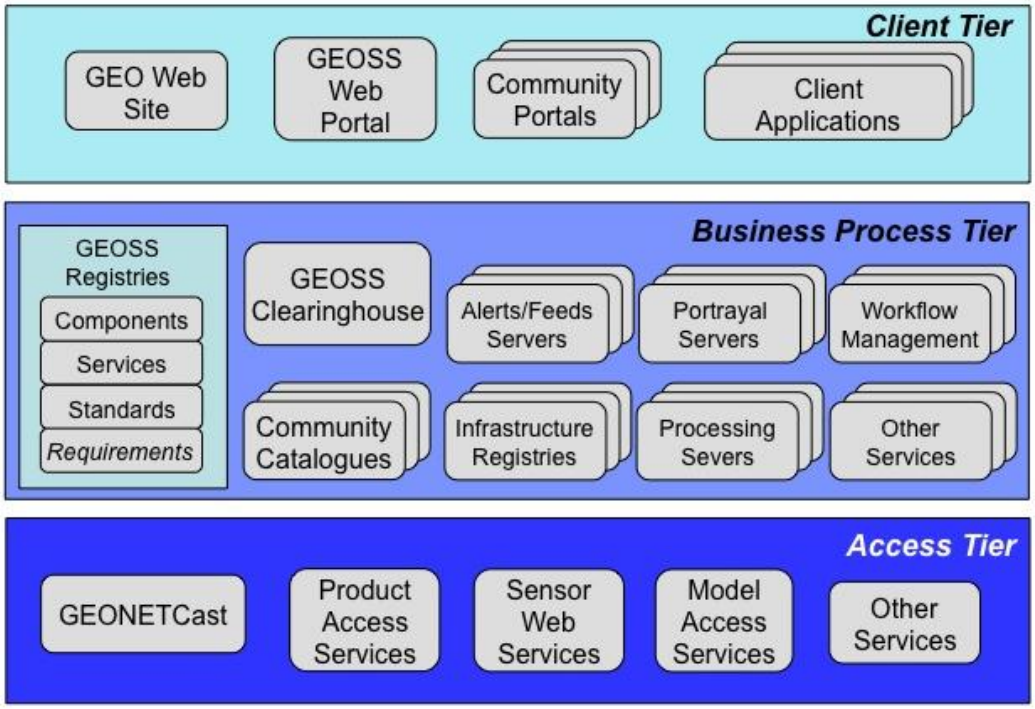


Figure 4-4: GEOSS Architecture – Engineering Viewpoint (GEOSS AIP CFP, 2008)

The results of the Architecture Implementation Pilot (Phase 1) are documented in the GEOSS Core Architecture Implementation Report (GEOSS CAIR, 2007) that was issued in November 2007. The Core Architecture is composed of the following components: GEO Web Portal, GEOSS Registries, and GEOSS Clearinghouse.

As highlighted in this report and illustrated in Figure 4-5, the GEOSS interoperability process follows a publish-find-bind pattern supported by several registries where components, services, and standards are registered. The role of the Standards and Interoperability Forum (SIF) is to facilitate the establishment of interoperability arrangements (standards or special).

The GEOSS Core Architecture Implementation Report provides a list of candidate interoperability arrangement standards for services and encodings. Most of these services and encodings refer to OGC specifications including some SWE services. Other services and encodings refer to OASIS specifications such as UDDI, WS-Notification, and BPEL.

The above services are also listed in section 4.2 of Annex B of the Call for Participation document (GEOSS AIP CFR, 2008) for the Phase 2 Architecture Implementation Pilot that was issued in June 2008.

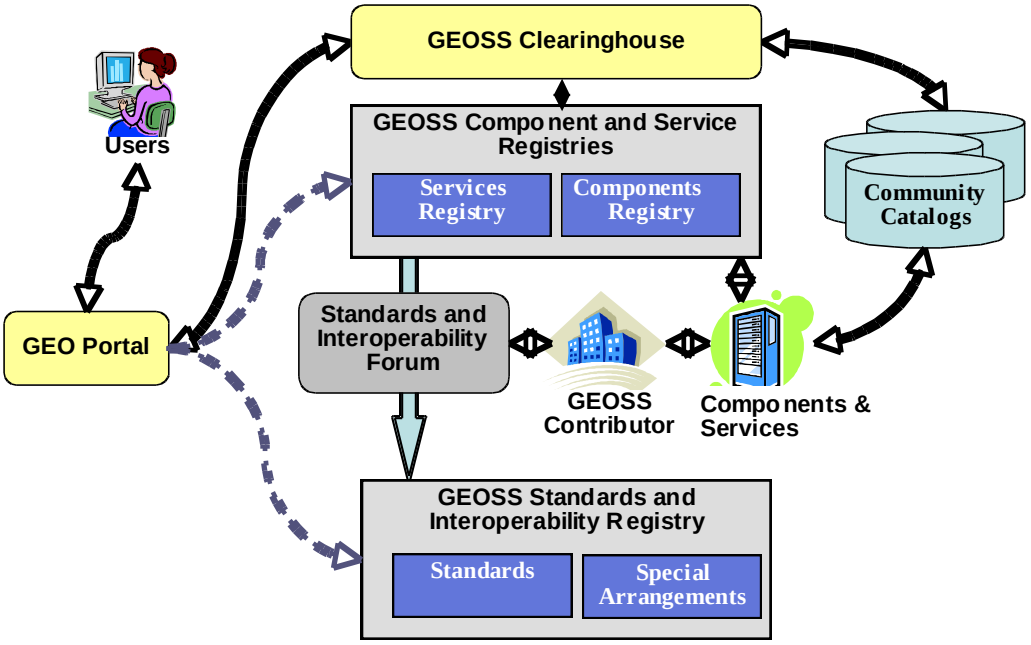


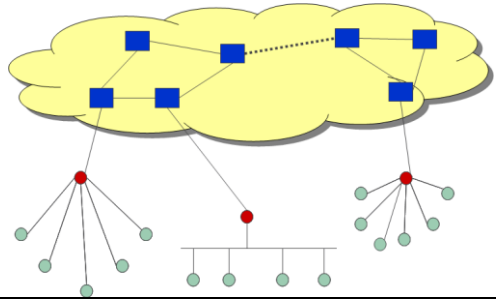
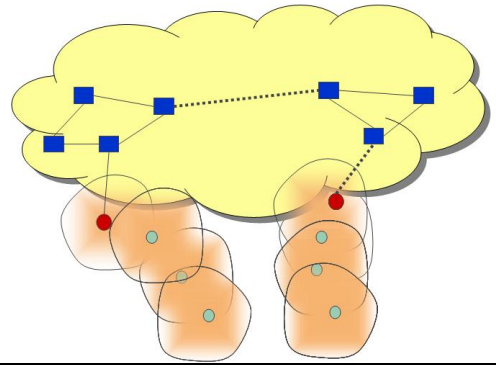
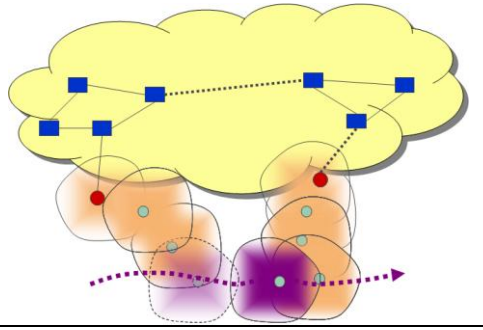
Figure 4-5: GEOSS Interoperability Process (from GEOSS CAIR, 2007)

The architectural requirements of GEOSS are in many ways similar to the architectural requirements encountered in GMES and INSPIRE (e.g. system of systems). As a matter of fact, as documented in The First 100 Steps to GEOSS document (GEOSS 100S, 2007), both GMES and INSPIRE are expected to provide important EU contributions to GEOSS. The document also highlights the contribution that some FP6 and FP7 EU projects, including ORCHESTRA and SANY, could make in support of building GEOSS.

4.5. Requirements of Sensor Networks

This section summarizes several common sensor network scenarios of (Watson/Kunz, 2007) and addresses the issues of network topology, communication, and information flow and processing. The specific requirements form the overall design approaches implemented within SANY. A sensor network is hereby understood as a collection of sensors and processing nodes in which information on properties observed by the sensors may be transferred and processed. A sensor network may be of an ad hoc nature. In this case communication links and/or nodes are not continually available or might change dynamically. An ad hoc sensor network is often, but not necessarily, based on wireless communication between nodes with limited resources (energy supply, processing power). It may include mobile sensors belonging to the network for a limited time or intermittently.

The different scenarios mainly distinguish stationary and mobile as well as wireless and wired sensors. These aspects are described in Table 4-1. They determine the adequacy of communication patterns and information flows.

No	Scenario	Topology	Communication	Information flow	Graphic
1	Sensors and data logger with fixed locations	Wired sensor networks with sensor nodes, data logger and central computer systems Sensor localization information provided externally	Sensor nodes communicate with data logger No intra-sensor node communication Wired connections; high bandwidth	Sensor nodes report observation to data logger Data logger provides necessary meta-information Data logger reports to central computing system	
2	Mobile sensors and fixed or mobile data logger	Mobile sensors with onboard GPS or other localization option Data logger mobile or fixed	Wireless link between sensor node and data logger Energy restrictions; bandwidth limitations	Data transmission energy-optimized Pre-processing/transfer ratio important	
3	Mobile sensors moving in different sensor networks	Sensor nodes migrate across networks boundaries	Sensor nodes adapt to data loggers dynamically	Central computing system needs to merge data from mobile sensors	

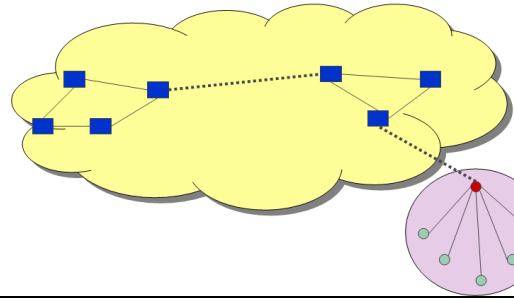
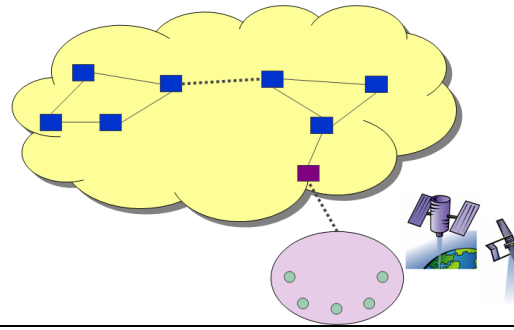
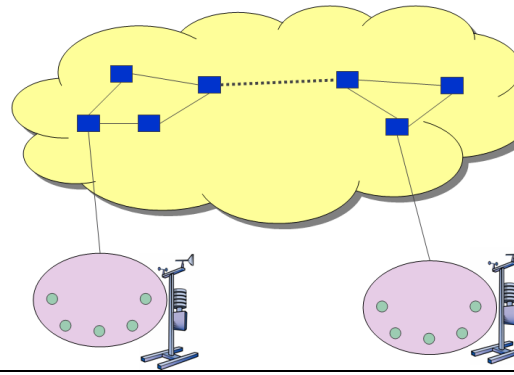
4	Mobile sensor cluster on vehicles (e.g. on ships) - block data transfer on demand	Sensor nodes and data loggers mounted on mobile platforms Platform devices not permanently connected to central computing system	Massive transfer in short time periods required	Pre-processing important	
5	Mobile earth observation sensors (satellite, airborne)	Remotely observing sensors Localization mainly calculated	Direct link between sensor and ground segment Several providers may provide access to data	Raw data repository often not accessible	
6	Mobile sensors with their own IP address	Sensors directly connected to the Internet Sensor node unambiguously identified by IP address	Permanent access via Internet	Direct data flow between sensor and Internet node Security settings by sensor owner	

Table 4-1: Overview about Sensor Network Topologies

4.6. User Requirements

4.6.1 Overview

This section summarizes the use cases of the application processes originally defined by the SANY application subprojects addressing different application domains:

- **Air Pollution Risks**
  - Management of plant pollution
  - Sophisticated data control to detect suspicious data
  - Conduct impact study
  - Merging data of different types
  - Odour measurements by field inspection and impact surveillance by real-time modelling
- **Marine Risks**
  - Oil Spill Trajectory Forecast Scenario
  - Bathing Microbial Risk and Beach Management Scenario
  - Short Term Ship Collision Risk Management Scenario
  - Long Term Ship Collision Risk Management Scenario
- **Geo Hazards**
  - Monitoring of the area around a tunnel construction
  - Sensor Network Management
  - Settlement Monitoring
  - Landslide Monitoring
  - Risk Zone Mapping
  - Rainfall Influence on Landslides

An analysis of the use cases has led to a set of requirements that have been grouped into functional blocks as described in the following sub-sections. The specific sensor network requirements are described in the SANY deliverable "Sensor Scenarios and Requirements" (Watson and Kunz, 2007) with references to the pertinent application processes and the use cases.

The requirements within the blocks are summarised below. Here, the term "SANY system" stands for the entirety of a sensor service network including its architecture and its hardware and software components.

Note: The SensorSA only covers a subset of these requirements. The coverage and the tracing of the requirements is documented in (Schimak and Watson (eds.), 2008).

4.6.2 Sensor Network

- **Plug & measure type of operation is required. As a requirement it is understood as** the degree of capability to add a new sensor node to a sensor and sensor service network without a manual re-configuration of the sensor network or sensor node.

- **Dependability** is required to provide data access and management services, in order to cope with the dynamic availability of possibly redundant sensor data sources, especially in the case of mobile sensors.

- **Sensor Network management**. Of particular interest here is the localisation of sensor nodes, e.g. for the planning and management of their deployment or the configuration of the measurement frequency in order to optimise network and battery load.

- **Deployment of mobile ad hoc sensor clusters.** Especially in the case of biological and chemical hazards, the responsible administration authority needs to measure air pollution or water quality in order to quickly assess the risk situation. However, in the affected area appropriate sensors are often absent.

- **Self-validation of sensor nodes** with regard to residual battery life and measurement capability (need for re-calibration or maintenance) is important for the assessment of node deployment and data quality.

- **Battery life optimisation** through selective data transmission is a necessary management capability to access the battery information via an interface to sensor nodes with self-diagnosis. It shall support the capability to automatically select alternative transmission routes for data transmission and/or the frequency of data transmissions, if the residual battery level of a sensor is too low.

4.6.3 Data and Information

- **Data sources** do not only include sensors and databases of archived data, but also data obtained from a laboratory analysis of samples, or data entered manually by humans. Data sources may also be results of fusion services.

- **Spatial and temporal information.** The SANY system shall be capable of associating the measurement with its spatial (sensor location; feature of interest) and temporal (measurement time; interpretation; reporting times) context.

- **Data type.** The SANY system shall operate with different data types (e.g. fields, coverages) associated with such measurement data as:

    - a single sensor measurement observation represented by a single value and unit located at a fixed location and detected at a particular time (time, date and year). The observation shall distinguish sampling time and result time.

    - a series of sensor measurement observations represented by a time series of triples {value and unit, feature of interest, time}. The time representation may be either absolute time (time, date and year) or a relative time representation (e.g. every day at midday starting from a certain date)

- **Geographic objects of several types.** The SANY system shall be capable of handling geographic objects of several types (e.g. the types specified by the ISO 19107 and ISO 19123 such as MultiPoint, MultiLineString, MultiPolygon, LineStrings, Polygons, MultiCurve, MultiSurface).

- **3D fields.** The SANY system shall be capable of representing and processing three-dimensional measurement data of an observed property (e.g. wind velocity, water current etc.)

- **Images** (e. g. IR/UV, SLAR, INSAR). As an example, such image data may be delivered by an earth observation sensor mounted on an aircraft or on a satellite with a certain spatial resolution.

- **Maps.** The SANY system shall be capable of handling a wide spectrum of maps (topography, roads, land usage etc).

4.6.4 Data Quality

- **Data quality.** To assess the quality (e.g. the measurement uncertainty) of delivered measured data some information about certain quality details (e.g. accuracy, tolerance, resolution, drift) has to be available. The SANY system shall be capable of accessing and using this sensor information.

- **Sensor level spatial and temporal uncertainty.** The SANY system shall be able to describe data quality levels depending on:

    - different available sensor data sources selected in an area of interest,
    - estimated data uncertainties (e. g. an interpolated temperature result at a given location depends on the distance to available sensors and on topological conditions), and
    - the time period and frequency of measurement observations (limited, for example, by a certain availability frequency of EO satellite images of an area).

- **Certification of data and its propagation.** The SANY system shall be capable of supporting the process of formally certifying data quality, e.g. by instructing a certified laboratory per email regarding an investigation of new microbial sampling at a specific beach.

- **Associate data quality with measurement context (validation).** The SANY system shall be capable of associating and storing data quality together with measurement context as meta-information, such as the spatial or temporal context. It shall be capable of visualizing this quality meta-information in its spatial and temporal context, e.g. in order to support and optimise the deployment of sensors.

4.6.5 Security

- **Authorized Access.** The provision of sensor data implies a large investment in equipment and in supporting services. Therefore sensor data normally cannot be

offered free of charge. Access control is the underlying mechanism for all use cases that require conditional data access like licensing and billing. The SANY system shall provide several security related services that facilitate the enforcement of access control policies:

  - Authentication (verification of user identities, support for multiple authentication mechanisms)
  - Authorisation (support of roles, authorisation for services, authorisation on the data level, etc)
  - User Management (storage and management of user profiles)

- **Data Integrity.** Another security topic is protection of measurement data against manipulation. In order to ensure data integrity cryptographic measures like digital signatures may be necessary at different operational levels (e.g. during transmission, storage in databases etc.).

- **Flexible security architecture.** The overall security architecture shall be able to integrate security measures without changes to the architecture as security requirements depend heavily on concrete use case requirements. For example, it may be a security problem to allow certain users of the SANY system to track sensor data over a period of time (e.g. a shipping company does not want competitors to know the exact routes of their ships).

4.6.6 Processing and Fusion

- **Interfaces for data processing services development.** The SANY system shall provide general data processing services such as merging data and extracting relevant data for reports. In addition, the SANY system shall be capable of handling meta-information. This includes the storage of intermediate information to assure that overall services (and service chains) can execute with acceptable speed.

- **Image analysis and feature extraction.** The SANY system shall be capable of processing images and extracting features (e.g. such as a road or a watercourse).

- **Homogenisation of spatial and temporal measurement resolution** The SANY system shall be capable of handling and adjusting different temporal and/or spatial resolutions of sensor data, e.g. through data interpolation and aggregation. Furthermore, the SANY system shall cope with missing values in data series.

- **Fusion of measurements of same phenomenon.** The SANY system shall be capable of processing and merging measurements of the same phenomenon using different sensor equipment. The information about the source shall be stored in meta-information. The different data, possibly generated by the equipment at different times, shall be processed using a fusion service.

- **Library of algorithms as (statistical) processing services.** The SANY architecture shall provide a mechanism for defining re-usable processing services, e.g. for data interpolation and data fusion.

- **Workflow.** The SANY system shall be capable of processing service chaining by workflows. The output results of a model service may be connected to the input of another subsequent process.

- **Visualisation.** In many use cases several services of visualisation are described, e.g. maps or diagrams. The SANY system shall offer flexible means to visualise data in different styles.

4.6.7 Events, Alerts and Alarms

- **Threshold surpass detection.** The SANY system shall be capable of detecting threshold surpassing. For instance in the field of air quality measurement, excess pollution of the environment can be asserted if sensor measurement data have reached defined thresholds.

- **Alert algorithms.** Typically, an alert will result if a combination of observed variables is no longer in a defined region (the event of variables departing from this region). Alerts cause application level procedures and/or workflows to be executed as a reaction to the event.

- **Interfaces for alarm management.** Alarms involve communication procedures with the emergency management agencies or the public to warn about an imminent hazard and to initiate emergency procedures (such as evacuation).

  Note: Although used here in the requirements section, the term “alarm” is not used in the SensorSA as a distinguished architectural term. It is considered to be a special type of an alert (see section 6.3.3).

- **Tracing.** Tracing requirements address the need to document what information sources were used as a basis for the decisions taken and the decision making process itself. The purpose is to be able to provide a retrospective justification of decisions made, which may later be contested by parties seriously affected.

- **Models.** The SANY system shall provide in general

  - a model service catalogue, where all available services may be selected by the user
  - an execution management service handling input/output and, optionally if expedient for performance reasons, data by reference
  - the capability of using the output of models and fusion algorithms as sensor values without changing the system.

- In addition to basic support for models and model wrapping (to integrate the models into the SANY service network), the following model-specific functionality is required:

  - Gathering of applicable source impact models
  - Domain skills compilation
  - Library of models as processes

- Library of geo-statistical analysis as processes
- Predictive models for adaptive sampling
- Spill advection & dispersion modelling
- Forecasting risks (water quality, bathing water, beach closed)
- Modelling long-term degradation of ecosystems
- Improved soil models

4.6.8 Decision Support

- **Providing supporting information for decisions.** The types of decisions to be taken imply requirements on the modelling, fusion and visualization of information as well as on auxiliary methods to compute utility functions and to undertake multi-criteria analysis.

4.6.9 User Management

- **User Registration.** The registration of a "new" user has to be supported, and the user account shall be verified by an administrator (accept/decline).

- **User Administration.** The administration of user accounts, including the selection, creation, deletion as well as the update of user related information, shall be supported. This includes management of user profiles (sets of attributes related to identities). A list of predefined profile attributes shall be established.

- **Policy Administration.** Permission assignment and removal on the user, group and role levels shall be supported. Various types of access restriction (permission types) shall be provided.

5. Sensor Model

5.1. Overview

The SANY Sensor Model is best described using a number of different views and is part of a general strategy to make use of abstract information models in order to optimize usability and flexibility for complex systems. Again, as for the design process of the SensorSA, the five viewpoints defined in the ISO RM-ODP are used (see section 3). The following discussion starts with the Technology Viewpoint, illustrating the view of a hardware manufacturer, and then reflects a "Sensor" from the Enterprise, Engineering, Service and Informational Viewpoints. The Sensor Model also encompasses definitions of the terms sensor network and sensor service network.

Note 1: In this discussion, the thing observed by sensors is called an "observed property" in line with the OGC Observations and Measurements model (Cox, 2007). An observed property identifies or describes the phenomenon for which the observation result provides an estimate of its value. Based on this definition, SANY defines a sensor to be an entity that provides information about an observed property at its output. A sensor uses a combination of physical, chemical or biological means in order to estimate the underlying observed property. Note that, basically, these means could be applied by electronic devices or by humans. In the former case, at the end of the measuring chain electronic devices produce signals to be processed. In the latter case, humans enter the observation results in a data acquisition system as a basis for further processing.

Note 2: The core of the Sensor Model described herein was submitted to OGC and is now part of the OGC Sensor Web Enablement Architecture, OGC Engineering Report 06-021r2 (Simonis, 2008).

5.2. Technology Viewpoint of a Sensor

From a technical point of view, we consider a sensor to be a device that responds to a (physical) stimulus in a distinctive manner, e.g. by producing a signal. This means that a sensor device converts the stimulus into an analogue or digital representation, the latter being of more interest within the IT domain. In contrast, an "actuator" transforms a signal into an action that has some sort of effect on the physical domain, i.e. the actuator produces a stimulus that can be observed by a sensor (note that actuators are not within the scope of SANY.) Figure 5-1 illustrates this definition.

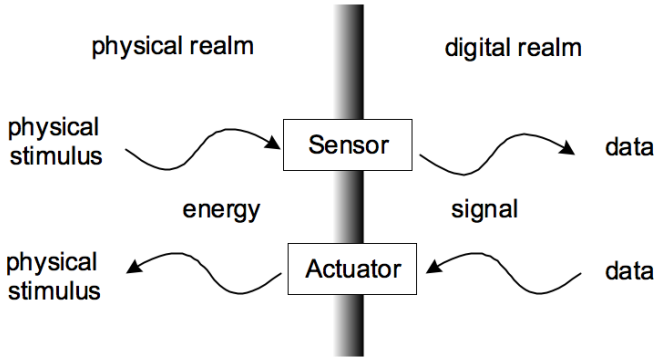


Figure 5-1: Sensor and actuator model (derived from (Ricker/Havens, 2005))

The following sections provide a more detailed discussion on sensors and distinguish between simple and complex forms of sensors and sensor systems.

5.2.1 Simple Form of a Sensor

The sensor observes an *environmental property* which may be a biological, chemical or physical property in the environment of a sensor, at a specific point in time (t0) at a specific location (spRef), i.e. within a *temporal* and *spatial context*. Note that the location of the sensor might be different from the location of the observed property. This is the case for all remote-observing sensors, e.g. cameras, radar, etc. For an in-situ observing sensor, locations of sensor and observed property are identical, i.e. the sensor observers a property in its direct vicinity. The simple form of a sensor provides information on a single observed property. Figure 5-2 shows the model of this situation.

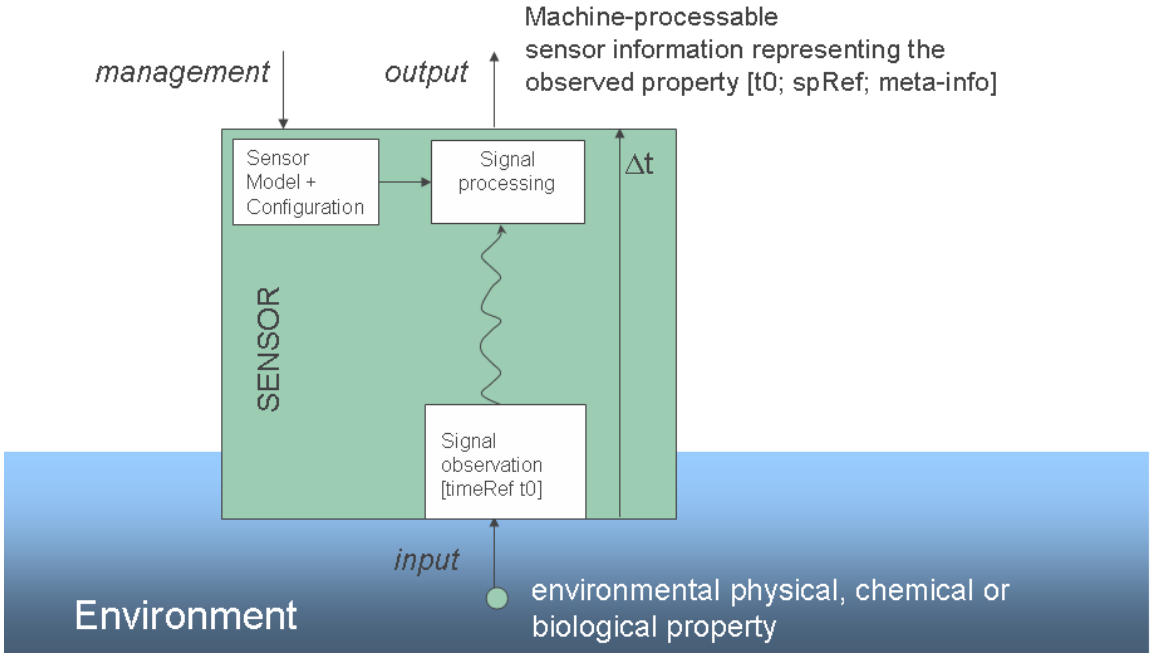


Figure 5-2: Model of a simple form of a Sensor

The observed property is usually converted to a different internal representation, usually electrical or mechanical, by the sensor. Any internal representation of the observed property is called a *signal*. Within the sensor any kind of signal processing may take place. Signal processing typically includes linearization, calculations based on calibration coefficients, conversions to different representations and any calculations to prepare the sensor data for output. The signal may also be transferred over longer distances.

Note: This transfer is not restricted to a signal transmission over a communication network but could also be a human carrying a chemical probe (e.g. a water probe from a river) to a laboratory.

The path from signal observation to the output of signal processing takes time and may also be distributed across several locations. However, the temporal context (t0) and the spatial context (spRef) of the signal observation must be preserved. As an example, consider the

water probe mentioned above: It is imperative to preserve the time and the location at which the probe has been taken. Depending on the application context, the time and location of the examination of the chemical probe in the laboratory might be an essential part of the probe data, or it may be considered as additional meta-information.

Finally, the observed property is accessible at the output of the sensor in a machine processable representation. The output provides information about the time (t0) and spatial context (spRef) during observation, though those parameters are usually provided in the form of meta-information and not as part of the observation result. Due to the delay, Δt, produced by the sensor during the observation, the information at the output of the sensor cannot be accessed before $t0+\Delta t$. This Δt can take any range from nanoseconds to several weeks or months.

Different sensors may provide different representations of the same observed property. They may differ in the units, the quality of the representation, the observation method or the internal signal processing that was used. The estimate of the value of the observed property may be a single value, a range of values, a choice between worst and best value, a sequence of values or a multi-dimensional array of values representing, for example, a picture. It may contain values for each point in spatial/temporal context or it may be a statistical representation in space or time. The description of the representation as well as all other observation related information has to be provided as sensor meta-information at the sensor output to be used by an application. A sensor may internally store representations of an older temporal context (history) or spatial context.

In addition to its output, a sensor may provide an interface to perform the management of the sensor itself. For instance, this interface may be used to tag the sensor with a name, to configure the internal signal processing, or to monitor the behaviour of a device.

5.2.2 Complex form of a Sensor

If an observed property cannot be observed with available sensor technology of simple form, it is possible to build a complex form of a sensor using several simple forms. This composite model is illustrated in Figure 5-3.

The information about the observed properties of the individual components of the complex form may be processed by any method of information processing (e.g. in fusion blocks). The output of the complex form of a sensor represents an observed property as defined by the sensor operator. This means that the linkage of the output of the complex form of a sensor to the output to the simple forms of a sensor is transparent. Still, even the complex form has to provide some information about the temporal and spatial context of its output data.

Note: Those contexts might be of different scales. A complex form of a sensor might provide forecast information for the next multi-week period in a large area, whereas the simple forms provides observations only at single points in time and space.

Thus, the resulting temporal context of complex forms of sensors is a function of the temporal contexts of the individual observed properties, represented in Figure 5-3 as *f(t0, t1)*. The same may be true for the spatial context, in Figure 5-3 represented as *spRef = f(spRef1, spRef2)*. The function should be provided as part of the meta-information, including

information about all processing steps at the output interface as well as on the management interface.

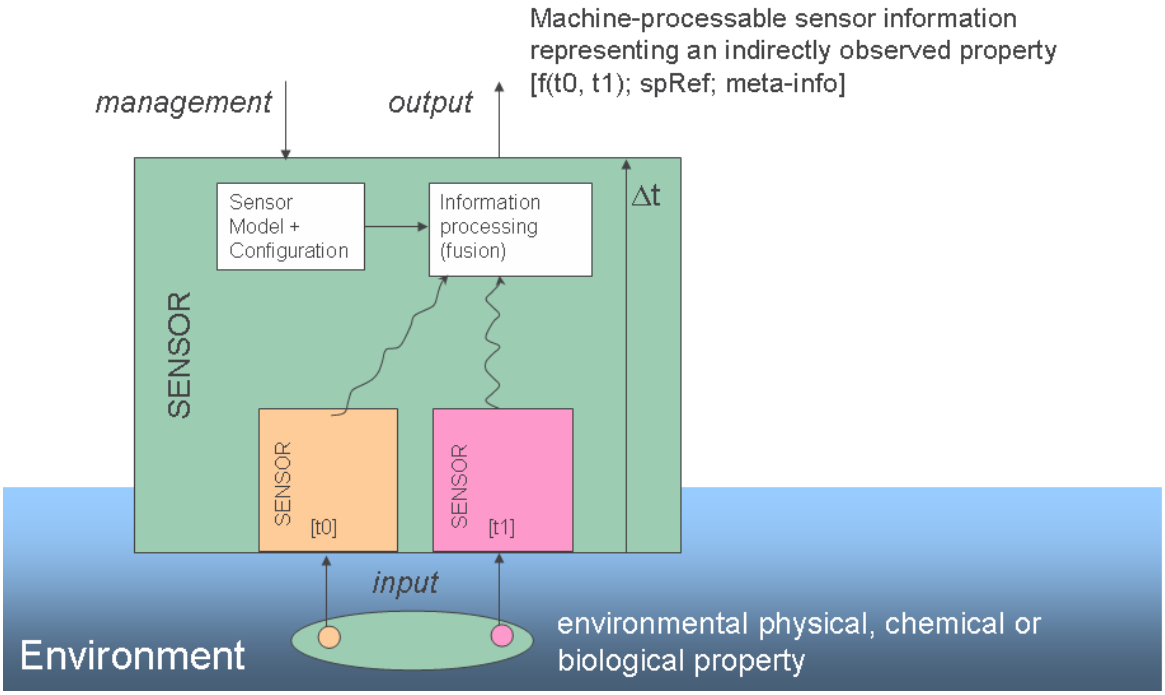


Figure 5-3: Model of a complex form of a Sensor

Depending on the application context, this complex form of a sensor may itself be aggregated into another complex form. In this case, the internal structure is a black box to the application.

5.2.3 Sensor System

Several sensors may be combined within a sensor system (see Figure 5-4) that allows the management of the system holding the sensors in addition to the management of each individual sensor separately. This is done through the management interface of the sensor system.

The key characteristic of a sensor system is its singular output and management interfaces that reflect its organisational unit. The organisational unit varies in type and nature. Having a sensor system doesn't necessarily mean that the individual parts of the system do not provide individual interfaces. In addition, each part of a sensor system might be unravelled into sub-systems or individual sensors with individual interfaces as well. The key characteristic of the system remains its single output- and management interface, independently of any kind of interface provided in addition.. Examples for sensor systems are satellites (whereas the physical structure of the satellite is a platform, not a sensor) with a number of remote-observing devices, weather stations with sensors for wind speed, temperature, and humidity, ground water observation systems used for surveillance of the environment around a chemical plant or a system of surface water observation points ordered on the surface and in the depth of a water body.

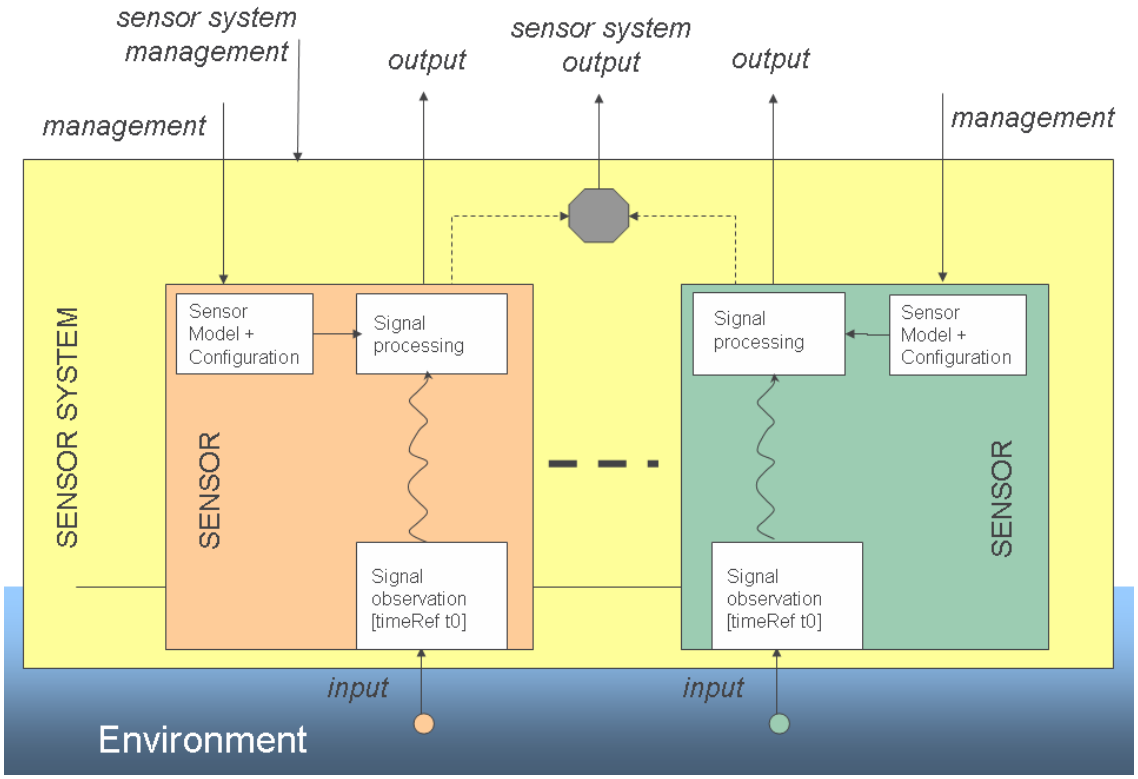


Figure 5-4: Model of a Sensor System

In contrast to a complex form of a sensor, the sensor system allows direct addressing of its individual parts as well as addressing of the sensor system as a unit. A complex form of a sensor provides only the management of the whole entity. Individual parts are not directly addressable. The difference affects the management interface, but has no influence on the response behaviour of both, complex form of a sensor as well as sensor system. Both might provide data that traces back to individual parts.

5.3. Enterprise Viewpoint of a Sensor

The enterprise viewpoint analyses the business context and the system and user requirements for environmental monitoring in terms of functionality, information demand and quality. It identifies the environmental phenomena that have to be observed with their temporal and spatial resolution and reflects this need with the types of sensors and models that are available. This activity may encompass a cost-benefit analysis if there are several options and offers of service providers. Furthermore, from the set of requirements basic patterns of sensor topologies are abstracted (see section 4.5). As listed in Table 4-1, a distinction has to be maintained between sensors and corresponding data loggers that are spatially fixed, i.e. bound to a given location, and mobile sensors such as cameras on aircrafts or satellites.

5.4. Engineering Viewpoint of a Sensor

Roughly speaking, the engineering viewpoint links components to a communication network. The network might be the Internet or any other open communication network. The components themselves implement purposes, functions, and content as described in the

service and information viewpoint below. Thus, the sensor model is extended with a network node component (e.g. an Internet node) as illustrated in Figure 5-5.

The SensorSA defines the resulting **sensor network** as a collection of sensors and optional processing nodes, in which information on properties observed by the sensors may be transferred and processed.

Internet nodes might be either connecting a single sensor (a) or a whole sensor network (c) to the communication network. Further on, a sensor system might even integrate all necessary components to act as one single network node, i.e. the sensor system is addressable and accessible within the communication network (b).

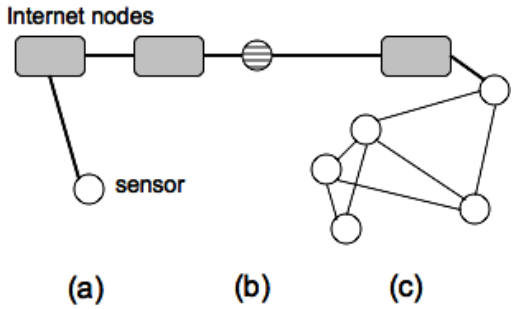


Figure 5-5: Sensors connected to a Communication Network (here: Internet node)

Depending on the available addressing options (see section 5.2.3), the sensor network appears to users as either a sensor system or a complex form of a sensor. This is the design decision of the sensor network engineer.

Let SN = {S1, S2,…,Sn} be a sensor network with $n \geq 0$ indicating the number of sensors in SN. There are the following properties of a sensor network with respect to membership of sensors.

- The membership of a sensor to a sensor network is time-dependent, i.e., sensors may join and leave sensor networks, or formally: $SN1(t1) \cap SN1(t2) \neq \emptyset$ with $t1 \neq t2$.
- Sensor networks may overlap, i.e., a sensor may be member in more than one sensor network at a given time t, or formally: $SN1(t) \cap SN2(t) \neq \emptyset$.
- Sensors may be moving, i.e. they may change their position. As a consequence of the movement of the sensor it may leave one sensor network SN1 and join another sensor network SN2, or formally: $Si \in SN1(t1) \wedge Si \in SN2(t2)$ with $t1 \neq t2$. The SensorSA refers to these sensors as roaming sensors. An example is a sensor node in a wireless sensor network that leaves the reachability zone of a data logger and gets into the reachability zone of another data logger.

5.5. Service Viewpoint of a Sensor

The service viewpoint is concerned with the functional decomposition of a sensor or a sensor system into a set of services that interact at interfaces. The transfer of this software modelling perspective into a more functional perspective of the sensor model leads to an even more

complex view. There are two perspectives for the service viewpoint: an internal perspective and an external perspective.

The internal perspective ignores the communication part for a moment and has a closer look at the physical device called a sensor by converting the black box sensor into a white box (see Figure 5-6).

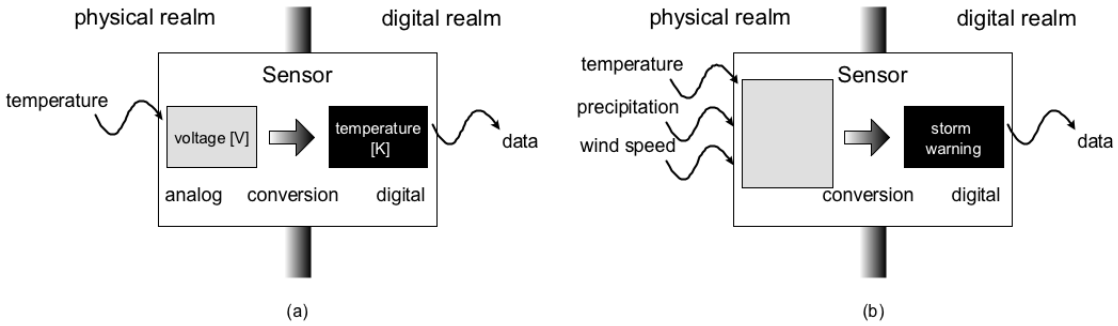


Figure 5-6: Service Viewpoint of a Sensor (internal perspective)

The sensor responds to the physical stimulus "temperature" with the generation of a certain voltage observed in Volts. Afterwards, the voltage gets converted into a digital representation of degrees Kelvin.

The external perspective represents the view of a software developer or a designer that aims at integrating a sensor into a network of services. From this perspective, a sensor may be seen as a component in a service network with two major logical interfaces:

- Information: an interface to access the information that represents the properties observed by the sensor (see the information viewpoint of a sensor described in section 5.6), and

- Management: an interface that enables the configuration and monitoring of the internal behaviour of the sensor (see the internal perspective) as well as the discovery of the sensor resources that are made accessible through the observation access interface.

Both logical interfaces have been illustrated before in Figure 5-2, Figure 5-3 and Figure 5-4. Technically, the SensorSA maps these logical interfaces upon the interface and service types of the OGC Sensor Web Enablement initiative. An example of an information access interface is the OGC Sensor Observation Service as described in section 8.2. An example of a management interface to a sensor is the OGC Sensor Planning Service as described in section 8.2.3.

From the service viewpoint, it often makes sense to consider a simulation model as a sensor, because a model can provide data for times in the past or future analogous to a sensor device. This view is, for example, found in (Botts, 2005) and (Cox, 2007). The main reason for this very broad usage of the term "sensor" results from research and standardization efforts within the domain of service-oriented architectures. As long as sufficient meta-information comes along with the data (e.g. how the data were produced, quality etc.), it does not make any difference for the client whether a physical device or a simulation models produced the data. This approach has the advantage that generic sensor applications may be built that retrieve their data from physical sensors (usually past observation results) in the

same way as from simulation or predictive models (i.e. calculated future observation results in the case of predictive models).

Services instances that provide access to sensor data are usually composed in so-called sensor service networks. The Sensor Model defines a service network as a set of service instances that interact in order to serve the objectives of applications (definition derived from RM-OA (2007)). Sensor service networks are variants of service networks that are compliant to the specifications of the SensorSA.

Let SSVN = {SV1, SV2,…,SVm} be a sensor service network with $m \geq 0$ indicating the number of services in a SSVN. In analogy to the membership of sensors to sensor networks, there are the following properties of a sensor service network with respect to the membership of service instances.

- The membership of a service instances to a sensor service network is time-dependent, i.e., service instances may join and leave sensor services networks, or formally: $SSVN1\ (t1) \cap SSVN1\ (t2) \neq \emptyset$ with $t1 \neq t2$.
- Sensor service networks may overlap, i.e., a service instance may be member in more than one sensor service network at a given time t, or formally: $SSVN1(t) \cap SSVN2(t) \neq \emptyset$.
- Sensor service networks may be re-configured, i.e. a service instance SVi may be removed from one sensor service network SSVN1 and assigned to another sensor service network SSVN2, or formally: $SVi \in SN1\ (t1) \wedge SVi \in SN2\ (t2)$ with $t1 \neq t2$.

The physical grouping of sensors into sensor networks and the logical grouping of service instances into sensor service networks is illustrated in Figure 5-7.

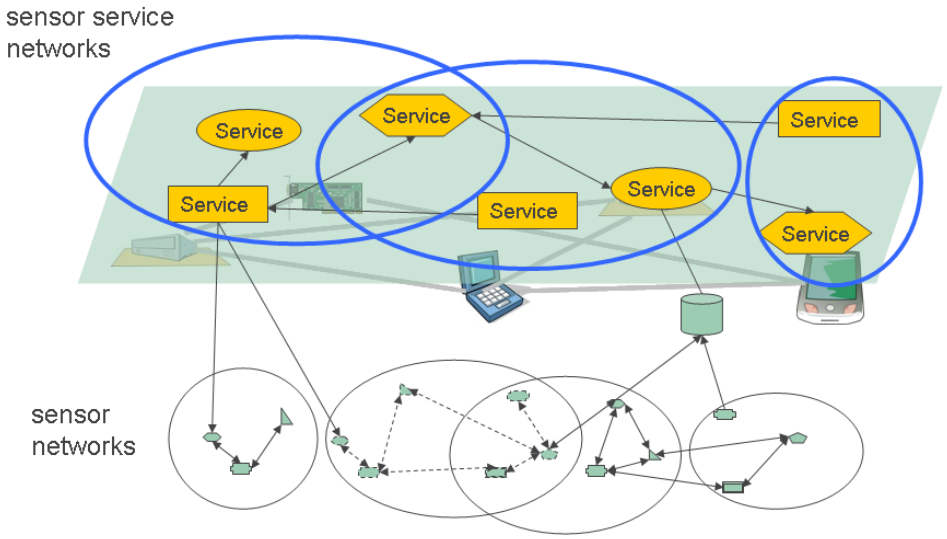


Figure 5-7: Sensor Networks and Sensor Service Networks

5.6. Information Viewpoint of a Sensor

The information viewpoint is concerned with the semantics of information and information processing. Thus, it discusses a sensor in regard to the semantics behind a sensor or a sensor system. The abstraction from the physical device described in the technology viewpoint becomes appropriate. Basically, SANY adopts here the OGC Observations and Measurements model as specified in (Cox, 2007) and described in section 7.2.

We speak of a sensor as a source that produces a value, within a well-defined value space, of an observed property which may represent a physical, biological or chemical environmental phenomenon. Sensors and sensor systems as well as simulation models fulfil this definition. If the semantics do not differentiate between data produced based on a physical stimulus or any other data, the distinction between model and sensor disappears.

The information viewpoint concentrates on the data that are provided in the form of observation results abstracting from the source of the observation data. These observation results have to follow the sensor data information model, i.e. the results have to reflect all aspects of the underlying viewpoints. In addition to the observation results, information about the observation procedure, spatial-temporal context, and organizational characteristics has to be provided. Such information is considered to be meta-information for the purpose of interpretation and further processing of the observation results (see section 6.3).

In order to identify and describe sensor networks the following information elements may be necessary:

- human-readable name and unique identifier of a sensor network
- status of the sensor network
- observation-related attributes, e.g. observed properties, features of interest
- topology of the sensor network (e.g. ring, star, bus,...)
- communication means of the sensor network (e.g. Zigbee)
- administrative attributes of the sensor network (e.g. provider, ownership)
- statement about how the membership of sensors in a sensor network is defined. Membership statements may be formulated explicitly, by defining a list of sensors, or implicitly, by providing a logical expression on attributes. Implicit membership statements may be based upon administrative attributes (example: “all sensors operated by the German Meteorological Service”), or based upon spatial-temporal conditions (example: “all air monitoring stations in Rome available in January 2009”).

6. Major Concepts of the Sensor Service Architecture

6.1. Overview

Before starting the detailed specification of the individual viewpoints of the SensorSA, the following major concepts of the SensorSA are described in order to facilitate the understanding of the subsequent specifications:

- functional domains of the SensorSA
- models of interactions including request/reply and event-based models
- resources and their identification
- approach for meta-information including handling of data quality
- management including resource discovery, sensor planning and sensor and sensor service monitoring
- security model with a focus on access control

Concrete concepts, e.g. interaction patterns between service instances and policies for service networks, are specified in the Engineering Viewpoint in section 10.

6.2. Functional Domains

Services in the SensorSA are designed to support applications that serve the needs of users. They may call other services if this is required to fulfil the functions offered at their interfaces. In this case, a service may itself be a client to another service. In an extended situation, chains of service operation calls may be defined in order to realise more complex functionality. In a service network every service instance may call operations of any other service.

The call of an operation of a service requires that the client know the name and the address of the operation. This knowledge may be acquired from some mediating instance in a discovery phase (e.g. a catalogue service, see below), however, it may also be acquired by other means (e.g. entered by a human or pre-configured).

Although there is no prescribed hierarchy of services, services may be grouped into functional domains for which they are basically designed. The SensorSA distinguishes between the following functional domains as illustrated in Figure 6-1:

- Sensor Domain

  Services in the sensor domain cope with the configuration and management of individual sensors and their organisation into sensor networks. Further examples are services that support the interaction among the sensors themselves, e.g. a take-over service in case of an impending sensor battery failure. Note that services in this domain that will be part of the SensorSA shall be abstractions from the proprietary

mechanisms and protocols of sensor networks. Proprietary mechanisms are part of an Implementation Architecture and are outside the scope of the present document.

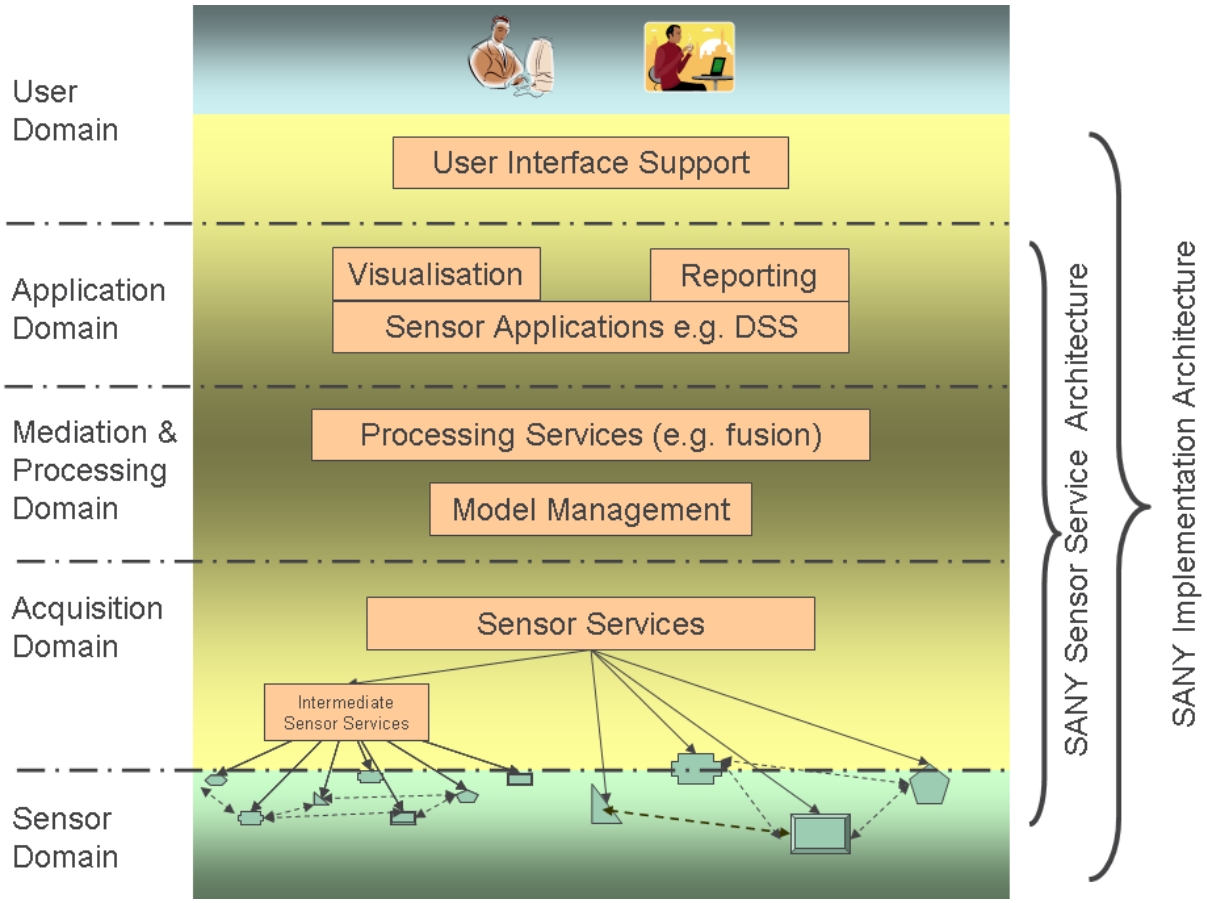


Figure 6-1: Functional Domains of the SensorSA

- Acquisition Domain

  Services in the acquisition domain deal with access to observations gathered by sensors. This includes other components in a sensor network (e.g. a database or a model) that may offer their information in the same way (i.e. as observations) as sensors do (see the discussion about the Sensor Model in section 5). They explicitly deal with the gathering and management of information coming from the source system of type "sensor". The information acquisition process may be organised in a hierarchical fashion by means of intermediate sensor service instances (e.g. using data loggers).

- Mediation and Processing Domain

  Services in the mediation and processing domain are not specific to the SensorSA. They are specified independently of the fact that the information may stem from the source system of type "sensor". They mediate the access from the application domain (see below) to the underlying information sources. They provide generic or thematic processing capabilities such as fusion of information (from sensors and other information sources), management of models or access to model results. In addition, service support for the discovery of sensors, data and services, naming resolution or service chaining are grouped in the mediation and processing domain.

- Application Domain

  Based on services of the acquisition and processing domain, services in the application domain support the rendering of information in the form of maps, diagrams and reports such that they may be presented to the user in the user domain.

- User Domain

  The functionality of the user domain is to support the interface to the end user. Functions in this domain are formally outside the scope of the SensorSA. Thus, the SensorSA does not specify dedicated services to support the user interface. However, when building concrete systems and applications, such functionality is essential. This functionality has to be specified in a dedicated implementation architecture that also may take proprietary components and products into account.

At the user domain layer, the user usually is provided with a graphical user interface that simplifies the operation of a sensor service application that is run on the application domain layer. For example, instead of typing commands into a console window or running shell scripts, the user uses forms, control bars, or joysticks to control the application. The communication taking place between the form and the sensor is opaque to the user. The form-providing application may communicate directly with the sensor or with any of the layers below. Similarly, every lower layer may communicate directly with the sensor or with any other lower layer. To the user, it appears as a direct communication with the sensor, although multiple intermediate steps might be involved. The following figure illustrates the various communication paths.

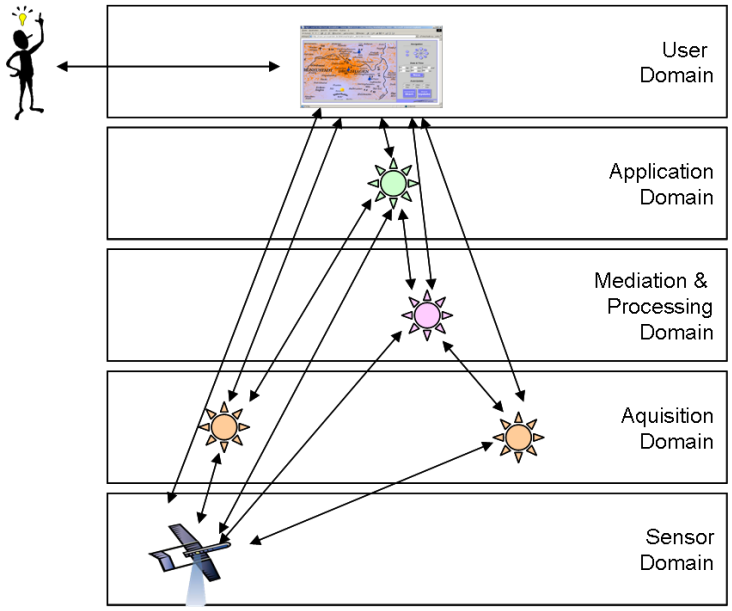


Figure 6-2: Communication paths between the user and the sensor

6.3. Models of Interaction

6.3.1 Overview

The SensorSA uses the taxonomy proposed in (Muehl/Fiege/Pietzuch, 2006) in order to characterise the interaction models between the service components in a sensor service network. The taxonomy is based upon the way interdependencies between the service components are established (see Figure 6-3).

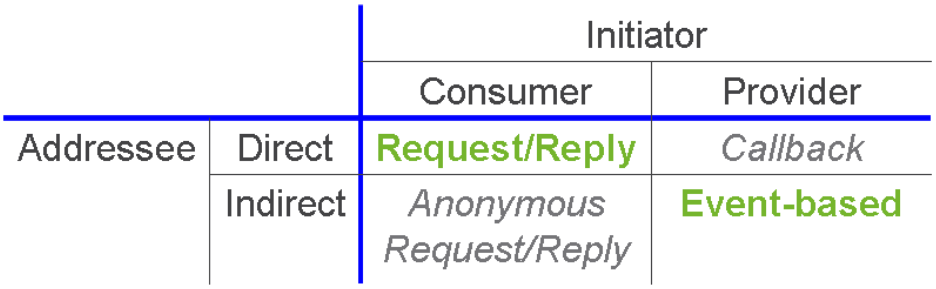


Figure 6-3: Taxonomy of Interaction Models (Muehl/Fiege/Pietzuch, 2006)

It is expressed by two attributes: the *initiator* attribute, which describes whether the consumer or the provider initiates an interaction, and the *addressee* attributes, which describes whether the addressee of the interaction is known (i.e. directly addressed) or unknown (i.e. indirectly addressed).The SensorSA currently supports the following interaction models:

- request/reply interaction model (see section 6.3.2)
- event-based interaction model (see section 6.3.3)

Note: These are interaction models between the logical service components without making assumptions about the underlying infrastructural means (e.g. communication protocols) to implement these interaction models. Furthermore, both interaction models may be applied to implement the two logical interface types that are distinguished in the service viewpoint of the sensor model (see section 5.5): the information and the management interface.

6.3.2 Request/Reply Interaction Model

In the request/reply interaction model the initiator is the consumer (also called client) that requests data or functionality from the provider (also called server). The initiator either expects data to be returned or it relies on a specific task to be done. The consumer knows the provider in the sense that it may directly address the provider. The address may have been acquired through pre-knowledge (configuration) or by means of resource discovery (see section 10.2).

The request/reply interaction model is applied in most of the services and interfaces specified in the Service Viewpoint in section 7.

Note: In cases where the addresses of the providers are not known, (Muehl/Fiege/Pietzuch, 2006) talk about the anonymous request/reply interaction model. This

interaction model may be applied in the sensor domain of the SensorSA functional domains (see Figure 6-1) and implemented by broadcast or multicast communication protocols in sensor networks. However, these communication protocols are conceptually out of the scope of the SensorSA., because, as explained in section 2.1, the SensorSA specification is independent of the specifics of a particular service platform.

6.3.3 Event-based Interaction Model

The event-based interaction model represents the basic form of interaction for cases in which timely delivery of observed actions is important but needs to be flexible. It is usually applied in event-driven processing systems (as defined in section 6.4.5). Flexibility and adaptability are among the key characteristics of event-driven processing systems, because event generators don't call any specific type of event receivers. Indeed, they don't even need to know them.

Events will be more fully defined in section 6.4, but the event-based interaction model relies on two basic concepts:

- an **event** that describes any happening of interest (i.e. anything that happens, or is contemplated as happening), and
- a **notification**, that transports the reified happening of interest.

In the event-based interaction model the initiator is the provider of the data, i.e. the producer of **notifications**. The essential characteristic of this interaction model is that producers do not need to know any consumers. Thus the addressing scheme is indirect, which means that the notifications are not addressed to any specific set of recipients but instead are mediated by a broker component which offers a notification service. A consumer may express its interest in notifications by subscribing to the notification service.

The SensorSA provides the means and mechanisms to define, generate, distribute, receive, and process events. Three causes of events are observed most frequently:

1. Events based on singular observations made by a single sensor,
2. Events based on aggregated observations made by one or multiple sensors, and
3. Events related to the operation of the sensor network or the sensor services.

The first type of event occurs if a sensor detects something that matches a previously defined event condition. The occurrence may take place in the environment of the sensor or internally. Examples are a temperature value that exceeds a threshold, the detection of hotspot pixels in a remote sensing image, or low battery power of a sensor.

The second type of event occurs if non-atomic conditions occur, e.g. both temperature and wind speed observation result values exceed thresholds. Events may be based on conditions that remain for a well-defined number of time intervals, e.g. temperature exceeds a threshold for n time intervals continuously (also known as time series analysis based events). In a common example, one event is produced when e.g. the temperature exceeds a threshold the first time. A second but different event is produced if the temperature again falls below the threshold. In this case the two different events follow a state change in the sensor.

The third type of event occurs when some state has changed in the sensor or service network configuration (e.g. addition of a new sensor or sensor service instance) or some unforeseen behaviour has been detected. The latter situation usually results in an exception on the software level. If deemed essential by the software engineer, such exceptions may be escalated to other components in the form of events.

In the context of the SensorSA, all event types are handled equally. The event type is transparent to the receiver of an event notification. It is created by the event observer and published or transferred to notification consumers.

Still, the SensorSA addresses a very heterogeneous environment with sensors and services provided by a number of institutions and organisations. The event-based architectural style of the SensorSA defined below takes these aspects into account.

6.4. Event-based Architectural Style

SensorSA defines different architectural styles that could be applied to the Sensor Web. The goal is to harmonize the various styles in order to facilitate a successful integration of different approaches in a single application. In the following, we define the Event-based Architectural Style. It is based on the existence of events that get communicated between various components within the Sensor Web.

Note: The SensorSA Event-based Architectural Style was developed in close cooperation with the Sensor Web Enablement Team of OGC to ensure a sustainable solution beyond the lifetime of the project SANY. The goal was to develop a general Event-based architectural style to be applied to Sensor Web applications. The results of this activity are published as OGC Engineering Report (Everding and Echterhoff (Eds.), 2009). With over 150 pages in length, this report goes beyond the scope of the SensorSA core document. Thus, we make do with an excerpt of the main findings at this stage and refer to the publicly available OGC report.

6.4.1 Event Definition

The ISO 19100 series of standards, which are, as leading standards in the geospatial IT domain, of overall importance to the SensorSA, provide two definitions of the term “event”. Common to both definitions is the reference to an “action”. It is therefore in line with the rather philosophic definitions such as the one provided by the Oxford English Dictionary, which defines an event as “something that happens or is thought of as happening”. In terms of our application domain, this leads to the rather problematic situation that the event is equated with the action itself, thus the event becomes an abstract, non-computable thing:

- ISO 19136: An event is “an action that occurs at an instant or over an interval of time.”

- ISO 19108: An event is “an action that occurs in an instant", an instant being "a zero-dimensional geometric primitive representing position in time”

Common to both terms is the reference to an action that occurs. The relation to a time instant versus a time interval manifests the crucial difference. According to ISO 19108, only

those actions qualify as events, that postulate changes, e.g. “it started raining at 20:00hrs CET”. Observations like “it rains” or “it rained from 2AM to 4PM” don’t qualify as events consequently. The definition in ISO 19136 is more relaxed. Here, both “it started raining at 20:00hrs CET” as well as “it rained from 2AM to 4PM” do qualify as events, only “it rains” does not. The latter would be called a “state” in ISO 19136.

In SensorSA, we adopt the definition of Luckham and Schulte (Eds.) (2008) and combine it with the definition of ISO 19136 in order to be consistent with ISO TC 211 nomenclature. Thus the term event is defined as follows:

- An **event** is anything that happens or is contemplated as happening at an instant or over an interval of time.

This definition emphasizes the fact that an event has a strong temporal aspect and may represent anything that happens in the real world but can as well be simulated or happen in software. The term *happening* encompasses *action, occurrence* and *situation of interest, state change* etc. which all represent something that happens.

To ensure better alignment with the domain of interest, i.e. information and communication technologies, we introduce further the term *event object*:

- An **event object** represents, encodes, or records an event, generally for the purpose of computer processing,

Thus, not the happening itself, but a record that signifies the happening is considered to be an event object. The happening of interest is sometimes referred to as *action* or *activity*. This action remains an action until something observes it and generates an event object that reifies it. We emphasise that event processing literature often does not distinguish between an activity that takes place and an object that represents that activity for the purpose of computer processing. This causes an overloading of the word “event”, which takes both meanings.

An event can be generated at any time during an observation. Given the sensor model as described in 5.2, an event can be produced at any stage during the observation process, i.e. from or after the first observation of an environmental property throughout all processing steps until the final observation is delivered as output of the sensor or sensing system. The event time will be assigned by the observation provider and equals the sampling time of the observation. The event model, which will be described in the next section, will elaborate the temporal aspects of events.

6.4.2 Event Model

6.4.2.1 Overview

Not every happening that might be of interest will be modelled as an event. It is up to the domain experts to make decisions about the event itself, its properties, associations, and its relationships to other events. The various levels of freedom need to be represented in the event model.

Events might but don't need necessarily be modelled as features, according to OGC Abstract Specification 5 and in accordance with ISO 19101 and the General Feature Model

(GFM) defined in ISO 19109. Here, features are entities that have a set of properties defining their characteristics. All events have one characteristic in common: The time the happening of interest took place. Some happenings last for a period of time, e.g. a sandstorm. SensorSA adopts the definition by Allen and Ferguson that events are temporally anti-homogeneous (Allen & Ferguson 1994). Thus, an event that happened over an interval of time did not happen during that interval, because the event would not have been completed yet. The event time equals the time of completion of the interval. This aspect needs to be considered for the modelling of event types.

Further on, the event model shall allow multiple events generated for a single happening of interest. Those events might represent different aspects of the happening, with only the time being shared across them. We will elaborate the lifetime of events in more detail in the following sections, before the event model aspects inheritance, constraints, properties and associations will be discussed in more detail. Before we address the lifetime of an event, we define further:

- A **notification** is a message that transports one or more events. The notification might be identical to the event object, if a single event object gets transported without any further packaging into a message container.

Note 1: An alert is a notification. The terms notification and alert are used synonymously throughout this document.

Note 2: Some use cases describe the dispatch of “alarms”. The SensorSA specification does not differentiate types of event notifications. Thus, an “alarm” is simply an event notification and shall not be used in architectural discussions, as its semantics differ considerably across applications. The term is better used in its verb-form, e.g. “…a notification will be sent to alarm the user…”.

6.4.2.2 Event Properties

SensorSA follows the General Feature Model defined in ISO 19109 to define the property types operations, attributes, and association roles of the feature type 'event type'. However, the SensorSA uses the RDF terminology, as defined by the W3C (W3C, 2004) because it doesn't differentiate between attribute and association roles, but calls both property types 'properties', which is more appropriate for the event model:

- Operations

  SensorSA defines a single mandatory operation to retrieve the event time: *getEventTime*. It returns a time instant or time interval. An event type might encode the time when the happening took place as a property, or the time gets computed on the fly once the *getEventTime* operation is called. Event types may provide additional optional operations.

- Fixed and Dynamic Properties

  Event objects have fixed and dynamic properties. Fixed properties cannot be modified once the value has been set. An example of a fixed property is the temporal or spatio-temporal characteristic of the event object. Event objects cannot be modified after

their release, but need to be superseded by updated new event objects, i.e. after release, all properties are fixed.

Nevertheless, there are some event objects that refer to the same logical happening, but differentiate in their property settings. An example would be a "lastEarthquakeEvent" object. This event object always refers to the last earthquake, thus some properties, such as epicentre or magnitude change from event object to event object. Here, every new event object represents an update of all earlier event objects, i.e. though the properties are fixed of each object, they seem to be dynamically changing.

- Temporal Properties

  Each event object may store a temporal property of type time instant or time interval to represent the time when the happening took place. Still, following the lazy loading pattern, the time might get computed in time when the getEventTime operation is called.

  Event objects might provide any additional temporal property, such as 'event creation time', 'event detection time' etc. To support distributed systems, each event object shall reference the temporal reference frame and clock to support proper event sequencing. Event object may represent happenings in the past, ongoing, or (simulated) in the future.

- Spatial and Location Properties

  Event objects may provide information on spatial extents related to the happening. Those properties may be of any geometry type and representation, i.e. vector or coverage based. A location identifier may be used instead of a geometry type.

- Thematic Properties

  Any domain expert is free to add as many thematic properties as necessary.

The formal definition of the event information model is provided in the Information Viewpoint in section 7.5

6.4.2.3 Event Lifetime

Temporal aspects of events have attracted considerable interest as a research topic over the past few years (Allen 1995). The core focus was on representing temporal aspects in geographic information systems and spatio-temporal data models. SensorSA doesn't contribute to this still ongoing discussion, but defines an event model that acknowledges the temporal aspect of events reifying actions in their respective environment as well as their causal catenation. SensorSA introduces a three-phases event lifecycle, with creation, update, and deprecation phase of an event.

- Creation of Events

  The Sensor Web requires events to be generated from a number of components and logical entities. Among those components are resource-constrained devices, such as

battery-powered sensors, temporarily unavailable components, such as human observers, and high-reliance systems such as emergency warning systems.

Basically, event creation is a two-step process. First, the action that is signified by the event needs to be observed; and second the observation must be transformed to a structure that can be processed within IT systems.

SensorSA doesn't apply any restrictions on the observation process, i.e. the activity can be observed by a sensor, a human, a piece of software that supervises another piece of soft- or hardware, event processing systems that generate events based on other events, or any other entity that is enabled to execute observations. The second step is the transformation of the observation to something that can be dealt with within IT systems, i.e. an event object that reifies the observed happening.

- Update of Events

Once generated, events can be updated to allow modifying the event content. The update of an event may have major effects on events further down the causal chain of related events. Causal chains and updating of events will be further elaborated in 6.4.2.6.

- Deprecation of Events

Events shall not be cancelled, i.e. an event exists and continues to be valid until declared void or deprecated. SensorSA calls this approach the **Reliant Event Model.** The rationale behind this approach can be found in the ambiguous semantics of the term cancellation. Cancelling an event could mean that the signified action never happened (e.g. false observation by sensing device), that it had happened but is not valid any longer (e.g. a "fire event" after the fire was extinguished), or that it was once true but was modified based on additional observations (e.g. reclassification of a storm once more observation data were received).

The validity of events shall be reflected in the event content model. Events are valid until they become explicitly deprecated.

6.4.2.4 Event Verbosity Levels

SensorSA differentiates three verbosity levels. Depending on the verbosity level, applications produce different amounts of events.

0. First event is generated when a pre-defined condition is matched. No further event is generated until a pre-defined condition is left, e.g. intrusion detected, observed value above threshold.

1. First event is generated when a pre-defined condition is matched; subsequent events get generated every time the observed value(s) change(s) and the pre-defined condition is still true.

2. Every observation that matches a pre-defined condition generates event, i.e. first event is generated when pre-defined condition is matched; subsequent events get generated every time at sample intervals as long as the pre-defined condition holds true.

Figure 6-4 illustrates the different verbosity levels for different phenomena.

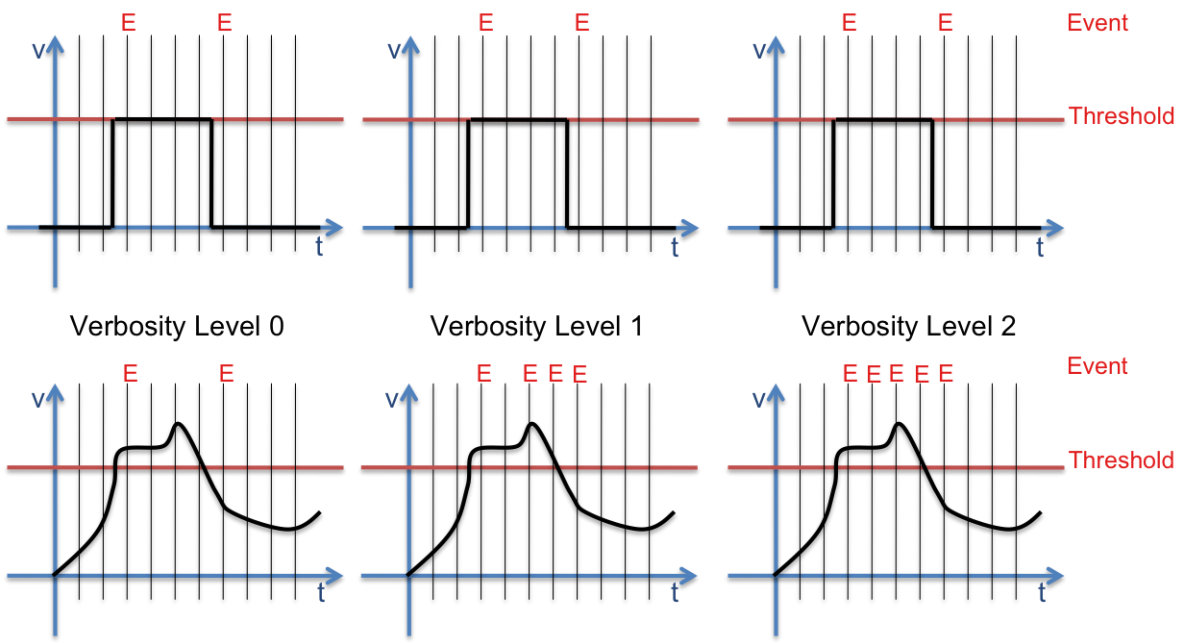


Figure 6-4: Event generation verbosity levels of type binary (upper row) and nominal (lower row)

Figure 6-4 illustrates the two extremes of binary and nominal scales. Other scales, such as ordinal or interval, work analogously.

The upper row illustrates events generated by an intrusion detection system. This binary system knows only two stati, i.e. “intrusion detected” or “no intrusion detected”. The lower row represents a temperature sensor. The y-axis represents the temperature value and is of type nominal scale. The sampling rate of both systems is identical. In both rows, the x-axis represents time, the y-axis observed value(s). The bold black line represents the current value. The sampling rate is indicated with vertical black lines. Generated events are labelled with a red “E” above the sampling lines. We see that the number of generated events is considerably higher for the temperature observation system than for the intrusion detection system, because the temperature may change its value above or below the threshold, whereas the intrusion detection system only knows two statuses.

6.4.2.5 Form of Events

Complex Event Processing, or CEP, is one of the major concepts of event processing. It deals with the task of processing multiple events with the goal of identifying the meaningful events within event clouds.

One of the key aspects of Complex Event Processing applications is the aggregation of event objects and the derivation of new information, reified as new events or event objects, respectively. SensorSA adopts to a large extent the concept developed by Luckham and Schulte (2008). According to them, an event, which is neither an abstraction nor a composition of other events, is a **Simple Event**. Whenever an event is an abstraction of other events, it is called a **Complex Event** (see also the specification of the vent information model in section 7.5). Complex events can but do not have to list the member events of which they

provide an abstraction. Furthermore, whenever an event is generated as a result of applying a well-defined procedure to one or more other events, we refer to a **Derived Event**.

The verbosity of events is an essential criterion in event messaging systems. It is the information model that defines the power of notification messages, i.e. which amount of information a notification may provide. In some cases a notification may only indicate that something of interest has happened, but no details - then clients would have to pull the additional information from a service; in other cases the notification itself might already contain all relevant information so that no additional service-request is needed.

6.4.2.6 Roles in Event Relationships

Events may have relationships to other features and thus also to other events. The specific relationships between events are domain dependent. However, some roles serve a general purpose and are thus defined in SensorSA. We can identify relationship roles for related events in general (see Table 6-1) and for events that are members of a complex event (see Table 6-2).

Role Identifier	Meaning
supersedes	The target event is superseded by the source event. This means that the target event is deprecated.
revokes	The target feature is revoked which means that the target event object should be considered as not having been issued.
caused	The source event caused the instantiation of the target event. More specifically, the source event is (one of) the reason(s) why the target event was instantiated.

Table 6-1: Roles implemented by a related event

Role Identifier	Meaning
causedBy	The source event was caused by the target event. More specifically, the target event is (one of) the reason(s) why the source event was instantiated.

Table 6-2: Member Event - defined values of the role property in an EventEventRelationship

Superseding an event object with another is the preferred way to implement changes applied to an event object. Let us explain this in more detail. Any modification of an event object can always be critical for other applications. When transmitting an encoded representation of an event object to other systems, what they get is only a snapshot if the event object has modifiable / dynamic properties. Computations that are based upon this snapshot will need to be repeated when a change to an event object is made later on. In the worst case this would lead to a rollback of the whole computation, possibly involving a rollback on other systems as well. This situation is unavoidable and applies to all systems that base their computations upon given information. At least we can make it easier for event processing systems to detect a change of a previously received event object by implementing such a change in a new event object that has a relationship to the original event with the role *supersedes*.

If it is recognized that an event was wrongly detected, initialized and distributed or the event object released on mistake, then this "happening of recognizing the failure" can be

represented by a new event and an event object be distributed which relates to the wrong event, which shall be *revoked*.

How an application reacts when receiving an event that either supersedes or revokes is up to the application and not defined here.

Whenever an event is detected based upon the information contained in other events then there exists a causal relationship between the detected event and the base events. A base event *caused* the detected event (maybe multiple other events). From the perspective of the detected event, it was *causedBy* one or more other events. The set of member events that caused a complex event is sometimes referred to as the *causal vector* of that event.

6.4.3 Event-Driven Processing System

6.4.3.1 Overview

Event-driven processing systems are applying the event-based architectural style as described above, i.e. they are centred on an asynchronous "push"-based communication model. They emphasise the basic idea of sense and respond: Sensors observe the current situation and alert receivers upon specific actions. Using techniques such as Complex Event Processing (CEP), it becomes possible to extract the information value of multiple events and data streams and alert/notify interested parties with minimum delay. Due to the permanent sensing, event-driven processing systems adapt perfectly to a continuously changing environment. Events get detected when they appear. The influence of pre-planned schedules is minimized.

Any event-driven processing system consists at least of two components, (1) a sensor sensing the event and emitting the notification, and (2) a consumer receiving this notification. In more complex scenarios, the consumer can act as a sensor itself, emitting new notifications in turn of received ones. Furthermore, any event processing systems supports the following three features (Chandy and Schulte, 2007):

1. Notifications are sent from the producer to a consumer using asynchronous messaging. The emission of notifications is triggered by the producer, not the consumer.

2. Consumers process event notifications as soon as possible to ensure timely end-to-end processes.

3. Notifications don't specify the operation that a consumer must perform upon receiving the notification. As the logic what to do with the notification remains in the consumer, developers can change their response systems without touching the producer. The logical coupling is minimized.

6.4.3.2 Event Processing Role Model

The SensorSA uses a role-based concept to differentiate the various participants involved in the event-driven processing systems[10]. Some of those roles can be implemented as services and will therefore be discussed under the service viewpoint (see section 8).

The SensorSA event role concept defines the following roles (Figure 6-5):

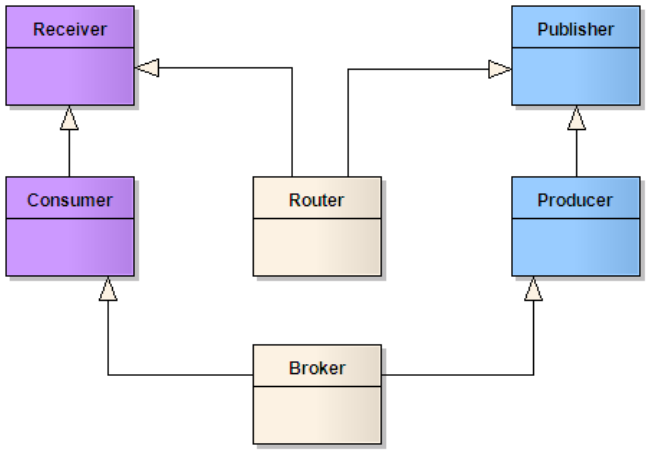


Figure 6-5: Event Processing Role Model

- **Receivers** do receive notifications from a priori known publishers. This simplest type of notification sink does not allow registering additional publishers. It does, however, support registering itself at new publishers. Example: A component that accepts only incoming notifications from known publishers, e.g. to make data persistent in a database.

- **Publishers** do publish notifications. Publishers don't offer subscription-interfaces. Example: Simple sensor without any interface exposed other than what is necessary to send notifications.

- **Routers** receive events and publish them again, i.e. they forward events from registered Publishers to registered Receivers. Example: A gateway that forwards data received over a thin wire from a sensor to a bunch of Internet clients. The Router is derived from Receiver and Publisher, i.e. it doesn't provide any interface to register new publishers, nor does it support subscriptions. Example: A component that receives notifications from a-priori known publishers and publishes them to a-priori known Receivers.

- **Consumer** consumes notifications and provides an interface that allows publishers to register with this Consumer. The Consumer then accepts notifications sent by the newly registered publisher. Consumers inherit the capability to subscribe with arbitrary Publishers from Receivers. Example: A service that accepts additional sensors to register themselves as publishers.

[10] The concept described herein is based on the OGC Engineering Report OGC 09-032 (Everding and Echterhoff (Eds.), 2009), but varies slightly, as it describes a later stage of discussion.

- **Producers** do publish data and offer subscription interfaces to clients. Example: Sensor that supports subscription and sends notifications.

- **Broker**: A Broker combines the roles of Consumers and Producers. Example: OGC Sensor Alert Service: it consumes data from sensors (therefore it is a Consumer) and offers pub/sub to clients (therefore it is a Producer).

The various roles result on the implementation of interfaces as described in section 6.4.3.3. In concrete implementations, the various components will certainly be called differently and may provide additional capabilities, as they will be based on specifications and definitions given elsewhere. Nevertheless, the role concept provides a guideline that helps distinguishing explicit behaviour of components. The following Figure 6-6 illustrates the core capabilities of the various roles.

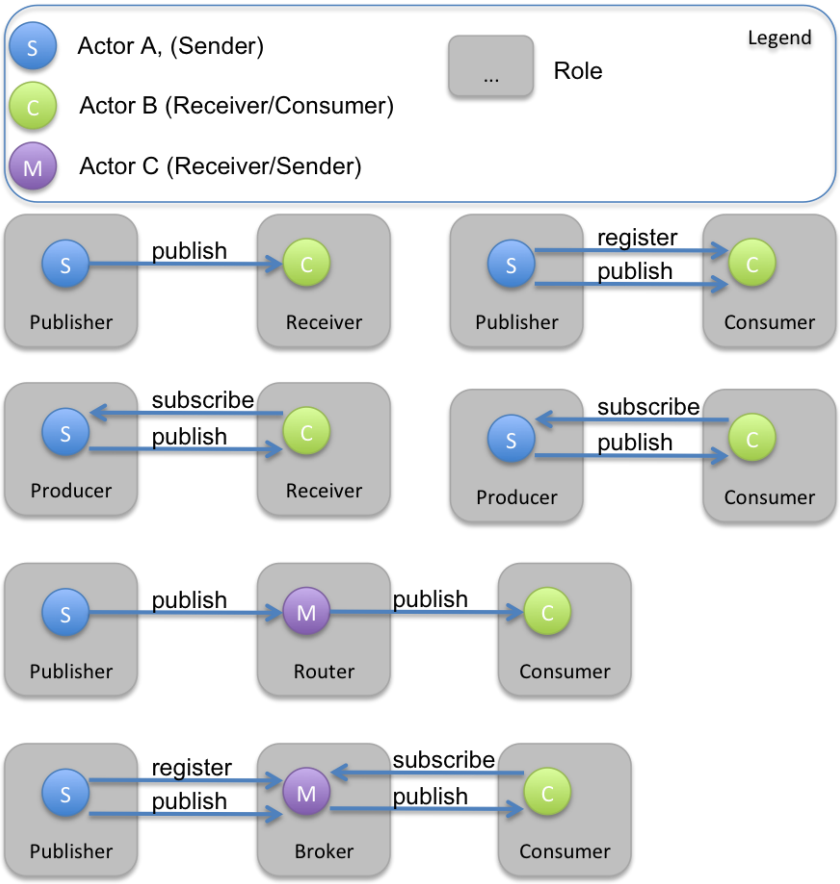


Figure 6-6: Event Processing Interaction Models

Blue dots represent actors taking on the roles of a publisher or producer, green dots represent consumers or receivers, and purple dots represent router and broker. The first row differentiates the different capabilities of receivers and consumers in case a publisher intends to send event notifications. The second row reverses the perspective: both receiver and consumer can subscribe with producers.

Row three and four illustrate the different capabilities of routers and brokers. Whereas the first behaves as a simple forwarder of information, the broker allows publishers to register and consumers to subscribe to the events its offers.

Naturally, all components can be included arbitrary chains. The following figure illustrates a realistic scenario. Here, a wireless sensor is connected to a gateway that forwards events generated by the sensor to other Internet nodes. Due to the limited capacities of the sensor, it is preconfigured to send all events to a dedicated event sink (Internet Gateway). The Internet Gateway doesn't provide any additional capabilities other than forwarding the events to a broker. Such a chain of components is often used to match dedicated security requirements. An additional router subscribes to the offerings of the broker. This router serves as an intermediate to the client. The router may only act as a protocol transducer, i.e. events received using Internet protocols might get forwarded using automated phone calls.

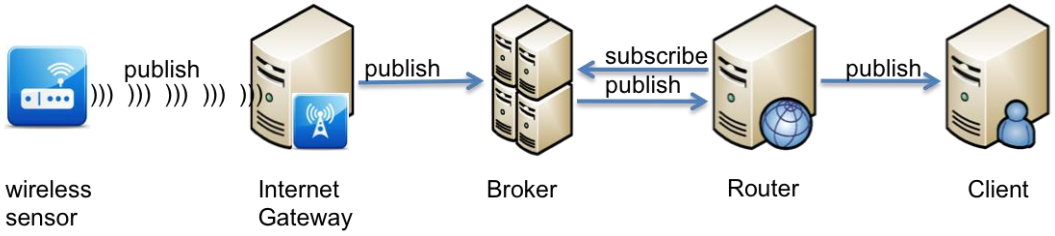


Figure 6-7: Event Processing Chain

Principally, brokers and router serve as mediators between event sources and event sinks and don't provide any further event processing capabilities. Though, both may implement further processing capabilities. The OGC Sensor Alert Service (see section 8.2.4) is an example of such a broker: Sensors register with the service and publish events or simple observations. The SAS instance acts as a complex event processor and analyzes all incoming data. Based on its internal settings, the service generates other types of events then it receives. An example is a blizzard warning services. The service receives data streams and events reporting various weather parameters. Based on the actual constellation of the various parameters in time and space, it then generates events of type "Blizzard". The generation of those new event types might be transparent or opaque to subscribers.

6.4.3.3 Event Role Interfaces

The components described in the Event Role Model (section 6.4.3.2) are differentiated by their functionalities. The SensorSA organises these functionalities in four orthogonal interfaces as illustrated in Figure 6-8. These interfaces can be implemented by components in order to take on a specific role within a distributed event-driven processing system.

Note: These interfaces are specified in section 8.5 of the SensorSA Service Viewpoint in an abstract, i.e. platform-independent form.

Figure 6-8 illustrates the four interfaces and the provided functionalities using UML notation. The interfaces relevant to publishers/producers are shaded in blue; those relevant to receivers/consumers are shaded in purple. All possible implementations are shaded in light yellow.

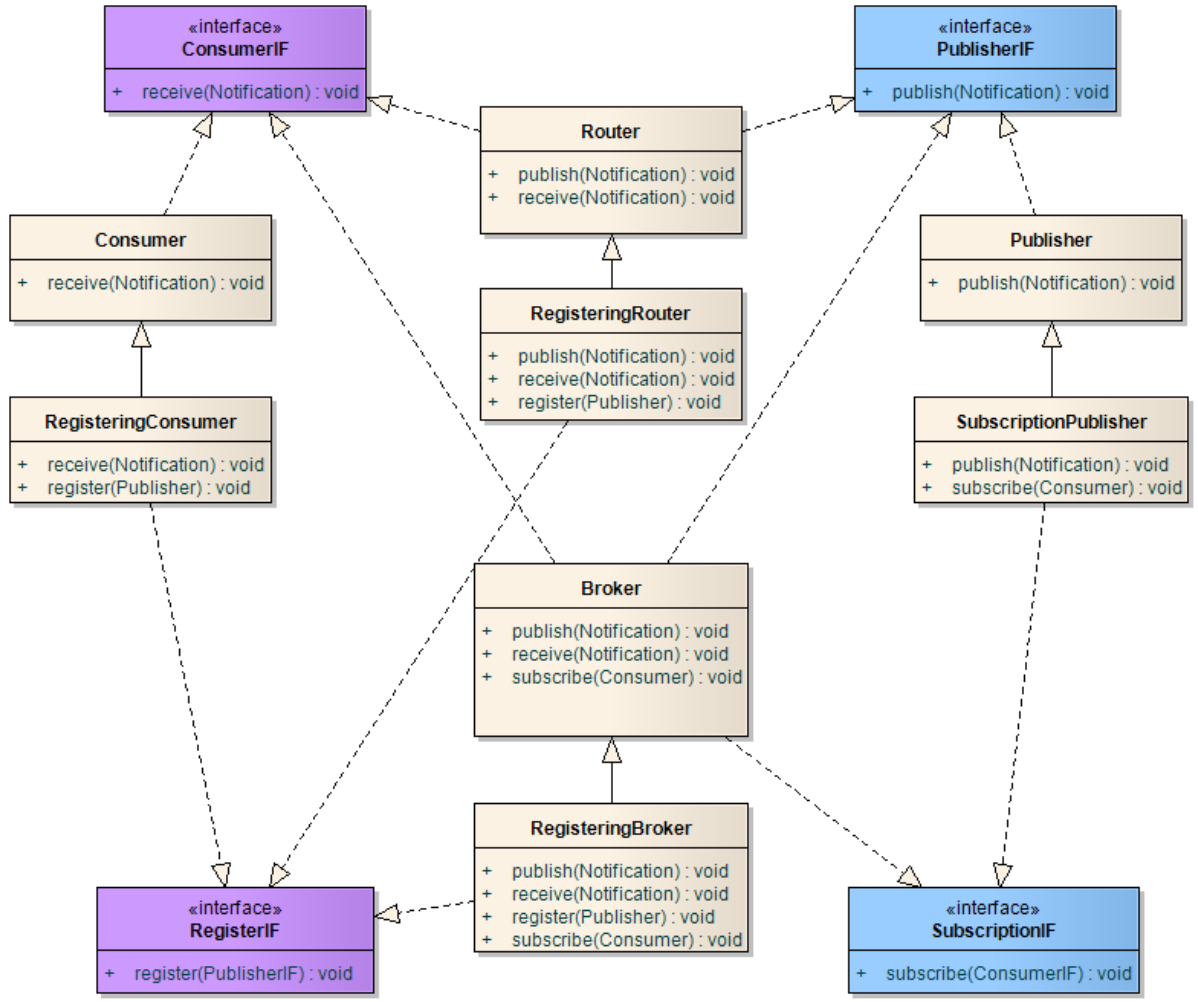


Figure 6-8: Event Processing Interfaces

6.4.4 Exemplary Event Types

The following list of event types has been identified as being of major importance for SensorSA applications[11]:

- Sensor Available: generated by a sensor or sensor system when a new sensor is connected to the sensor network. In general, it's useful for "Plug and Measure" scenarios, especially for event triggered catalogue harvesting

- Sensor Unavailable: Generated by a sensor or sensor system when an existing sensor is unavailable, e.g. disconnected from the sensor network. It may be used for event triggered catalogue harvesting and "Plug and Measure".

- Sensor Timeout: Generated by a sensor or sensor system when a sensor has not responded since a defined period of time.

- Sensor Properties Changed: Generated by a sensor or sensor system when sensor properties change (e.g. recalibration, location change in the case of mobile sensors).

[11] This is a non-exhaustive list without any claim of completeness or lack of redundancy.

- New Sensor Data: Generated by a sensor or sensor system when the sensor acquires new data, e.g. new data is inserted into a Sensor Observation Service.

- Service Trigger: Generated by a service orchestration environment in order to trigger other services, e.g. in service chains.

- Service Created/Deleted: Generated by a service in case of creation or deletion of a service. It may be used to trigger the harvesting of service capabilities.

- Service Capabilities Updated: Generated by a service when the capabilities of a service have been changed. It may be used to trigger the harvesting of service capabilities.

- Threshold Exceeded: Generated when a value has exceeded a given threshold. It may be used in environmental monitoring applications.

- Sensor Battery Low: Generated when the remaining power of a sensor energy supply system (e.g. a battery) has fallen under a given level.

6.5. Resources and their Identification

6.5.1 Resources

In general, the SensorSA denotes by the term "resource" anything that's important enough to be referenced as a thing itself (Richardson/Ruby 2007). Examples of resources in the SensorSA may be sensors, functions (possibly provided by means of services), data objects (possibly but not necessarily modelled as feature types), views upon data objects or descriptions of data objects or services (capabilities).

The SensorSA focuses on a service-centric computing paradigm and puts the services and their interfaces into the foreground. However, these services access and manipulate underlying resources with quite complex schemas (e.g. the elements of the observation and measurement model described in section 7.2). Typically, these resources provide views (subsets) upon data sources driven by the needs of the user. The effects of the service operations heavily depend on the meaning and the status of the underlying resources that are selected by the caller of an operation when setting the operation parameters. An alternate paradigm would be to focus upon these resources and their representations when defining a service platform. This approach is followed by the resource-oriented architecture (ROA) realised by so-called RESTful Web services (Richardson/Ruby 2007).

The SensorSA aims at conceptually linking a service-oriented and a resource-oriented view upon a sensor service network in order to gain flexibility. This approach follows the architectural principles of "technology independence"(see section 4.1.3) and "component architecture independence" (see section 4.1.5).

The concept "resource" is covered in the SensorSA by considering the following aspects:

- The identification of resources is defined below in sections 6.5.2 to 6.5.3.

- A resource model that links basic concepts of a ROA to the concepts “service” and “interface” is defined in the Information Viewpoint in section 7.6.3.

- A SANY RESTful Web Service platform is defined in the Technology Viewpoint in section 9.2.3.

6.5.2 URN Namespace for SANY Resources

Interoperability in sensor network applications depends to a very large extent on a mutual agreement about identifiers of resource types and their underlying semantics. The SensorSA distinguishes between the identifier itself and its semantics:

- The identifier scheme shall be based on Uniform Resource Names (URN) to unambiguously reference location-independent identifiers.

- The semantics of an identified resource are provided when URNs are resolved. As a minimum, the semantics shall be provided in the form of free text descriptions. Optionally, the semantic description may be enriched and made more concrete by references to taxonomies or ontological concepts.

Basically, resources used in the SensorSA are identified in the “Uniform Resource Name (URN) Namespace for the Open Geospatial Consortium (OGC)” (RFC 5165) which is structured as follows:

urn:ogc:{OGCresource}:{ResourceSpecificString}

For resources that are defined in the scope of the SANY Sensor Model (see section 5) or for the purpose of SANY applications, the sub-tree of the OGC resource type “def” as defined by the URN resolver of the OGC Naming Authority (http://www.opengeospatial.org/ogcna) shall be used. The resulting naming scheme for SANY URNs is:

urn:ogc:def:objectType:authority:[version]:code

with the following definitions:

- "objectType" denotes concepts in the form of a controlled list currently defined in table 3 of OGC 06-023r1. In particular, the SensorSA shall use object types that are related to the Sensor Model such as

  - “phenomenon” (observable property definition) or

  - “uom” (unit of measure definition).

  - In addition, the term "eventType" shall be used to denote events (e.g. "sensor available", or "threshold exceeded"). The event URN usually comes with a related phenomenon URN. For instance, the URN urn:ogc:def:eventType: SANY:2009.03:occurrence is related to urn:ogc:def:phenomenon:SANY: 2009.03:earthquake.

- "authority" denotes organisations (e.g. standardisation organisations such as ISO or OGC, but also projects such as SANY) that define the resource identifiers. It is a controlled list defined as follows:

  authority := EDCS | EPSG | OGC | SI | UCUM | SANY

  Note: This list is an extension of table 1 of OGC 06-023r1 by adding the authority "SANY".

- "version" denotes the version of the resource identifier definition as defined by the authority.

  Note 1: For the authority OGC the version is optional (as stated in section 7.2 of OGC 06-023r1): *"The "version" part of these URNs can be omitted when the referenced definition does not have a version, and the referenced definition is not specific to an authority version. When included, the "version" shall be recorded in the format specified by the authority. The version format is sometimes "N.N.N" or "N.N", where each "N" stands for an integer. If no other version identification is provided by the authority, a year or other date can be used. No "v" or other version prefix shall be included."*

  Note 2: For the authority SANY the version is mandatory when defining a URN and it shall be provided in the format "N.N", where each "N" stands for an integer. When referring to a URN and the version number is missing, the resource that is associated with the highest version number shall be taken by default.

- "code" denotes a human-readable name that identifies the resource.

Note: URN namespaces (for OGC and for SANY) must not be confused with XML namespaces used in schema documents of the eXtensible Markup Language (XML). According to the W3C Recommendation "Namespaces in XML 1.0" (http://www.w3.org/TR/xml-names), "XML namespaces provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references".

6.5.3 Naming principles

Section 6.5.2 described how resources are identified in a global scope in form of an URN. To guarantee globally unique identifiers it is necessary to set up policies to generate and administer such identifiers. Such a naming policy must take into account that resources (in particular sensor nodes) in sensor networks have been manufactured by different vendors and/or are operated by independent authorities, all with their own, possibly proprietary, resource identification scheme. Thus, global uniqueness of resource identifiers cannot be assumed to have already been achieved in the sensor domain (see section 6.2). The only solution is to restrict the scope of such resource identifiers to the sensor network in which they have been uniquely defined. It is then the task of the acquisition domain (e.g. an instance of a sensor observation service) to guarantee global uniqueness with respect to other services in the acquisition or other domains.

The following situation highlights the need for this approach: If more than one service instance provides access to the same set of resource instances and if it is necessary for an

application to treat them as one instance (e.g. one instance of a sensor), the identifiers of the resource instances shall be globally unique in order to enable the correlation of the resource instances at a higher level. This is illustrated in Figure 6-9 where one resource instance may be accessed through two resource providers A and B. Here, the resource providers are abstractions of the service instances that provide access to the resources.

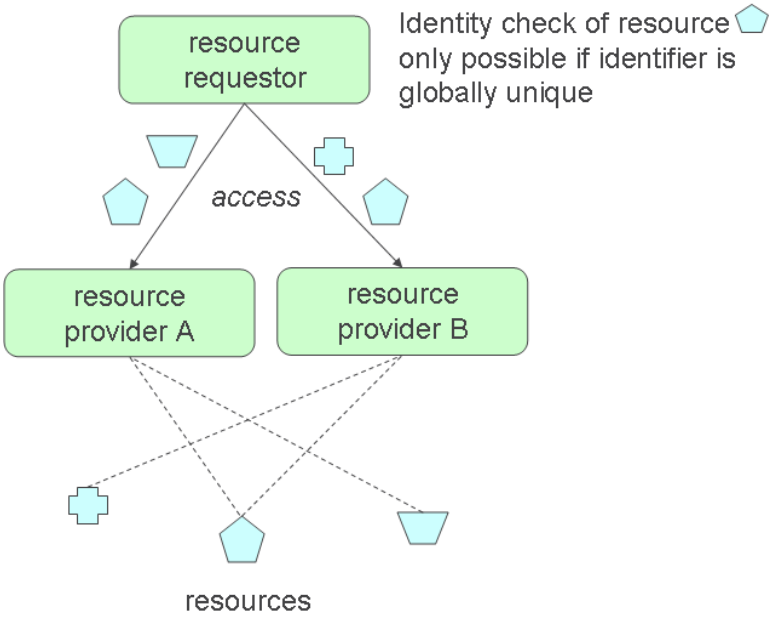


Figure 6-9: Naming requirements for resources

A resource may be defined as composite resource, i.e. its identity relies on the identity of other resources. In this case the identifier of a (composite) resource is defined by a composition of the identifiers of its defining characteristics (attributes). Its global uniqueness is then dependent on the global uniqueness of the identifier of its composing identifiers.

An important example is the resource *observation*. According to the Observation and Measurement model (see section 7.2) the identifier of an *observation* is defined as a tuple consisting of the identifier of the *feature of interest*, the *observed property*, the *procedure* (see the definition of these terms in section 7.2) and the time of occurrence.

Section 4.5 distinguishes between several topologies of sensor service networks. Looking at them from the perspective of how to identify the resources involved, the topologies differ in the relationship between the resource providers and the resources themselves as well as in the cardinalities of this relationship. This has consequences for the requirements about local or global resource identifiers.

As an illustrating example let's consider a service instance acting as a resource provider to access a sensor (e.g. an instance of the Sensor Observation Service (SOS), see section 8.2.2). The sensor is specified as a *procedure* which provides *observations* according to the SOS information model described in section 7.3.

Note: The method by which a resource (here: a sensor) registers itself to its resource provider (here: the SOS instance) is outside the scope of the SensorSA because it is specific to a given sensor network solution.

Two particular relationships are distinguished:

1. The relationship between a procedure and SOS instances

   There is a single SOS instance that acts as a resource provider to a given procedure. In this case the scope of the procedure is the SOS instance, i.e. there is a 1:1 relationship between a procedure and an SOS instance. Thus, the identifier of the procedure needs only to be unique within that SOS instance. However, depending on the sensor network topology, the procedure may also be connected to other service instances (SOS instances or instances of other service types) at the same or at different times. Examples are mobile sensors that connect to different (stationary) SOS instances from time to time. In this case more than one service instance acts as a resource provider for this procedure, i.e. there is a 1:n relationship between a procedure and a service instance.

   Table 6-3 shows the minimum requirements for the scope of the identifiers that result from the different cardinalities between a resource provider (here: SOS instance) and a procedure as a function of the sensor network topologies. A "local" scope of the procedure identifier means that the uniqueness of the identifier is only guaranteed in the context of the SOS instance that acts as a resource provider to access the procedures.

Sensor Network Topologies	Cardinality SOS Instance : Procedure	Scope of Procedure Identifier
Sensors and data logger with fixed locations	1:n	local
Mobile sensors and fixed or mobile data logger	n:n	global
Mobile sensors moving in different sub networks	n:n	global
Mobile sensor cluster on vehicles (e.g. on ships) - block data transfer on demand	1:n	local
Mobile earth observation sensors (satellite, airborne)	1:n	local
Mobile sensors with their own IP address	n:n	global

Table 6-3: Procedure Identifiers in different Sensor Network Topologies

2. The relationship between an SOS instance and the observations.

   More than one SOS instance may provide access to the same set of observations. Each SOS instance may access different but possibly overlapping subsets of observations. Thus, basically, there may be a either a 1:n or an m:n relationship between SOS instances and observations. Note that for this relationship the cardinality of this relationship is not a function of the sensor network topologies. As a consequence, the demand for a local or a global scope for the observation identifier is as follows:

   – In case of a 1:n relationship, the scope of an observation identifier may be local (i.e. only unique in the scope of the SOS instance).

   – In case of an m:n relationship, the scope of an observation identifier shall be global.

6.6. Management

6.6.1 Overview

The management aspects of the SensorSA are discussed in a three-dimensional space as illustrated in Figure 6-10.

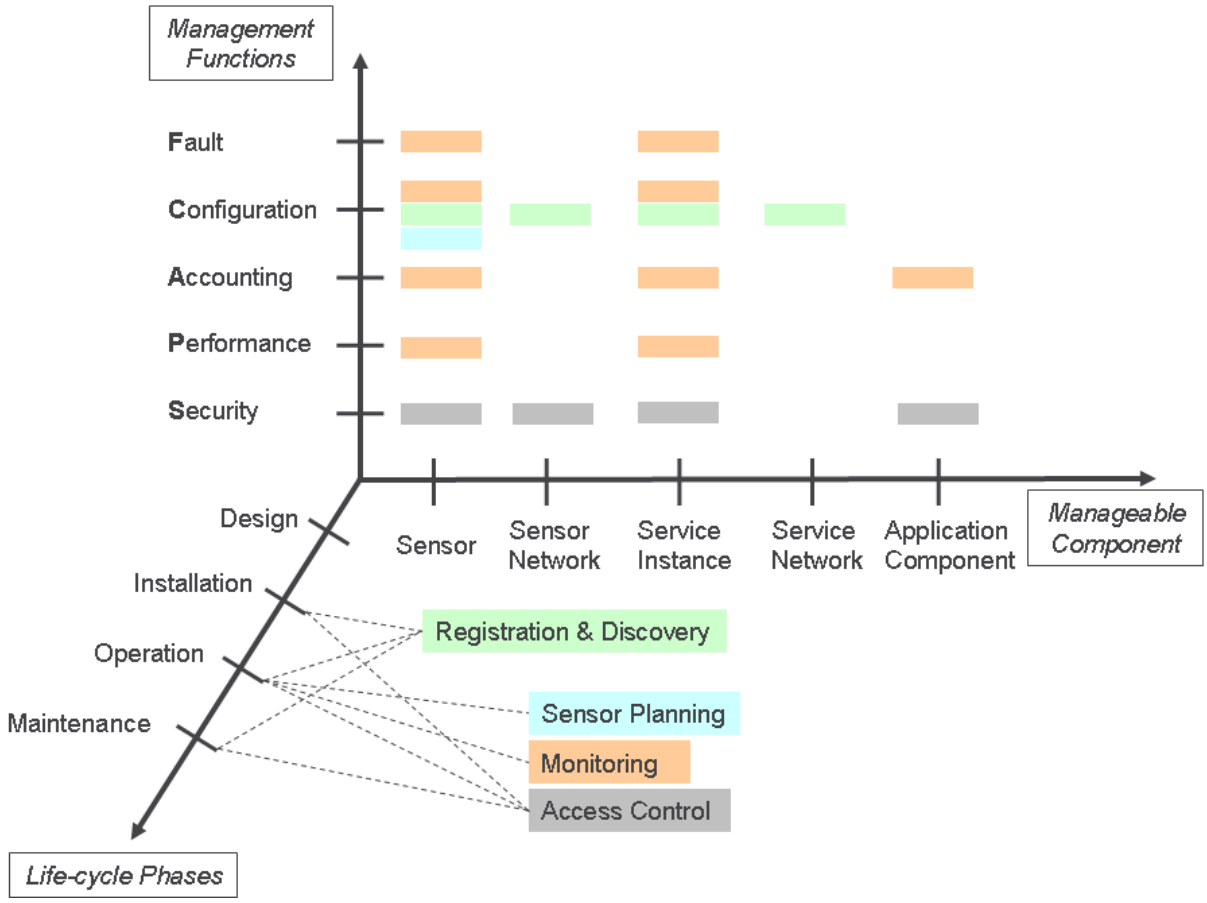


Figure 6-10: Management Space in the Sensor Service Architecture

The first dimension corresponds to the "management functions". These are grouped according to a non-orthogonal classification into functional areas of the ISO system management reference model (ISO/IEC 7498-4):

- Fault management traps and handles faults occurring in managed entities.
- Configuration management modifies the configuration of the manageable components.
- Accounting management tracks resource usage according to different criteria.
- Performance management measures system performance for resource optimisation.
- Security management configures multi-level secure processing domains, detects and traps security violations.

The second dimension is dedicated to the "manageable components" to which the management functions are applied. These are sensors (in the broader sense as defined in the

SANY sensor model in section 5), sensor networks, service instances, service networks and application components whose meanings are defined the SANY glossary (see section 2.4).

The third dimension constitutes the different life-cycle phases that a SANY sensor and sensor service network may follow. These are design, installation, normal daily operation and maintenance phases where the configuration changes with respect to component failures or system evolution.

The SensorSA does not support all management functions applied to all manageable components in all phases of the life-cycle. Table 6-4 summarises the management aspects that are covered in V1 of the SensorSA and provides references to the sections where the concepts and the policies (as part of the Engineering Viewpoint) are described.

Management aspect	supports functional area	applied to components	supported life-cycle phases	concepts described in section	policies described in section
registration and discovery	configuration	sensor sensor network service instance service network	installation operation maintenance	6.6.3	10.2
sensor planning	configuration	sensor	operation	6.6.4	10.4
monitoring	fault configuration accounting performance	sensor service instance	operation	6.6.2	10.3
access control	security	sensor sensor network service instance application component	installation operation maintenance	6.8	10.5

Table 6-4: Management Aspects covered in the SensorSA

6.6.2 Management Architecture

The basic architectural decision taken for the management aspects “monitoring” and “sensor planning” in the SensorSA is that the concepts, models and services of the OGC Sensor Web Enablement (SWE) initiative, in particular the Observations and Measurement model (see section 7.2) and the major OGC SWE services, are applied to the management of manageable components themselves.

This approach may be applied to the management of service instances in all functional domains. However:

- The focus of the SANY project is on sensor service networks.
- In the functional domains above the sensor and acquisition domain there is a conceptual overlap with other standard management architectures such as ISO

Common Management Information Services and Protocol (CMIS/CMIP) and IETF Simple Network Management Protocol (SNMP) for the Internet community.

Consequently, the SensorSA management approach only applies to the manageable components sensor, sensor networks and service instance of the sensor and acquisition domain. Gateways to other management standards may be developed but are currently out of scope of the SensorSA.

As a consequence, the central concept of this approach is the "sensor" as defined in the sensor model (see section 5). By looking at the sensor model from the management perspective it can be seen that the sensor exposes its management capabilities through a dedicated management interface (see section 5.5) in order to enable the configuration and monitoring of the internal behaviour of the sensor

Within a sensor network a variety of sensors exist that provide different types of functionality and data to the sensor client. These sensors might rely on their own proprietary management technology. Thus the management of a sensor network containing tens or even hundreds of sensors would be a very difficult and expensive task in the absence of a standardised management interface.

When addressing the general problem of sensor integration, the management aspect has to be considered, especially when dealing with topics such as sensor monitoring and configuration. Standardised management interfaces and protocols (i.e. the schema of management information and rules about how to exchange it) across all sensors have the following advantages:

- The sensor network management task is simplified by eliminating the plethora of management applications that a service network manager has to use. Instead a single but generic management client is imaginable.

- The integration of a sensor into a service network is reduced mainly to the implementation of the management interface.

The sensor management information, i.e. the management view upon the sensor, must be accessible through a sensor management endpoint. The implementation behind a management endpoint has to be capable of retrieving and manipulating the management information related to the sensor.

Following the terms of ISO (ISO 7498-4) and the IETF management architectures, this set of management information is called a Management Information Base (MIB).

Note: In the SensorSA, a MIB is conceptually equivalent to meta-information for the purpose of monitoring (see section 6.7.5). Thus, it is basically specified as an application schema following the rules of the (RM-OA, 2007). More specifically, the SensorSA models monitoring information as observations according to the OGC Observation and Measurement model (see section 7.2), and configuration information as tasking parameters according to the sensor planning approach (see section 6.6.4).

When specifying a sensor-related MIB, three management views can be distinguished: sensor management, sensor service management and sensor network management. From the sensor model perspective the sensor services and the sensor system may also qualify as

sensors, i.e. they may be modelled as sensors. The CPU temperature, CPU load, file system and network usage are examples of infrastructure management information belonging to the sensor system and modelled as observed properties.

6.6.3 Resource Discovery

6.6.3.1 Introduction

Discovery is the act of locating a machine-processable description of a resource that may have been previously unknown and that meets certain functional, informational or qualitative criteria. Applied to a SANY Sensor Service Network it is the process of searching for information and services (both together referred to as resources). It involves matching a set of functional and other criteria with a set of resource descriptions. The search is based on meta-information (see section 6.3) whose schema has been designed for this purpose. It is directed at a store of meta-information, populated by entries that represent the resources of a SANY Sensor Service Network. In general, the externally visible functionality of such a meta-information store is provided by means of a discovery service. According to the Web Services Architecture (W3C, 2004), a discovery service is used to publish and search for descriptions meeting certain functional or semantic criteria.

Resource discovery in SANY realises the publish-find-bind pattern (OGC 03-040) as illustrated in Figure 6-11. This basic pattern supports the dynamic binding between resource providers and requestors because sites and applications may frequently change in a distributed sensor service environment.

There are three essential roles:

- **Resource provider**: publishes resources to a broker and delivers resources to resource requestors. Note that resource providers are usually software components that represent the resources as a kind of resource surrogate.

- **Resource requestor**: performs resource discovery operations on the resource broker to find the resource providers and to get the information it needs to bind to the resource provider. Using the bind information as address it then accesses the resource providers for provision of the desired resource.

- **Resource broker:** helps resource providers and resource requestors to find each other. Resource brokers provide a functional interface to an underlying meta-information store. In geospatial service environments such as a SANY Sensor Service Network, resource brokers are usually called (geospatial) catalogues (ISO 19119). The SANY catalogue service is described in section 8.2.

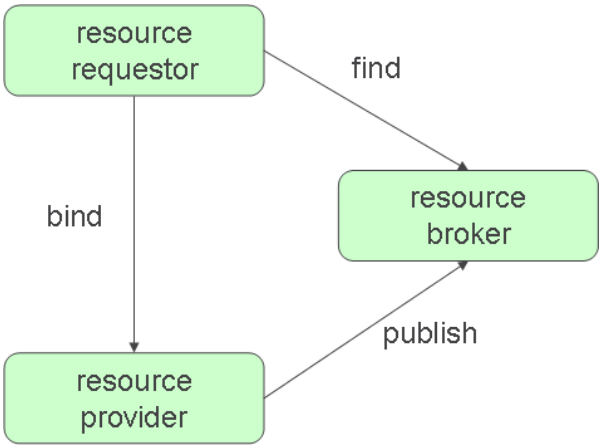


Figure 6-11: Publish-Find-Bind Pattern

The publication of resources of the resource provider to the resource broker may follow either a push or a pull paradigm. In a push paradigm the meta-information entries are created, updated and deleted by actively calling corresponding operations of the catalogue service. In this case, the resource providers act in a client role. In a pull paradigm, the resource broker acts in a client role and retrieves meta-information (e.g. the capabilities documents of resource providers) by calling corresponding operations of the resource providers. The SANY catalogue service supports both paradigms.

The following outlines what spectrum of resource types need support for discovery in a sensor service network. Discovery policies, typical catalogue queries and the service interaction patterns that realise the typical queries are presented in section 10.2

6.6.3.2 Resource and Catalogue Types

The discovery process is designed such that it may discover resources of any type that are possibly available in a geospatial resource network. However, this specification focuses particularly on the resource types that are specific to a SANY Sensor Service Network. The main resources to be discovered follow the concepts that are defined by the information model about Observations and Measurements (see section 7.2). This model also describes the relations between the different types of resources.

The discoverable resource types are

- feature of interest (FOI) that represents the observation target,
- observed property of a FOI that describe the phenomenon to be observed,
- procedure which encompasses sensors but also algorithms or simulations,
- observation about the phenomenon that has been generated by the procedure.
- service types and instances which deal with the resources listed above. Examples are services to obtain observations (Sensor Observation Service, see section 8.2.2) and services to influence the way the measurement is taken (Sensor Planning service, see section 8.2.3), and
- sensor networks as a container for a set of interconnected sensors (procedures).

It is important to distinguish between a resource type and an individual physical instance of a resource. A user may query for all observations processed by a specific method. In this case the user references a resource type. The user may also be interested in all observations that are provided by a specific physical sensor. In the latter case the user references a physical instance of a resource.

Usually, there is no direct access to physical sensors in a SANY Sensor Service Network. Instead, resource providers (usually service instances) act as surrogates for physical sensors. It is important to note that more than one service instance may contain different views of the same instance of a resource (e.g. a sensor or an observation). Each service provides meta-information about itself and the resources it handles. For example a Sensor Observation Service provides meta-information about its own instance (e.g. service provider) but also on the FOI, the observable properties and the procedure used.

A catalogue, i.e. an instance of the Catalogue Service (see section 8.2), acts as the primary resource broker in discovery models. Table 6-5 shows several types of catalogues depending on the set of meta-information about resources types that are stored in the catalogue. Note that the term sensor in this table is used according to the SANY sensor model, including physical devices but also simulation models as defined in section 5.

The meta-information schema is described in detail in section 7.6.3 as part of the Information Viewpoint of the SensorSA.

Catalogue Type	Description
Full Catalogue	catalogue containing information about all defined resources types
Feature Type Catalogue	catalogue containing definition of the feature types, feature attributes and feature associations occurring in one or more sets of geographic data, together with any feature operation that may be applied (see (ISO 19110:2005), but referred to there as a feature catalogue)
Property Type Catalogue	catalogue containing definition of the property types including at least their identifiers, their names and human-readable descriptions (possibly in multiple languages), semantics, synonyms and default units
Sensor Type Catalogue	catalogue containing definition of the sensor types including their classification scheme, their names and human-readable descriptions (possibly in multiple languages) and references to property types made available by a sensor of this type
Sensor Catalogue	catalogue containing information about instances of sensor types available in a SANY Sensor Service Network.
Service Catalogue	catalogue containing information about instances of service types available in a SANY Sensor Service Network.

Table 6-5: Catalogue Types in a SensorSA

6.6.4 Sensor Planning

6.6.4.1 Introduction

The term sensor planning is used in this architecture specification in a very broad way and covers the aspects of sensor configuration (sensor tasking), sensor tasking feasibility analysis as well as updating and modifying sensor tasking instructions at runtime.

Note 1: The term "sensor planning" is used as it is the standard term defined by OGC, although the term "sensor tasking" would be more appropriate.

Note 2: The term "sensor" is used according to the sensor model described in section 5, i.e. it includes simple forms of physical sensors, complex forms of physical sensors as well as models and all sorts of combinations. Additionally, sensor planning allows the tasking of actuators.

Note 3: Although actuators are not in the focus of the current SANY Sensor Service Architecture, they are mentioned at this stage as they play a major role in context of tasking. Sensors are usually mounted on some form of platform. Often the tasking addresses the platform (which is an actuator) rather than the sensor itself if, for example, the "sensor" is sent to a new location.

The goal of sensor planning is to hide the complexity of the sensor from the user. The same operation shall be provided to the user to task a buoy observing wave heights somewhere in the ocean, a simulation model calculating the weather for the next day, or a simple A plus B operation. The user shall only be confronted with a list of parameters that they might set (so called *tasking parameters*). All other complexity shall be hidden.

Sensor Planning takes place in each of the functional domains identified in section 6.2. However, the same interface type is used to provide a façade to the tasking of each specific domain layer. This means that the sensor planning interface shielding the sensor domain differs from the interface shielding the mediation and processing domain only by its tasking parameters, not by the interface itself. The general information model, encoding, and operations remain the same.

As an interface to the sensor domain, sensor planning allows (re-)configuration and managing of individual sensors, e.g. changing the sampling frequency. Sensor planning of the acquisition domain allows the tasking of individual missions. An example would be the tasking of a set of sensors that observe a specific area: a satellite with a mounted radar sensor, another satellite with electro-optical-sensors as well as some in-situ observations on ground are triggered to produce a complex data set of the area of interest. Sensor planning on the mediation and processing domain allows the integration of processing steps. Here, sensor planning may act as a process orchestration and chaining engine. A user might provide a set of interface locators that will be used to build a processing chain on the fly. The application domain as well as the user domain usually aggregate various sensor planning services and provide interfaces to the users. A user will be provided with a form that allows easy entry of tasking parameter data. These data are then sent to a sensor planning service on the application domain to execute necessary actions.

6.6.4.2 Sensor Planning Information

The communication between a Sensor Planning Service and a client consists of the following information items:

- Meta-information about the service

  The service shall be self-describing (section 7.7.2.)

- Meta-information about tasked sensors

  The service shall provide all information about the sensors that will be tasked by the service (e.g. type of sensor, location, accuracy etc.)

- Tasking parameters

  The service shall describe the kinds of parameters required to submit a tasking request. Those parameters might be in direct relation to the sensor, e.g. the looking angle in degrees for a frame camera, or they are more abstract parameters, e.g. the different modes “normal”, “severe”, or “fatal” for an observation campaign. The tasking parameter has to be semantically defined, though the executed actions might be transparent to the user of the service. The client provides values of the required parameters in order to start the tasking or check the feasibility of a potential tasking request.

- Status information of tasking requests

  The service shall describe if a tasking request is still in queue, currently executed, idle etc.

- Status information of feasibility requests

  Analogous to the status information of tasking requests, the service shall be able to report the status of a feasibility check.

6.6.4.3 Service Planning Functions

In the following the major functional requirements of a service that realises sensor planning are listed from an abstract point of view.

- Sensor Description

  The service shall be able to describe the sensor itself. The sensor and the tasking parameters are different aspects. Therefore, the sensor description shall focus on the sensor itself rather than on the description of the tasking parameters of the sensor. The sensor description primarily serves the purpose of identifying the service instance as an instance that provides access to a specific sensor with its features. This information can be stored in registries or catalogues to foster discovery.

- Description of tasking parameters

The service shall be able to describe the tasking parameters of the sensor. The description shall follow design and encoding rules in order to allow usage of previously unknown sensors. The level of detail of those rules defines the level of interoperability, as only very strict rules allow tasking of new sensors fully automatically, i.e. without human intervention.

- Feasibility Analysis

  The service shall be able to test a tasking request for feasibility. This allows the user to determine if and under what conditions a tasking would be feasible before submitting a potentially expensive tasking request. The service itself shall provide detailed responses, indicating the feasibility itself as well as potential alternatives.

- Submission of tasking requests

  The service shall be able to accept tasking commands. The tasking instructions shall follow the definitions provided by the service in its asset tasking description.

- Update of submitted requests

  The service shall allow the user to update any submitted request, either feasibility test or concrete tasking command. Additionally, the service itself shall be able to request update information from the client. This situation occurs when the tasking had been stopped and additional information becomes necessary in order to proceed with the tasking.

- Cancellation of submitted requests

  The service shall allow the cancellation of previously submitted feasibility test requests or tasking requests. It is the responsibility of the service to bring the tasked asset back into a safe position. This should be transparent to the user.

- Status description

  The service shall provide information about the current status of a submitted feasibility test or tasking request. The response sent by the service shall contain all available information about the current status of a request. As this information depends to a large extent on the concrete application, the service shall be able to describe the parameters of the status report in its self-description or as part of other request responses.

- Description of Result Access Mechanisms

  The service shall be able to provide information about how to access potential data that are produced in response to a tasking instruction.

In the SensorSA, Sensor Planning activities are based on the Sensor Planning Service (SPS) as specified by the Open Geospatial Consortium (OGC 07-014r3) and described in section 8.2.3.

6.7. Meta-information Approach

6.7.1 Introduction

The approach to describing meta-information within the SANY Architecture is based on the ORCHESTRA meta-information approach, as described in Annex A3 of (RM-OA, 2007). According to this approach a conceptual meta-information model, which is a human understandable representation of the meta-information needed, has to be developed for a given purpose. Based on the user requirements (see section 4.6), the following purposes for meta-information have been identified:

- Data and Service Integration
- Interpretation
- Discovery
- Monitoring
- Authentication and authorisation
- User profiling
- Quality control / management

6.7.2 Data and Service Integration

For the purpose of service integration a service has to provide meta-information that describes the service. This meta-information comprises the structure of the implemented interfaces, the descriptions of the operations that can be performed including the descriptions of the parameter and return types, the location of a service instance (e.g. its URL) or additional characteristics of the service (e.g. costs) that enable service selection.

For the purpose of data integration a service has to expose meta-information regarding the data it provides describing the structure, the location (where can it be accessed), geo-spatial information, quality and precision information, measurement unit, measured phenomenon, and the measurement and processing (e.g. filtering) procedure, among other information.

6.7.3 Interpretation

Meta-information is needed for the explanation and understanding of resources (data and services). Resource descriptions shall contain explicit semantic descriptions or pointers to vocabularies (dictionaries) in order to ensure the self-description of services and data and their semantically correct integration.

6.7.4 Discovery

Meta-information is extensively used for resource discovery and is described in detail in section 6.6.3. The conceptual meta-information model is defined in section 7.6.3.

Typical examples of meta-information necessary for the support of the search functionality are keyword-lists, spatial-temporal information and bounding areas. Examples of meta-information for the purpose of navigation are descriptions of the content and structure of catalogue content.

The discovery of services requires a specific meta-information model and dedicated query languages to access the meta-information entries. Meta-information may be semantically annotated in order to increase the quality and the recall of the discovery process.

Note: For automatic service discovery meta-information based on semantic service descriptions (e.g. OWL-S or WSMO) could be provided. However, as there is not yet a generally accepted standard, semantic service specifications will not be considered in the scope of the SANY project.

6.7.5 Monitoring

According to section 6.6.2 monitoring is applied to the manageable components “sensor” and “service”. Meta-information for the purpose of monitoring includes status, actual load, usage statistics (e.g. amount, quality, resolution and time span of downloaded data, used processing time), execution traces, etc. This meta-information is especially useful as input for composite services (e.g. services resulting from service orchestration) that rely on the data provided by other services. Furthermore, accounting applications that audit the usage of resources (e.g. as a pre-requisite for billing) rely on such monitoring information. Meta-information concerning accounting is a combination of the principal and some measure (quota) for resource usage.

6.7.6 Authentication and Authorisation

Authentication and authorisation rely on meta-information necessary for controlling the access to services and enforcing access control policies (see section 6.8.2).

Typical meta-information for the purpose of authentication includes the identifier of the principal that uniquely identifies the subject.

For the purpose of authorisation, meta-information necessary to enable restriction of the usage of resources on a per-principal basis have to exist. An authorisation process is used to decide whether a principal is allowed to access a certain resource or not. This type of meta-information is directly related to the services implementing the authorisation paradigm and is of minimal or no relevance for anything else. A specific set of meta-information makes up the authorisation context that is used by an authorisation service to decide on the authorisation for a given request before allowing access to the requested service operations.

6.7.7 Quality control and management

Quality control and management is concerned with meta-information needed to enhance quality of information and services as well as to increase trust in information, data and services. Quality control and management is needed when certain criteria need to be fulfilled by data and/or services. The SensorSA currently focuses on the following quality aspects of data:

- Information about the measurement and data preparation process, e.g. measurement principle, calibration, spatial and temporal resolution
- Uncertainty of measurements or model calculations, e.g. absolute and relative errors of measurement data or computational errors of data processing services.
- Quality assurance of measurements, e.g. information about whether the measurements have been validated by machines or by humans.

Each of these aspects is more or less relevant for a given application scenario. Often this level of detail is not necessary in order to classify the quality of data. The SensorSA allows an application designer to use the parts that are specifically relevant to their application.

6.7.7.1 The measurement process

The process used to take measurements obviously has a big influence on the quality of the gathered observations. Since for most applications this information is important when processing the observations, information about the measurement process has to be provided together with the observations. Examples of things that influence the measurement process can be:

- Environmental conditions when taking the measurement
- Type, manufacturer, model, etc of the measurement device
- Operating parameters of the measurement device
- Status of the measurement device (error conditions, etc)
- Calibration processes applied to the measurement device
- Amount of processing that has been applied to the data (whether raw or filtered data)

Although this information is very important, it is very dissimilar in different application domains. Even within application domains (e.g. air quality) differences exist because of different legal regulations in different countries, for example. Thus only a generic data model can be specified to describe the measurement process. The SensorSA uses the schema defined by the OGC SensorML specification (Botts, 2005) for the description of measurement processes. Furthermore, information that is specific to each measurement shall be encoded using the Observations & Measurement schema defined in (Cox, 2007) and described in section 7.2.

6.7.7.2 Uncertainty

All data in SensorSA has an associated uncertainty depending on the available meta-information on how the data was observed (measured) or derived from other data sources. We first address measurement uncertainty and then uncertainty of general data.

Following ISO GUM 1993, Barry N. Taylor and Chris E. Kuyatt (1994) and UKAS (2007), measurement uncertainties may be classified into two categories:

- Type A: uncertainty arising from a random effect; evaluated by statistical methods
- Type B: uncertainty arising from a systematic effect, evaluated by other methods

A common way of evaluating a type A uncertainty is to compute the standard deviation of the mean of a series of independent observations. A second common technique is an analysis of variance (ANOVA) and random effects in data in dependence of experimental parameters.

Type B uncertainty is evaluated using scientific judgement. A typical cause is measurement bias due to the calibration of the measurement instrument or its behaviour in given environmental conditions (e.g. temperature, air pressure), or over time (deterioration of instrument, measurement drift). It is evaluated based on information about the instrument and environment. The measurement values may be corrected to compensate for known systematic effects.

Note the distinction between the terms error of a measurement and uncertainty. Error is the difference between the measured value and the (in general unknown) 'true value' of the measured property. Uncertainty is a quantified description of the doubt about the measurement result. The error of a measurement may be small, even though the uncertainty is large.

In SensorSA data arises not only from sensor measurements and observations, but also from data processing with specific services, e.g. a kriging algorithm to generate a spatial coverage from a set of measurement points, or a time series analysis to produce a temporal interpolation. The results of such data processing steps are themselves uncertain, on the one hand due to the uncertainty of the input data, on the other hand due to the probabilistic or approximate nature of the processing itself.

Uncertainty of data is typically expressed with one of the following

- Probability density function, e.g. a normal distribution with known mean and variance. The data value would then lie within one standard deviation of the mean with probability 68% and within two standard deviations with probability 95%.
- Intervals (the data value lies in [a,b]). This does not a-priori assume a uniform distribution on this interval; this would however be the case if the distribution of maximum entropy were chosen. An important special case is when then the measurement instrument can assert that the data value is below or above a given threshold, but can provide no further information.
- Statistics such as standard deviation and moments, or quantiles (the data value lies in [a,b] with probability 95%).

Within the SensorSA, the uncertainty of data sets is described using the UncertML (Williams et al, 2007). UncertML, which was developed within the INTAMAP project[12], allows the information modeller to describe the uncertainty of a specific data set in an interchangeable way using an XML document conforming to the UncertML schema. This XML document can be embedded in a SensorML document to express information about the uncertainty of some process. In addition, UncertML can also be embedded in an Observation & Measurement document (Cox, 2007) to express the uncertainty of a specific sensor observation.

6.7.7.3 Quality assurance

In some application domains, observations sampled by a sensor have to be quality controlled by some automatic or manual process before they can be further processed. Mostly depending on legal regulations, these quality assurance procedures are often specific to an application domain and/or to some organisational unit (e.g. a country). Thus, similar to information about the measurement process, this information has to be described using some generic data model.

Again, the OGC SensorML specification and the OGC Observation & Measurement specification provide mechanisms to describe such information. Depending on the granularity of the information it can be described in a SensorML document if it applies to the measurement process as a whole, or in an Observation & Measurement document (Cox, 2007) if it is specific to a measurement value.

6.8. Security

6.8.1 Introduction

Security aspects are an integral part of the SensorSA as most of the measures that aim at achieving a certain level of security have, at least to some extent, effects on applications and their interactions. Following (SOA-RA, 2008), *"security is one aspect of confidence – the confidence in the integrity, reliability, and confidentiality of a system"*, here sensor networks and sensor service networks.

The provision of an overall model for all aspects of security is out of the scope of the current SensorSA specification. In the SensorSA the focus lies on the regulation of arbitrary access to resources through a service interface (see section 6.5.1). The security model as part of SensorSA does not distinguish between accidental actions or malicious intent of a user to compromise the access to a resource. However, it does not provide dedicated means to respond to the following potential security threats of a malicious user as listed in the OASIS Reference Architecture for Service-oriented Architecture (SOA-RA**,** 2008)

- Message alteration: an attacker is able to modify the content (or even the order) of messages that are exchanged without the legitimate participants being aware of it.

- Message interception: an attacker is able to intercept and understand messages exchanged between participants.

[12] See the INTAMAP Web site at http://www.intamap.org/

- Man in the middle: an attacker attempts to convince each participant that they are their correspondent; whereas in fact they are not.

- Spoofing: an attacker convinces a participant that they are really someone else – someone that the participant would normally trust.

- Denial of service attack: an attacker attempts to prevent legitimate users from making use of the service.

- Replay attack: an attacker captures the message traffic during a legitimate interaction and then replays part of it to the target.

- False Repudiation: a malicious user completes a normal transaction and then later attempts to deny that the transaction occurred.

There are known counter measures using security concepts for each of the listed threats that can be taken on different levels of a protocol stack. However, in a sensor (service) network, this may require physical protection of hardware (deployed sensors), intrusion detection in source systems or protection against eavesdropping of communication channels, application and situation-dependent actions. This is outside the scope of the SensorSA. However, all key security concepts listed in ISO/IEC 27002 to counter the different threats aspects like confidentiality, integrity, availability, authentication, authorisation and non-repudiation (see the definition of these concepts in the glossary in section 2.4.2) need access control as a basic mechanism to regulate access to resources in the first place.

Many security measures in a sensor (service) network are dependent on the sensor service and sensor network topology. Additionally, specific threats require specific counter measures that in most cases cannot be handled on an abstract architectural level only. As a consequence, the SensorSA specification focuses on those concepts that can be defined independently of the underlying use case specific security requirements. Platform-specific security concepts have to be specified as part of the implementation architecture (see Figure 2-1).

To summarise, the SensorSA security model focuses on resource protection based upon a flexible access control pattern. It provides a solution that can serve as the foundation of most other security concepts and adjunct topics like protection against malicious system interaction, licensing and digital rights management.

6.8.2 Access Control

Access control is understood as the ability to permit or deny the use of a particular resource by a particular entity. In general, access control mechanisms ensure that only authorised entities may access resources using well defined methods that comply with the security policy of the system.

The way access control is performed in the SensorSA is described as follows:

1. An abstract access control pattern is introduced in Figure 6-12. It is the basis for all use cases that require access control, including licensing and the management and enforcement of digital rights.

2. Access control tasks that underpin the access control pattern are described. These tasks comprise Profile Management, Identity Management, Authentication, Authorisation Policy Enforcement and Policy Management.

3. The access control architecture comprises the supporting access control services (see section 8.3) and the underlying access control information model (see section 7.4).

The SensorSA realises access control according to an abstract access control pattern. This pattern has been introduced as a data-flow diagram in the OASIS standard of the eXtensible Access Control Markup Language (XACML) (OASIS 2005) and applied, among other places, in the OGC Geospatial Digital Rights Management Reference Model (OGC 06-004r4). An extended version of the pattern is illustrated in Figure 6-12. It seizes on general ideas of *policies* as the basic mechanism to support the governance of service-oriented architectures. In (SOA-RM, 2006) a policy is defined as "the representation of a constraint or condition on the use, deployment, or description of an owned entity as defined by any participant".

The abstract access control pattern explains a controlled call of a service operation request. It assumes that the rules that express the constraints and conditions defining "who may access which resource using which action" are recorded in an access control policy statement of the service. The access control pattern uses the following components:

- The *Subject* is the requestor of the service operation. It represents the acting entity (e.g. user).

- The *Identity Provider* (IdP) issues a ticket as proof of successful provision of a subject's credentials.

- The *Authentication Provider* (AP) has the task of verifying issued tickets.

- The *Policy Enforcement Point (PEP)* receives a service operation request, enforces the access control policy of the service and forwards the service operation request to the protected service.

- The *Policy Decision Point (PDP)* responds to an authorisation request with an authorisation decision.

- The *Policy Information Point (PIP)* holds the services policy information.

- The *Policy Administration Point (PAP*) provides an interface to perform administrative tasks on policy level.

Note 1: This pattern also applies to event-based systems as long as the exchange of events is handled through notification services as discussed in section 6.3.3.

Note 2: In OASIS and related work the term PDP is used for the actual software implementation of the concept "PDP" as defined in the SensorSA.

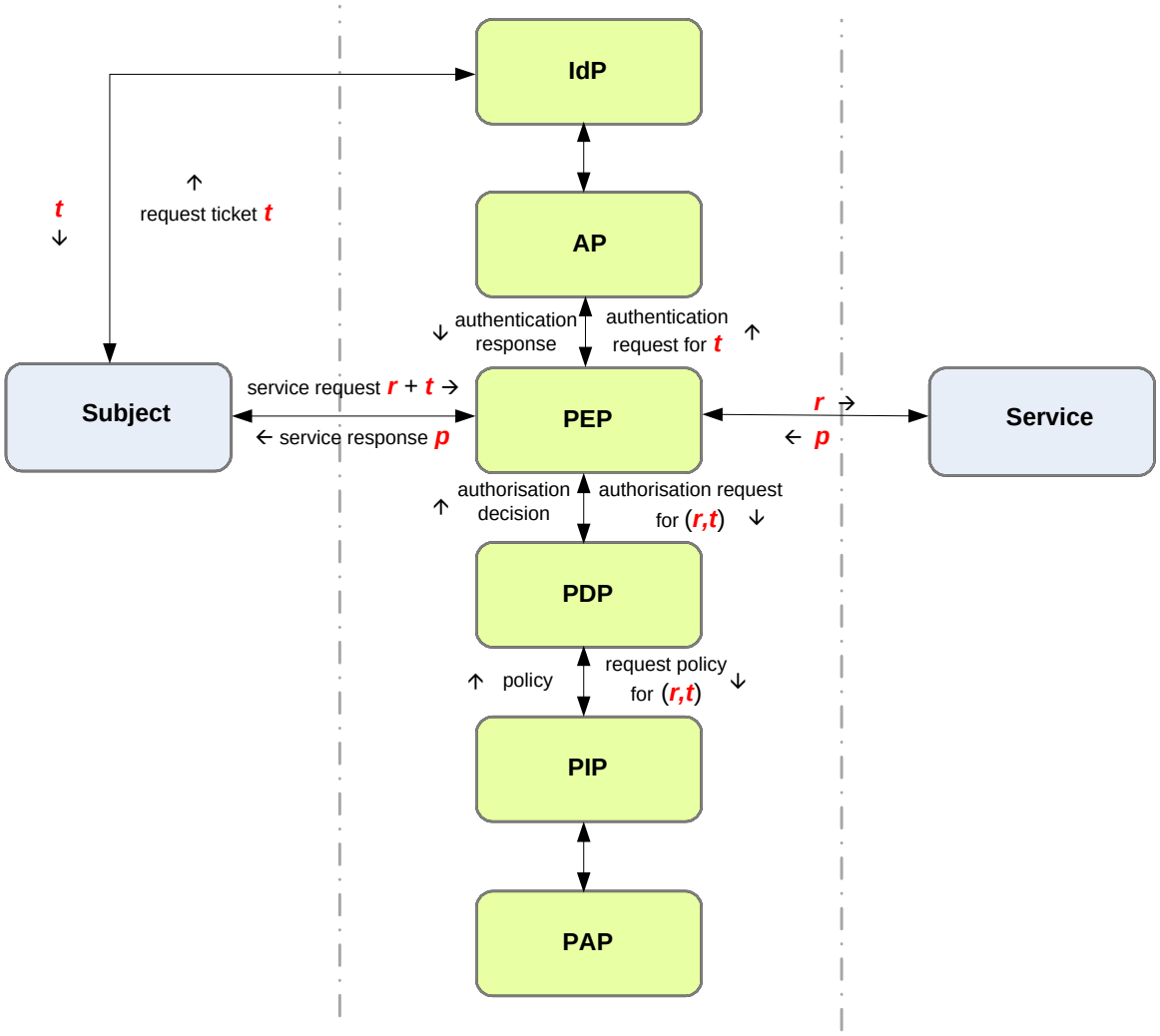


Figure 6-12: Abstract Access Control Pattern

6.8.3 Access Control Tasks

The major tasks of access control comprise

- Profile Management (section 6.8.3.1),
- Identity Management (section 6.8.3.2),
- Authentication (section 6.8.3.3),
- Authorisation (section 6.8.3.4)
- Policy Enforcement (section 6.8.3.5) and
- Policy Management (section 6.8.3.6)

In the SensorSA, these tasks are supported by a set of services that are designed for an evolving heterogeneous environment and offer a high level of flexibility. Each of these tasks is defined in the following sub-sections. Their position in the abstract access control pattern is illustrated in Figure 6-13.

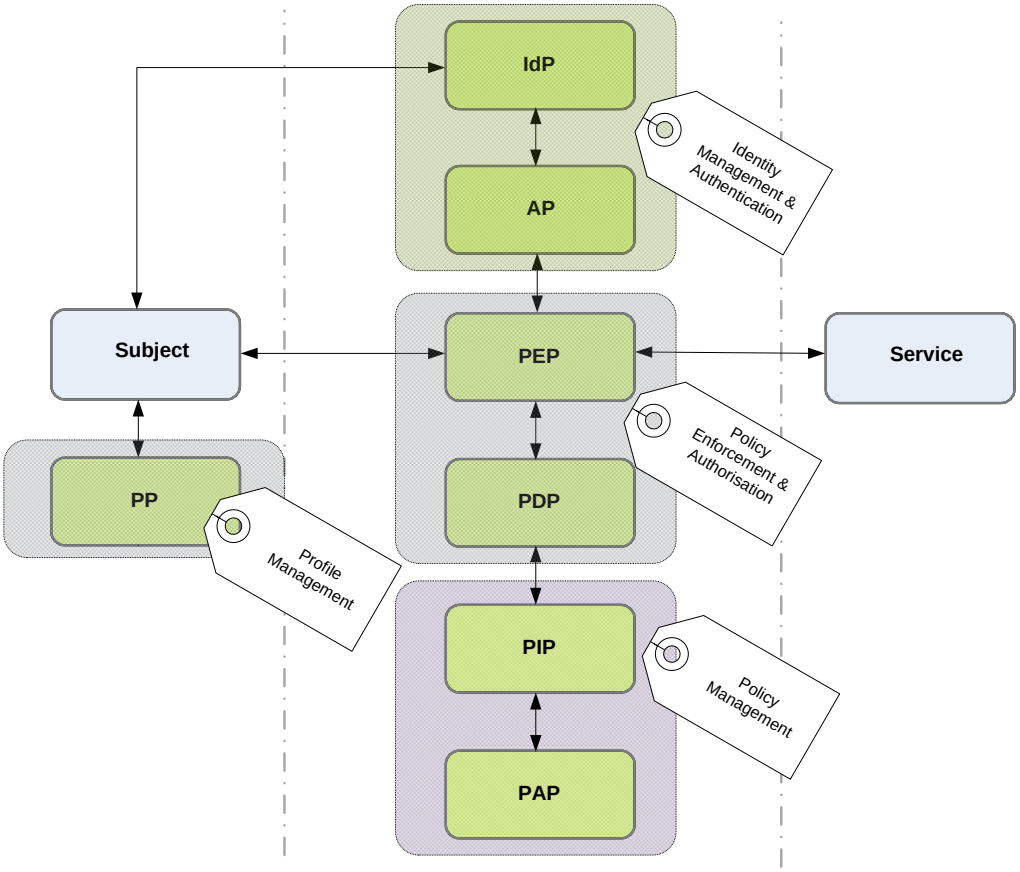


Figure 6-13: Abstract Access Control- Pattern and Access Control Tasks

6.8.3.1 Profile Management

Profile management is an essential basis of a sound security architecture. The major objective of profile management in the SensorSA context is to map a real world user to a representation in a sensor service network (user registration). This representation is called a profile. A profile represents an acting entity which may be a human user or software component like a service. In order to support multiple authentication mechanisms simultaneously and to keep authorisation irrelevant information out of the access control mechanism, profiles and their identities (aka principals) are separated. The concept of an identity constitutes the key entity on which an authorisation decision is mounted, regardless of the underlying access control mechanism. In contrast to that, a profile provides information on the real world entity. The relation between profiles and identities is reflected in the profile and identity model specified in section 7.4.

The main functions covered by Profile Management are the creation, update and deletion of instances of profiles and related information in particular references to identities.

6.8.3.2 Identity Management

An identity is the core information required to realize access control. To interlink access rules and acting entities (subjects), access rules refer to identities and associated properties (attributes), issued by an identity provider and verified by an authentication provider.

The main functions covered by Identity Management are

- management of identity related information e.g. credential management, identity attribute management, and
- definition and management of identity groups.

6.8.3.3 Authentication

According to (SOA-RA**,** 2008), authentication concerns the identity of the participants in an exchange. Authentication refers to the means by which one participant can be assured of the identity of other participants.

When applied to the SensorSA profile and identity model as introduced in section 7.4, the participants are profiles of real world entities that are represented by their identities. Authentication in the SensorSA is the process of verifying the identity of a certain profile. During the authentication process an entity proves that it is allowed to act with the corresponding identity. This proof normally depends on the acting entity's credentials that can be, for example, what somebody has (e.g. key, smart card), what somebody knows (e.g. password), what somebody is (e.g. biometrical data), the place somebody resides (e.g. a certain computer) or the skills of somebody (e.g. a handmade signature). The SensorSA uses the term ticket to denote the result of an authentication process.

Note: Synonymous terms for ticket are assertion (e.g. in the context of SAML, see section 7.4.6.1), session (as a temporarily valid ticket) or token.

The issuer of an assertion acts as delegate for all service providers accepting assertions from this authentication authority. In this way an acting entity is not forced to present its credentials (e.g. a secret) at each service call and authentication can be done centrally.

Assertions can be verified and are used for all actions that require proof of identity. In general, an assertion encompasses all identity related information that is required to perform an authorized request. Moreover assertions may contain information about the authentication provider, expiry date, etc.

The main function covered by Authentication is the verification of identity related information

6.8.3.4 Authorisation

According to (SOA-RA**,** 2008), authorisation concerns the legitimacy of an interaction. Authorisation refers to the means by which an owner of a resource may be assured that the information and actions that are exchanged are either explicitly or implicitly approved.

When applied to the SensorSA access control information model (section 7.4), authorisation is the process of determining whether an identity is allowed to have specified types of access to a particular resource. This is done by evaluating applicable access control information mainly consisting of an authorisation request and a policy. This information is used by the authorisation service to determine an authorisation decision. Usually,

authorisation is carried out on the basis of successfully authenticated identities that are part of an authorisation request.

6.8.3.5 Policy Enforcement

Policies or access rules can be expressed in many ways from simple access control lists to complex statements in policy languages like the OASIS eXtensible Access Control Markup Language (XACML) (OASIS 2005). The actual application of access rules is performed through the combination of Authentication (section 6.8.3.3) and Authorisation (section 6.8.3.4) and the actual enforcement of access control decisions.

6.8.3.6 Policy Management

Access control tasks include the provision of means to manage access rules.

The main functions covered by Policy Management are

- creation, update and deletion of instances of policies,
- definition and management of policy templates for certain frequently used access control patterns, and
- distribution of policy templates.

6.8.4 Access Control Service Architecture

As illustrated in Figure 6-13, access control in the SensorSA is accomplished through the interaction of services, each of which fulfils one or more of the access control tasks described above:

- The Profile Management Service (see section 8.3.2) manages profiles and their relations to identities.
- The Identity Management & Authentication Service (see section 8.3.3) is responsible for the management of identities, their authentication and the management of credentials. An instance of the Identity Management & Authentication Service acts as both authentication provider (AP) and identity provider (IdP).
- The Policy Management and Authorisation Service (see section 8.3.4) supports the management of policies, acting as policy administration point (PAP) as well as policy information point (PIP). Moreover, as an instance of the authorisation service interface it acts as policy decision point (PDP)
- The Policy Enforcement Service (see section 8.3.5) handles the necessary interaction (authentication & authorisation) to obtain the required access control decision and is independent of the controlled service (generic).
- The Service Proxy mimics the controlled service and delegates the service request to the Policy Enforcement Service.

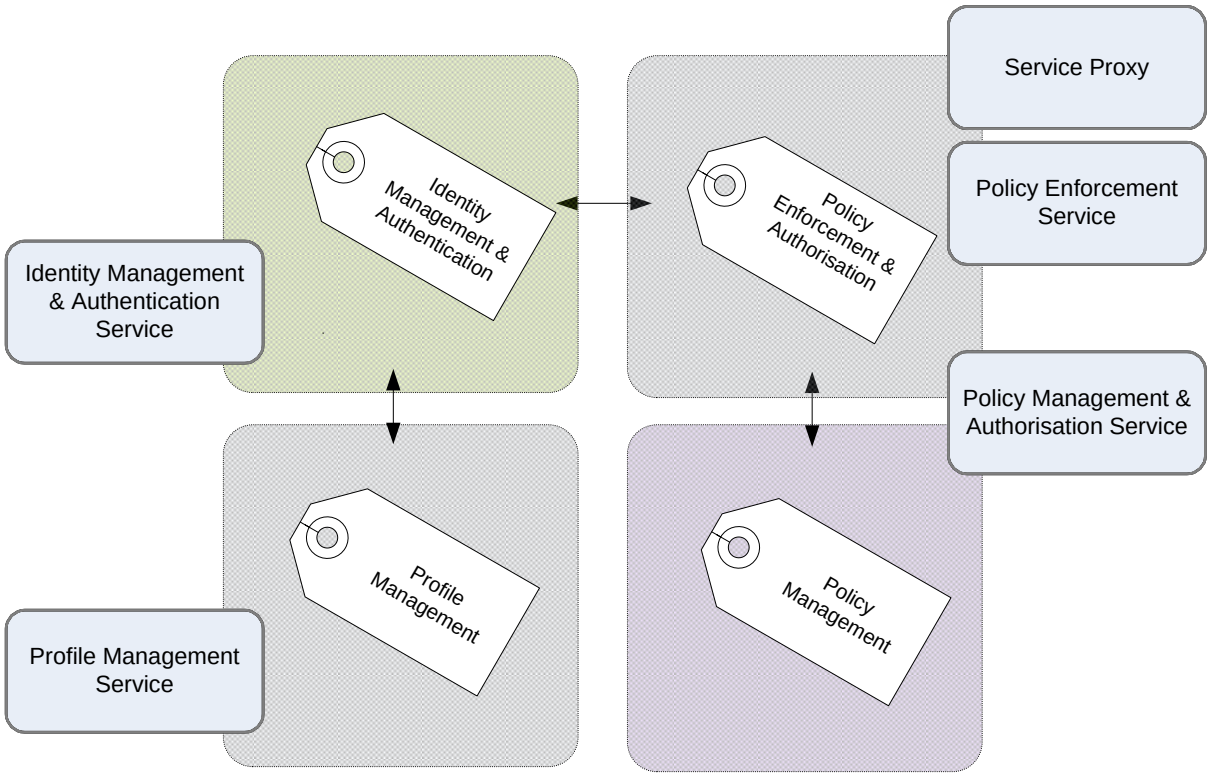


Figure 6-9: Abstract Access Control- Tasks & Services

In addition to the access control service infrastructure, the profile and identity model (see section 7.4) as one vital part of the underlying information model, plays a key role in the access control service architecture and enables the separation of concerns. As an example, the support of different authentication methods, without compromising the whole service architecture, is made possible due to the decoupling of profiles and identities as well as the management of identities in different instances of the Identity Management and Authentication Service, each possibly supporting a different authentication method.

Based on the Abstract Access control Pattern (section Figure 6-12) the workflow involving relevant services can provide non intrusive access control (i.e. realisation with a minimal impact on existing software components) for all services specified in the SensorSA service viewpoint (see section 8.3). Implementation options for non intrusive security on service and data level are described in section 10.5.1.

6.9. Conceptual Building blocks for “Plug-and-Measure”

The SensorSA aims at supporting a plug-and-measure type of operation. Plug & measure hereby refers to the degree of capability to add a new sensor to a sensor network, register it in a sensor service network and access its observations through sensor services in all functional domains of a sensor service network without additional manual intervention. Together with self-healing and self-configuration characteristics of sensor networks, plug-and-measure is an application of the re-configuration capability of a sensor network that has effects upon all functional domains of a sensor service network as defined in section 6.2. Version 1 of the SensorSA offers the following basic conceptual building blocks in order to support dynamic reconfiguration of sensor networks and sensor service networks:

- naming principles for resources (see section 6.5.3),
- automatic discovery of resources based on a modular meta-information schema (see section 6.6.3),
- monitoring of sensors and service instances that access sensors (see section 6.6.2) to detect sensor failures,
- transactional interface to the Sensor Observation Service (see section 8.2.2) to enable the registration of a sensor with the SOS and to insert observations,
- sensor planning capabilities to influence the behaviour and the configuration of sensors (see section 6.6.4), and
- inclusion of events and their processing in the sensor service architecture (see section 6.4).

Section 10.12 explains the use of these building blocks and specifies typical plug-and-measure scenarios based on the SensorSA capabilities.

7. Information Viewpoint

7.1. Overview

The Information Viewpoint of specifies rules and guidelines about how to define SensorSA (meta-)information models. These provide the structure of the information that is being accessed and exchanged in a service network. Principal guidance is given by the meta-model for information defined in (RM-OA, 2007) as an extension of the ISO 19109 General Feature Model (GFM).

The following sections define the information model that is specifically defined to be used for the services of the OGC Sensor Web Enablement initiative. These services are described in the Service Viewpoint in section 7.

7.2. Information Model for Observations & Measurements (O&M)

The SensorSA basically adopts the specification of the Observations and Measurements (O&M) model as defined in (Cox, 2007). This information model is of core relevance for the access and interpretation of the data provided through the Sensor Observation Service. It defines an *observation* as "an act associated with a discrete time instant or period through which a number, term or other symbol is assigned to a phenomenon".

The phenomenon is a *property* of an identifiable object, which is the *feature of interest* of the observation. The observation uses a *procedure*, which is often an instrument or sensor but may be a process chain, human observer, algorithm, computation or simulator. The key idea is that the observation *result* is an estimate of the value of some property of the feature of interest, and the other observation properties provide context or meta-information to support evaluation, interpretation and use of the result.

The model for O&M describes the semantics of the observation and its related feature of interest from a user view point. In contrast, sensor-oriented models emphasise a process or data provider viewpoint. The O&M model is illustrated in Figure 7-1. It defines pre-defined feature types and their relationships, thus extending the ISO 19109 General Feature Model (GFM).

Note: The SANY Information Viewpoint also supports further extensions of the GFM as specified in the meta-model for information of (RM-OA, 2007).

The key concept of the O&M model is the pre-defined feature type *observation* and its related feature type *process.* These pre-defined feature types are to be used as building blocks of project-specific application schemas. An *observation* has the following characteristics:

- An *observation* is modelled as a feature type whose instances are created at a specific time point or time period, the *samplingTime*. An *observation* may have been processed after sampling. The *resultTime* reflects the time when the result of the observation was produced. Observations may be members of *ObservationCollections*.

- The key properties of an *observation* (modelled as association roles in Figure 7-1) are its *featureOfInterest*, *observedProperty*, *procedure* and *result*:

- The *featureOfInterest* (FOI) is a feature of any type (ISO 19109, ISO 19101), which is a representation of the observation target, being the real-world object regarding which the observation is made
- The *observedProperty* identifies or describes the phenomenon for which the observation result provides an estimated value. It must be a property associated with the type of the feature of interest.
- The *procedure* is the description of a *process* used to generate the result. It must be suitable for the observed property.
- The *result* contains the value generated by the procedure.

  Note: The schema of the result data is not determined by the O&M model. The SensorSA recommends a self-describing schema, e.g. by using the definitions of the SWECommon specification.

- As further properties, an observation may have meta-information, e.g. the responsible actor for the observation and an indication of the event-specific quality.

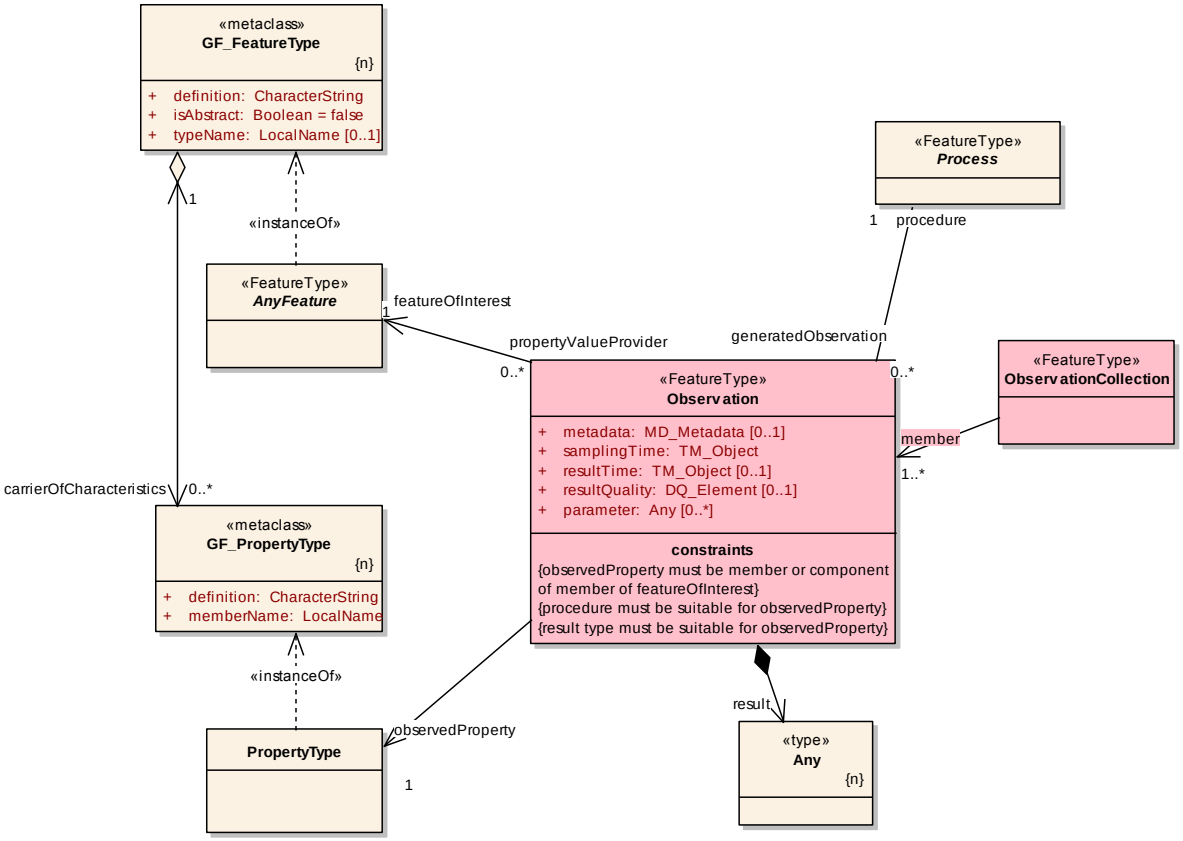


Figure 7-1: Information Model Observation &Measurement from OGC 07-022

7.3. Information Model of the Sensor Observation Service

The information model that underpins the Sensor Observation Service (SOS) (see its description in section 8.2.2) follows the concepts of the O&M Model previously described in section 7.2. The resulting SOS information model is illustrated in Figure 7-2.

The SOS collects observations in a concept called *observation offering*. An *observation offering* is specified by instances of the related O&M concepts *observation*, *observed properties* (which points to a *phenomenon)*, *procedure* and *feature of interest*. Observations and procedures can be part of more than one offering. Further properties of an observation offering are its name, the temporal context (time period) and the spatial context (region). The spatial context of the offering is usually defined as a bounding box which includes all locations where observations are taken.

The O&M concept of a *procedure* abstracts from the source that produces the value of an observation. This may involve a sensor as a technical device (see the technology viewpoint of the SANY Sensor Model in section 5.2), an analytical procedure, a simulation or other numerical processes.

The following describes how the SOS operations use the concepts of the SOS information model.

The *getCapabilities* operation is used to discover the *observations* provided by an SOS and returns the service capabilities. Detailed information is included about all available *observation offerings*. This comprises the *observed properties*, *procedures* and *feature of interest* included in the offering where

- A *procedure* is used to produce an estimate for an observed property.

- A *phenomenon* and the related unit of measurement are defined by a URI (Universal Resource Identifier).

  Note: Details of the phenomenon and the unit may be defined in a dictionary.

- The *feature of interest* is a single feature instance or a collection of feature instances that represent the object on which the observations are made. The feature of interest may have a location property that is expressed in GML.

  Note: Different GML feature types are allowed. All such features are expected to include the optional *boundedBy* element with a GML envelope if the location is known.

Notes:

1. In Figure 7-2, the associations used to resolve the *getCapabilites* request are shown in green.

2. The *getCapabilities* operation does not return information on the relationship between a sensor and its measured phenomena or between a procedure (sensor) and the related feature of interest.

The SOS operation *getFeatureOfInterest* is used to obtain detailed information (such as the location) of features of interest, as in all other operations the feature of interest is only referenced by its identifier.

The consumer of an SOS uses the *getObservation* operation to retrieve observation data for an observation offering as instances of the concept *observation* defined by the O&M specification (see section 7.2.):

- An observation is an event and contains an *eventTime* that describes the time at which the observation has been taken. An observation is also related to the procedure that estimated the value.

- It is bound to the feature of interest which describes the feature for which the observation was taken.

- An observation captures one or more observed properties. The observations can also be requested for a set of observed properties or a set of procedures within the observation offering.

- An observation from a procedure only contains observed properties which are included in the output section in the corresponding SensorML document. The output section of SensorML describes the observed properties of the sensor that has produced the observation values.

- The user can further restrict the amount of data returned by using temporal or spatial filters. The allowed filter types are reported by the *getCapabilities* operation.

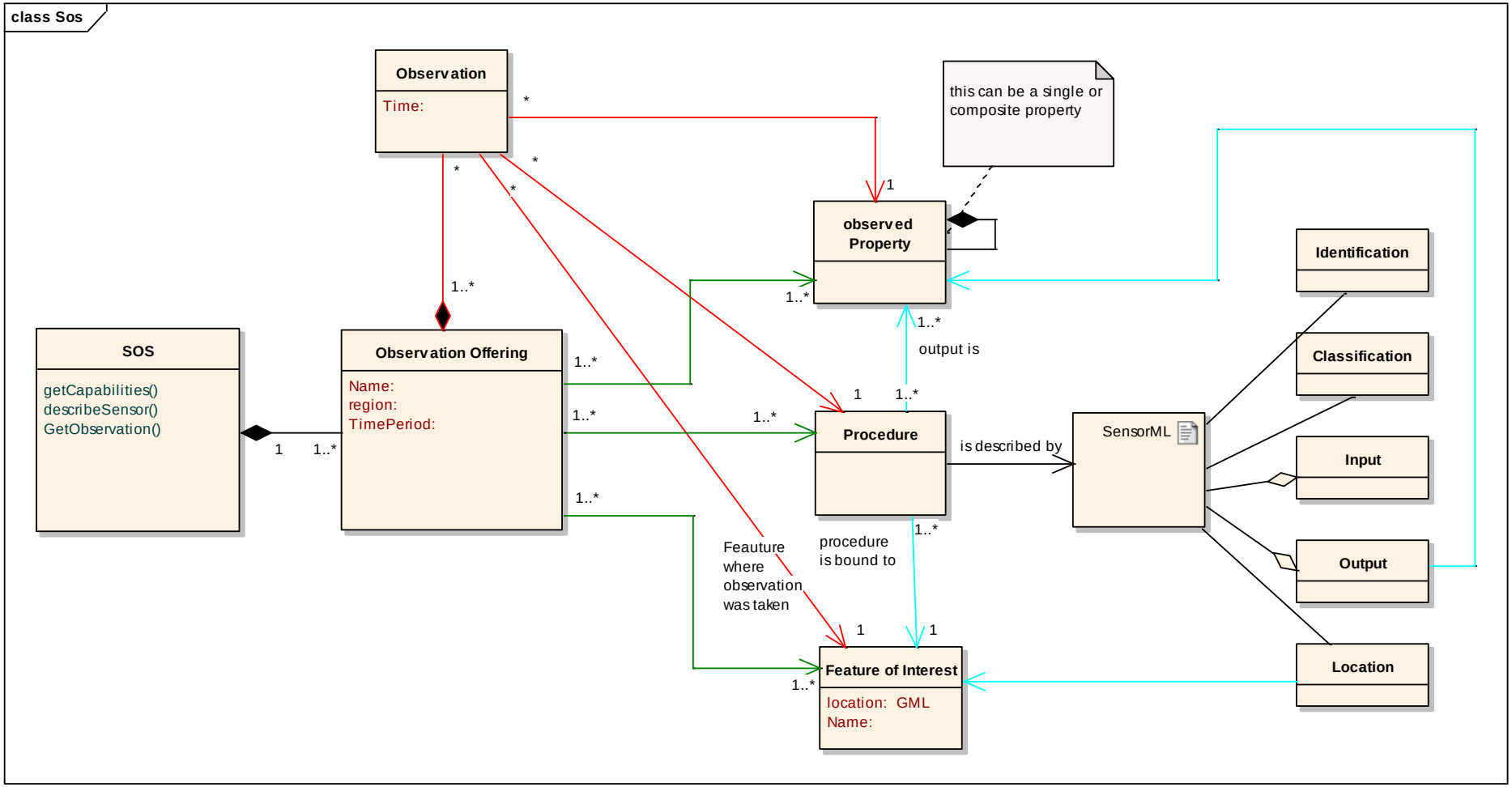


Figure 7-2: Information Model of the Sensor Observation Service

7.4. Access Control Information Model

In the following information model for the realisation of access control is presented with a reference to the corresponding OASIS standards.

7.4.1 Model for Subject Related Information

In the description of the SensorSA access control concepts in section 6.8.2 a **Subject** has been introduced as the requestor of a service operation who represents the acting entity, whereas an acting entity may be a user or a software component, e.g. a service instance. The profile and identity model defines the basic concepts related to a subject that support the tasks of the access control pattern introduced in section 6.8 and related services as described in section 8.3. The following elements of a subject-related information model are defined:

- Profile
- Identity
- Group
- Role
- Policy

7.4.2 Profiles and Identities

A **profile** is the abstract representation of a subject, i.e., it is characterised by a set of attributes that describe a subject. The profile information comprises associated identities and may therefore serve as container of logical pointers to identities. The definition of a complete profile schema is out of the scope of the SensorSA information model for access control as it is application dependent. For instance, a profile attribute could be the social security number to be able to identify the real word user behind the subject, or it could contain the phone number to contact a responsible person or the signature of a software agent. The definition of profile attributes is a application design decision.

In order to perform an authorised action (e.g. a service request) a subject represented by his profile has to provide a proof of authenticity so that a service provider can decide whether the requested action is in line with the service provider's access policy. A subject may present different identities, possibly authenticated with different authentication mechanisms, for different actions. Thus, a single profile may have multiple identities .By decoupling profiles from identities, on the one hand information not relevant for authorisation decisions can be kept away from the access control mechanisms, and on the other hand requirements like single sign-on (SSO) can be easily supported. The separation of profiles and their associated identities reflects the real life situation in which a single subject (i.e. a single profile) may have an arbitrary number of identities for particular purposes. A subject may authenticate one or more of its profile's identities and thus accumulate access rights in disparate security domains (systems, networks and organisation).

Profiles are managed (created, deleted, etc.) using an instance of the Profile Management Service (see section 8.3.2). Profile attributes are used to store arbitrary profile related information (e.g. first name, last name, address, e-mail).

Identities are managed (created, deleted, etc.) using an instance of the Identity Management and Authentication Service (see section 8.3.3)

. Identity attributes are used to store information that can be the basis for an authorisation decision. The Identity Management and Authentication Service acts as an identity provider. The manner in which identity information is managed is up to the particular identity provider as different authentication mechanisms (e.g. public key/secret infrastructures or login/password) require different identity related information. In this way the task "profile management" (see section 6.8.3.1) can act independently of the methods applied in the task "authentication" (see section 6.8.3.3) and vice-versa.

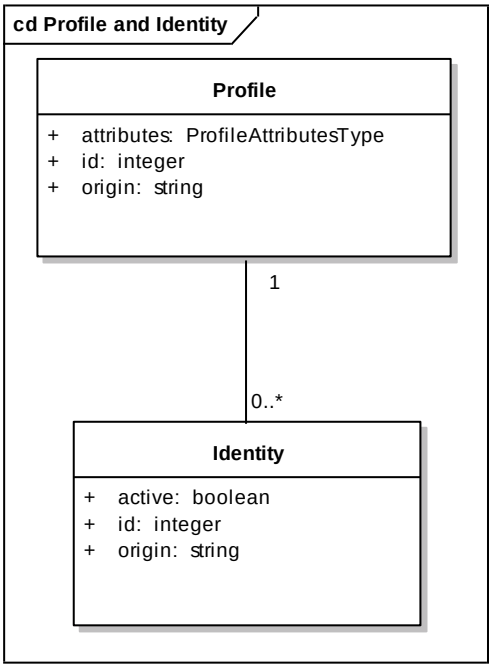


Figure 7-3: Profiles and Identities

7.4.3 Groups

A *group* is modelled as a special type of identity that is composed of a set of identities. Group attributes are used to hold common properties of its members' identities. An identity that is a member of a group automatically inherits all identity attributes of that group as after login all related group identities are included in the SAML assertion (session information). Thus, groups facilitate administrative operations which need to be applied to a number of identities of a single Identity Management and Authentication Service instance.

A group can be treated as an ordinary identity by an instance of the Policy Management and Authorisation Service (see section 8.3.4). Therefore, writing policies for groups does not differ from writing policies for any other identity. Management of groups is done according to

the management of identities in instances of the Identity Management and Authentication Service (see section 8.3.3).

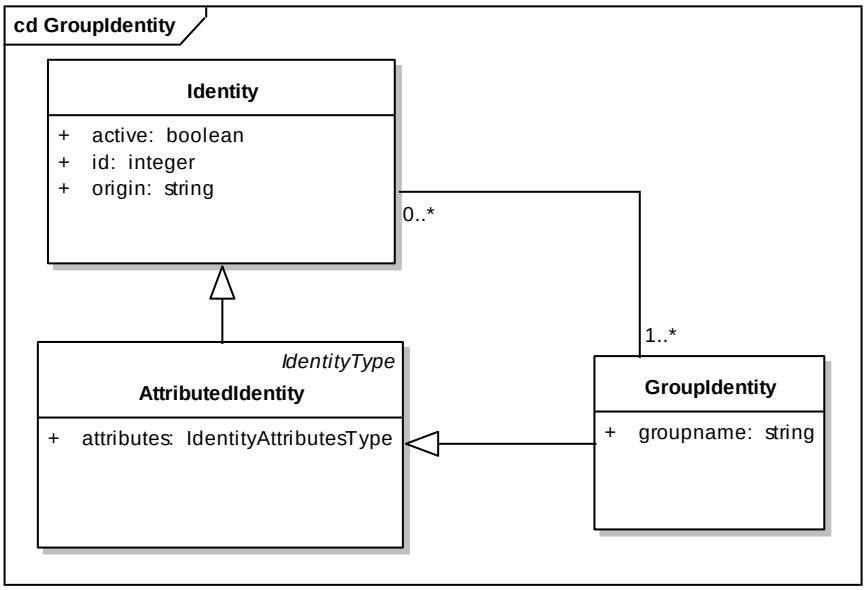


Figure 7-4: Groups are special Identities

Groups can only be defined on the level of a single Identity Management and Authentication Service instance, since the concrete representation of identities, and thus groups, may vary from instance to instance. To allow cross domain Identity Management and Authentication, and thus cross security domain enforcement of access rights, the concept of roles is used.

7.4.4 Roles

A *role* is an abstract concept (e.g. “administrator”) that corresponds to a related policy. The usage of roles is a powerful yet simple way to facilitate large-scale authorisation management of complex systems. The general concept is applied in most databases and operating systems. Roles can reduce administration effort significantly, especially when similar access restrictions have to be enforced for many users of different organisations.

As suggested by (OASIS 2004) in SensorSA “roles are expressed as XACML Subject Attributes”, apart from the fact that SensorSA does not predefine policies or sets of policies that narrow down the concept of roles to a fixed set of permissions. In our approach a role is modelled as a special identity attribute named “role”. If a mutual semantic agreement on the common role attribute exists among different security domains the role concept allows the enforcement of access rights across all security domains regardless of individual identity representations or domain specific identity attributes.

In contrast to groups no distinct interface for the management of roles is provided by SensorSA at the moment.

7.4.5 Policies

Policies contain sets of rules to express access restrictions for particular resources. At the conceptual stage no assumption about the policy language, namely 'how' a policy is encoded, are made. The resource side basis of every access control decision is in the SensorSA policies.

7.4.6 Assertion and Policy Encoding

In SensorSA the access control mechanisms rely on the usage of OASIS Security Standards.

- SAML (Security Assertion Markup Language) is used to encode identities and related information in a SAML Assertion.

- XACML (eXtensible Access Control Markup Language) is used to define access rules, for the above mentioned identities.

7.4.6.1 SAML (Security Assertion Markup Language)

SAML is a language to encode security related information. In SensorSA SAML is used to encode Identity related information. SAMLS is summarised by (OASIS 2006b) as follows:

"SAML consists of building-block components (…) The components primarily permit transfer of identity, authentication, attribute, and authorization information between autonomous organizations that have an established trust relationship. The ***core*** *SAML specification defines the structure and content of both assertions and protocol messages used to transfer this information.*

SAML assertions carry statements about a principal that an asserting party claims to be true. The valid structure and contents of an assertion are defined by the SAML assertion XML schema. Assertions are usually created by an asserting party based on a request of some sort from a relying party, although under certain circumstances, the assertions can be delivered to a relying party in an unsolicited manner. SAML protocol messages are used to make the SAML-defined requests and return appropriate responses. The structure and contents of these messages are defined by the SAML-defined protocol XML schema.

The means by which lower-level communication or messaging protocols (such as HTTP or SOAP) are used to transport SAML protocol messages between participants is defined by the SAML bindings. Next, SAML profiles are defined to satisfy a particular business use case, for example the Web Browser SSO profile. Profiles typically define constraints on the contents of SAML assertions, protocols, and bindings in order to solve the business use case in an interoperable fashion."

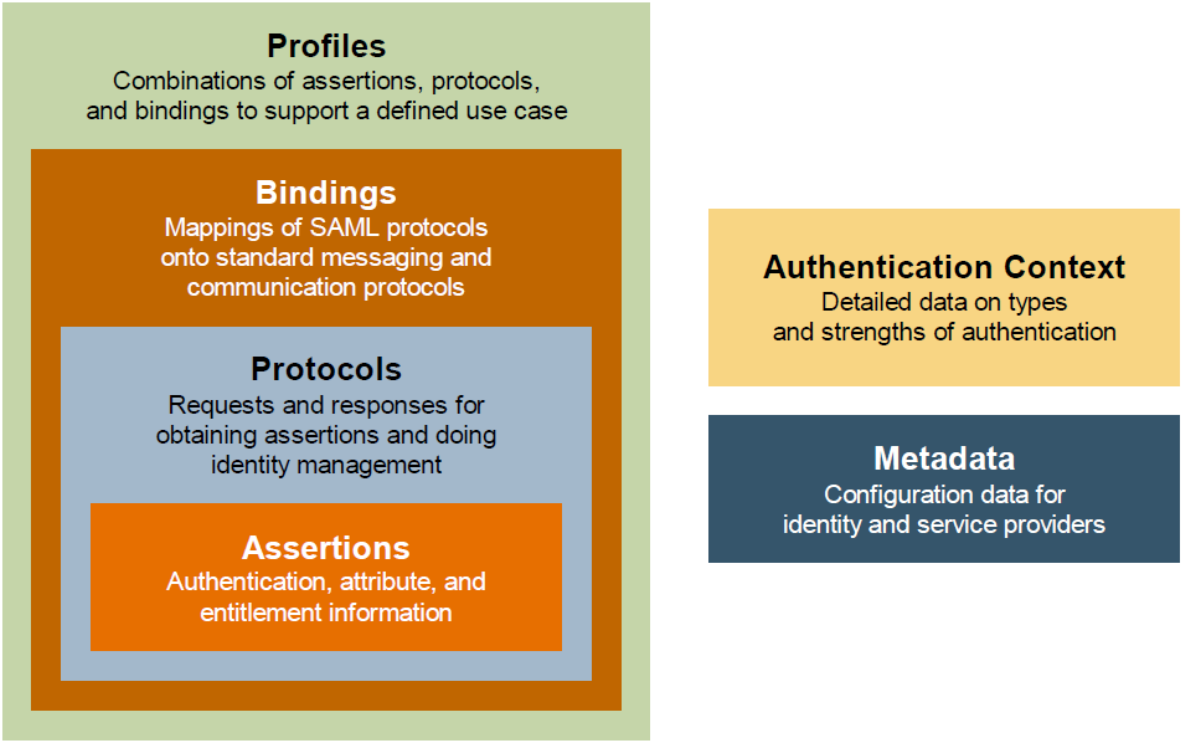


Figure 7-5: Basic SAML concepts (OASIS 2006b)

In SensorSA SAML core (assertions and protocol) is used exclusively or in other words no bindings or profiles have been defined. The use of SAML in the context of the access control services is further described in section 10.5.4.

7.4.6.2 XACML (eXtensible Access Control Markup Language)

"XACML is language for expressing access control policies. The language is used to standardise the process of access control management. Access control management consists of some or all of the following steps: writing, reviewing, testing, approving, issuing, combining, analysing, modifying, withdrawing, retrieving and enforcing of policies" (OASIS 2005). XACML delivers a language to encapsulate security rules in policies in a standardised manner. If the same mechanisms of access control are used throughout the components of an information infrastructure, it is possible to manage the enforcement of policies in a consistent and to some extent interoperable way. In the SensorSA XACML and extensions like GeoXACML can be used as a common access control language.

The following paragraph describes XACML's basic elements and establishes a connection between the concepts described in section 7.4 and XACML. Furthermore, it provides an overview of the GeoXACML extensions.

7.4.6.2.1. XACML Basic Concepts

The following paragraphs are slightly modified extractions from the eXtensible Access Control Markup Language (XACML) standard (OASIS 2005) intended to give an overview of the XACML components and principles.

- Rule

  A rule is the most elementary unit of a policy. A rule can be evaluated on the basis of its contents and the request. The main components of a rule are a target, an effect and a condition.

- Rule target

  The target defines the set of resources, subjects, actions and environment to which the rule is intended to apply (it is possible to further refine the applicability by the target with conditions (see OASIS 2005). An XACML PDP verifies that the matches defined by the target are satisfied by the subject, resource, action and environment attributes in the request context, e.g. role. In summary, targets are used to determine which rules match the given request.

  Where subjects and their attributes are used to encode the identities and related attributes like' role' described in section 7.4.4 .

- Effect

  The effect of a rule indicates the rule-writer's intended consequence of a "True" evaluation for this particular rule. Two values are allowed: "Permit" and "Deny".

- Policy

  A policy is a container for rules and other information, e.g. a general policy target or a particular rule combining algorithm to support different matching policies for an authorisation request.

- PDP functionality

  After receiving a request the PDP matches the request against the policies to determine the policies to be considered, where matching simply means the evaluation of the target, which comprises functions on the attributes of the elements subject, resource, action and environment as described in the paragraph Rule. In case one of the targets matches, the effect is either "Permit" or "Deny". It is possible that there is more than one matching rule, in this case the defined combining algorithm is used to provide an authorisation decision, e.g. if the combining algorithm is "Deny-overrides" then one occurrence of a "Deny" overwrites all occurrences of "Permit".

7.4.6.2.2. GeoXACML: The geospatial extension of XACML

Geospatial eXtensible Access Control Markup Language (GeoXACML) is an OGC Implementation Standard and "defines an extension to XACML for spatial data types and spatial authorisation decision functions. Those data types and functions can be used to define additional spatial constraints for XACML based policies." (OGC 07-026r2)

It makes use of existing XACML extension points to be fully compatible to the XACML standard. This means that a "GeoPDP" is not only able to evaluate GeoXACML decision queries but standard XACML queries as well.

"GeoXACML extends XACML by only one new data type that is named "urn:ogc:def:dataType:geoxacml:1.0:geometry"." (OGC 07-026r2) It contains geometric data types described in (OGC 06-103r3). There are also two extensions to the GeoXACML implementation specification that define GML encoding for GML version 2 (OGC 07-098r1) and 3 (OGC 07-099r1).

The XACML extension point for data types is illustrated in the XML fragment. As the DataType attribute is of type anyURI the additional geometry data type can be used.

```
<xs:element name="AttributeValue" type="xacml:AttributeValueType" substitutionGroup="xacml:Expression"/>
<xs:complexType name="AttributeValueType" mixed="true">
  <xs:complexContent mixed="true">
    <xs:extension base="xacml:ExpressionType">
      <xs:sequence>
        <xs:any namespace="##any" processContents="lax" minOccurs="0"maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 7-5: AttributeValue extension point of XACML

```
<xs:complexType name="AttributeDesignatorType">
  <xs:complexContent>
    <xs:extension base="xacml:ExpressionType">
      <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
      <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
      <xs:attribute name="Issuer" type="xs:string" use="optional"/>
      <xs:attribute name="MustBePresent" type="xs:boolean" use="optional" default="false"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 7-6: AttributeDesignatorType extension point of XACML

```
<xs:complexType name="AttributeSelectorType">
  <xs:complexContent>
    <xs:extension base="xacml:ExpressionType">
      <xs:attribute name="RequestContextPath" type="xs:string" use="required"/>
      <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
      <xs:attribute name="MustBePresent" type="xs:boolean" use="optional" default="false"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 7-7: AttributeSelectorType extension point of XACML

There are 34 new functions of two different conformance classes defined by GeoXACML, nineteen functions of conformance class BASIC and fifteen of conformance class STANDARD. The different functions cover several aspects of geographic evaluation:

- Topological functions (conformance class BASIC)
- Bag functions (conformance class BASIC)
- Set functions (conformance class BASIC)

- Geometric functions (conformance class STANDARD)
- Conversion functions (conformance class BASIC)

A "GeoPDP" has to implement all functions of conformance class BASIC to be considered a BASIC GeoXACML conformant PDP implementation. A STANDARD conformant PDP implementation has to implement all functions of conformance class STANDARD in addition to all functions of conformance class BASIC.

The XACML extension point for function types is shown in the following figure. As the FunctionId attribute is of type anyURI any additional function may simply be used. GeoXACML does not define any additional or changed XSD schema elements to XACML to stay XACML conformant.

```
<xs:element name="Function" type="xacml:FunctionType" substitutionGroup="xacml:Expression"/>
<xs:complexType name="FunctionType">
  <xs:complexContent>
    <xs:extension base="xacml:ExpressionType">
      <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 7-8: Function type extension point of XACML

7.5. Event Information Model

With respect to the OGC baseline, an event is a feature. SensorSA provides a cross-domain application schema for events, called *event information model*. Although each domain needs to build their own, well-adapted event model, a cross-domain application schema serves as a common denominator or kind of crystallization point for further extensions.

The event model is organized in a package stereotyped <<ApplicationSchema>>. It has dependencies on a number of packages from the ISO 19100 Harmonized Model, as shown in Figure 7-9.

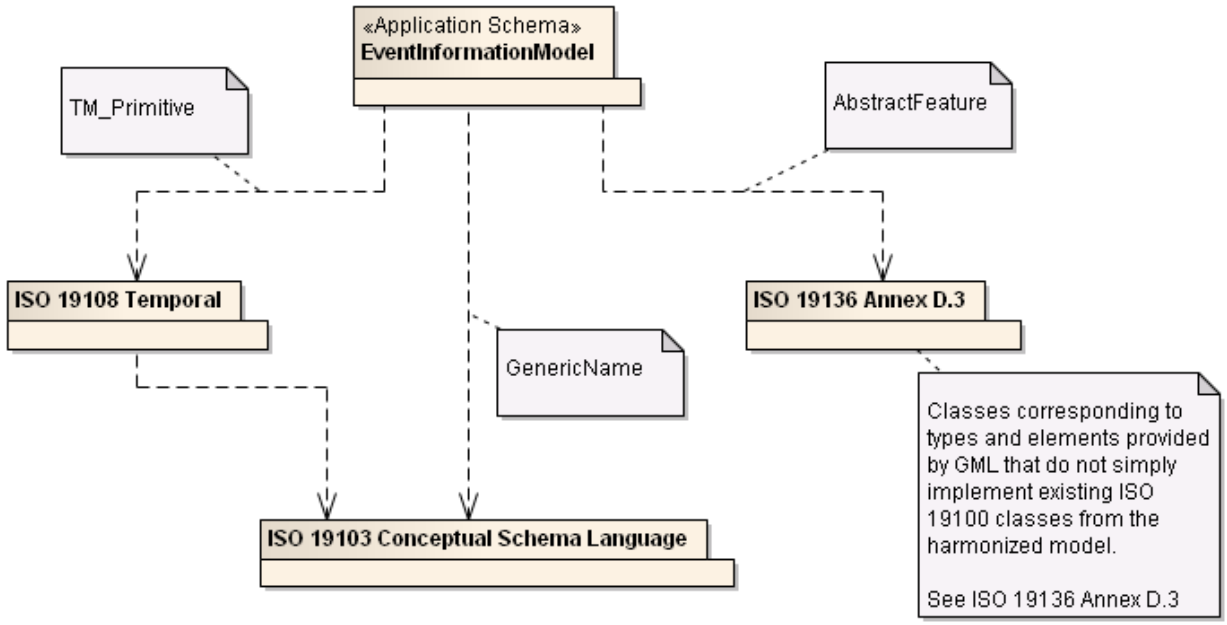


Figure 7-9: Event model dependencies on packages from the ISO 19100 Harmonized model

Only one sub-package currently exists in the event model (see Figure 7-10).

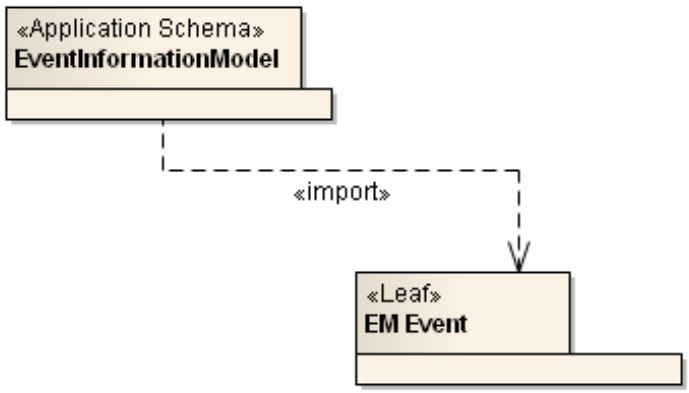


Figure 7-10: Event model package structure

This package will be explained in the following.

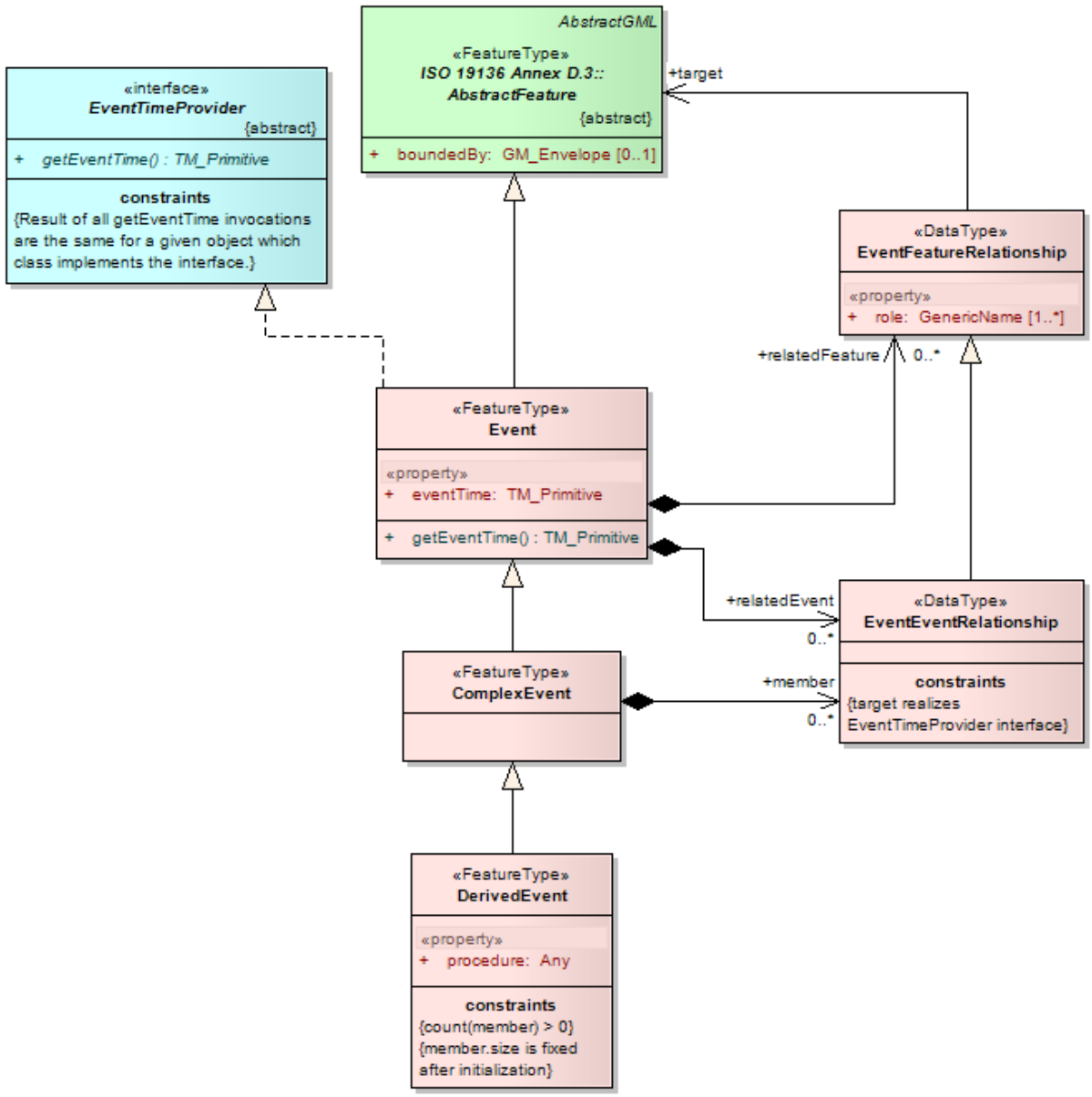


Figure 7-11: Event type and specializations

Note: The class named "AbstractFeature" represents the set of all classes with stereotype <<FeatureType>>. In an implementation a concrete class representing a feature type from a domain of discourse will substitute this abstract class. This class is implemented in GML by the element gml:AbstractFeature.

An *Event* is a feature. It has a required property containing the time when the event happened. The *eventTime* is modelled as a TM_Primitive from ISO 19108. As such, the value may be a temporal geometry primitive (instant or interval) or a temporal topology primitive (node or edge). The Event class realizes the *getEventTime* operation from the *EventTimeProvider* interface, which provides access to the value of the eventTime property.

Note: in ISO 19136, a feature is modelled as a gml:AbstractFeature which contains a boundedBy property that can only describe a spatial or spatio-temporal boundary, but not a pure temporal one. In the case of events, the temporal aspect is of primary interest. This also applies

to non-geospatial features, which contain complex temporal properties. Thus, the boundedBy property of gml:AbstractFeature should be modified to support both spatial, temporal and spatio-temporal bounding / extent information.

An event may be related to other features. One to many role properties characterize each EventFeatureRelationship. This primarily supports use cases in which an event object is exchanged between multiple domains where the relationship target may incorporate a different association role and thus may have a different semantic as property of the event.

Example: an earthquake event may have a relationship to a street feature, indicating that the earthquake *affected* the street and that it actually *destroyed* the street. Similarly, a hurricane event may have a relationship to a butterfly, indicating that the hurricane was *causedBy* the butterfly and that the hurricane *blewAway* the butterfly.

An event may also be related to other events via an *EventEventRelationship*. This type inherits from EventFeatureRelationship and thus has AbstractFeature as target property. However, a constraint is added to the EventEventRelationship, which requires the target to realize the EventTimeProvider interface. This allows us to establish relationships to features that can be considered as events, regardless of whether they derive from the Event type defined in this application schema or not.

For further relationships between an Event and another feature/event, we refer to the OGC engineering report OGC 09-032, as mentioned at the beginning of this chapter.

Specializations of the *Event* type – which itself is neither an abstraction nor a composition of other events (it may nevertheless be related to other events) and thus represents a simple event – can be a *ComplexEvent* or a *DerivedEvent*.

An abstraction or aggregation of multiple events - its members - is called a ComplexEvent. The relationship to a member event is modelled through the EventEventRelationship. A ComplexEvent does not necessarily have member events. This situation will likely be the case when it is known that an event happened and that it was caused by other events but that these 'causing' events cannot be determined at the moment. The *ComplexEvent* will then be an abstraction of the happening caused by these unknown events.

This also implies that the event time of a ComplexEvent is - like for a simple Event - assigned by the entity that creates the event object. It can but does not have to be related to the event times of member events.

Note: Because the event time is a genuine property of an event, it shall not be modified in an event object. This also applies to complex events. Therefore, if the event time of a ComplexEvent object was computed (at creation time) based upon - for example - the temporal bounding box of the member events that were available and if later the set of members is modified then the event time of the original event object shall not be modified. Instead a new (complex) event object should be created which encompasses the modified set of member events and the new event time. This new event may for example have a relationship to the old event with a role of *supersedes*. Otherwise, if a modification of the member set has no implications upon the event time, then a new event object is not necessary.

If a well-defined procedure was used to detect an event based upon (the existence or absence of) one or more other events, the resulting event is called *DerivedEvent*. The procedure may be any process or algorithm that is capable of detecting an event based upon the (existence or absence of) information in member events. A *DerivedEvent* therefore has at least one member. The event time of a derived event is determined by its procedure.

7.6. Resource Model

7.6.1 Introduction

As motivated in section 6.4, the SensorSA defines a conceptual link between a service-oriented and a resource-oriented view upon a sensor service network. In this section a resource model is defined as a technology-independent basic information model of a Resource-Oriented Architecture (ROA). Possible applications of the resource model in the context of a sensor service network are described in section 7.6.3 after the specification of the ROA concepts.

The term ROA denotes the architectural concepts and the set of rules that aim at accessing and manipulating uniquely identified resources based on a uniform interface. The SensorSA understands ROA itself as a technology-independent concept, although it is usually discussed together with its realisation in a Web environment, i.e. based on the basic technologies of the World Wide Web. (Ruby/Richardson, 2007) describe ROA as a "way of turning a problem into a RESTful web service: an arrangement of URIs, HTTP and XML that works like the rest of the Web".

This section describes the major ROA concepts as an extension to the meta-model for information and services defined in (RM-OA, 2007). This enables the specification of resource models as application schemas in an information viewpoint specification of a SensorSA.

7.6.2 ROA Concepts

The basic concepts of an ROA are abstracted from the specification of RESTful Web Services according to (Ruby/Richardson, 2007). These are:

- resource
- resource name
- resource representation
- resource links

These ROA concepts are modelled as feature types and types according to (RM-OA, 2007). The basic resource model is shown in Figure 7-12. Furthermore, one of the major characteristics of a ROA is the access to the resource and the manipulation of its state through a uniform interface. The modelling of uniform interface is defined in section 7.6.2.4.

Note: The resource model has been submitted to OGC as OGC 07-156r1 (Usländer, 2008).

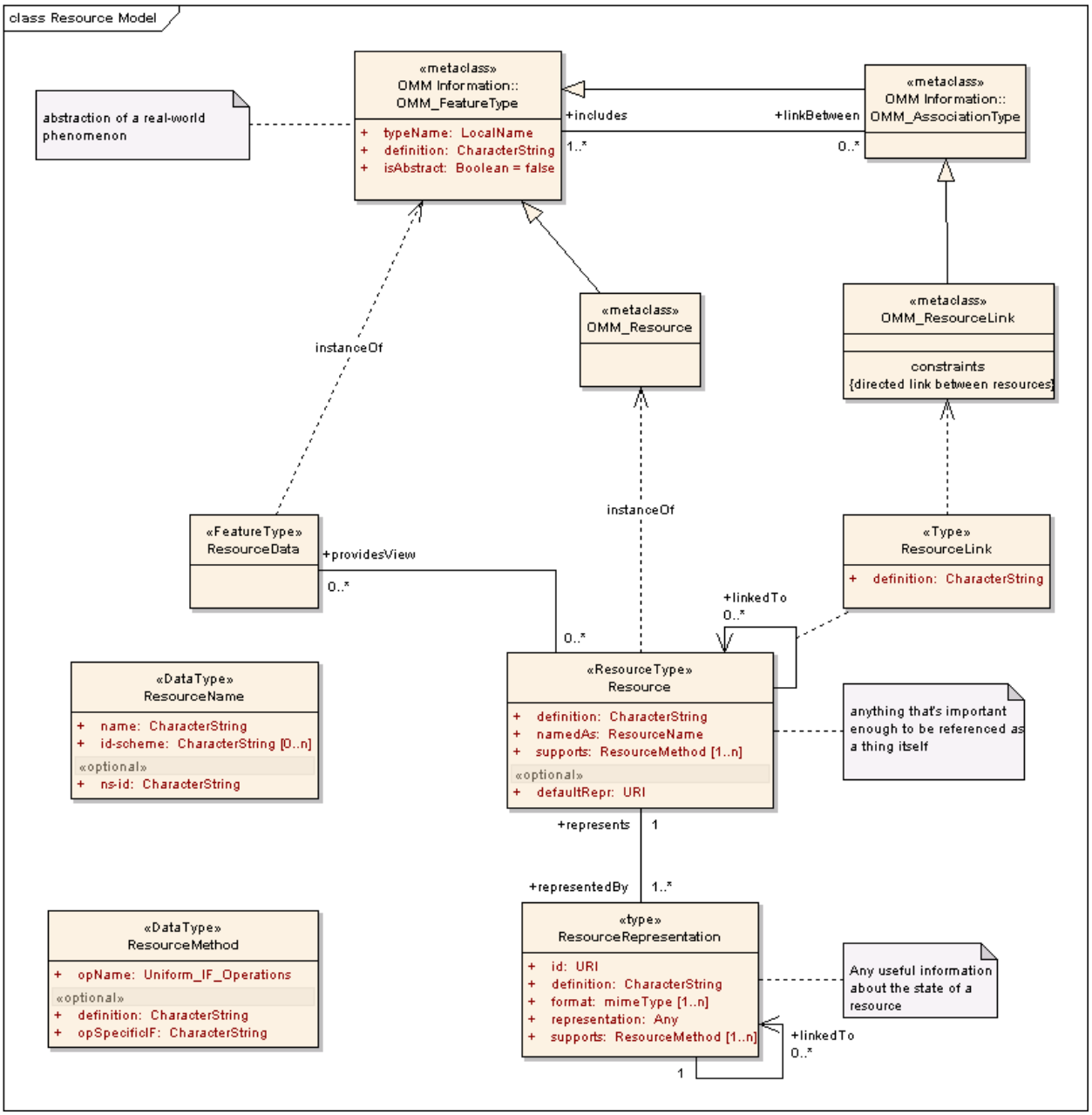


Figure 7-12: Resource Model

7.6.2.1 Resource

A resource is anything that's important enough to be referenced as a thing itself. A resource is understood as a specialisation of a feature type and has the following properties:

- *definition*: Human-readable description of the purpose of the resource.
- *namedAs*: Name of the resource. It is modelled by the type *ResourceName* (see section 7.6.2.3)
- *supports*: Provides a list of those methods of the uniform interface that are supported by the resource (see section 7.6.2.4).

- *providesView (optional)*: list of feature instances that provides the underlying data of the resource.

- *linkedTo:* list of zero or more resource types to which representations of this resource types may be potentially linked. This link is further described by the association class *ResourceLink* (see section 7.6.2.4).

- *representedBy*: List of identifiers of possible representations of this resource. One resource may have one or more possible representations. This property is modelled by the type *ResourceRepresentation* (see section 7.6.2.2).

- *defaultRepr*: identifier of the default representation of the resource.

7.6.2.2 Resource representation

The representation of a resource comprises any useful information about the current state of a resource. A resource may have (and usually has) several representations. It has the following properties:

- *id:* unique identification of the resource representation.

- *definition*: Human-readable description of the purpose of the resource representation.

- *format*: MIME-type format in which the information is presented to the client.

- *representation*: Information that is returned to the client when the representation is retrieved.

- *supports*: Provides a list of those methods of the uniform interface that are supported by the resource (see section 7.6.2.4).

- *linkedTo*: identifier of zero or more resource representations to which may be navigated from the resource representation.

7.6.2.3 Resource name

The resource name denotes the resource. It shall indicate the intended purpose of the resource to a human user. It has to be distinguished from the identifier of the representations of the resources which also provide an address (a path) by which to access the resource (see section 7.6.2.2). It has the following properties:

- *name:* Provides the name of the resource.

- *id-scheme*: Defines the ways how identifiers for representations of the resource may be built.

Note: The resource model just assumes that the identifiers are built hierarchically. The namespace attribute *ns-id* (see below) shall define the scope of the identifier such that all identifiers of representations are unique.

- *ns-id (optional)*: Defines the namespace for the identifiers of the representations of the resource. It may but need not be the same as the name of the resource.

Note: When mapping to a service platform, the relationship between the identifier of the resource and the identifiers(s) of its representations has to be defined. This is done in an identifier scheme and has to be defined as part of the Engineering Viewpoint of the system's design. In the case of RESTful Web Services, a typical identifier scheme is to combine the *ns-id* into one hierarchical URI such that the start of the URI denotes the resource name and the rest denotes the identifier. The boundary between the start and the rest is specific to the resource.

7.6.2.4 Resource link

The possibility to link a representation of a resource to other representations of the same or a different resource is one of the key features of a resource-oriented architecture. It is modelled by the association class *ResourceLink*. A resource link is a directed link that determines the navigation direction between the corresponding representations.

It has the following properties:

- *definition* (optional): Human-readable description of the purpose of the link.

7.6.2.5 Uniform Interface

The uniform interface is an instance of the meta-class OMM_InterfaceType of the service meta-model of (RM-OA, 2007). As illustrated in Figure 7-13, it requires that the following operations be defined for a given platform. Examples for HTTP 1.1 as the mandatory transport protocol of the SANY service platforms (see section 9.2) are given in brackets:

- *createResource (cR):* create a new resource (e.g. HTTP PUT to a new resource representation, HTTP POST to an existing resource representation)
- *getResource (gR):* retrieve a representation of a resource (e.g. HTTP GET)
- *deleteResource (dR):* delete an existing resource (e.g. HTTP DELETE)
- *updateResource (uR):* modify an existing resource (e.g. HTTP PUT to an existing representation)

The following operations are to be optionally provided:

- *getResourceCapabilities (gC)*: check which methods are supported by a particular resource (e.g. HTTP OPTIONS)
- *getResourceMetadata (gM)*: retrieve the descriptive information about a representation (e.g. HTTP HEAD)
- *createSubordinateResource (cS)*: create a resource representation in the context (e.g. namespace) of a given resource representation (e.g. HTTP POST)

- *appendToResourceState (aS)*: create additional information about the state of a resource (e.g. HTTP POST).

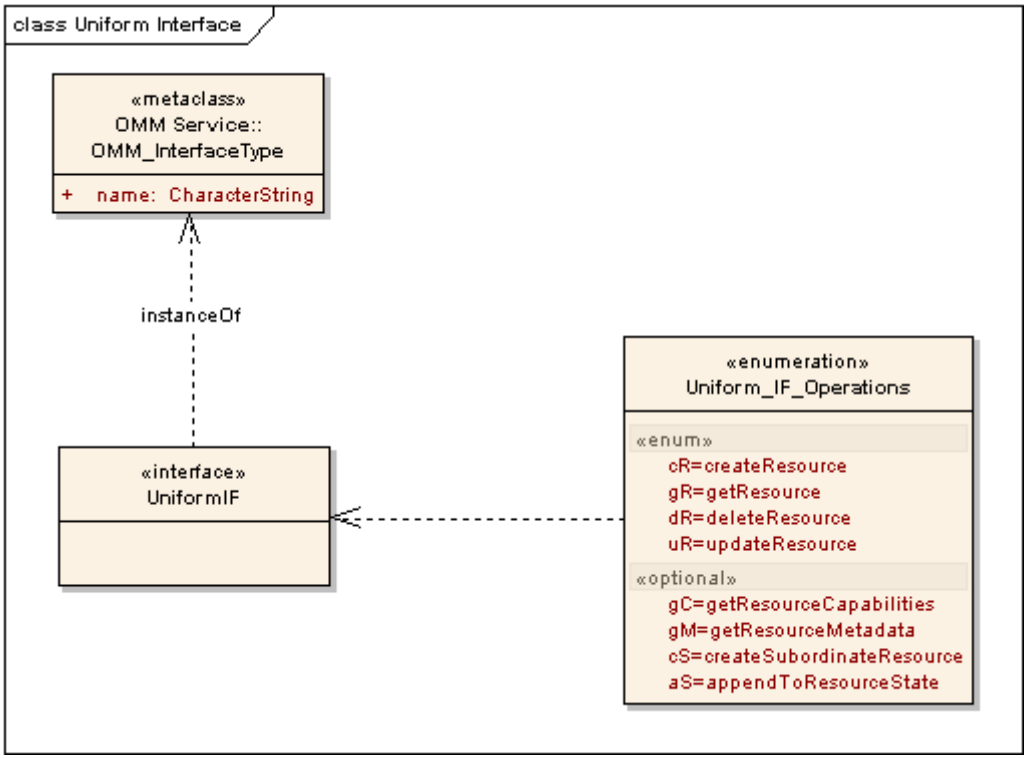


Figure 7-13: Model of the Uniform Interface

7.6.2.6 Resource Method

The resource method defines a method that is supported by a resource and its possible mapping to an operation of another interface type. It has the following properties:

- *opName*: Method that is supported by the uniform interface (see section 7.6.2.4)

- *definition (optional)*: Human readable description of the meaning of the method for the resource.

- *opSpecificIF* (optional): name of an operation of another specific interface (other than the uniform interface) that is semantically equivalent to this resource method. This property is essential when the uniform interface is defined in addition to an existing interface of a given service type.

  Example: The *getResourceRepresentation* operation of the resource type "*observation*" is semantically equivalent to the *getObservation* operation of the interface *Core Operation Profile* of the Sensor Observation Service (see section 8.2.2).

7.6.3 Relationship between Resources, Services and Features

The conceptual relationships between resources, services and features cannot be described without having a purpose of the resource modelling in mind. Figure 7-14 illustrates possible relationships derived from the basic assumption of the RM-OA that a service provides one or more interfaces, and each interface consists of one or more operations. Operations access underlying data in a read and write mode.

The left-hand side of Figure 7-14 shows the traditional approach of OGC services accessing underlying data whose structure is modelled as an ISO/OGC application schema. The right-hand side shows a complementary resource-modelling approach. Here, operations are modelled together with their underlying data in form of resources and their representations.

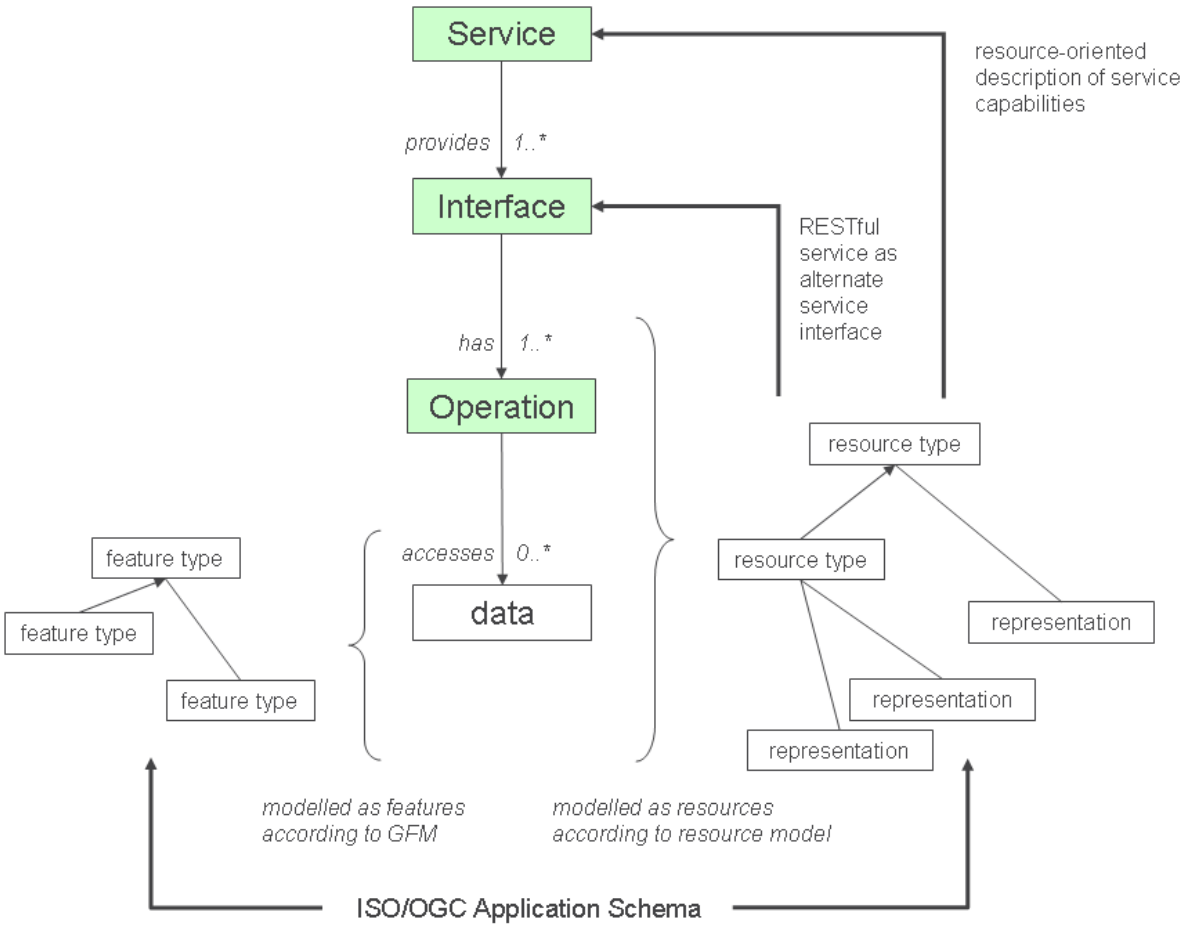


Figure 7-14: Services, features and resources and possible relationships

There are two possible purposes and applications for this modelling approach:

- The capabilities of a service may be specified in a resource-oriented way. Typically, the resulting resource model mirrors the basic elements of the underlying application schema of a service. As an example section 10.2.5 of the Engineering Viewpoint specifies an resource model for the Sensor Observation Service (SOS) (see section 8.2.2). Here the SOS capabilities are interpreted as resources that reflect the basic O&M concepts such as offerings, features of interest, observation collections or observed properties. They may

be used by client application as navigational means for discovery of and the access to the observations provided by the SOS.

- An extended approach is to provide a RESTful service as an alternate interface based on a resource model on top of a service instance (or by combining several service instances). Such a RESTful service would then provide a selected view (typically just a subset) upon the capabilities and operations of the underlying service. This approach has been implemented for the as a prototype in the SANY project.

The modelling of resources according to this resource model may serve the following purposes in the context of a sensor service network:

- It may provide a modelling bridge between the Information and the Service Viewpoint in a system design. The modelling of services in terms of the resources and their representations which they provide to a client may facilitate the understanding of the functionality of the service to a system designer.

- When specifying a resource-oriented view upon a service in addition to its "specific" interface and operations, the system designer gains flexibility in mapping the abstract service specification to an implementation specification. For example, the system designer may then map it to the SANY W3C Web Service platform (see section 9.2.1), to a SANY OGC Web Service platform (see section 9.2.2), or to a RESTful Web Service platform (see section 9.2.3).

- The provision of a resource-oriented view upon a service may facilitate the discovery of the service as the notion of resources may be closer to the "universe of discourse" of the user as it's the case for the signatures of specific services. Having this application in mind, the resource application schema should then be stored as meta-information entries in catalogue systems.

7.7. Meta-information Schema for Discovery

7.7.1 Overview

The discovery of resources is an important consideration in a sensor service network. Several kinds of resources need to be discovered. It is possible to distinguish between two main resource types which can be described by a meta-information document:

- services, describing meta-information about available service instances

- data, describing meta-information about available datasets

It is possible to directly search for meta-information describing these two main resource types in common OGC catalogues. This structure was also used in the default ORCHESTRA meta-information schema. The schema provides general elements for the description of the above resource types. It is possible to directly re-use this meta-information schema for SANY to create meta-information documents describing resources in the Application Domain of SANY. However, several extensions of the meta-information schema are needed for the discovery of resources of the Mediation & Processing, Acquisition and Sensor Domains. The meta-

information schema was especially extended to address the need for sensor observation specific discovery. To achieve this, the O&M model (see section 7.2) was taken into account.

The following text and UML diagrams are describing the structure and contents of the resulting application schema for meta-information (AS-MI). All meta-information types are prefixed with "MI" for meta-information. The AS-MI contains classes for the two main resource types:

- MI_Service for the description of services
- MI_Data for the description of data

To address the need for sensor observation specific discovery the data resource type was refined into several subtypes derived from MI_Data. The following types are reflecting the structure of the O&M model:

- feature of interests (FOI) resource types can be described with MI_Data_FeatureOfInterest,
- procedure resource types can be described with MI_Data_Procedure, and
- observed property resource types can be described with MI_Data_ObservedProperty.

Besides these O&M related types the AS-MI defines a resource type for the description of sensor networks (MI_Data_SensorNetwork). Figure 7-15 provides an illustrative overview of the available resource types. Each resource type is described by means of several mandatory and optional sections. The mandatory sections for each resource type are listed in the comment boxes that are attached to each resource type box.

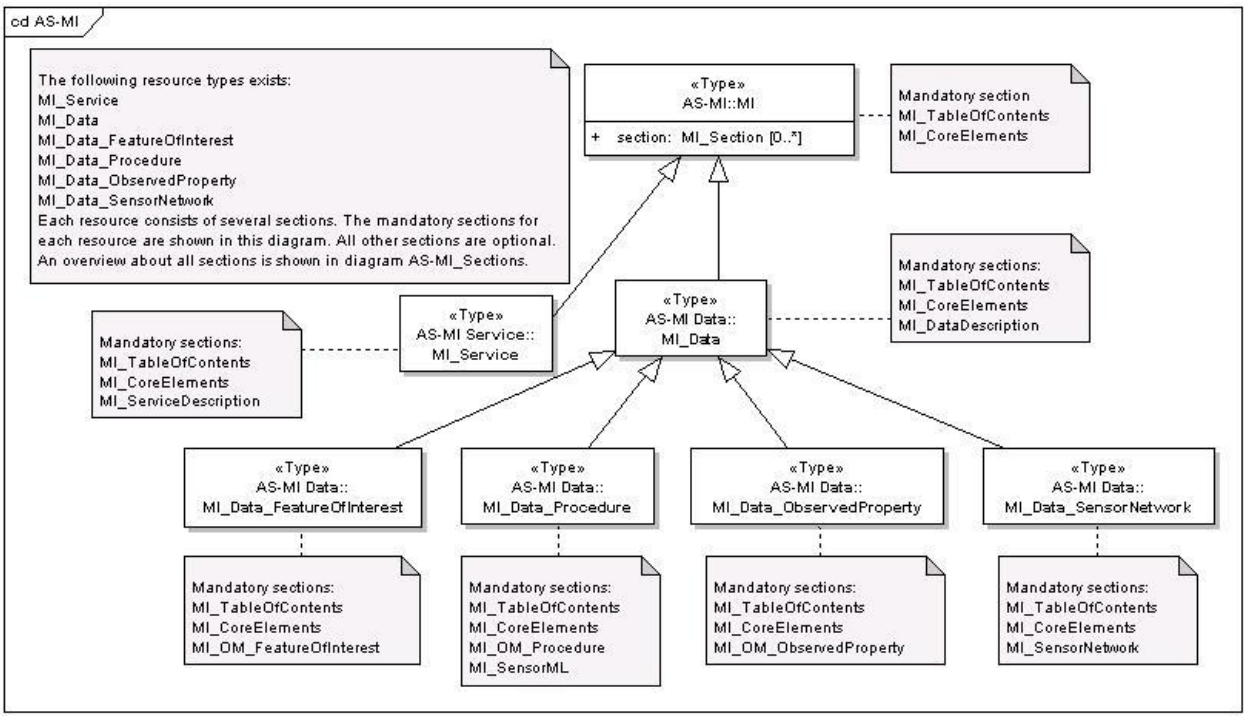


Figure 7-15: Meta-information resource types

The meta-information schema provides the possibility to create relationships between different resource types. It is possible to link services to a specific data type via the MI_DataConnector. The data connector can be restricted to specific time-intervals. This ensures that observations can be discovered by time constraints. Conversely, it is also possible to create links between data resource types and services realised by the MI_ServiceConnector.

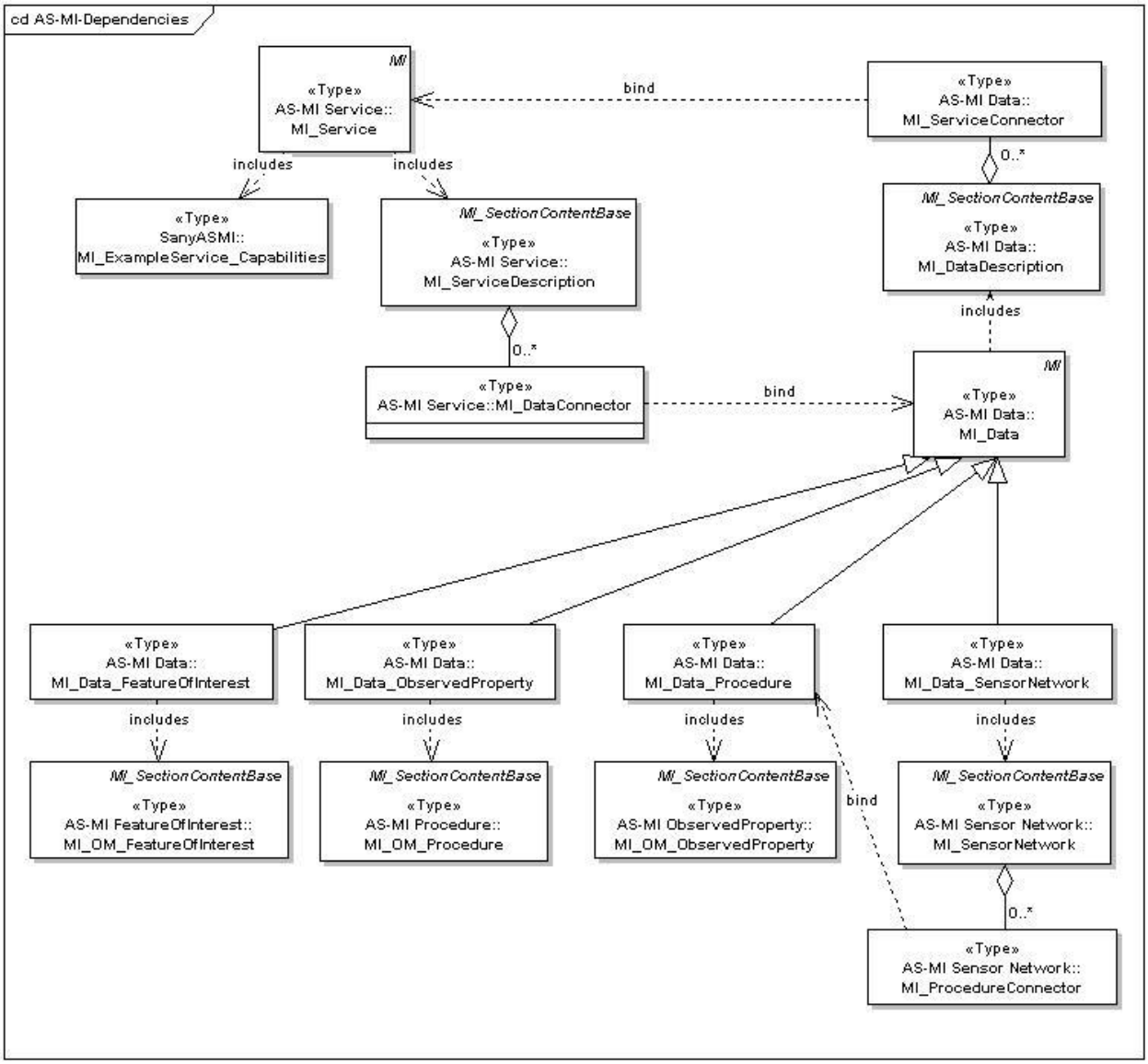


Figure 7-16 shows the dependencies between the different resource types.

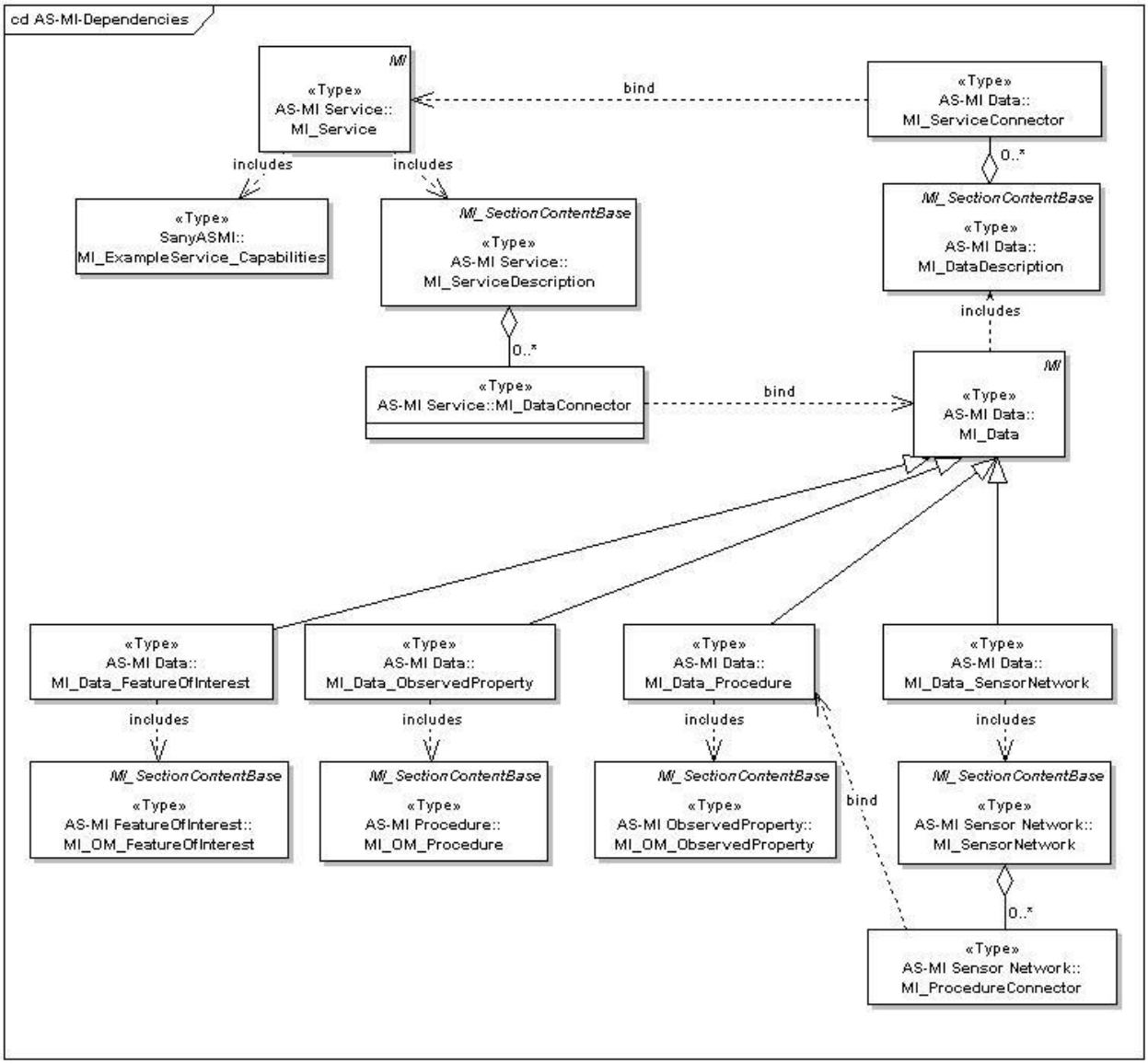


Figure 7-16: Dependencies between Resource Types

7.7.2 Generic Meta-information Sections

Each meta-information document describing one resource type consists of several sections. All defined sections are summarised in Figure 7-17. All sections are derived from the class MI_SectionContentBase.

A particular role is played by the MI_TableOfContents section. It defines the contents of the meta-information document and is mandatory for all resource types (see Figure 7-18). Each meta-information section is described by its name and a description. The combination of all sections described in the table of contents sections builds the meta-information document.

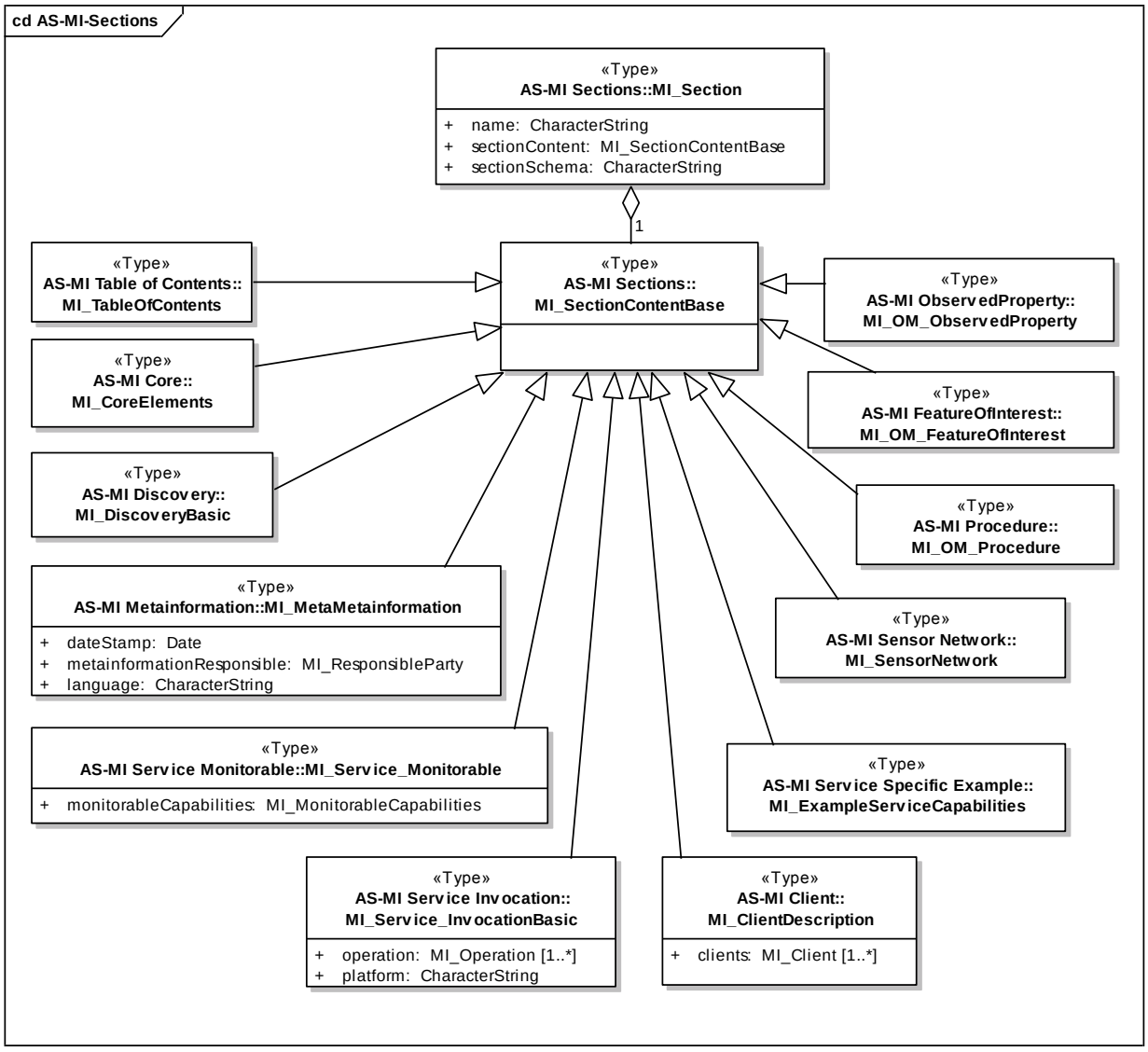


Figure 7-17: Defined Meta-information Sections

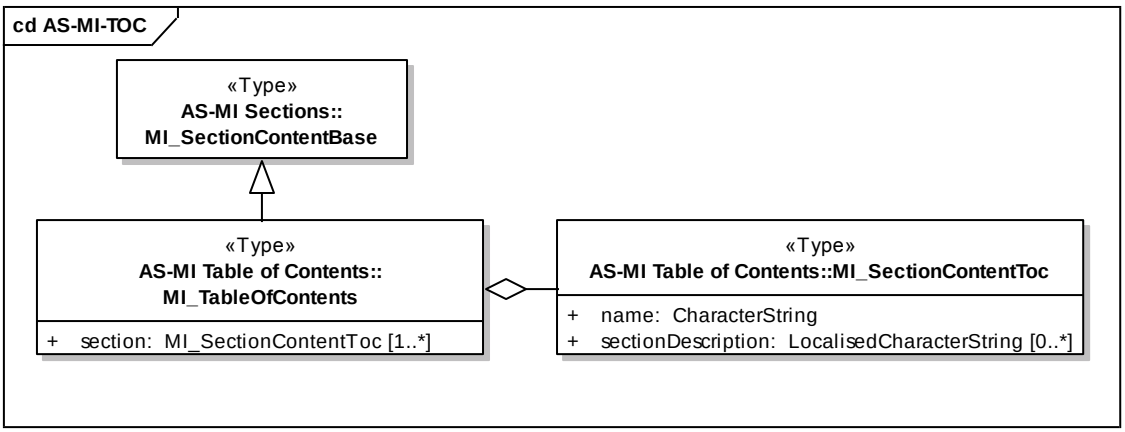


Figure 7-18: Table of Contents Section

Another important section is the section describing the common core elements (MI_CoreElements) of the resource. It contains basic information like the ID of the resource, the

name of the resource, a description of the resource and basic access to the resource via a URL (see Figure 7-19).

Catalogue entries may be time-dependent, i.e. they are only valid for given point or period of time. This may be required for validation and processing purposes. Examples are:

- A service instance or a procedure may no longer be operational (for example, because the underlying sensor is no longer accessible).

- A service may provide access to different procedures at different times. This permits the deployment of several possibly redundant sensors that are used together to provide an aggregate measurement. From the perspective of the resource requestor, they are accessible together by means of a single proxy, i.e. a single service instance.

As a consequence, the meta-information entries of the catalogue contain information about the temporal validity of meta-information entries. It is possible to search for a catalogue entry filtered with the queryables *StartDate* and *EndDate*, which correspond to the time interval of the entry (see the validation interval as defined in the mandatory core section in Figure 7-19).

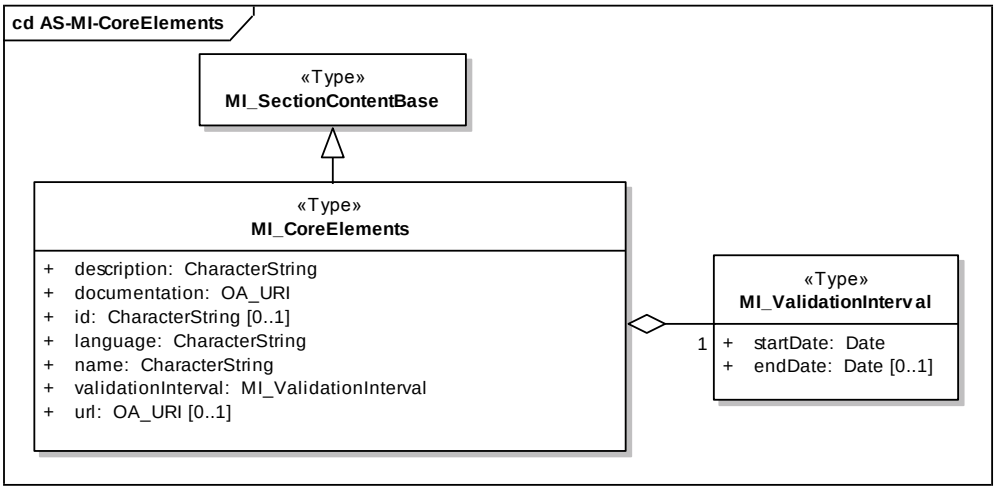


Figure 7-19: Core Meta-information Elements

Figure 7-17 shows all available sections of the meta-information schema including the centrals sections "table of contents" and the sections about the core elements. The sections that deliver the meta-information that is needed for sensor observation-specific discovery are described as follows:

- The discovery basic section (MI_DiscoveryBasic) is described below.

- The meta-meta-information section (MI_MetaMetainformation) describes meta-information about the meta-information entries themselves. This section includes a timestamp of the last update of the meta-information document in the catalogue.

- The service monitorable section (MI_Service_Monitorable) is a section which can be provided for the description of monitorable services.

- The service invocation section (MI_Service_InvocationBasic) is a section which can be provided for the description of service invocation details. It contains elements to define the access to the service. Access URLs are given together with operation names and input and output parameters of the available operations.

- The client description section (MI_ClientDescription) is a section which can be provided for the description of the clients of services. It contains access information about clients for the described service resource.

- The service specific section (MI_ExampleServiceCapabilities) is a placeholder for a section describing the specific capabilities of a service. The contents of these sections are usually described in the specification of the described service. An example for the SOS service is given below.

- The procedure section (MI_OM_Procedure) is described below.

- The feature of interest section (MI_OM_FeatureOfInterest) is described below.

- The observed property section (MI_OM_ObservedProperty) is described below.

- The sensor network section (MI_SensorNetwork) is described below.

- The SensorML section (MI_SensorML) is described below.

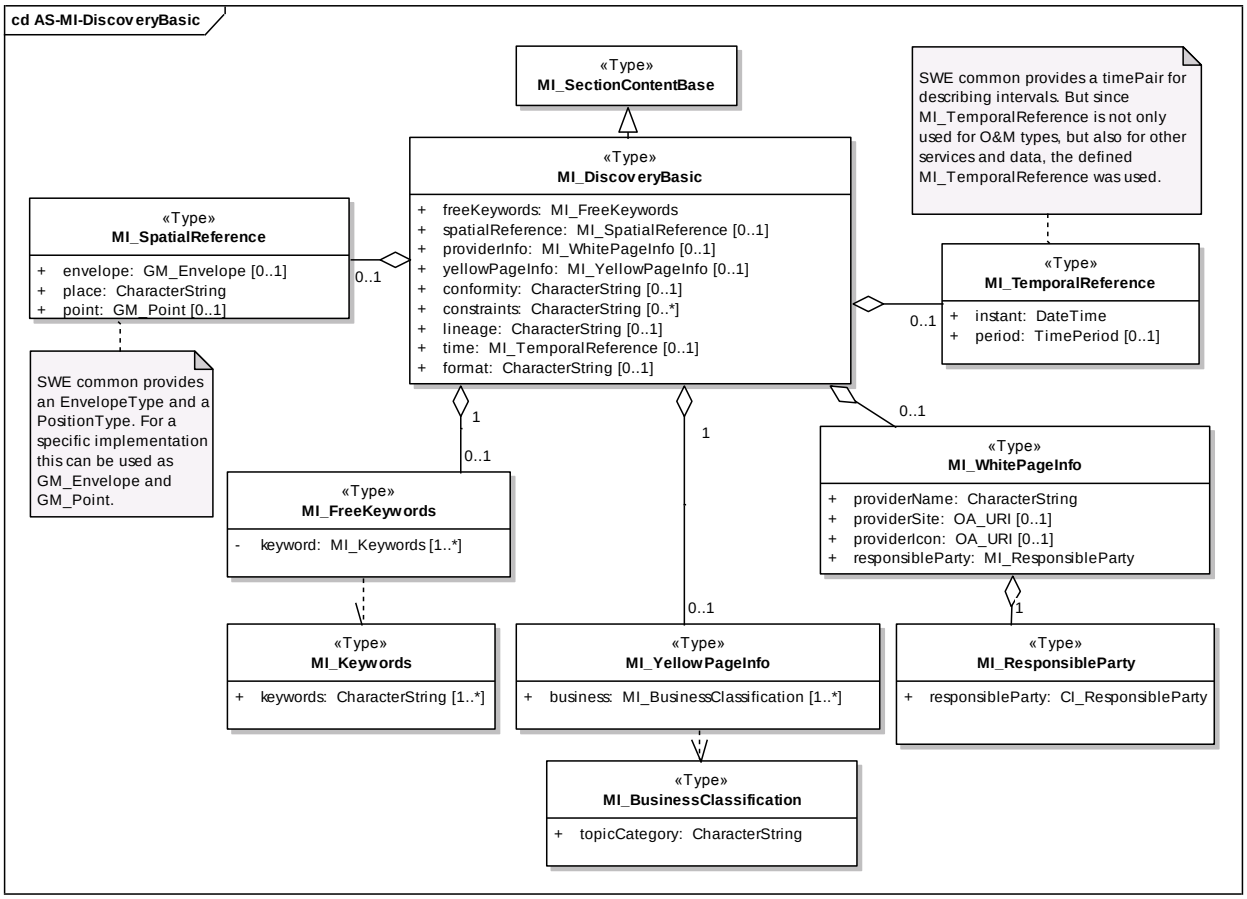


Figure 7-20: Common Meta-information Elements

The optional section MI_Discovery_Basic further details common information that may be useful for the description of the meta-information about different kinds of resources (see Figure 7-20). It includes spatial references to define the location of the described resource.

The discovery basic section also contains a temporal reference. It is possible to define a time stamp or a time interval. The defined temporal types of SWE common are not used, because the meta-information schema and especially the discovery basic section is common and not exclusively tied to the description of sensor observation meta-information. However, for the implementation of a specific catalogue containing O&M resources the usage of SWE common types for spatial and temporal reference is possible. The discovery basic section also defines free keywords to be used as matching constraints in catalogue search operations as well as white page information containing detailed contact information about the resource provider and yellow page information for the classification of the meta-information document. The white page information part uses well known types defined by ISO19115.

7.7.3 Meta-information Sections Related to Observation Discovery

The sections MI_OM_FeatureOfInterest, MI_OM_ObservedProperty and MI_OM_Procedure represent the O&M model types. These sections can be used to define meta-information that describes observations (see Figure 7-21, Figure 7-23 and Figure 7-22). For the description of the meta-information of observations there is no need for a complete mapping of the O&M elements. Instead a well defined list of available types describing the O&M types is needed. The feature of interest section is mandatory for the creation of a meta-information document of the resource type feature of interest. It includes a feature of interest type defined with a URN, which shall be defined in a list of available features of interest and a spatial reference which provides information about the location of the feature of interest.

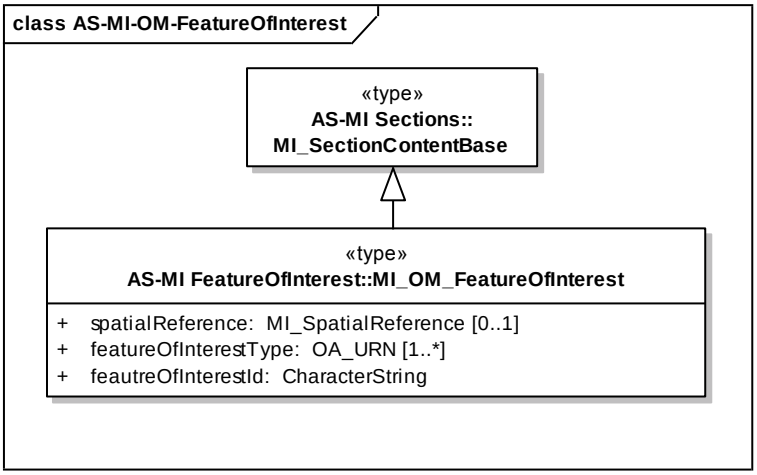


Figure 7-21: Feature of Interest Section

The procedure section is mandatory for the creation of a meta-information document of the resource type procedure. It includes a procedure type defined via a URN, which shall be defined in a list of available procedures. Like the feature of interest section it is also possible to define an area of interest of the procedure via the spatial reference attribute. Additionally the procedure section contains information about the procedure input, procedure output, accuracy,

process and responsible party which describes the meta-information for information fusion processes of procedures (see section 10.6.3).

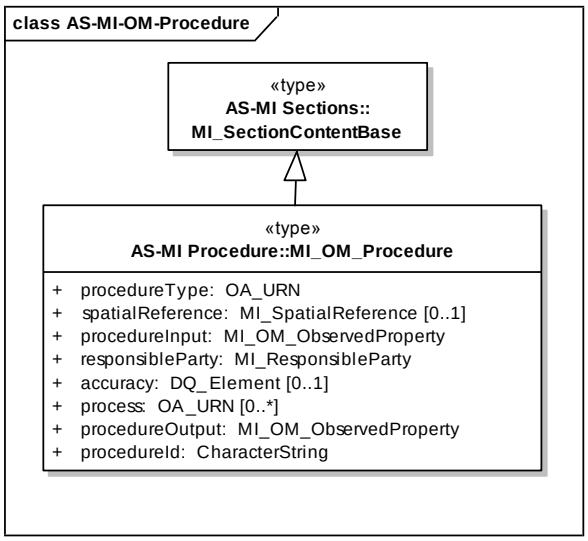


Figure 7-22: Procedure Section

The observed property section is mandatory for the creation of the resource type observed property. It includes an observed property type defined via a URN, which shall be defined in a list of available observed properties (see section 6.5.2). Additionally the section contains a placeholder for a more detailed description of the observed properties. This element can be used if there is a need for a registry containing observed properties.

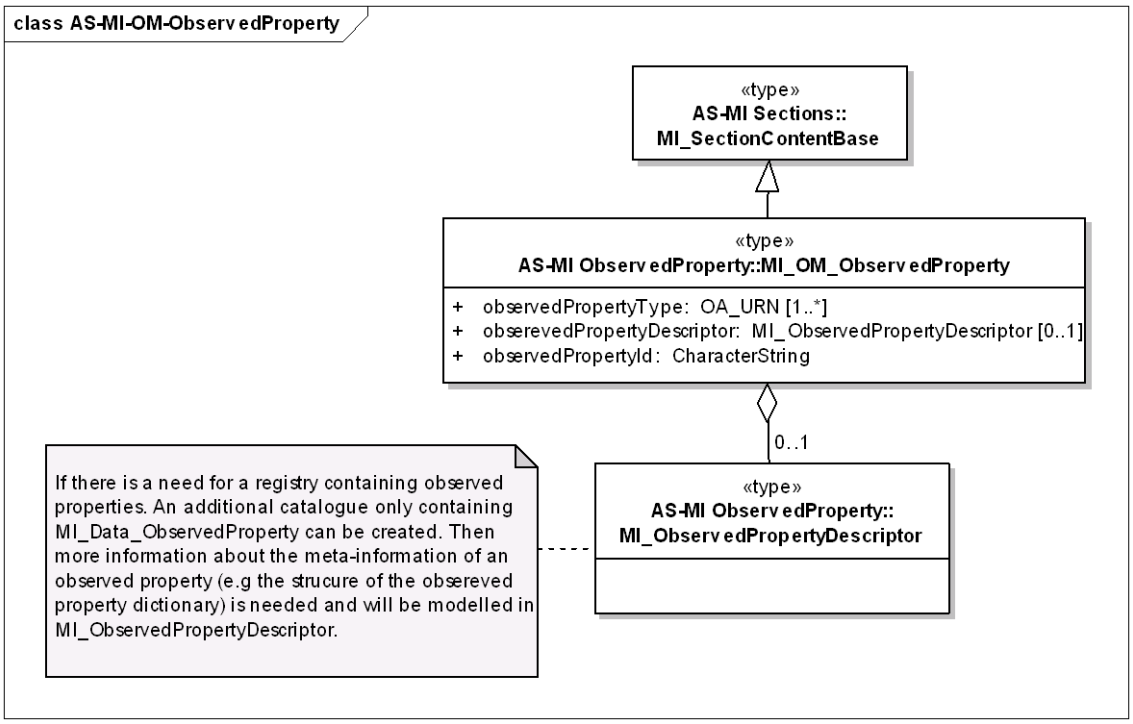


Figure 7-23: Observed Property Section

The section MI_SensorML (see Figure 7-24) provides the possibility to include a SensorML document that contains detailed information about a procedure (which, according to the O&M model, represents a sensor (see section 7.2). Using this section the creation of a catalogue as sensor registry is possible.

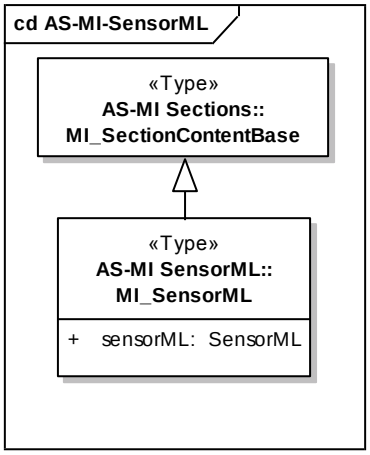


Figure 7-24: SensorML Section

For each service type there is a need to define a section that defines the specific capabilities of a service. As an example the section MI_SOS_Capabilities defines the capabilities of an SOS service (see Figure 7-25). The same structure shall be used for the creation of specific sections for other services.

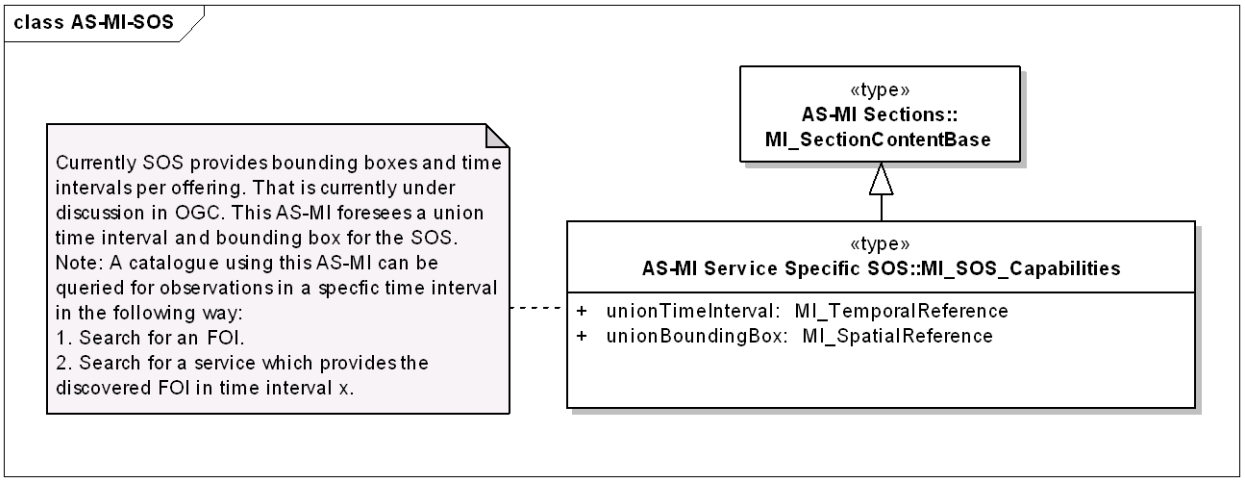


Figure 7-25: Example Section describing Specific Capabilities of a Service

Instances of the OGC Sensor Observation Service (SOS) (see section 8.2.2) provide offerings as a means to combine observations that share common characteristics. For each offering a bounding box and a time interval is provided. This principle is currently under discussion in the OGC Sensor Web Enablement Working Group. Therefore the section describing the SOS capabilities in the meta-information schema assumes an SOS having a single offering and provides a single time interval and bounding box. The MI_SOS_Capabilities type will be adapted to the result of the ongoing discussion.

It is also possible to include original OGC capabilities in a meta-information document. An example including the capabilities of an SOS is defined in the section MI_OGC_SOS_Capabilities (see Figure 7-26)

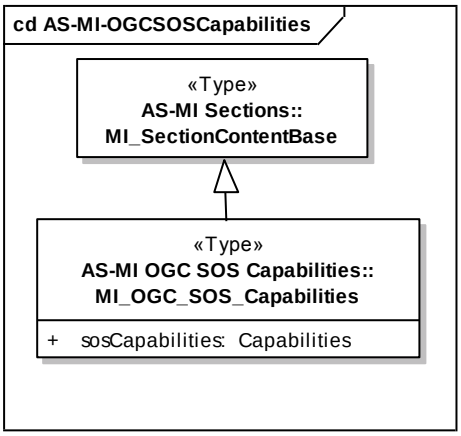


Figure 7-26: Example Section including original OGC SOS Capabilities

There is a placeholder in the section MI_SensorNetwork to define meta-information about sensor networks (see section 5.4). From the modelling point of view, a sensor network is at present simply represented by a container of a set of sensors.

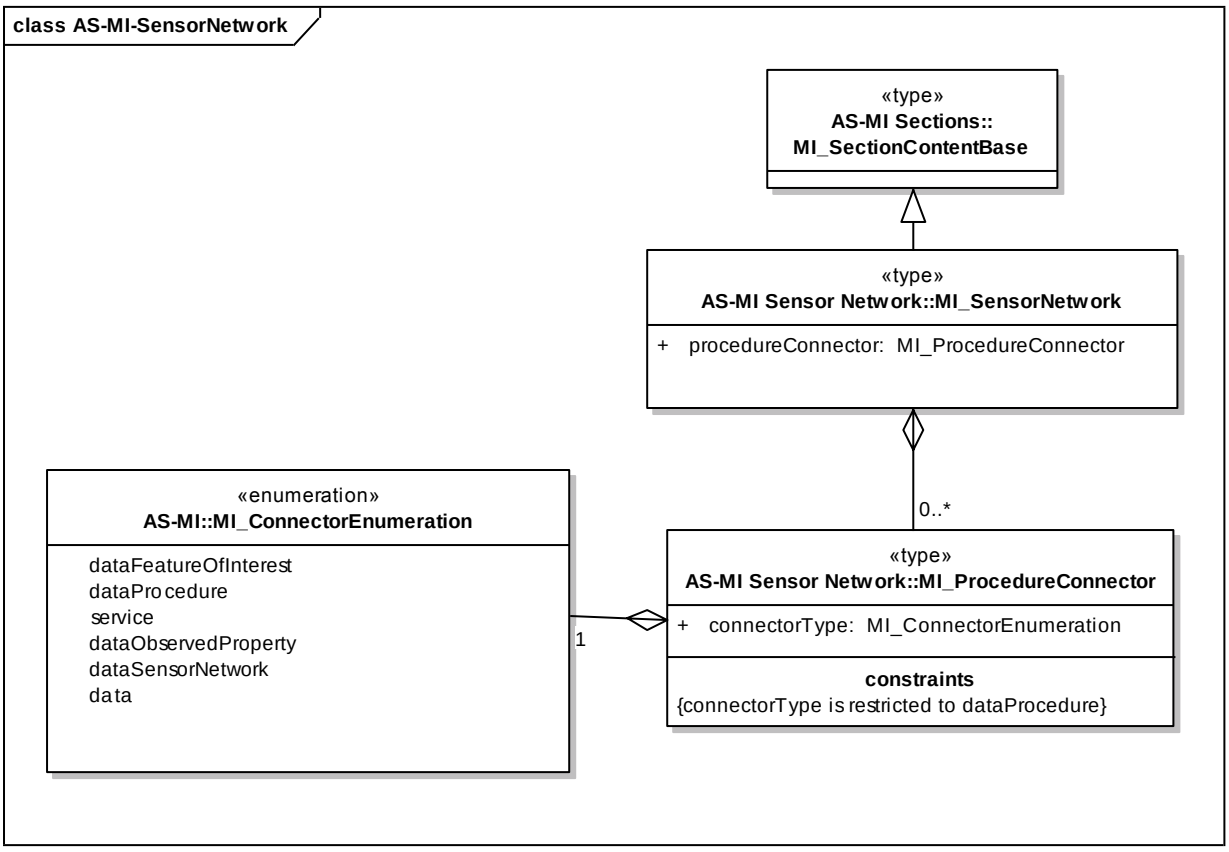


Figure 7-27: Sensor Network Section

The sections described above can be used for the discovery of different resource types in a catalogue. For the discovery of sensor observations there is a need for links between the different resource types. Figure 7-28 shows the MI_Service section that describes common meta-information about a service. Using a so-called data connector, resources information about services may be connected to different data resource types such as features of interest, procedures or observed properties. As a consequence, a query for services may include search criteria for these resources. The following describes an example workflow for the discovery of SOSs providing observations yielded by a specific procedure:

1. Search in the catalogue for the specific procedure type. The catalogue will return a MI_Data_Procedure type document.

2. Search in the catalogue for SOSs which are connected to the MI_Data_Procedure type document. The MI_Data_Procedure instance is identified via its ID. The catalogue will return MI_Service documents describing SOSs which use the specific procedure type.

3. The MI_Service document provides access information about the SOS. The access to the observations of the procedures is achieved via the means of the SOS.

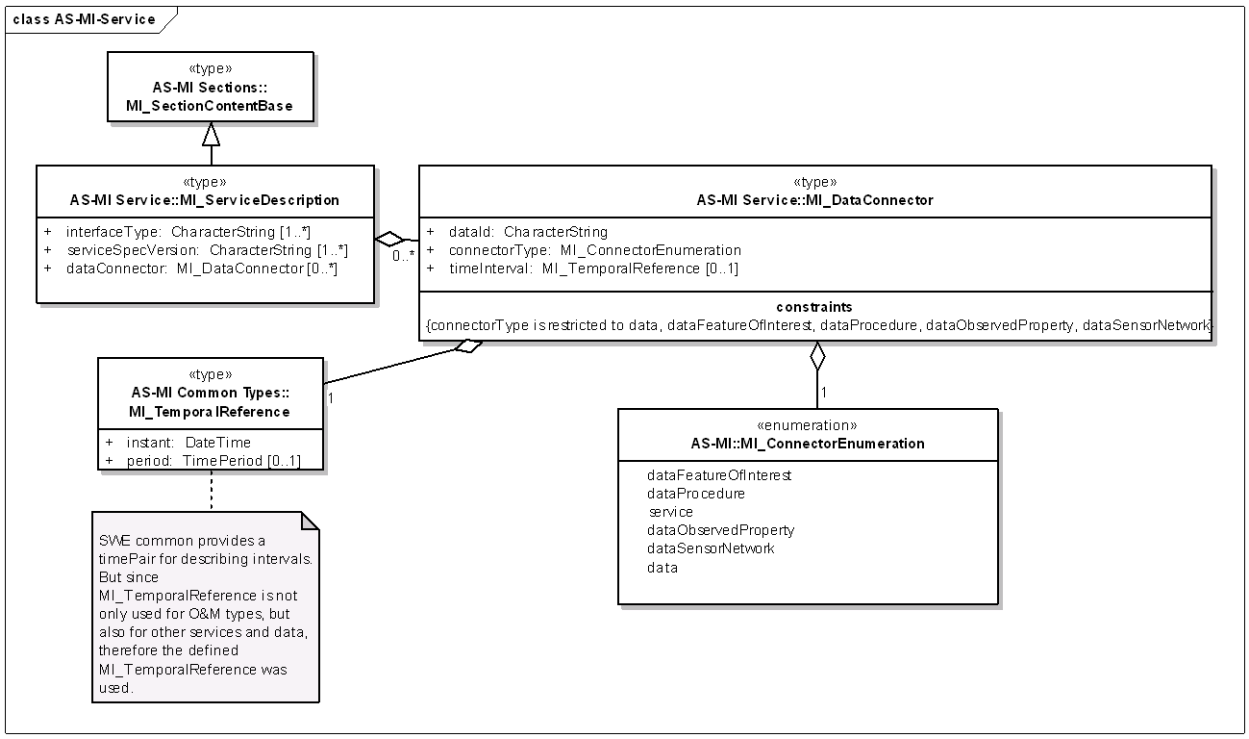


Figure 7-28: Service Description Section

Figure 7-29 shows the OA_MI_Data section that describes a data resource. In analogy to the section about services, resource information of type service can be connected to data.

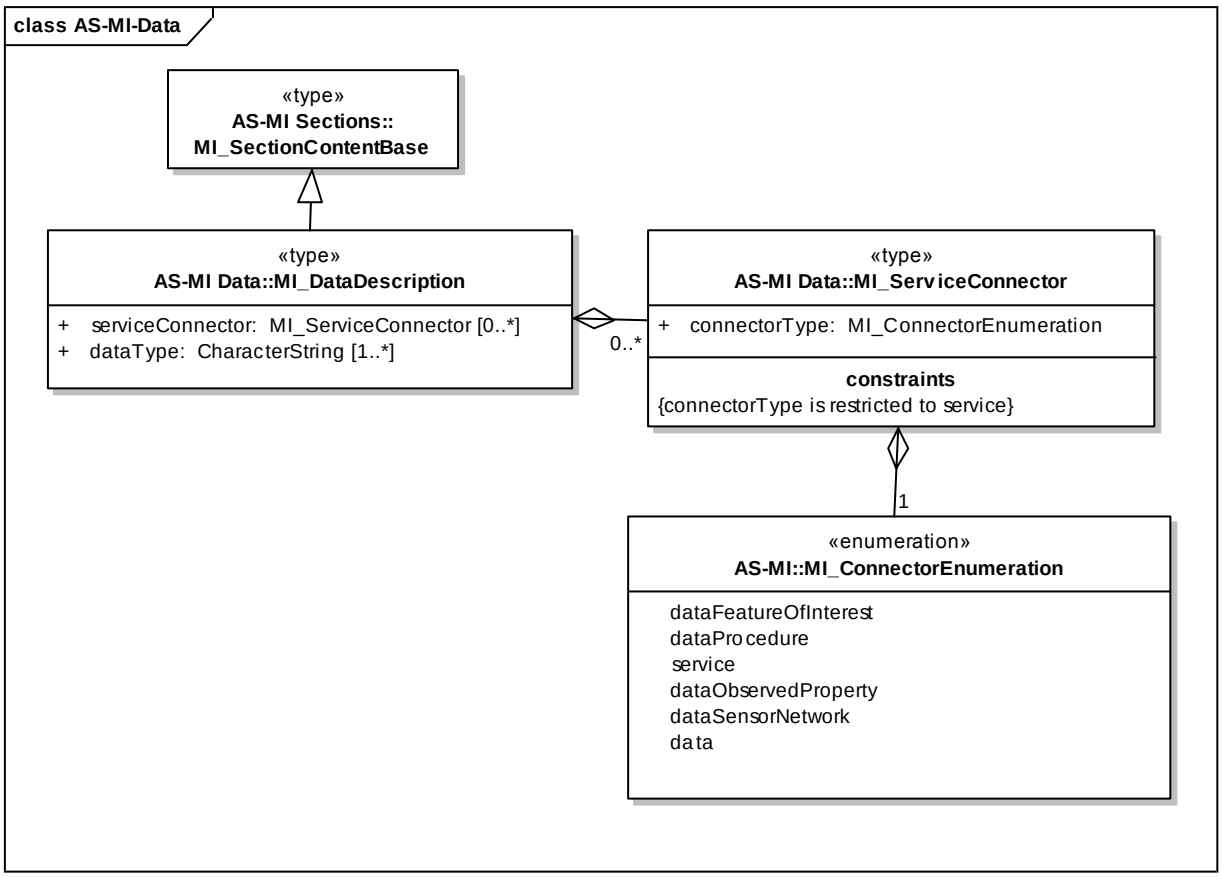


Figure 7-29: Data Description Section

8. Service Viewpoint

8.1. Overview

The service viewpoint of the SensorSA specifies the interface and service types that aim at improving the syntactic and semantic interoperability between services, source systems and applications. These interface and service types principally cover all of the functional domains as illustrated in Figure 6-1 and described in section 6.2.

The services are described according to the service description framework of the (RM-OA, 2007) and are structured as follows:

- Section 8.2 focuses on the services of the acquisition domain on the basis of the services of the OGC Sensor Web Enablement initiative.
- Section 8.3 focuses on the services that support the abstract access control pattern as introduced in section 6.8.2.
- Section 8.4 provides the descriptions of the remaining architecture services of the mediation, processing and application domain.

Note: Services of the sensor domain are out of scope of the current version of the SensorSA. The user domain is not specified on an abstract level in the SensorSA. Instead, implementations of user-oriented services in a Web-based environment may be found in the Service Support Environment (ESA SSE, 2007).

8.2. Services of the OGC Sensor Web Enablement

8.2.1 Overview

Service and Interface Type Name	Overview Description	Reference
Sensor Observation Service	Provides access to observations from sensors and sensor systems that is consistent for all sensor systems including remote, in-situ, fixed and mobile sensors.	section 8.2.2
Sensor Planning Service	Provides an interface to task any kind of sensor to retrieve collection assets.	section 8.2.3
Sensor Alert Service	Provides means to register for and receive sensor alert messages.	section 8.2.4
Web Notification Service	Provides means by which a client may conduct asynchronous dialogues with one or more other services using a range of communication protocols.	section 8.2.5

Table 8-1: Sensor Web Enablement Services

8.2.2 Sensor Observation Service

Name	Sensor Observation Service
Standard Specifications	OGC 06-009r4 Sensor Observation Service OpenGIS® Implementation Specification
Description	The goal of the Sensor Observation Service (SOS) is to provide access to observations from sensors and sensor systems in a standard way that is consistent for all sensor systems including remote, in-situ, fixed and mobile sensors. An SOS organizes collections of related sensor system observations into Observation Offerings. The SOS leverages the Observation and Measurements (O&M) specification for modelling sensor observations and the TransducerML and SensorML specifications for modelling sensors and sensor systems. The approach that has been taken in the development of SOS, and the SWE specifications on which it depends, is to carefully model sensors, sensor systems, and observations in such a way that the model covers all varieties of sensors and supports the requirements of all users of sensor data. SOS leverages the standard properties of these two data types (sensors and observations) to provide specialized operation signatures for observation data. The Sensor Observation Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Provides information about both common and specific capabilities. • *CoreOperationProfile*: Provides the basic functionality to retrieve observations. • *TransactionOperationProfile:* Provides functionality to register a Sensor with the SOS and to insert Observations • *EnhancedOperationsProfile:* Provides FOI centred operations as well as convenience operations
Interface *ServiceCapabilities*	
getCapabilities	Informs the client about both common and specific capabilities of a Service Observation Service instance.
Interface *CoreOperationProfile*	
describeSensor	Requests detailed meta-information about a sensor and delivers a document describing the sensor system.
getObservation	Retrieves observation data structured according to the Observation and Measurement specification.
Interface *TransactionOperationProfile (Optional)*	
registerSensor	Registers a new sensor system with the SOS as part of the transactional profile. The response to a *registerSensor* request contains an *AssignedSensorId* which is the identifier assigned by the SOS to designate the new sensor.
insert Observation	Inserts new observations for a sensor system.
Interface *EnhancedOperationsProfile (Optional)*	

getObservation ById (optional)	Returns an observation based on an identifier.
getResult (optional)	Obtains sensor data in a repetitive fashion from the same set of sensors without having to send and receive requests and responses that largely contain the same data except for a new timestamp.
getFeature OfInterest (optional)	Returns detailed information (such as location) about one or more features. These features are typically used in the Sensor Observation Service to identify the geographical location where observations are being made.
getFeature OfInterestTime (optional)	Returns the time periods for which the SOS will return data for a given advertised feature of interest. Delivers a GML time primitive which lists one or more time periods for which observations from that feature of interest are available.
describe FeatureType (optional)	Returns the XML schema for the specified GML feature advertised in GetCapabilities. This may be used to obtain a description of the type of an observation feature-of-interest.
describe Observation Type(optional)	Returns the XML schema that describes the Observation type that is returned for a particular phenomenon.
describe ResultModel (optional)	Returns the schema for the result element that will be returned when the client asks for the given result model by the given ResultName.
Example usage	A sensor data consumer is interested in obtaining sensor observations from one or more sensors. The consumer would perform service discovery using service capabilities information, usually via a catalogue service, in order to find SOS service instances that can provide the desired sensor observations. After initial discovery the consumer could directly obtain observations from services or could perform additional discovery at the service level or get sensor meta-information before obtaining sensor observations. Service-level discovery involves invoking the *GetCapabilities* operation to return information about the offerings that are available from each service. Detailed sensor meta-information can be obtained after extracting the sensor system identifiers out of each observation offering by invoking the *DescribeSensor* operation.
Comments	This service description is abstracted from the OGC 06-009r4 Sensor Observation Service OpenGIS® Implementation Specification.

Table 8-2 : Description of the Sensor Observation Service

8.2.3 Sensor Planning Service

Name	Sensor Planning Service
Standard Specifications	OGC 07-014, OpenGIS® Sensor Planning Service Implementation Specification
Description	The Sensor Planning Service (SPS) provides a standard interface to task any kind of sensor to retrieve collection assets. It is a service by which a client can determine collection feasibility for a desired set of collection requests for one or more sensors/platforms, or a client may submit collection requests

	directly to these sensors/platforms. Not only must different kinds of assets with differing capabilities be supported, but so must different kinds of request processing systems, which may or may not provide access to the different stages of planning, scheduling, tasking, collection, processing, archiving, and distribution of requests and the resulting observation data and information that is the result of the requests. The SPS is designed to be flexible enough to handle such a wide variety of configurations. The Sensor Planning Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Provides information about both common and specific capabilities. • *SensorTasking:* Provides as set of operations to task sensors.
Interface *ServiceCapabilities*	
getCapabilities	Informs the client about both common and specific capabilities of a Service Planning Service instance.
Interface *SensorTasking*	
describe Tasking	Requests the information needed in order to prepare an assignment request targeted at the assets that are supported by the SPS and selected by the client. The server will return information about all parameters that have to be set by the client to perform a submit operation.
getFeasibility (optional)	Provides feedback to a client about the feasibility of a tasking request. Dependent on the asset type covered by the SPS, the SPS server action may be as simple as checking that the request parameters are valid and are consistent with certain business rules, or it may be a complex operation that calculates the availability of the asset to perform a specific task at the defined location, time, orientation, calibration etc.
submit	Submits the assignment request. Depending on the covered asset, it may perform a simple modification of the asset or start a complex mission.
getStatus (optional)	Requests information about the current status of the requested task.
update (optional)	Updates a previously submitted task.
cancel (optional)	Cancels a previously submitted task.
describeResult Access	Retrieves information about how and where data that were produced by the asset can be accessed. The server response may contain links to any kind of data accessing OGC Web services such as SOS, WMS or WFS.
Example usage	Imagine a user who wants to get an overview of the current situation in a specific building. This building is what we call the area of interest (AOI). The first step would be to call up a catalogue to provide descriptions of all sensors that have an area of service (AOS) which overlaps with the AOI. However, a catalogue may only contain high-level information about the observable properties, locations and contact information. It might be necessary to call a Sensor Observation Service to retrieve the SensorML descriptions for the sensors found. In this example, one of the descriptions provides information about a video camera located inside the building.

	Now the user wants to find out more about the service. Note that a single SPS instance might be a façade covering hundreds of sensors with even more tasking parameters. The user sends a *getCapabilities* request to the SPS instance. The response shows that the service supports all SPS operations and offers only one taskable sensor. Before the user moves forward in tasking the sensor they want to find out whether they can access the sensor data. Note: The SPS is an interface to task an asset or asset system. It is not an interface to access the observed data produced by it. Observed data can be made accessible by a number of services. In most cases it might be a Sensor Observation Service, but TML Data Streaming Service, Web Feature Service, Web Coverage Service, or Web Map Service are other options.
Comments	This service description is abstracted from the OpenGIS® Sensor Planning Service Implementation Specification

Table 8-3: Description of the Sensor Planning Service

8.2.4 Sensor Alert Service

Name	Sensor Alert Service
Standard Specifications	OGC 06-028r4, OpenGIS® Sensor Alert Service Implementation Specification
Description	The Sensor Alert Service (SAS) provides the means to register for and receive sensor alert messages. The service supports both pre-defined and custom alerts and covers the process of alert publication, subscription, and notification. More specifically, the SAS defines an interface that allows nodes to advertise and publish observational data or alerts and corresponding meta-information. It allows clients to subscribe for these data – or any other data that are produced by the SAS based on incoming messages from sensors – within specific thresholds. Observational data sent from a sensor are referred to in this specification as sensor data. These sensor data might be a single observation result, a complex observation result or even an alert. The SAS sends alerts if conditions are matched, supports the integration of new sensors and uses XMPP (Extensible Messaging and Presence Protocol) to send messages. The Sensor Alert Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Provides information about both common and specific capabilities. • *SensorAlert*: interface allowing nodes to advertise and publish observational data or alerts plus corresponding meta-information. It allows clients to subscribe for data produced by the SAS based on incoming messages from sensors – within specific thresholds
Interface *ServiceCapabilities*	
getCapabilities	Informs the client about both common and specific capabilities of a Service Alert Service instance.
Interface *SensorAlert*	

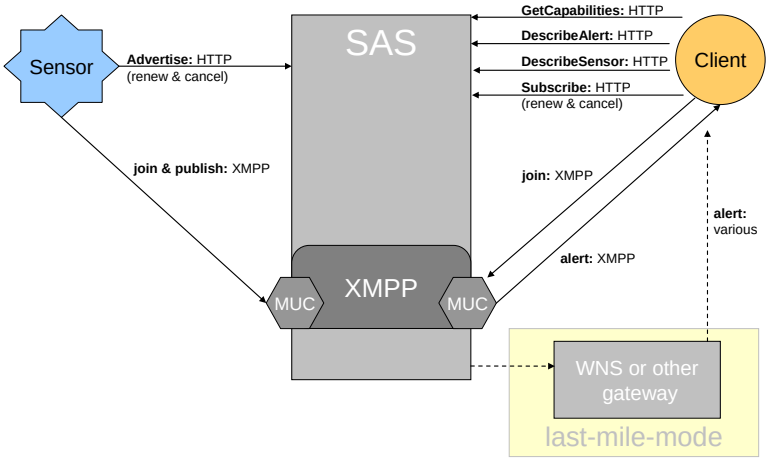
getSupported Operations (optional)	Delivers a document that describes the interface in terms of the operations that are supported by the service instance. A typical format may be a WSDL document in a W3C Web Service platform environment[1].
advertise (optional)	Allows producers to advertise the type of information published.
cancel Advertisement (optional)	Cancels an advertisement.
renewAdvertisement (optional)	Renews an advertisement when the advertisement set by the SAS has expired.
subscribe	Allows consumers to subscribe to alerts.
cancel Subscription	Cancel a subscription.
renew Subscription	Renews a subscription.
describeAlert	Delivers a template of the alert message structure.
describeSensor	Requests information about a sensor, encoded in SensorML.
Example usage	Figure 8-1illustrates a high level view of the SAS and the protocols used at the different steps.  **Figure 8-1: Overview about the Sensor Alert Service (Simonis, 2006)**
Comments	This service description is abstracted from the OpenGIS® Sensor Alert Service Implementation Specification.

Table 8-4: Description of the Sensor Alert Service

[1] In OGC 06-028r4, the corresponding operation is called "getWSDL".

8.2.5 Web Notification Service

Name	Web Notification Service
Standard Specifications	OGC 06-095r1, OpenGIS® Web Notification Service Discussion Paper
Description	The Web Notification Service (WNS) is a service by which a client may conduct asynchronous dialogues (message interchanges) with one or more other services using a range of communication protocols. This service is useful when many collaborating services are required to satisfy a client request, and/or when significant delays are involved in satisfying the request and data have to be pushed to receivers using various protocols. The "way-of-notification" palette may include e-mail, http-call (as HTTP POST: in case of sophisticated clients that act as web services themselves), SMS, Instant Message, phone call, letter or fax. A WNS has to provide at least one of the described notification mechanisms. The Web Notification Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Provides information about both common and specific capabilities. • *WebNotification*: Performs functions to send notifications to registered users and deal with responses.
Interface *ServiceCapabilities*	
getCapabilities	Requests the description about the capabilities of a service. In the particular case of a Web Notification Service, the response of a getCapabilities request is general information about the service itself, specific information about the available notification protocols, and mandatory operational parameters. The following questions have to be answered in the capabilities: • What notification protocols are supported? • Which parameters are mandatory for registering? • What kinds of response protocols are supported?
Interface *WebNotification*	
getWSDL	This operation allows a client to request and receive the WSDL definition of the server interface.
register	Registers users to receive further notification. The user address and the communicationProtocol have to be provided. The WNS must provide a UserID.
unregister	Allows a user to unregister from that service instance
updateSingle User Registration	This operation allows a client to update a previous registration by providing a new communication endpoint (e.g. an email address or a telephone number).
opdateMulti User Registration	This operation allows a client to update a previous MultiUserRegistration by adding or deleting individual group members.
doNotification	Initiates the notification of a user.

getMessage	This operation allows a client to retrieve a message which has not been delivered by the WNS because of restrictions set by the chosen transport protocol. If notification via SMS or phone call is desired then the WNS will forward the contents of the ShortMessage element of the *DoNotification* request together with a unique ID assigned to that message (for later retrieval of the complete message via the GetMessage operation).
Example usage	Observations that require preceding collection feasibility studies, complex control and management activities, or intermediate and/or subsequent user notifications are not conducive to synchronous operations, but instead favour asynchronous operations. For these cases, especially when a Sensor Planning Service comes into play, asynchronous communications need to be supported. For example, in a request for a satellite image, the user submits a collection feasibility request through a Sensor Planning Service and then subsequently requests collection of the desired observations. For this case, if any procedures are finished, interrupted, delayed, timed out, or cancelled, the user must be notified. If the feasibility request returns a positive response, the user would then request the observations. The requested data will probably not be ready for immediate retrieval; there will likely be a delay. It is also possible that the service would not be able to provide an exact retrieval date-time. Thus, a notification mechanism becomes necessary. The communication is never initiated by the Web Notification Service. This service acts as a message transducer exclusively.
Comments	This service description is abstracted from the OpenGIS® Web Notification Service Discussion Paper.

Table 8-5: Description of the Web Notification Service

8.3. Access Control Services

8.3.1 Overview

Service and Interface Type Name	Overview Description	Reference
Profile Management Service	Creates and maintains (user) profiles and their associations to identities.	section 8.3.2
Identity Management- and Authentication Service	Creates and maintains identities. Supports the management of groups (of identities) as a special kind of identity. Proves the genuineness of identities using a set of given credentials and issues session information (IdP).	section 8.3.3
Policy Management- and Authorisation Service	Acts as an external policy decision point (PDP) and policy administration point (PAP). The service provides a decision on whether some identity (e.g. a user or a service) is authorised to access a certain resource. Allows the management (create, update, delete) of XACML policies.	section 8.3.4
Policy Enforcement Service	A dedicated policy enforcement point (PEP) that handles authentication and sends authorisation requests to the PDP for non-security enabled web services.	section 8.3.5

Table 8-6: Access Control Services

Note: The abstract access control pattern specified in section 6.8.2 introduces the concept of a Policy Enforcement Point as the entity that enforces an access control policy. The SensorSA introduces a "Policy Enforcement Service" to handle policy enforcement as mainly a coordination of the authentication and authorisation request tasks. A Service Proxy (Service Side Façade) in conjunction with a Policy Enforcement Service should be used if the Policy Enforcement task should be performed in a non intrusive manner. This implementation pattern is described in section 10.5.1.1.

8.3.2 Profile Management Service

Name	Profile Management Service
Standard Specifications	The following RFC has been used as a template to define profile attributes in the SANY implementation of the Profile Management Service: • IETF RFC 2251-RFC2256 Lightweight Directory Access Protocol (LDAP) (v3) • IETF RFC 2256 - A Summary of the X.500(96) User Schema for use with LDAPv3
Description	The Profile Management Service is used to create and maintain profiles. In general, profiles (of users, services, etc.) represent entities that need to be authenticated. They are not authenticated themselves but rather represent a point of contact and management feature for authentication and authorisation

	purposes. A profile is decoupled from authentication. This decoupling is done by separating identities from profiles. An identity of a profile is defined in an Identity Management Interface instance. Management of profiles includes the association to identities as well as storage of profile attributes. Profile attributes can be arbitrary key / value-list pairs and are currently defined by an LDAP schema The Profile Management Service provides its functionality through the following interfaces: • *ServiceCapabilities* • *ProfileManagement*.
Interface *ServiceCapabilities*	
getCapabilities	Informs the client about both common and specific capabilities of a Profile Management Service instance, e.g. the supported LDAP schemas.
Interface *ProfileManagementInterface*	
createProfile	Creates a profile.
deleteProfile	Deletes a profile including the deletion of all associations to identities and profile attributes.
updateProfile	Updates a profile. Can be used to change profile related information, e.g. profile attributes.
addIdentityTo Profile	Associates an existing identity to an existing profile.
removeIdentity FromProfile	Removes a previously assigned identity from a profile.
getProfiles	Enumerates all profiles of the current service instance. Accepts a query parameter to narrow the list of returned profiles.
Example usage	The Profile Management Service provides the functionality to register and update user profiles. The result of a successful registration is a profile entry in the Profile Management information base. Moreover, the Profile Management Service's information base contains information about the profile's identities whereas authentication of associated identities and the provision of session information is provided by an Authentication Interface instance.
Comments	The Profile Management Service replaces the former User Management Service described in the RM-OA (2007).

Table 8-7: Description of the Profile Management Service

8.3.3 Identity Management and Authentication Service

Name	Identity Management and Authentication Service
Standard Specifications	The Authentication Interface uses the following standard for the encoding of session information: • OASIS Security Assertion Markup Language (SAML) v2
Description	Identities and their attributes are managed (created, deleted, etc.) using an Identity Management Interface instance. The Identity Management Interface

	acts as an identity provider (IdP). The manner in which identity information is managed is up to the particular identity provider as different authentication mechanisms (e.g. asymmetric public key/secret infrastructure or login/password) require different identity related information. In this way Identity Management can be independent of authentication methods. Please note that the association between profiles and identities is performed using an instance of the Profile Management Service. In this way authentication remains independent of Profile Management tasks related to identities. The Authentication Interface verifies genuineness of identities using a given set of credentials. The authentication mechanism, which means the way authentication is performed, is up to the service implementation. The kind of credentials an Authentication Interface needs as well as the way they are passed is specific to the authentication mechanism used. The present specification of the Identity Management and Authentication Service supports a username / password authentication mechanism. A SAML ticket (session information) returned after a successful authentication can be used to invoke services demanding authenticated identities. The Identity Management and Authentication Service provides its functionality through the following interfaces: • *ServiceCapabilities* • *Authentication.* • *IdentityManagement*
Interface *ServiceCapabilities*	
getCapabilities	Informs the client about both common and specific capabilities of an Identity Management and Authentication Service instance.
Interface *Authentication*	
login	Performs a login using the credentials and identity (e.g. username / password) supported by this Authentication Interface instance. Returns a SAML ticket that contains the authenticated identity and related attributes and possibly a set of authenticated group identities that is associated to the authenticated identity. Note: a SAML ticket serves as a asserted and temporarily valid record of a subject's identity including identity properties (e.g. age) that can serve as a basis for an authorisation decision.
verifySession Information	Verifies the SAML ticket (session information) previously issued by the same Authentication Interface instance. Returns a status value indicating the validity of the SAML assertion stated in the SAML ticket.
Interface *IdentityManagement*	
addIdentity	Creates an identity. The identity's representation is specific to the supported authentication mechanism. The present specification of the Identity Management and Authentication Service supports UserNamePassword Identities apart from the obligatory GroupIdentities.
delete Identity	Deletes an existing identity. Deletion of identities implies the need to update the corresponding Profile Management Service instance as well as any policy

	referring to it.
update Identity	Updates an existing identity. The identity to be updated as well as information to be changed, e.g. a new username, additional or modified attributes, shall be provided as input.
add Credentials	Adds credentials to a certain identity. Credentials are specific to the authentication mechanism used. For a username/password authentication the credential is a password.
update Credentials	Updates credentials (e.g. password) for a certain identity (e.g. username).
deactivate Identity	Deactivates an identity without removing it. The identity, e.g. username to be deactivated and additional information, e.g. a time period for deactivation, shall be provided as input.
activate Identity	Activates an existing, formerly deactivated identity. The identity, e.g. username to be activated and additional information, e.g. a point of time for activation, shall be provided as input.
getIdentities	Executes a query and returns Identities that match the query conditions. The query language depends on the different implementations of the service instance.
addItentityTo Group	Associates an existing group with an existing idnetity. The identity must reside in the same Identity Management Interface instance.
removeIdentity FromGroup	Removes the association between a given identity and a given group. The removed identity is not deleted.
Example usage	Multiple instances of Identity Management and Authentication Services may coexist in a network and each organisation may maintain their own instance of the Identity Management and Authentication Service. This favours cross-organisational sign-on or single-sign-on (SSO) since identities represent only the identity of a (user) profile and one profile may refer to multiple identities, each registered at different instances.
Comments	The Identity Management and Authentication Service replaces the former Authentication Service described in the RM-OA (2007).

Table 8-8: Description of the Identity Management and Authentication Service

8.3.4 Policy Management and Authorisation Service

Name	Policy Management and Authorisation Service
Standard Specifications	The following standards are used for the definition of policies and authorisation request and responses: • OASIS Security Assertion Markup Language (SAML) v2.0 • OASIS eXtensible Access Control Markup Language (XACML) TC v2.0 • OASIS SAML 2.0 profile of XACML v2.0

Description	The Authorisation Interface evaluates an authorisation request of a policy enforcement point (PEP) and returns the authorisation decision. The authorisation decision is based on an XACML authorisation request passed from the PEP or a security-enabled service. The authorisation request comprises the authenticated identities of the service requestor including all identity attributes relevant for an authorisation decision as well as specific environment attributes, for example individual state variables of the service. The Policy Management Interface is responsible for the management of access policies and thus plays the role of a policy information point (PIP) and policy administration point (PAP). Access policies can be expressed in the XACML access control policy language. The Policy Management and Authorisation Service provides its functionality through the following interfaces: • *ServiceCapabilities* • *Authorisation* • *PolicyManagement*
Interface *ServiceCapabilities*	
getCapabilities	Informs the client about common and specific capabilities of a Policy Management and Authorisation Service instance.
Interface *Authorisation*	
authorise	This operation uses the SAML 2.0 profile of XACML 2.0 to request an authorisation decision. The authorisation decision is currently provided as a compliance value indicating how to treat the request (e.g. permit or deny).
Interface *Policy Management*	
createPolicy	Creates a new policy.
deletePolicy	Deletes an existing policy.
getPolicy	Retrieves a policy identified by a unique ID.
getPolicies	Retrieves a sequence of policies maintained by the Policy Management Interface instance..
updatePolicy	Updates an existing policy.
Example usage	Access policies can be expressed in the XACML access control policy language. XACML allows the definition of very flexible policies that can be evaluated against any kind of environment attributes. Such environment attributes may be derived from boundary conditions of a service request as well as from the underlying data source. By defining an appropriate policy for e.g. a WMS and a SOS the Policy Management and Authorisation Service may restrict access to a certain layer or offering.
Comments	The Policy Management and Authorisation Service replaces the former Authorisation Service described in the RM-OA (2007).

Table 8-9: Description of the Policy Management and Authorisation Service

8.3.5 Policy Enforcement Service

Name	Policy Enforcement Service
Standard Specifications	The following standards are by the Policy Enforcement Service: • OASIS Security Assertion Markup Language (SAML) v2.0 • OASIS SAML 2.0 profile of XACML v2.0
Description	The Policy Enforcement Service is a dedicated policy enforcement point (PEP) that handles the necessary interaction with Authorisation Service and Authentication Service. The PEP comprises the service independent part of a proxy solution for non-security enabled web services and thus is one important component of non-intrusive web service security for services compliant with the SANY W3C Web Services Platform (section 9.2.1). It enables both security-enabled and non-security-enabled clients to access a proxied web service via the same interface. The PEP always works in conjunction with a service specific proxy and/or a service specific client facade. As suggested in OASIS WS-Security standards, the optional security information encoded in SAML is provided in the SOAP header while the actual service request in the SOAP body remains unchanged. The Policy Enforcement Service provides its functionality through the following interfaces: • *ServiceCapabilities* • *PEP*
Interface *ServiceCapabilities*	
getCapabilities	Informs the client about both common and specific capabilities of a Policy Enforcement Service instance.
Interface *PEP*	
doRequest	This operation performs a service request and enforces access restrictions by calling a service that implements the Authorisation Interface. In general doRequest is called by a proxy and/or client facade..
Example usage	The Policy Enforcement Service is designed to interact with the Authentication and the Policy Management and Authorisation Service. It verifies the genuineness of the security information by calling the Identity Management and Authentication Service and then delegating the evaluation of the access policies to an external policy decision point (PDP), the Policy Management and Authorisation Service.
Comments	none

Table 8-10: Description of the Policy Enforcement Service

8.4. Services of the Mediation, Processing and Application Domain

In the following, an overview is given of those architecture services of the SensorSA that belong to the mediation and processing as well as the application domain. See Table 8-11 for the list of ORCHESTRA architecture services as specified in (RM-OA, 2007) that could immediately be applied in a SANY Service Network. Their abstract specifications are available under (ORCH-AbstrServ, 2007), and their implementation specifications are found under (ORCH-ImplServ, 2007).

Note: For better readability and self-containment of the present document, some of these services are additionally described in the present SensorSA. This is especially the case when their description has been extended or tailored to the SANY purposes or if the service type is used in the description of a service interaction pattern in section 10.

Service and Interface Type Name	Overview Description	Reference
Basic Interfaces	Interface types enabling a common architectural approach for all ORCHESTRA Services: – self-description of service instances (capabilities) – synchronous and asynchronous interactions – transactional support Furthermore: – predefined exception types	RM-OA, 2007
Catalogue Service	Ability to publish, query and retrieve descriptive information (meta-information) for resources (i.e. data and services) of any type. The SANY Catalogue Service is an extension of the ORCHESTRA Catalogue Service and is described in section 8.2.	section 8.4.1
Document Access Service	Supports access to documents of any type (textual documents, images). A document is referenced by a document descriptor which is considered to be a specific kind of a feature type.	RM-OA, 2007
Feature Access Service	Selection, creation, update and deletion of feature instances and feature types[1] available in a service network. Features provided are instances of a certain feature type defined in an ORCHESTRA Application Schema. Interface may be re-used by more specific access services using interface inheritance.	RM-OA, 2007
Map and Diagram Service	Enables geographic clients to interactively visualise geographic and statistical data. Transforms geographic data (vector or raster) and/or numerical tabular data into a graphical representation using symbolization rules. The main output of this service is an	section 8.4.3

[1] As in (RM-OA, 2007), SANY adopts the ISO 19101 definition of a feature as being an "abstraction of a real world phenomenon" but explicitly subsumes hypothetical worlds under the term "real world", too. Thus, for instance, a "model" may also be understood to be a feature type.

	image document which may be a map, a diagram or a thematic map (visualization of the spatial distribution of one or more statistical data themes).	
Ontology Access Interface	Supports the storage, retrieval, and deletion of ontologies as well as providing a high-level view on ontologies. As an optional Knowledge Base interface, it provides operations to query and update models contained in a knowledge base	RM-OA, 2007
Name Service	Encapsulates the implemented naming policy for service instances in a service network, e.g. creates globally unique service instance names using a defined naming policy. Important if several service networks across different platforms are to be interconnected.	RM-OA, 2007
Processing Service	Describes a common interface for services offering processing operations on spatial (vector as well as raster) and non-spatial data. Examples of processing operations are statistical or geospatial calculations, image processing and analysis or, in general, computer algebra operations.	section 8.4.2
Schema Mapping Service	Provides functionality for the mapping of features from a source into a target schema.	RM-OA, 2007
Service Monitoring Service	Provides an overview about service instances currently registered within service network, e.g. 1. Actual status (e.g. running, stopped, offline) 2. Statistical information (e.g. average availability, response times)	RM-OA, 2007

Table 8-11: Architecture Service applicable for a Sensor Service Network

8.4.1 Catalogue Service

Name	Catalogue Service
Standard Specifications	The Catalogue Service has been derived from the approach to the handling of meta-information in SensorSA (see section 6.3). Thus, the following series of catalogue standards and specifications has been considered, but the goal has not been to specify a service that is exactly compliant to one of these services. • OASIS UDDI Version 3.0.2 Specification (http://uddi.org/pubs/uddi_v3.htm) • OGC 06-079r2 EO Application Profile for CSW 2.0 (Status: Pending) • OGC 06-131r1 EO Products Extension Package for ebRIM (ISO/TS 15000-3) Profile of CSW 2.0 (Status: Discussion Paper) • OGC 07-006r1 OpenGIS® Catalogue Service Implementation Specification V2.0.2 • OGC 07-038r1 OGC™ Cataloguing of ISO Metadata (CIM) using the ebRIM profile of CS-W (Status: Discussion Paper)

	• OGC 07-045 OpenGIS® Catalogue Service Specification 2.0.3 ISO Metadata Application Profile (Status: Implementation Specification final) • OGC 07-110r1 OpenGIS® Web Registry Service – ebRIM profile of CSW V1.1.0 (Status: pending) • ORCHESTRA Catalogue Service – Abstract Specification V1.1 (http://www.eu-orchestra.org/docs/OA-Specs/ Catalogue_Service_Specification_v1.1-BRGM-IITB.pdf) However, the functionality of these specifications for basic search and publication is supported by the Catalogue service such that it may be mapped onto corresponding service implementations. In addition, the Catalogue Service provides the following interfaces: • A Semantic Interface for semantic extensions • A Catalogue Management Interface for the management of cascaded catalogues The Catalogue Service does not define a meta-information schema by itself. The intention of the SANY Catalogue Service is to provide a flexible service type which can be adapted to the particular purposes of the application environment.
Description	The Catalogue Service supports the ability to publish, query and retrieve descriptive information (meta-information) for resources (i.e. data and services), meta-information about sensors and source systems and instances of feature types and defined extensions (e.g. observations, document descriptors, schema descriptors). The Catalogue Service is not tied to a particular schema of a meta-information standard (e.g. ISO 19115); instead it supports application schemas for meta-information (AS-MI) that are designed according to the rules of the ORCHESTRA meta-model (RM-OA, 2007). Because it is independent from any specific meta-information standard the catalogue can be used to store meta-information about services and data according to the meta-information schema used in the catalogue. Therefore a catalogue instance can be used as a data catalogue, service registry or both if multiple meta-information types are used in the catalogue instance. The multi-linguality of the catalogue is dependent on the multilingual capabilities of the meta-information schema used inside the catalogue. Meta-information entries in catalogues represent resource characteristics that can be queried and presented for evaluation and further processing by both humans and software. The Catalogue Service supports the discovery of registered resources within an information community and returns binding information that allows a user to locate and access the resource (e.g. a URI). The Catalogue Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Provides information about both common and specific capabilities.

	• *CatalogueSearchInterface:* The interface for search provides a means for searching for information in the catalogue. The client asks the catalogue capabilities for the available catalogue entry types. Each entry type is associated with a meta-information type and its corresponding query languages. With this information the client can query the catalogue entry type with the appropriate query language. • *SemanticInterface:* The semantic interface contains all operations that work on the basis of ontologies in order to improve the query and result assessment phase of a search. • *CataloguePublicationInterface:* The interface for publication is responsible for including, updating and deleting meta-information in the catalogue. It is pushing information into the catalogue. It provides operations for filling the catalogue. The needed meta-information could be created with some kind of meta-information editor, in which the user is specifying the meta-information about resources to be registered in the catalogue, or it could be collected through the collection interface. • *CatalogueCollectionInterface:* The collection interface provides operations which are helpful for the automatic update of catalogue content, in contrast to the publication interface which just fills the catalogue with provided content. It is pulling meta-information into the catalogue. The operations in this interface should be able to be triggered from the outside of the catalogue and it should be possible to define a periodic update from the catalogue content. • *CatalogueManagementInterface*: The management interface provides operations for the management of the underlying catalogue services of a cascading catalogue scenario.
Interface *ServiceCapabilities*	
getCapabilities	Informs the requestor about both common and specific capabilities of a Catalogue Service instance. Examples include information about query languages, whether or not the catalogue service instance is the main catalogue of a service network (the "OSN Catalogue" as introduced in (RM-OA, 2007), and the meta-information types used in the Catalogue Service instance.
Interface CatalogueSearchInterface	
search	Returns a list of identifiers and some corresponding meta-information attributes of discovered catalogue entries, given a request expressed in a particular query language. In a cascaded environment it is possible to specify a list of catalogues in the request to forward the search only to specific underlying catalogues.
getMeta Information	Returns the entire associated meta-information instance, given some identifiers of catalogue entries managed by the catalogue as returned by a previous search operation call.
getQuery Domain	Returns the domain of values that are applicable to a property of the meta-information type and queryable elements of the catalogue. This is used by catalogue clients. Using this operation by giving the parameters of interest,

	the client shall be provided told which values (e.g. list of values, range of values) are allowed for meta-information properties and queryable elements.
getMeta Information Type	Returns the associated meta-information type, given a list of catalogue entry types managed by the catalogue.
Interface SemanticInterface	
improveQuery	Returns semantically connected keywords (e.g. parents, children or related concepts in an ontology) related to a given search request. This operation enables interactive or automatic query expansion.
activate Ontology	Activates a specific ontology known by the catalogue to be used in the semantic extension (e.g for *improveQuery* or the ranking of search results). Available ontologies shall be provided by the *getCapabilities* operation and uploading of ontologies shall be managed via operations of the Ontology Access Interface (RM-OA, 2007).
Interface *CataloguePublicationInterface*	
createMeta Information	Pushes information into the catalogue. The task of this operation is to insert catalogue content into the catalogue. The operation receives the meta-information to be stored and returns information about the update of the catalogue.
setMeta Information	Updates the catalogue content. The operation receives the meta-information types to be stored and returns information about the update of the catalogue.
deleteMeta Information	Deletes catalogue content from the catalogue. The input is a constraint to identify the catalogue content which is to be deleted. The operation returns information about the update of the catalogue.
Interface *CatalogueCollectionInterface*	
collectMeta Information	Pulls meta-information into the catalogue. The operation receives one reference of a source of meta-information and a catalogue entry type. This catalogue entry type is the type in which the meta-information is going to be stored in the catalogue. The operation returns information about the update of the catalogue.
collectMeta Information Periodic (optional)	Receives one reference of a source of meta-information, the catalogue entry type and the time interval between two collections and a date to stop the collect. The catalogue entry type is the type in which the meta-information is going to be stored into the catalogue. The operation is processed periodically according to the given intervals and stores the resulting meta-information in the catalogue.
Interface *CatalogueManagementInterface*	
setCatalogue	Publishes a new underlying catalogue to the list of available catalogues of the cascaded catalogue. A list of available underlying catalogues shall be provided via the *getCapabilities* operation.
getCatalogue	Returns information about a specific underlying catalogue.
delete Catalogue	Deletes an underlying catalogue from the list of available catalogues
activate Catalogue	Activates or deactivates a specific underlying catalogue for the cascaded search of the cascaded catalogue.
Example usage	A possible usage scenario of the catalogue is the use of a catalogue for discovering maps and displaying them in a map viewer. The following steps need to be accomplished for this scenario:

	1. The catalogue needs to be initialized with meta-information about the maps and a service capable of displaying the maps. The meta-information can be written into the catalogue using operation *createMetaInformation*. 2. The user performs a search for available maps on the catalogue using the *search* and *getMetaInformation* operations. 3. The user performs a search for an available map viewer, again using the *search* and *getMetaInformation* operations. 4. The user displays the maps in the map viewer, using the retrieved meta-information about the maps and the map viewer.
Comments	The abstract specification leaves the question of the meta-information creation open. It could be created by the user with the help of a meta-information editor or automatically either within the catalogue inside *collectMetaInformation* or with the usage of other means and services inside *collectMetaInformation*. The support of multi-linguality depends on the meta-information schema used in the catalogue. Meta-Information about data and services inside the scope of a service network will be described with the help of the service capabilities.

Table 8-12: Description of the Catalogue Service

8.4.2 Processing Service

Name	Processing Service
Standard Specifications	• OGC 05-007r7 Web Processing Service (WPS), version 1.0.0 (OGC Standard) including the WPS Corrigendum OGC 08-091r1 Instantiations and examples of how to use the Processing Service in a geospatial application domain may be found in: • ORCHESTRA Application Architecture (ORCHESTRA Deliverable D4.1.2)
Description	The Processing Service describes a common interface for services offering processing operations on spatial (vector as well as raster) and non-spatial data. Examples of processing operations are statistical or geospatial calculations, image processing and analysis or, in general, computer algebra operations. The Processing Service provides mechanisms to identify the data required by the calculation, initiate the calculation, and manage the output so that it can be accessed by the client. It is also possible that the client cancels the process if its output is not needed any longer. The client can also choose to be notified about the status of a process execution via a Web Notification Service. This removes the burden of constant polling by the client to achieve the same level of information. The Processing Service provides the functionality through the following interface: • *ServiceCapabilities:* Informs the client about both common and

	specific capabilities of the Processing Service. • *ProcessingService:* provides the means to invoke and cancel a process as well as retrieving information about the current process status.
Interface *ServiceCapabilities*	
getCapabilities	Informs the requestor about both common and specific capabilities of a Processing Service instance. Examples of specific capabilities are types and versions of procedures and algorithms supported by the processing service.
Interface *ProcessingService*	
describeProcess	Requests and receives detailed information about one or more processing operation(s) that can be executed by an execute operation, including the input parameters and formats, and the outputs.
execute	Executes a specified processing operation implemented by the Processing Service, using provided input parameter values. The process can be executed synchronously or asynchronously. In the case of the latter the client can be notified through a WNS about the completion of the process. The processed values can then be retrieved at the specified location.
getStatus	Retrieves information about the current status of a process. Such information includes the progress of an executing process and also the URL where the output of the process can be retrieved by the client after it has finished,
cancel	The Cancel operation allows a client to cease the execution of the specified process.
Example Usage	An instance of a Processing Service may offer 'local' and 'zonal' operations that are performed on coverage features. The local operations cover the arithmetic binary operators and a reclassify operation based on local values. The zonal operators cover a spatial aggregation function and a reclassification, both based on a zone feature. Another example is the support for processing chains for the purpose of information fusion as explained in section 10.9.
Comments	This service description is abstracted from the OGC Web Processing Service Specification. The Processing Service as described here is a kind of template. It is highly generic and is the service of choice in the SensorSA for encapsulating general purpose processing such as fusion and models. This means that the semantics of the Processing Service, i.e. the meaning of the processing when calling the *execute* operation, shall be specialised in a further specification and an associated information model. The original OGC processing service is enhanced with two operations (*getStatus, cancel*) which give the client more control over a process that is executed.

Table 8-13: Description of the Processing Service

8.4.3 Map and Diagram Service

Name	Map and Diagram Service

Standard Specifications	The Map and Diagram Service is a functional extension of the following standards: • ISO 19128:2005 - Geographic information -- Web Map Server Interface • OGC 06-042 Web Map Service (WMS) Implementation Specification V1.3.0 The extensions refer to the generation of diagrams, legends, and the detailed layer descriptions that are needed for fine-grained user-styling, and the management of layers and styles. Data sent to the Map and Diagram Service may be structured according to: • ISO 19136 Geographic information -- Geography Markup Language (GML) An alternative data source may be a feature store that provides feature instances according to: • OGC 04-094 Web Feature Service (WFS) Implementation Specification) V1.1 The following standards are used for the symbology definition: • OGC 02-070 Styled Layer Descriptor (SLD) Implementation Specification V1.0 • OGC 04-095 Filter Encoding Implementation Specification V1.1 These are extended with symbolizers for diagrams.
Description	The Map and Diagram Service is a service that dynamically portrays geographic and statistical data using style definitions and symbolisation rules. Its main task is to produce maps and diagrams from geographic data (vector or raster) and/or statistical data (e.g. census data or results of a statistical analysis) as digital image files suitable for display on a computer screen. This service is able to create maps and diagrams based not only on data hosted on the server, but also on data provided by external services (e.g. by a Feature Access Service) or directly included in the request message as GML (sent as an optional part of the Styled Layer Descriptor (SLD)). In addition, the Map and Diagram Service is able to create maps and diagrams based on data provided by an offering of the Sensor Observation Service (see section 8.2.2). The main output of this service is an image document. The image document can be a map (visualization of geographic information and spatially referenced data), a diagram (visualization of statistical data) or a legend of a portrayed layer. Optionally, this service offers the possibility of querying information about the features portrayed in an image document or getting a comprehensive layer description (including count, minimum, maximum, sum, mean, standard deviation, and histogram for every attribute when appropriate). The output of these operations is an information document containing the requested information (when available).

	The maps and diagrams represent a visualisation of the data. These data are generally rendered in a pictorial format such as PNG (recommended as default format), GIF or JPEG. Portrayal in vector-based graphical elements as Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) is also allowed if it is acceptable for the service provider. However, it is usually preferred that the original data cannot be reconstructed from the portrayal. The Map and Diagram Service provides its functionality through the following interfaces: • *ServiceCapabilities:* Provides information about both common and specific capabilities. • *MapDiagram:* Allows a client to request and receive maps, diagrams and, optionally, information about the visualized features according to specifications, as well as to both put data and styles on the server for visualization and to remove them.
Interface *ServiceCapabilities*	
getCapabilities	Informs the requestor about both common and specific capabilities of a Map and Diagram Service.
Interface *MapDiagram*	
getMap	Returns a map of spatially referenced geographic and thematic information as an image document with the characteristics specified by the client application. The characteristics of the output image are specified by the *outputAttributes* parameter (image format, width, height, transparency, etc.) as well as the *mapAttributes* parameter (list of layers and their corresponding styles, coordinate reference system, global bounding box). Optionally, the map parameters can be provided using an SLD document.
getDiagram	Returns a diagrammatic representation of numerical data as an image document with the characteristics specified by the client application. The characteristics of the output image are specified by the *outputAttributes* parameter (image format, width, height, transparency, etc.) as well as the *diagramAttributes* parameter (list of tabular data layers and their corresponding styles – diagram type, diagram characteristics). Optionally, the diagram parameters can be provided using an SLD document. This operation expects that the data to be rendered are in tabular format.
getLayerDescription	Returns a layer description document containing schema information for a layer: attribute names, types, units, statistical information when applicable (like value ranges, max, min etc.). This information is needed by clients in order to create their own styles and symbolization rules based on attribute values.
getLayerLegend	Returns a legend symbol (corresponding to a layer) as an image document with the characteristics specified by the client application. The characteristics of the output image are specified by the *outputAttributes* parameter (image format, width, height, transparency, etc.) as well as the *styledLayer* parameter (name of the layer for which the legend should be generated and its corresponding styles). If the

	styles corresponding to the layer are not available on the server, then the styles have to be defined and sent again by the client (optionally, also as a SLD document).
getFeatureInfo	Returns information about the features rendered in a certain point of a map or diagram layer as a document. The request must specify the attributes of the query point (x and y coordinates of the point in the image coordinate system, the layer name, and the number of features for which is expected to receive information) as well as a copy of the request that generated the image.
getStyle	Returns the style of a hosted layer. The operation confirms the success of the request by returning a Boolean "TRUE".
Example Usage	A requestor accessing this service wants to create a map that shows the spatial distribution of information provided through a SANY sensor service network. As a map background the requestor wants to have the shaded relief, the road network, the hydrological network, and the urban areas. The measurements coming from sensors are to be displayed either as a diagram layer with bar charts or to be interpolated and classified within a choropleth. For this purpose the necessary background layers are either hosted on the server or are accessible by means of a Feature Access Service/OGC Web Map Service or OGC Web Feature Service. The requestor now invokes a *getMap* operation by passing a styled layer descriptor document which defines the location of the data (for sensor data an appropriate SOS offering) and the symbolization corresponding for each layer. The response of the service will be a map provided in the requested format.
Comments	It is beyond of the scope of this service to provide a human interface like the geographic viewer in human interaction services. Other map service instances, a geographic viewer or even a web browser could act as a client to this service.

Table 8-14: Description of the Map and Diagram Service

8.5. Event Based Interaction Services

This section gives an overview of the SensorSA services that enable event-based interaction between sensors, services, and clients. They rely upon the base and the brokered variant of the OASIS WS-Notification standards, see sections 8.5.1 and 8.5.2. Table 8-15 provides a comparison between these OASIS standards and the functionalities of the OGC Sensor Alert Service as described in section 8.2.4.

In addition, the SensorSA supports the OGC Web Notification Service (WNS) as presented in section 8.2.5. In contrast to the OASIS WS-Notification and the OGC SAS the WNS is a simple protocol transducer. The WNS supports the registration of identities (including their name, e-mail address, phone or fax number, etc.) and their notification by means of the notify operation (which is an HTTP-based message). The notify operation uses the notification means that is associated to the identity, i.e. it may send e-mails, short message services, facsimiles and so on. Note that the OGC WNS may be used in combination with both the OGC SAS and the OASIS WS-N in order to abstract from the concrete notification mechanism.

Criteria	OASIS WS-Notification	OGC Sensor Alert Service
Publish interaction	Yes	No
Subscribe-Publish interaction	Yes	Yes
Register-Publish interaction (Advertise-Publish)	Yes	Yes
Brokered interaction	Yes	Yes
Transport	SOAP	XMPP / HTTP
Hierarchical organisation of event types	Yes	No
Event Filtering	Based on topic (type) only	Yes / complex based on notification content (eg. spatial)
Provides description of the notification payload	Not by the means of the specification	Yes
Information model and schema for the payload	No	Yes

Table 8-15: Comparison between OASIS WS-Notification and the OGC Sensor Alert Service

8.5.1 Interfaces of WS-Base Notification Specification

Name	WS-BaseNotification
Standard Specifications	OASIS Web Services Base Notification 1.3 (WS-BaseNotification) Committee Specification, 31 July 2006. http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-cs-01.pdf
Description	OASIS describes the WS-BaseNotification as follows: "The WS-Notification family of specifications defines a standard Web services approach to notification. The WS-BaseNotification specification is the base specification on which all the other specifications in the family depend. It defines the normative Web services interfaces for two of the important roles in the notification pattern, namely the NotificationProducer and NotificationConsumer roles. This specification includes standard message exchanges to be implemented by service providers that wish to act in these roles, along with operational requirements expected of them. In the Notification pattern a Web service, or other entity, disseminates information to a set of other Web services, without having to have prior knowledge of these other Web services. This specification defines a role called the NotificationProducer. A NotificationProducer is capable of producing a set of Notification messages. A NotificationProducer accepts incoming Subscribe requests. Each Subscribe request contains a reference to a NotificationConsumer

	and identifies the subset of the Notifications the NotificationProducer should produce. This subset can be described by identifying one or more boolean filters, including filtering by Topic, as described in [WS-Topics]. The NotificationProducer agrees to produce Notification Messages as requested in the Subscribe request, or returns a fault if the subscription cannot be handled. In addition to the message exchanges described in this specification, a NotificationProducer may also support the required message exchanges defined in the [WS-ResourceProperties] specification and may support the optional message exchanges defined in the WS-ResourceProperties specification. In such a case, this specification defines several resource properties which MUST conform to the schema defined in the WS-BaseNotification specification." Interfaces of the WS-BaseNotification specification: • *NotificationConsumer:* Enables reception of notifications produced by a NotificationProducer as a result of a subscription. • *NotificationProducer:* Enables production of notifications to those NotificationConsumers that have subscribed based on occurring events and on the supplied parameters during subscription. • *PullPoint:* Supports destruction of PullPoint resources and retrieval of notifications from a PullPoint resource. • *CreatePullPoint:* Manages creation of PullPoint resources. • *SubscriptionManager:* Manages renewal and cancelation of Notification subscriptions. • *PausableSubscriptionManager:* Enables pausing and resuming production of notifications for a given subscription.
Interface *NotificationConsumer*	
Notify	This action is implemented by the consumer and invoked by the producer (produces Notification message) when publishing Notifications.
Interface *NotificationProducer*	
Subscribe	Used by the notification subscriber to register its interest to receive a subset of the notifications that the producer can publish.
GetCurrentMessage	Upon invocation of this operation the NotificationProducer returns the last published Notification on a given topic.
Interface *PullPoint*	
GetMessages	The PullPoint interface supports the NotificationConsumer interface in order to allow accumulation of Notifications by enabling the NotificationProducer to send notifications to the PullPoint. The GetMessages operation enables the requestors to retrieve (or pull) Notification Messages from the PullPoint.
DestroyPullPoint	Invoked by a requestor in order to destroy an existing PullPoint.
Interface *CreatePullPoint*	

CreatePullPoint	This operation is invoked by a requestor on an endpoint supporting the PullPoint interface in order to create a new PullPoint resource.
Interface *SubscriptionManager*	
Renew	The Renew operation is used to modify lifetime of an existing Subscription.
Unsubscribe	This operation is invoked by a requestor in order to terminate an existing Subscription.
Interface *PausableSubscriptionManager*	
PauseSubscription	This operation is invoked in order to temporarily pause the production of notifications for a given Subscription.
ResumeSubscription	This operation is the counterpart of the PauseSubscription operation being issued in order to resume the production of notifications for a given Subscription.
Example Usage	Event based automatic catalogue harvesting can be realised by implementing the NotificationProducer and NotificationConsumer interfaces. Given the dynamic aspects of an existing sensor network, the Catalogue responsible for the discovery of resources must be able to react to changes in the sensor network (e.g. new sensor connected to the network). This can be realised by a service of the sensor network implementing the NotificationProducer interface and providing subscriptions to notifications concerning new sensors. Following the Catalogue has to implement the NotificationConsumer interface and subscribe to these notifications. Based on the received Notifications the catalogue service can start harvesting for information concerning the sensor network and updating its contents.
Comments	It is beyond the scope of this specification to describe an information model for or taxonomy for events. It relies on the WS-Topics specification that enables the definition of event/notification types and categories that a consumer can subscribe to. Notification producer might support the message patterns defined by the WS-ResourceProprieties specification.

Table 8-16: Description of WS-BaseNotification Service

8.5.2 Interfaces of WS-Brokered Notification Specification

Name	WS-BrokeredNotification
Standard Specifications	Web Services Brokered Notification 1.3 (WS-BrokeredNotification) OASIS Standard, 1 October 2006. http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf
Description	OASIS describes the WS-BrokeredNotification as follows: "The Event-driven, or Notification-based, interaction pattern is a commonly used pattern for inter-object communications. Examples exist in many domains, for example, in publish/subscribe systems or in system and device management domains. Message brokers are involved in many of these systems, such as the ones provided by Message Oriented Middleware vendors.

	This specification defines the Web services interface for the NotificationBroker. A NotificationBroker is an intermediary between message Publishers and message Subscribers. A NotificationBroker decouples NotificationProducers and Notification Consumers and can provide advanced messaging features such as demand-based publishing and load-balancing. A NotificationBroker also allows publication of messages from entities that are not themselves service providers. This is very similar to a traditional Message Oriented Middleware model. The NotificationBroker interface includes standard message exchanges to be implemented by NotificationBroker service providers along with operational requirements expected of service providers and requestors that participate in brokered notifications. In addition to the message exchanges described in this specification, a NotificationBroker my also support the required message exchanges defined in the WS-ResourceProperties specification and may support the optional message exchanges defined in the WS-ResourceProperties specification." Interfaces of the WS-BrokeredNotification specification: • *NotificationBroker:* Enables the implementing service to act as a middleware between Publishers and Subscribers of Notifications. A service implementing this interface must implement the following interfaces from the WS-BaseNotification specification: NotificaitonConsumer, NotificationProducer. It may also implement the CreatePullPoint interface. In addition the following interfaces must be implemented. • *RegisterPublisher:* Handles the publisher registration. • *PublisherRegistrationManager*: Handles the termination of a Publisher registration (unregistration).
Interface *RegisterPublisher*	
RegisterPublisher	This action is implemented by the broker in order to allow for to Publishers to advertise on their ability to publish Notifications on a set of Topics.
Interface *PublisherRegistrationManager*	
DestroyRegistration	Used to destroy the PublisherRegistration resource thus effectively terminating the publishing of Topics from a Publisher. Additionally the PublisherRegistrationManager may also support the required message exchanges defined in the WS-ResourceProperties specification.
Example Usage	A NotificationProducer registers with a Broker and by doing so the Broker exposes the Topics that the Publisher is able to provide. Upon successful registration a NotificationConsumer can subscribes to the Broker to receive notifications on a given Topic. The Broker mediates this subscription by subscribing for Notifications at the NotificationProducer and forwarding the received Notifications to the NotificationConsumer.
Comments	

Table 8-17: Description of WS-BrokeredNotification Service

9. Technology Viewpoint

The Technology Viewpoint of the SensorSA specifies the technological choices of the concrete service platform and its operational issues. To accommodate the requirements of Sensor Networks as introduced in section 4.5 the SensorSA refines the guidelines and requirements for platform specifications as defined in the Technology Viewpoint of the ORCHESTRA Reference Model (RM-OA, 2007). These guidelines comprise:

- a general approach for how to specify a service platform (see section 9.1),
- the specification of the SANY service platform
- a description of how access control mechanisms are being implemented (section 9.3.1),
- an agreement on data formats (see section 9.3.2), and
- optionally a set of restrictions to be observed for a particular platform.

9.1. Properties of a Service Platform

As a general guideline, the specification of a service platform shall be conformant to the OASIS Reference Model for Service Oriented Architecture 1.0 (SOA-RM, 2006). This implies that the platform is being described according to the SOA-RM by the following predefined platform properties:

- Platform Name

  Name of the platform and if applicable the exact version number of the platform specification.
  In the case of a standard platform, a reference shall be provided.

- Reference Model

  If the platform specification is based on a specific reference model, the name and the exact version number of the reference model shall be provided.

- Interface Language

  Specification of the formal machine-processable language used to define SOA-RM Service Interfaces. In the case of a standard language, a reference shall be provided.

- Execution Context

  Specification of the SOA-RM Execution Context. The Execution context is an agreement between service providers and consumers. It contains information that can include preferred protocols, semantics, policies and other conditions and assumptions that describe how a service can and may be used. This includes, for example, the specification of the transport and the security layer, the format of the messages exchanged between service providers and consumers, etc. In the case of a standard SOA-RM Execution Context, a reference shall be provided.

- Schema Language

  Specification of the schema language used to define SOA-RM Information Models.

- Schema Mapping

  Specification of how to map the abstract level (UML) to the schema language used for this particular platform.

- Information Model Constraints

  Specification of the constraints on the SOA-RM Information Model, especially the constraints on the message format which is required to accomplish the SOA-RM Action model.

9.2. The SensorSA Service Platform

The SensorSA service platform has to consider multi-platform aspects like disparate protocol bindings and request and response schemas, without either jeopardizing service interoperability or putting an unnecessary burden on client and service developers. This is achieved by separating the platform specification into a core mandatory part and one or more extended optional parts as illustrated in Figure 9-1.

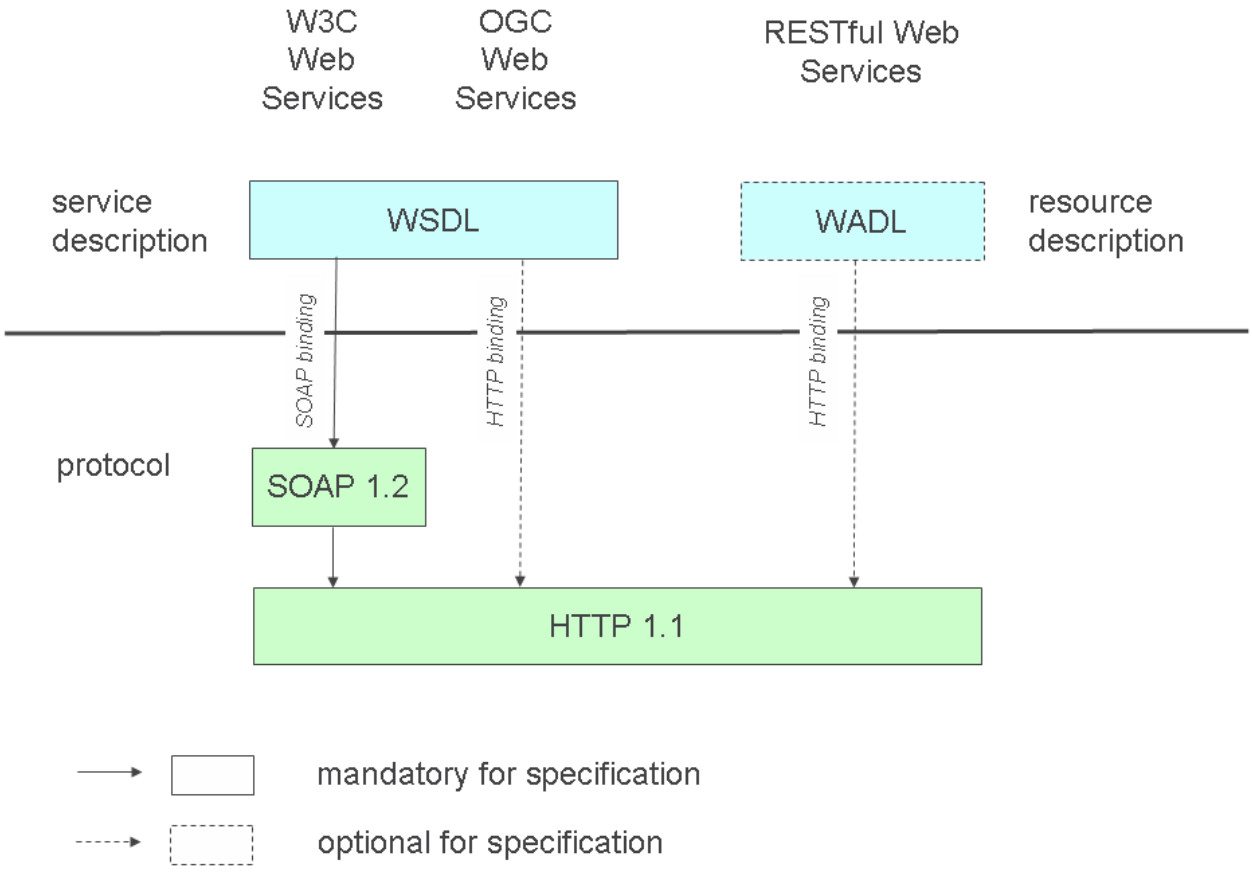


Figure 9-1: Structure of the SensorSA Service Platform

The core mandatory part specifies only the transport protocol and the common interface description language. The actual protocol binding and the request-response schemas are selected by a separate platform specification. This core part in conjunction with a respective platform specification is obligatory for the specification and development of SensorSA services.

The extended part may specify a different combination of protocol bindings and request-response schemas and should also use the common interface description language. Furthermore, it should provide a formal mapping to other protocol bindings and request-response schemas in order to make a protocol transformation possible, ideally automatically. This multi-platform approach will facilitate the reuse and integration of existing software components and the evaluation of other service paradigms.

The different platforms currently taken into consideration are based on the following service paradigms:

- W3C Web Services (section 9.2.1)
- OGC Web Services (section 9.2.2)
- RESTful Web Services (section 9.2.3)

This leads to the following possible options regarding the request and response schema and the protocol bindings. The transport protocol is always HTTP, as defined in the core mandatory part of the platform specifications.

Topic	Options
Transport	HTTP
Request	• KVP (Key Value Pair) • XML (plain XML or SOAP Messages)
Response	• HTML • XML (plain XML or SOAP Messages) • Binary (any MIME Type)
Protocol binding	• HTTP GET, POST, PUT and DELETE • SOAP (HTTP POST)

Table 9-1: Options for the SensorSA Service Platform

Although not all permutations of protocol bindings and request and response types are suitable (e.g. HTTP GET and SOAP), it should be obvious that the development of services and clients which support all of the possible options by default is highly impractical. In consequence, a recommendation for a default platform shall be given, which has to define the most reasonable combination of protocol binding and request and response schema.

Considering the different options in terms of interoperability and keeping in mind the desire for automatic protocol transformation the default platform for the specification of SensorSA services shall be based on the W3C Web Services Architecture. For compatibility

reasons, extensions to existing OGC SWE services shall be specified according to the OGC Web Services Platform.

9.2.1 Specification of the SensorSA W3C Web Services Platform

The SensorSA W3C Web Services Platform is an instance of the W3C Web Services Architecture (W3C, 2004). It comprises of a well-defined selection of standards and specifications related to and defined by the W3C Web Services Architecture. It allows the specification of W3C Web Services in relation to the requirements of the SensorSA.

W3C Web Services offer the following options regarding the transport protocol, the request and response schema and the protocol bindings:

Topic	Options
Transport	HTTP
Request	XML (SOAP Message)
Response	XML (SOAP Message)
Protocol binding	SOAP (HTTP POST)

Table 9-2: Options for the SensorSA W3C Web Service Platform

Since the KVP encoding used by OGC Web Services and also to a certain extent by RESTful Web Services may be encoded in XML and wrapped by a SOAP message, the SensorSA W3C Web Services Platform is the premier choice for the obligatory core platform. The SensorSA W3C Web Services Platform is characterized by the following SOA-RM properties:

- Platform Name

  The name of the platform is “SensorSA W3C Web Services Platform” following the Web Service infrastructure as defined by the W3C specifications (W3C, 2004).

- Reference Model

  The SensorSA W3C Web Services Platform is based on the W3C Web Services Architecture (W3C, 2004).

- Interface Language

  The formal language that is used to define the SOA-RM Service Interfaces is the Web Service Description Language (WSDL), Version 1.1 (W3C, 2001).

  Note: If required and supported by the tools used, WSDL 2.0 (W3C, 2007) may also be used.

- Execution Context

  The execution context of the SensorSA W3C Web Services Platform is defined by the

following properties:

- Transport Protocol and Message Format:

  SOAP 1.2 HTTP binding as defined in *SOAP Part 1: Message Framework, Version 1.2* (W3C, 2003) and *Hypertext Transfer Protocol (HTTP), Version 1.1* (W3C, 2006). The message style that shall be used is document/literal non-wrapped since it is the most widely accepted and interoperable message style.

- Security

  The common security aspects of the different SensorSA Service Platforms are discussed in section 9.3.1. The following aspects, however, are specific to the SensorSA W3C Web Services Platform:

  Session Information: The transport of session information may be accomplished by using platform specific mechanisms, such as the inclusion of a session key in the SOAP header.

  Encryption: Optional encryption of SOAP messages shall be accomplished by Web Services Security: 4 SOAP Message Security 1.1 (OASIS, 2006).

- Schema Language

  The general schema language used to define the SOA-RM Information Models is the eXtensible Markup Language (XML) 1.0 (XML, 2006). Section 9.3.2 will list further XML-based schema and modelling languages.

- Information Model Constraints

  There are currently no immediate constraints on information models themselves.

9.2.2 Specification of the SensorSA OGC Web Services Platform

The SensorSA OGC Web Services Platform is specified on the basis of the OpenGIS® Web Service Common Implementation Specification (OGC, 2007). The OGC Common Specification "specifies many of the aspects that are, or should be, common to all or multiple OGC Web Service (OWS) interface Implementation Specifications" (OGC, 2007). The SensorSA OGC Web Services Platform adopts the most general aspects of the OWS Common Specification and extends it by SensorSA specific aspects to ensure compatibility with existing OGC Web Services and facilitate the specification of new SensorSA services.

OGC Web Services offer the following options regarding the transport protocol, the request and response schema and the protocol bindings:

Topic	Options
Transport	HTTP
Request	• KVP (Key Value Pair) • XML (plain XM)
Response	• HTML

	• XML (plain XML) • Binary (any MIME Type)
Protocol binding	HTTP GET and POST

Table 9-3: Options for the SensorSA OGC Web Service Platform

The SensorSA OGC Web Services Platform is characterized by the following properties:

- Platform Name

  The name of the platform is "SensorSA OGC Web Services Platform".

- Reference Model

  The SensorSA OGC Web Services Platform is based on the OpenGIS® Web Service Common Implementation Specification (OGC, 2007).

- Interface Language

  The formal language that is used to define the SOA-RM Service Interfaces is the Web Service Description Language (WSDL), Version 1.1 (W3C, 2001).

  Note: If required and supported by the tools used, WSDL 2.0 (W3C, 2007) may also be used.

- Execution Context

  The execution context of the SensorSA OGC Web Services Platform is defined by the following properties:

  - Transport Protocol and Message Format:

    Operations are invoked by HTTP requests. In the case of HTTP POST the requests are XML-encoded, whereas in the case of HTTP KVP encoding of parameters shall be used. The response shall be either an XML document or a binary document (e.g. an image). In any case the format of the response has to be made transparent to the requestor, for example in the interface description or in a capabilities document of the service. An XML response shall be described by a corresponding XML-Schema, and a binary response by a MIME-Type (e.g. image/png). The complete rules are defined in the chapter entitled "Operation request and response encoding" of the *OGC Common Implementation Specification* (OGC, 2007).

  - Security

    The common security aspects of the different SensorSA Service Platforms are discussed in section 9.3.1. The following aspects, however, are specific to the SensorSA OGC Web Services Platform:

    Encryption: Optional transport-layer encryption of HTTP requests and responses shall be accomplished by *SSL 3.0* (Netscape, 1996).

- Schema Language

  The general schema language used to define the SOA-RM Information Models is the

eXtensible Markup Language (XML) 1.0 (XML, 2006). Section 9.3.2 will list further XML-based schema and modelling languages.

- Information Model Constraints

  There are currently no immediate constraints on information models themselves.

9.2.3 Specification of the SensorSA RESTful Web Services Platform

The SensorSA RESTful Web Services Platform is based on architectural principles introduced in Architectural Styles and the Design of Network-based Software Architectures (Fielding, T.R., 2000). Although similar to the SensorSA OGC Web Services Platform it further defines several constraints on the specification of service interfaces.

RESTful Web Services offer the following options regarding the transport protocol, the request and response schema, and the protocol bindings:

Topic	Options
Transport	HTTP
Request	• KVP (Key Value Pair) • XML (plain XM)
Response	• HTML • XML (plain XML) • Binary (any MIME Type)
Protocol binding	HTTP OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE and CONNECT

Table 9-4: Options for the SensorSA RESTful Web Service Platform

The SensorSA RESTful Web Services Platform is characterized by the following properties:

- Platform Name

  The name of the platform is “SensorSA RESTful Web Services Platform”.

- Reference Model

  The SensorSA RESTful Web Services Platform is based on the architectural principles introduced in Architectural Styles and the Design of Network-based Software Architectures (Fielding, T.R., 2000)

- Interface Language

  The formal language that may used to define the SOA-RM Service Interfaces is either the resource model as defined in section 7.6 or the Web Application Description Language (WADL) (Hadley, M. J., 2006) which has been specifically developed for the description

of RESTful Web Services.

- Execution Context

  The execution context of the SensorSA RESTful Web Services Platform is defined by the following properties:

  - Transport Protocol and Message Format:

    The HTTP POST, GET, PUT and DELETE methods are used to perform generic create, read, update and delete (CRUD) operations on resources. Resources are addressed and uniquely identified using uniform resource identifiers (URI). A POST, PUT or UPDATE request is typically XML-encoded. In the case of a HTTP GET or DELETE request KVP encoding of parameters shall be used. The response shall either be an HTML, XML or binary document (for example an image) or a string representing the URI of the resource upon which the operation has been performed. For example, in the case of a PUT request the URI of a newly created resource shall be returned. In any case the format of the response has to be made transparent to the requestor, for example with an interface description or in the capabilities document of the service. An XML response shall be described by a corresponding XML-Schema (e.g. the SensorML schema), a binary response by a MIME-Type (e.g. image/png).

  - Security

    The common security aspects of the different SensorSA Service Platforms are discussed in section 9.3.1. The following aspects, however, are specific to the SensorSA W3C Web Services Platform:

    Encryption: Optional transport-layer encryption of HTTP requests and responses shall be accomplished by *SSL 3.0* (Netscape, 1996).

- Schema Language

  The general schema language used to define the SOA-RM Information Models is the eXtensible Markup Language (XML) 1.0 (XML, 2006). Section 9.3.2 will list further XML-based schema and modelling languages.

- Information Model Constraints

  There are currently no immediate constraints on information models themselves.

9.3. Specification of Further Platform Properties

9.3.1 Selection of User Management, Authentication and Authorisation Mechanisms

This topic comprises the specification of how security and access control mechanisms are intrinsically supported by the platform. The current draft of the SensorSA security model implies that there could be a co-existence of different access control solutions. The platform

specification has to define whether the use of disparate implementations of security services and disparate communication channels for security related information are permitted.

9.3.2 Agreement on Data Formats and Application Schemas

This topic comprises the agreement on the usage of specific data formats (e.g. non-GML representation of coverages). There are currently no extensions required.

.

10. Engineering Viewpoint

10.1. Overview

Based on the major concepts of the SensorSA, the specification of information models and services in the Information and Service Viewpoint of a SensorSA, the following sections provide definitions of policies for the set-up and operation of sensor service networks.

Policies are defined for the following aspects:

- resource discovery (see section 10.2)
- sensor and service monitoring (see section 10.3)
- sensor planning (see section 10.4)
- access control (see section 10.5)
- processing of quality information (see section 10.6)
- handling of large data sets (see section 10.7)
- cascading sensor observation services (see section 10.8)
- processing and fusion support (see section 10.9)
- integration of mobile sensors (see section 10.10)
- event handling (see section 10.11)
- plug-and-measure support (see section 10.12)

Here, the SensorSA follows the basic idea of the (RM-OA, 2007) to consider qualifying characteristics of a service network in terms of policies.

10.2. Resource Discovery Policy

10.2.1 Introduction

The process of resource discovery may be carried out in multiple ways. However, for a given service network it has to be specified in detail in order to enable interoperability.

If not otherwise specified, a sensor service network is qualified as a “mediated service network” and follows the policy of a “centralised discovery”. According to (RM-OA, 2007) this means that there shall be a distinguished instance of a catalogue service (in the following simply called the “SANY Catalogue” or simply “the catalogue”) that serves as the discovery entry point to a sensor service network. The meta-information schema of the SANY catalogue is specified in section 7.6.3.

In the following, the service interactions for resource discovery (i.e. querying of the catalogue and underlying services) and for resource registration (i.e. creating and updating of catalogue entries) are specified in more detail as illustrative examples.

10.2.2 Query Models

The queries may be executed in one step or may be broken down into several queries. The first query is always sent to the SANY Catalogue. The sequence of queries may follow one of the following two basic query models:

- query chaining

  In the case of query chaining, queries are executed in different steps that are controlled by the resource requestor. Each step reduces the result set of possible resources. Between two steps the resource requester may process the result of the previous step and decide how to continue with the next step. The result from the previous query is typically passed as an argument (a condition) for the next query. The queries in the different steps may be sent to the resource brokers of the same type (homogeneous queries) or to resource brokers of different types (heterogeneous query). Typically a service has the role of a resource provider for the previous step and acts as a resource broker for the next step.

  Example: The first query returns a set of SOS instances, and the next step uses the meta-information of the SOS by calling *getCapabilities* to query for resources within the SOS.

  Note: Query chaining may be implemented for a predefined application purpose using service chaining if no user interaction is needed.

- query cascading

  In the case of query cascading queries are broken down into sub-queries that are individually sent to one or more other resource brokers of the same or of different types, after which the results are assembled into one result set. In contrast to query chaining, query cascading is transparent to the resource requester. Instead of having all meta-information available to process a query, the resource broker relies on meta-information entries of additional resource brokers

10.2.3 Typical resource discovery policies

Resource discovery in the SensorSA is supported by a combined usage of the SANY Catalogue that provides the interfaces of the Catalogue Service (see section 8.2) and instances of other SANY service types, especially the Sensor Observation Service (SOS) (see section 8.2.2) if observation and related observation attributes have to be discovered. Depending on the search target the following typical query types may be distinguished from a user's point of view:

- search for "features of interest (FOI)"
  - all FOI with a specific set of observable properties in a specific area
- search for "observations"

  - all observations related to FOI
  - all observations for specific observable properties in a specific area and time range
  - all observations for specific observable properties in a specific area and time range produced by a call of procedures
  - all observations whose attributes fulfil specific conditions (for example quality)
- search for “procedures”
  - all procedures (e.g. sensor types, models) in a specific area
  - all procedures that have specific properties described by their SensorML document
- search for “services”
  - all services that deal with a procedure

As examples, the following sequence diagrams show the typical chain of service operation calls for two selected query types:

1. The search target for the first sequence is an observation according to the Observation & Measurements model (see section 7.2). To keep this example simple it is assumed that the user wants to find observations without using procedure information as selection criteria.
2. The second sequence shows how to discover procedures. The user wants to discover a procedure that is similar or identical to a known procedure.

10.2.3.1 Discovery of Observations

The first sequence diagram (Figure 10-1) shows how a user may discover observations and retrieve their attribute values. It is divided into three parts:

1. The user starts to discover a Feature of Interest (FOI) within an area by sending a search request to the catalogue using a bounding box as a spatial condition. Additional conditions, such as the type of the FOI may be added here. The catalogue returns a list of feature instances that fulfil the search conditions. This list contains the IDs of the feature instances together with a set of core attributes (e.g. according to the Dublin core schema). If additional attributes are required, the user can retrieve the complete meta-information entry that is available in the catalogue by issuing a *getMetaInformation* operation request for a selected FOI instance. Finally, the user decides which FOI(s) they want to move further along in the discovery process.

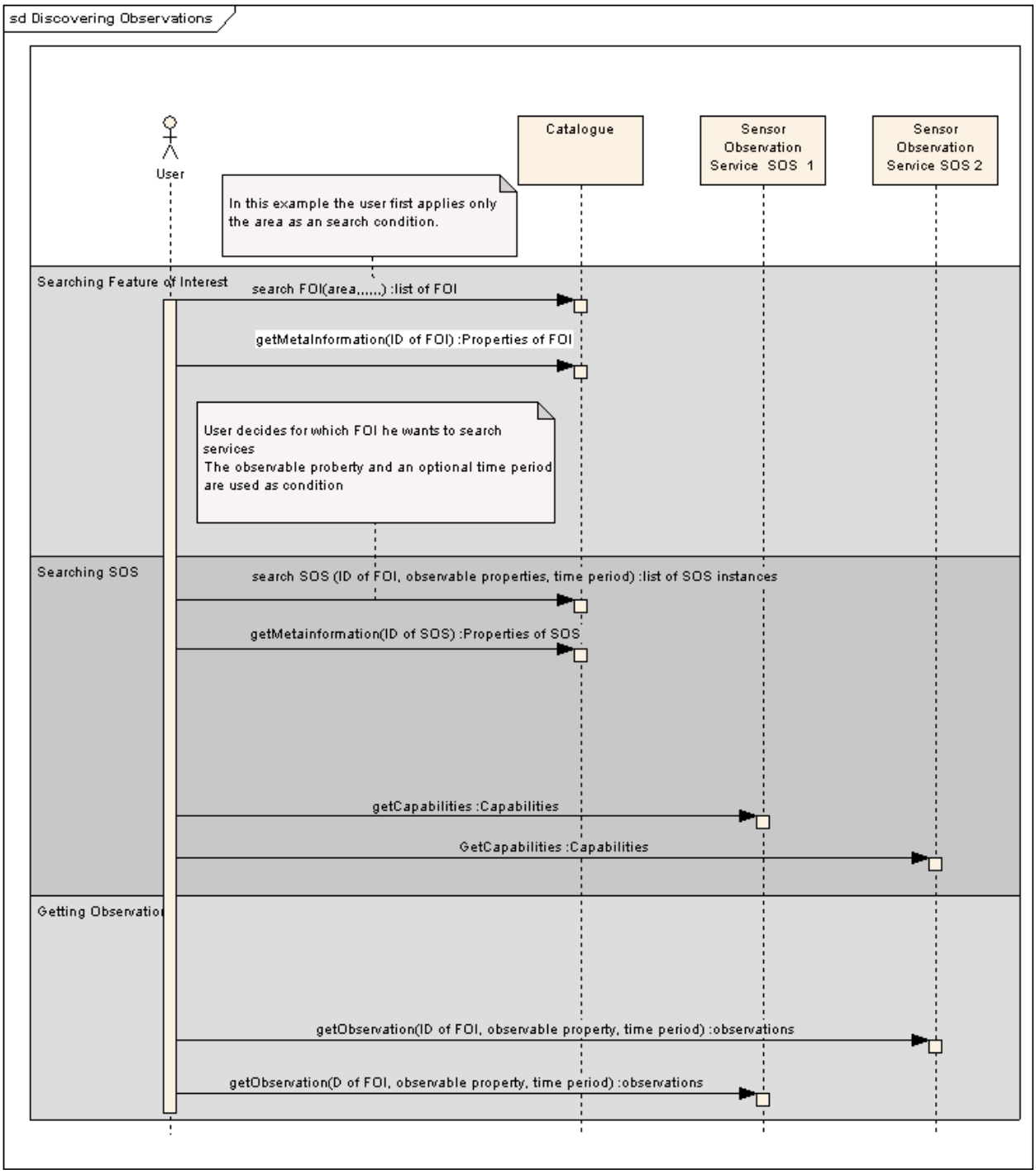


Figure 10-1: Discovery of Observations

2. The user continues by searching in the catalogue for the SOS instance that provides observations for the selected FOI(s). Typically they add one or more observable properties and a time range, for which the SOS should provide observations as an additional condition. The user sends a search request to the catalogue to find an SOS service instance and retrieves a list of SOS instances that meet the search conditions. Again the user can get additional meta-information about the services instances by issuing a *getMetaInformation* operation request to the catalogue. If the meta-information entry of the catalogue doesn't contain the full set of capabilities provided by an SOS

instance or if the catalogue information is not sufficiently current for the user's purpose, they may retrieve the capabilities directly from the SOS instance by issuing a *getCapabilities* operation request to the SOS instance. The capabilities document of the SOS instance usually provides more detailed information about the service such as possible result models and the procedures used to get the values of an observation. Based on this information the user now selects one or more SOS instances from which they want to get observations.

3. The user gets the observations by issuing a *getObservation* operation request to the SOS instance. It is important to note that the user does not get the observations from the catalogue. The user finally decides which observations will be used in the application. Possible criteria for this decision are contained in the attributes of the observation, such as quality attributes.

Depending on prior knowledge the user may skip parts of the sequence. As an example, the user may directly start with step 2, the search for an SOS instance, without having previously searched an FOI in step 1. In this case the user may replace the ID of the FOI with a condition for the spatial context in the search SOS request to the catalogue.

10.2.3.2 Discovery of Procedures

The sequence diagram in Figure 10-2 explains the discovery of procedures. For this scenario, it is assumed that the user is already using a selected SOS instance to get observations. Now, the user wants to find procedures that are similar or identical to the one that produces the value for an observation. The user then decides to configure this procedure using the Sensor Planning Service (SPS) (see section 8.2.3).

This scenario may be realised as follows:

1. The user gets detailed information about the procedures available in a given SOS service instance by invoking *getCapabilites* in order to get the information about the involved procedures. By means of the *describeSensor* operation of the SOS the user then retrieves information about the procedure, here in the form of SensorML documents. These documents may then be parsed in order to retrieve the required information subset.

2. The user provides parts of this information as search condition to the catalogue to retrieve a list of (similar or identical) procedures that are known by the catalogue. Examples are the type of the procedure (e.g. image-delivering sensors), the types of the output values produced by the procedure, or a spatial condition (e.g. all sensors located within a circle of 10km of the currently used sensor).

3. After having selected one or more procedures the user searches for an instance of the Sensor Planning Service in the catalogue and finally uses this service to configure the procedure by calling the *describeTasking* and *submit* operations of the selected SPS.

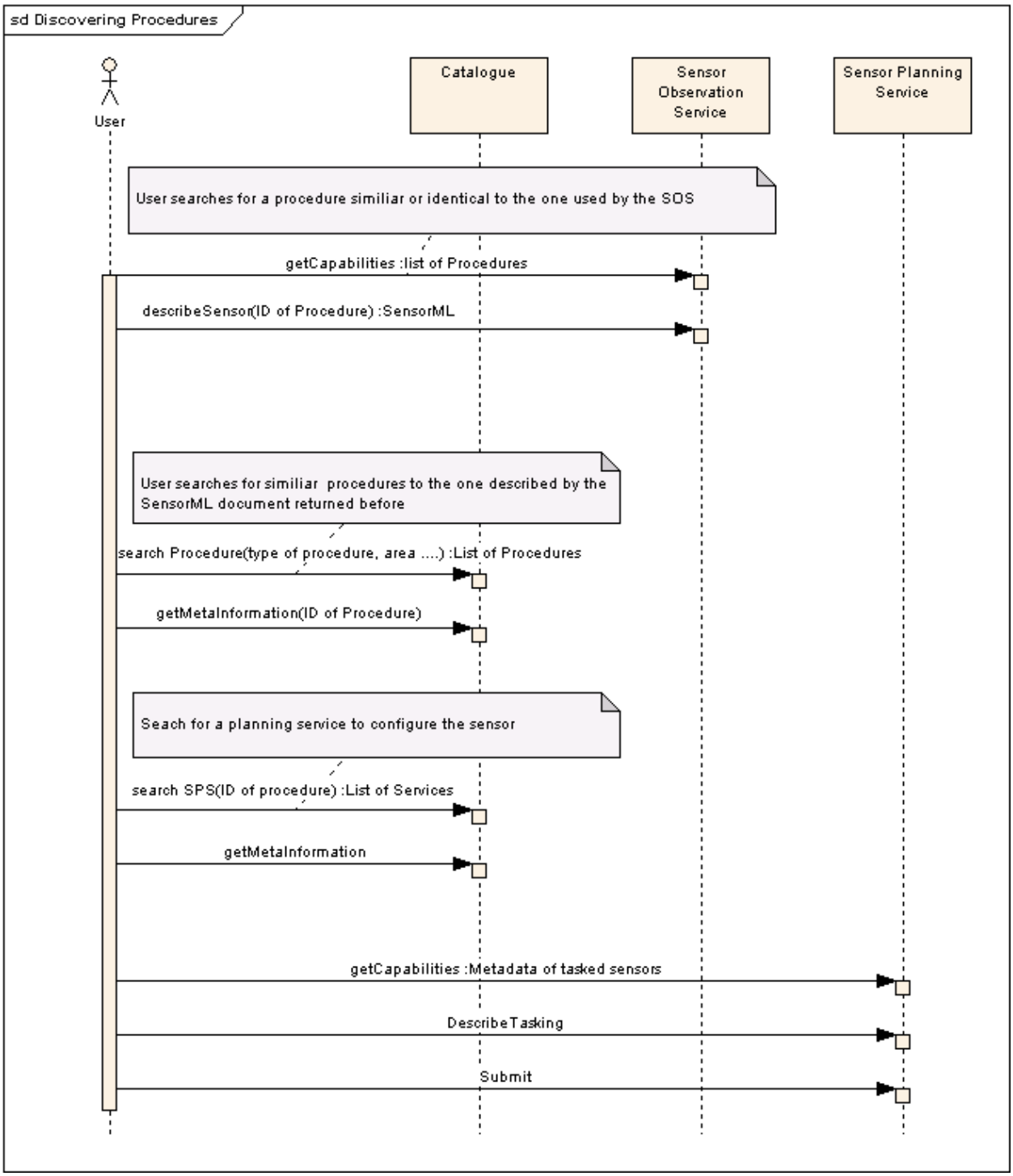


Figure 10-2: Discovery of Procedures

10.2.4 Harvesting of SOS Capabilities

This section describes how meta-information can be automatically created from domain specific resources and stored into a SensorSA catalogue. This procedure is usually called **harvesting**. As an example, the Austrian Federal Environment Agency (Umweltbundesamt) needs to provide INSPIRE-compliant meta-information for their air quality data resources in accordance with the CAFE Directive (CAFÉ, 2008). This data is provided via an Sensor Observation Service (SOS)

(section 8.2.2). The goal is to reuse the capabilities provided by the SOS for the creation and publication of INSPIRE related meta-information and store them in an OGC Catalogue Services supporting the ISO 19115 and ISO19119 metadata schema in order to fulfill the INSPIRE duties. Such a catalogue service is, for instance, embedded into the prototypical INSPIRE geo-portal[1].

Besides the INSPIRE requirements there is the need for additional meta-information elements:

- CAFE is concerned with air quality data. For a CAFE meta-information document, specific validation information is needed, such that users of the meta-information can precisely decide to which level they can trust the resource data. The means CAFE requires additional information regarding data quality, including quality assurance levels and completeness of measurements. The required validation information exceeds the basic description dictated by INSPIRE.

- The SensorSA formulates the need for the realization of information discovery in the sensor related domain as SensorSA users search for services, sensors, features of interest and observable properties. To support these specific requirements the SensorSA has extended the INSPIRE and ISO 19115/19119 meta-information schemata as described in section 7.7.

In the following it is described how the meta-information elements of the SOS services that may be retrieved through the operations *GetCapabilities* and *DescribeSensor* can be mapped to INSPIRE metadata resources:

- The SOS service itself can be described in a metadata document describing a service.

- Data provided by the Umweltbundesamt via the SOS needs to be described in metadata documents describing datasets. Each offering of the SOS is interpreted as a single dataset.

- Common resource information like title, abstract and resource location can be found in the capabilities of the SOS.

- The Geographic location and temporal extent can be provided in the descriptions of the SOS offerings in the capabilities of the SOS.

- Access constraints can be provided in the SOS capabilities.

- A responsible party can be described in the SOS capabilities.

- Quality and validity information can be described via the means of SensorML and therefore included into responses of the *DescribeSensor* operation.

The gathered metadata is used to create ISO 19115 (for datasets) or ISO 19119 (for services) documents, following the INSPIRE Technical Guidelines. ISO 19115 and ISO 19119 contain some mandatory metadata elements, which are not mandatory for INSPIRE. These elements also need to be included into the resulting metadata documents. This results in metadata documents conformant to both: INSPIRE and ISO.

[1] http://www.inspire-geoportal.eu

However, two issues remain when following this approach:

1. The SOS supports the provision of the above mentioned information, but many elements are not mandatory for an SOS to be compliant to its specification. Thus, if an SOS shall be used for the creation of INSPIRE related metadata, one must make sure that the metadata required for INSPIRE is provided by the SOS.

2. Even if all possibilities in the current SOS schemas are used, some metadata required by INSPIRE remains which is not provided by the SOS. Some of these metadata elements are to be picked from fixed keyword lists provided by the INSPIRE Commission Regulation. A schema was defined gathering all missing metadata elements. It is possible and conformant to the current SOS specification to include links into the *GetCapabilities* response of a SOS. A link to an online resource can be defined in the ServiceProvider section of the capabilities[1]. This link can lead to an external document that is structured according to the defined schema and containing the missing metadata elements.

Using the defined mapping and the additional information provided for gap-filling, it is possible to create INSPIRE related metadata as well as SANY meta-information from SOS resources. Figure 10-3 shows an implementation architecture and the interactions between the system components for this harvesting task (Hilbring and Schleidt, 2009).

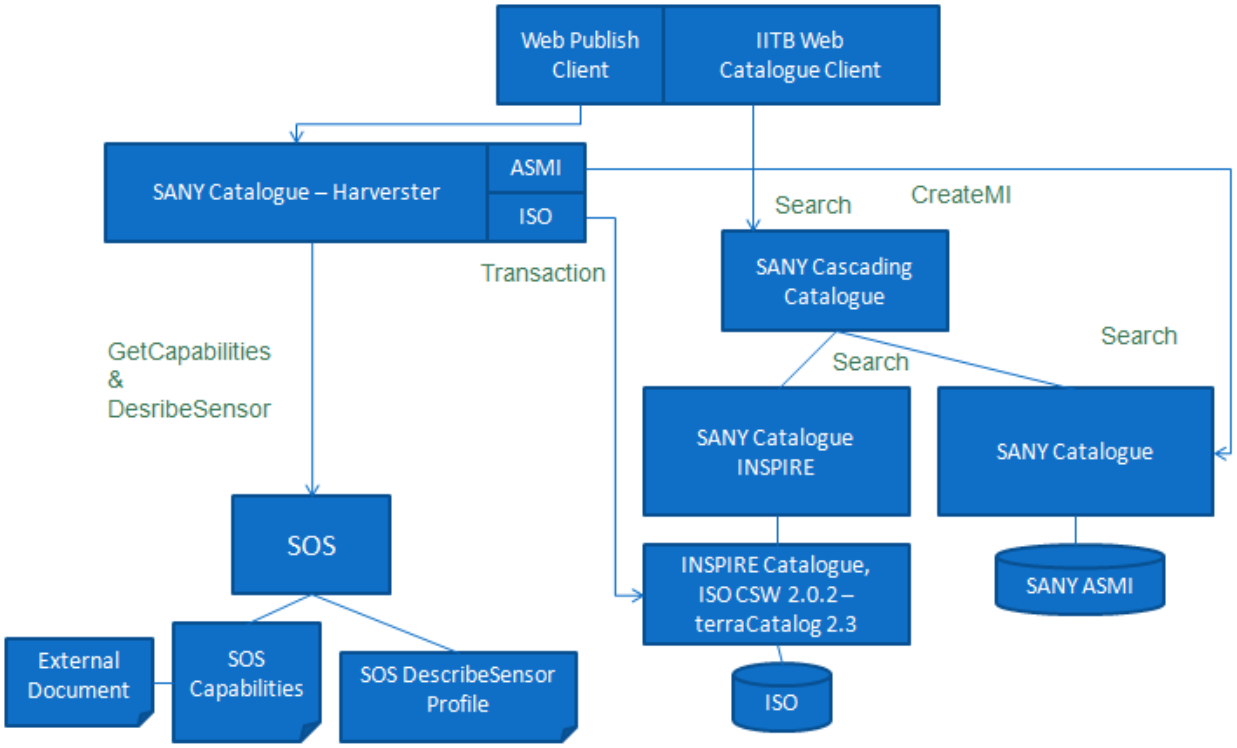


Figure 10-3: Creation and publication of INSPIRE and SANY related meta-information

The Catalogue Harvester calls the operations *GetCapabilities* and *DescribeSensor* of the SOS and uses the retrieved information for the metadata mapping. For the creation of INSPIRE related metadata the mapping described in the section "Mapping Solution" is used. The results are metadata documents structured according to the ISO 19115 or ISO 19119 schemas. They can

[1] sos:Capabilities/ows:ServiceProvider/ows:ServiceContact/ows:ContactInfo/ows:OnlineResource

be included into an OGC Catalogue Service 2.0.2 supporting the ISO Metadata application profile. This catalogue could be accessed directly from users or it could be included into a cascading catalogue realized by the INSPIRE geo-portal.

The architecture also shows the integration with a SANY Catalogue Server according to the SensorSA Catalogue Service (see section 8.4.1). A different mapping is performed in the Catalogue Harvester to create the SANY metadata documents according to the SensorSA Application Schema for Meta-information (AS-MI) schema (see section 7.7). These documents can be published via the publication interface of the Catalogue Service.

10.2.5 Event-based Harvesting

The harvesting solution described above in section 10.2.4 requires that the update of the meta-information is triggered by a management interface of the client accessing the Harvesting component. This may not be sufficient as people using the management interface might lack of knowledge about new developed services or changed data views in existing services. They need to be informed about changes, such that they can adapt the configuration of the management interface. This can lead to outdated meta-information in the catalogue. The alternative is to apply the event-based architectural style as it has been introduced as a SensorSA concept in section 6.4.

An event-driven processing system must be configured. The client of the harvesting component must provide a function which can be used for the subscription to the three event types service created, service updated and service deleted. Figure 10-4 shows the subscription phase:

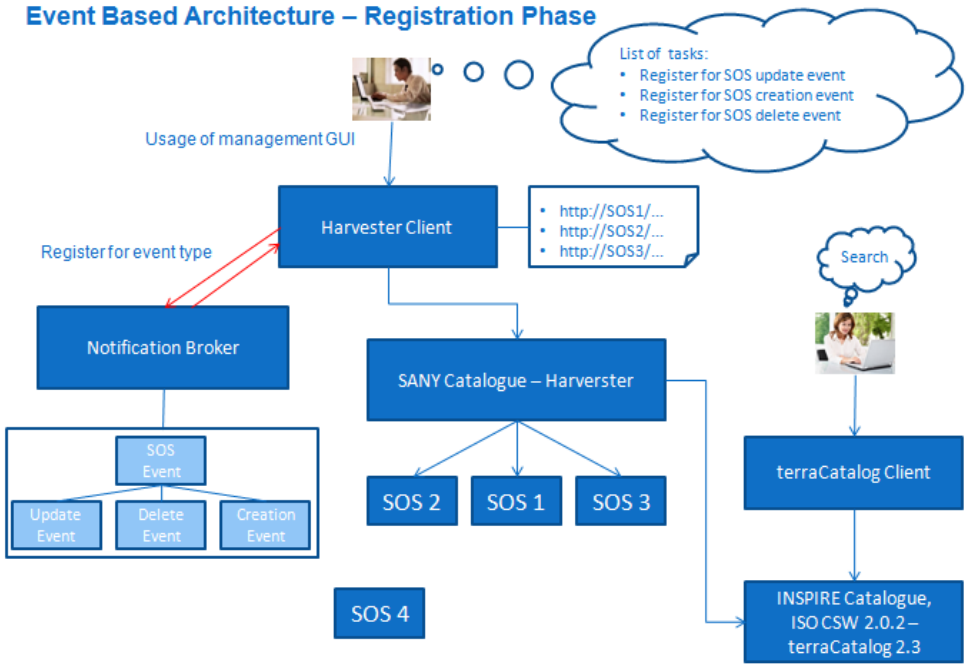


Figure 10-4: Event-based Harvesting - Registration Phase

After this configuration the system is ready for receiving events:

1. An event must be created. Typically the service automatically sends an event if the service has changed.
2. The notification broker receives the event and informs the interested party which subscribed for this event type. In case of the harvesting, the client of the Harvesting Component will be informed when services have been created, updated or deleted.
3. The client of the Harvesting Component will update the service configuration list and performs the meta-information harvesting function

Figure 10-5 shows the operational phase. In this phase, all services contained in the service configuration list are harvested according to a harvesting policy, e.g. periodically after a given time period that may be set by the client of the harvesting component. The service configuration list of the harvester has to be updated depending on the type of the event received. In case of a "service creation" event, a new entry is added. In case of a "service deleted" event an entry is deleted.

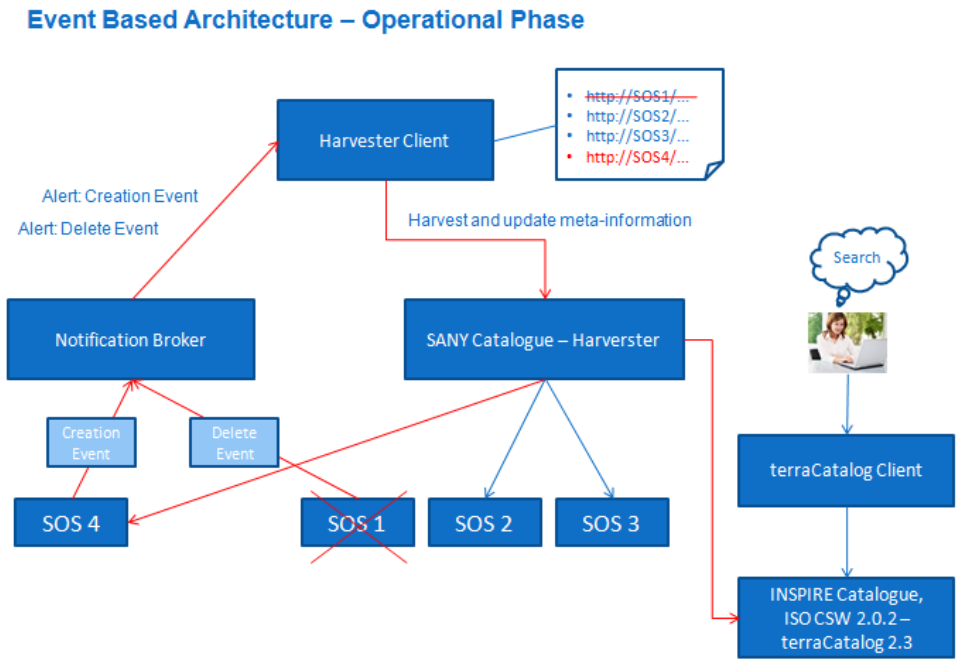


Figure 10-5: Event-based Harvesting - Operational Phase

10.2.6 SOS Resource Model

The resource model as specified in section 7.6 may be used to structure the discovery and the access to information provided by the SensorSA services. As an example SOS resource types (see section 8.2.2) are defined. These are oriented at the O&M model (section 7.2) and shown in Figure 10-6.

For each resource type an example identifier scheme is given as a constraint. The configuration of resource types in a resource type network is illustrated in Figure 10-6. The directed links in Figure 10-7 represent the navigation possibilities between the representations of these resource types. Selected resource types are described in more detail in subsequent sub-sections. The following resource types are defined:

- *SOS-instance*: This resource type is a sub-type of the resource type *ServiceInstance* and provides the description of the capabilities of an instance of the SOS. It provides access to procedures (through a selection in a *ProcedureList* as a sub-type of a *SelectionList*) and makes offerings which may be selected by means of an *OfferingList* as a further sub-type of a *SelectionList*. An SOS-instance is access by the path ./sos followed by the URL and the local id of the SOS-Instance.

- *Procedure*: This resource type reflects the O&M concept of a procedure. A procedure is linked to an offering and bound to a *FeatureOfInterest*. A procedure is accessed by the path {SOS-instance}/procedures followed by the name of the procedure.

- *Offering*: This resource type reflects the SOS concept of an observation offering. It contains *ObservedProperties* to which may be navigated by means of a *PropertyList* as a sub-type of a *SelectionList* It is linked to an *ObservationCollection* that delivers all observations of all procedures in one resource representation. An offering is accessed by the path {SOS-instance}/offerings followed by the name of the offering.

- *ObservationCollection*: This resource type reflects a collection of the O&M concept of an observation. An observation collection is accessed either by the path {ObservedProperty}/observations or by the path {Offering}/observations followed by the name of the observation collection.

- *FeatureOfInterest*: This resource type reflects the O&M concept of a feature of interest. It is linked to *ObservedProperties*. A feature of interest is accessed by the path {Offering}/features followed by the name of the feature of interest.

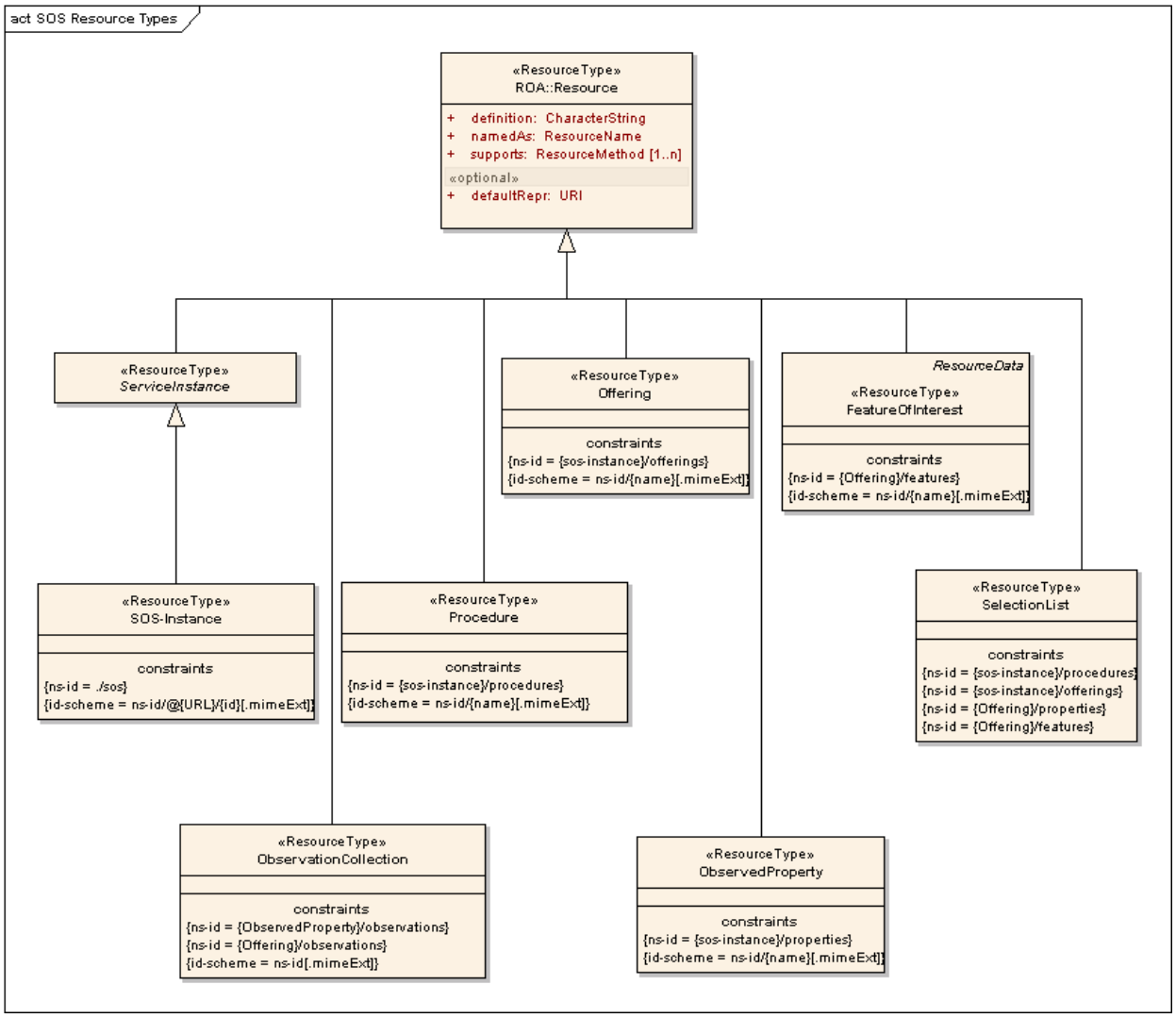


Figure 10-6: Resource types for the access to sensor observations

- *ObservedProperty*: This resource type reflects the O&M concept of an observed property. An observed property is linked to an *ObservationCollection* that delivers observations of this property for a given time period. An observed property is accessed by the path {SOS-instance}/properties followed by the name of the feature of interest.

- *SelectionList*: This resource type is an auxiliary resource type that is used to select a resource representation out of a list. In this example, it is used to select properties, offerings, procedures and features of interest. The access path to a selection list is dependent on the resource representation to be selected. For instance, for the selection of features of interest, it is the path {Offering}/features.

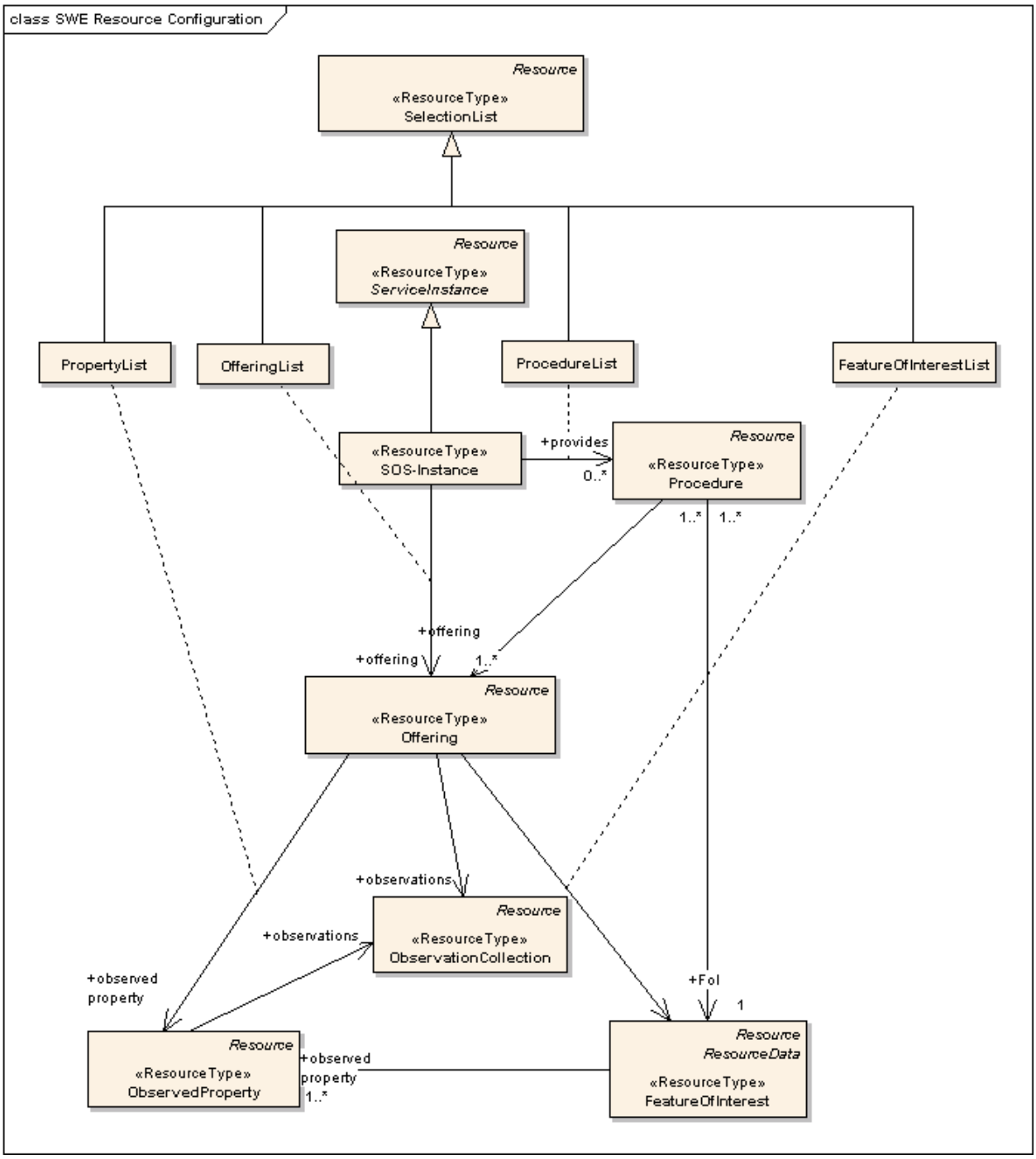


Figure 10-7: Resource type network for the access to sensor observations

Figure 10-8 shows an example of an SOS resource type network embedded in a Web-based application that allows one to navigate. Representations of observations in terms of diagrams, tables or maps may be retrieved by a direct URL or by navigating through the resource type network.

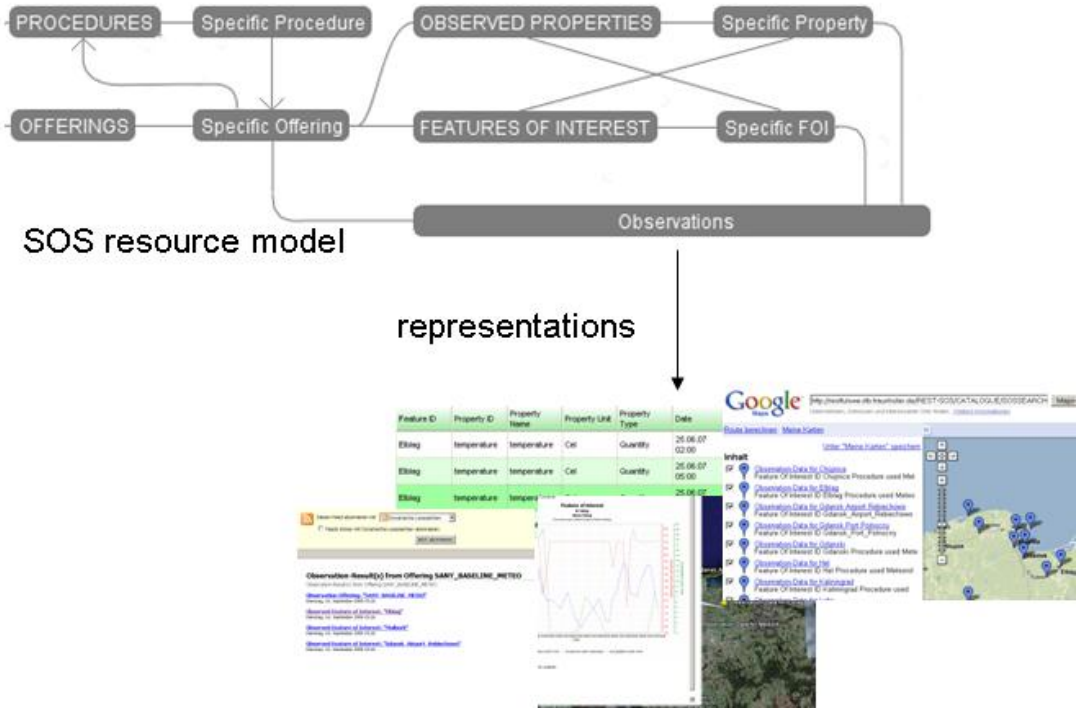


Figure 10-8: Example Representations of the SOS Resource Type "Observation Collection"

10.3. Policies for Sensor and Service Monitoring

Service monitoring is an important aspect of management which must be dealt with in a sensor network. This includes simple checks of the life status of a sensor or service as well as more complex monitoring of tasks such as average load or uptime. From the SANY point of view the various monitoring aspects in a sensor network are considered to be observed properties. The discrete monitoring results have a timestamp, can be associated to a *featureOfInterest* containing the location of the monitored resource, and have a procedure describing the measurement process.

Thus we can safely say that the information resulting from the monitoring process can be modelled as an observation according to the OGC Observation and Measurement model (Cox, 2007). A simple example would be monitoring the CPU temperature of a measuring station computer. Similarly event notifications can also be modelled as observations.

This view enables a harmonised integration of monitoring into the SensorSA. This approach implies the use of the OGC Sensor Observation Service (see section 8.2.2) and Sensor Alert Service (see section 8.2.4) for the configuration of the monitoring process, storage and access to the observation data related to the monitored sensors.

Getting the observation data into the SOS is implementation specific, but two patterns can be identified, as depicted in Figure 5-17.

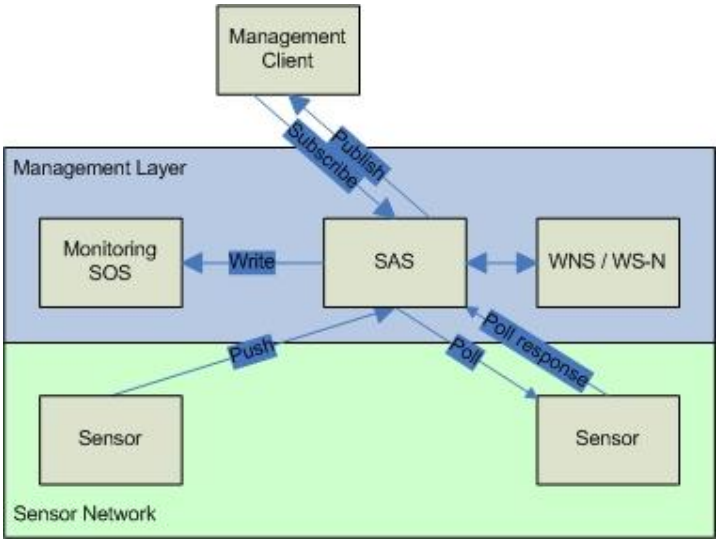


Figure 10-9: Monitoring SOS

In the first one the sensor directly pushes the status information to the Sensor Alert Service. This implies that the sensor knows the location of the SAS and is configured to do so. In the second and most usual pattern the Monitoring SAS polls, on demand or periodically, the status information from the sensors. In both cases it is the responsibility of the SAS implementation to write the observations to the Monitoring SOS.

In a sensor network used for environmental monitoring it is a common property that event notifications will be generated whenever certain conditions exceed some defined thresholds. Usually the most important task of the environmental monitoring application is to notify someone (a decision maker) about the event. The SAS provides the functionality needed for the configuration of event conditions and the underlying publish/subscribe notification mechanism. The notification acknowledgement functionality is essential for a decision support system and is part of the notification mechanism. The notification acknowledgements are necessary in order to enforce the delivery of notifications and allow for policy based delivery of notifications. For example, a notification policy might enforce resending an unacknowledged notification to a recipient on a higher level in the decision making hierarchy.

In the SensorSA the notification functionality is accomplished by two types of services which act in the backend of the SAS, depicted in Figure 10-3 as WNS/WS-N. The OGC Web Notification Service (WNS) and OASIS WS-Notification (WS-N) communicate with the SAS and serve as mediators between the SAS and the Management Client.

Coupled with the notification itself is the need for notification tracing, especially when something goes wrong and the decisions and steps that have been taken need to be revised. As stated in the beginning of this sub-chapter the event notifications and acknowledgements can be modelled as observations based on the OGC Observation and Measurement specification. Such notification tracing can simply be added to the service network by storing the messages in a time-series data store. In the SensorSA the OGC Sensor Observation Service is used for this purpose (Monitoring SOS).

Sensor monitoring is important in most measurement scenarios. In general there are a multitude of sensor aspects that require monitoring in order to ensure the availability and

integrity of the measurement data. The more complex form of monitoring consists if there are multiple sensors at the same time. An example is the video monitoring of a measurement station in order to detect external factors that might affect the measurement process. Other monitoring scenarios could involve sensor service degradation. The main limitation in scenarios involving mobile sensor networks is the power supply (usually batteries) of each individual network node. Depending on the network technology and geographic distribution of the nodes a failure of one node might render the entire network unusable. In these cases a vital monitoring aspect is the node residual power.

Based on this information a sensor network administrator can make management decisions. Examples of such decisions include selection of network optimisation strategies (e.g. adjusting the network routing policies or node reporting /sampling frequencies in order to reduce the power consumption of individual nodes or entire network). In the later scenario a good indication of the residual power of a node is the power supply voltage. On a conceptual level as described in section 6.6.2 in the SensorSA Management Architecture this information constitutes the actual observation data and should be handled the same way as any other observation coming from the sensor (e.g. ambient air temperature or humidity). Therefore, the observation data about power supply voltage shall be encoded as an observation according to the OGC Observation and Measurement model (Cox, 2007). Consequently, the observation data can be stored and accessed by means of an OGC Sensor Observation Service (see section 8.2.2). Additionally the Sensor Alert Service (see section 8.2.4) can be used to define event conditions (and generate notifications) about the observation data, e.g. to send an e-mail notification to the sensor network administrator whenever a node battery voltage drops under a defined threshold.

10.4. Policies for Sensor Planning

Examples of sensor planning tasks in a sensor network include sensor configuration, sensor calibration or the actual initiation of a measurement.

Whenever a measurement is triggered or prepared, the sensors involved must be configured for the specific measurement (or measurement series). This can be achieved by the Sensor Planning Service (SPS) as described in section 8.2.3. Although the same operation (submit) is invoked for both planning and configuration the slight difference is the observation response. For planning the response encompasses observation data whereas the result returned upon configuration will contain the success status of the configuration step. One obvious advantage is the possibility of planning configuration tasks.

In general, sensor planning includes different interaction models or patterns. Some sensors allow synchronous interaction patterns, i.e. the service responds directly to incoming requests. An example would be an instance of an SPS that provides a facade for a simple forecasting model. This service, at least theoretically, could start unlimited parallel processes. Concurrent users don’t compete for limited resources and the service can report the successful execution of the requested tasking right away.

Other sensors require asynchronous interaction patterns. This is the case if multiple users have to share a limited resource and the execution of the tasking cannot be handled instantaneously. An example would be a satellite that could at any moment in time observe a single scene only. If this satellite is equipped with an optical sensor, the observation depends,

among other factors, on the cloud coverage. Thus, the tasking request might consume any amount of time before being fully executed.

10.5. Policies for Access Control

The services performing the access control tasks (see section 8.3) cover major parts of the abstract access control pattern as introduced in section 6.8.2. In the following some interaction patterns and methods to use this Access Control Framework are presented.

10.5.1 Patterns for Non Intrusive Access Control

The stateless nature of SOAs causes that in principal no lasting connection between client and service is established. Therefore, each service message has to include the application context. This is one of a number of reasons why security measures, especially access control, have to be positioned on the message level (Kanneganti and Chodavarapu, 2008). Note, however, that established Internet security measures such as SSL (Secure Socket Layer) for the encryption on transport layer may be used in addition.

One of the more challenging goals of SOA security is to minimize interference with the actual service communication in order to relieve service designers and developers that are experts for their particular domain from including security aspects in their work. This property is called “Non Intrusive”.

Flavours of the “Non Intrusive” property in the SOA context are:

- The protected service remains untouched (specification and implementation).
- The body of the message that is derived from the interfaces of the service content is „not“ modified by security mechanisms.
- From the viewpoint of the client the interface to the secured service must be unchanged so that it just depends on the access control policy on the server side if an operation is permitted or not.

Measures towards a “Non Intrusive” access control architecture affect services as well as service messages.

10.5.1.1 “Non intrusive” at service level

The abstract access control pattern explained in section 6.8.2 implies that a Policy Enforcement Point exists for every security enabled service. One implementation pattern is a transparent service proxy that shields an unsecured Web service and mimics the secured service. For instance, in the SensorSA W3C Web Service platform (see section 9.2.1) this means that it provides the “same” WSDL document as the underlying unsecured Web service.

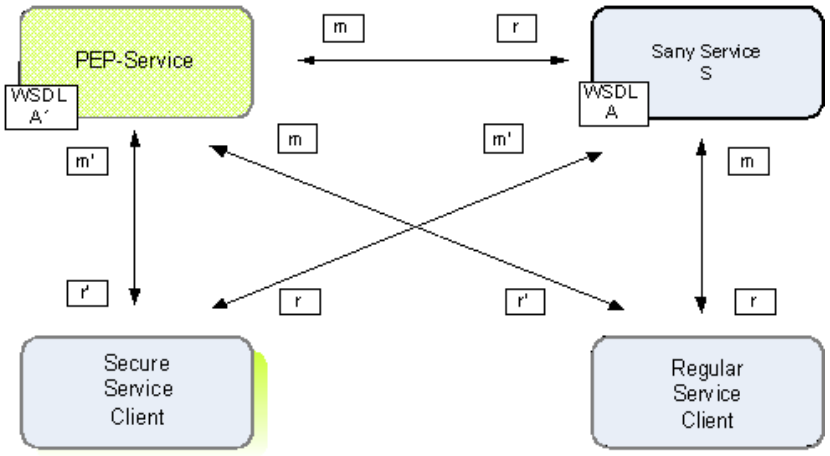


Figure 10-10: Non Intrusive "compatible" Approach

Note: See Figure 10-11 for definitions of m, m',r and r'. WSDL A' equals WSDL A, apart from the service endpoint.

A transparent service proxy is a software component that provides the same interfaces as an underlying service but allows one to offer additional functionality that is transparent to the client. Thus, the SensorSA does not consider it to be an actual service in the sense that it does not provide operations of its own. Nevertheless, Table 10-1 provides the description of the transparent service proxy in the same description style as the SensorSA services in section 8.3.

Name	Service Proxy
Standard Specifications	The following standards are used by the Service Proxy: • OASIS Security Assertion Markup Language (SAML) v2.0 • OASIS SAML 2.0 profile of XACML v2.0
Description	The Service Proxy "intercepts" service requests for the proxied service and delegates the requests to the Policy Enforcement Service. The Service Proxy can be automatically provided by a Proxy Generator. In contrast to the entirely generic Policy Enforcement Service, in addition to the mimicked service interface the Service Proxy may contain service specific elements. As suggested in OASIS WS-Security standards, the optional security information encoded in SAML is provided in the SOAP header while the actual service request in the SOAP body remains unchanged.
Interface *Proxy*	
The Proxy Interface does not define operations by itself but mimics the interface of an arbitrary web service. A software component, the Proxy Generator, is used to automatically generate a service specific Proxy.	
Comments	The Service Proxy is not to be confused with a firewall. The Service Proxy does not contain any security mechanism on the transport layer level and does not prevent a service requestor from accessing the proxied service.

Table 10-1: Description of Service Proxy

10.5.1.2 "Non Intrusive" at message level

A non intrusive realisation of access control aspects at message level requires that the actual message body is not altered. Instead, there is meta-information dedicated to access control in the message header. This approach ensures backwards compatibility in the sense that existing clients need not to be changed.

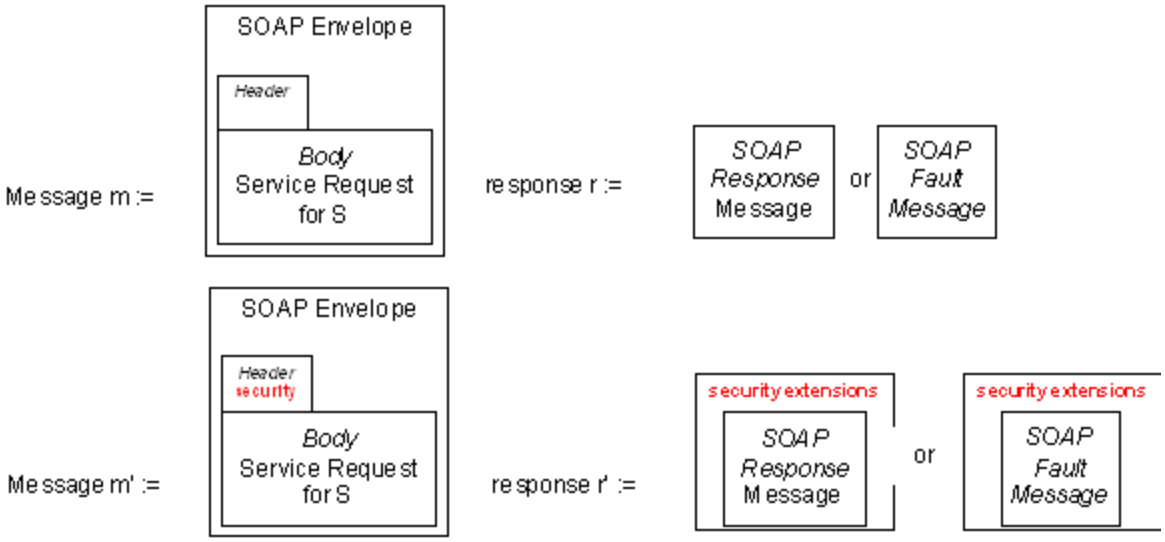


Figure 10-11: Security Information in the SOAP Header

As shown in Figure 10-11 the SOAP protocol distinguishes between SOAP Body containing the actual service message and a SOAP Header for "meta information". Obviously, the nature of the SOAP protocol therefore directly provides the means for "Non Intrusive" access control on the message level.

10.5.2 Patterns for Access Control in Service Chains

From service- and information viewpoint, access control for service chains needs no particular attention as service chains and their elements are perceived as services and therefore all presented concepts already apply to them. However, the SensorSA Access Control Framework leaves a high degree of freedom to the engineer with the task to design, set-up and deploy a security domain using concepts and tools provided. To tackle security issues for service chains, the engineering viewpoint perspective leaves several thinkable approaches induced by application specific requirements (that could be even based on legal requirements).

Before a Service Chain Access Control design decision is made several questions may affect the decision.

- Who may define the service chains policy?
    - Who may define/edit/delete a service chain?
    - Who may invoke a certain service chain?
- Who is in charge of the policies of the service chain's elements and how are these policies managed?

- What is the level of trust that can be established between a service chain and its elements?

- What is the authentication method for all services involved (chain & elements)?

For the following we assume that a service chain is encapsulated by service (e.g. a WPS), that could be implemented as a BPEL process. In practice we find variations of two different approaches.

10.5.2.1 Delegate (Anonymous) Service Chain

A Delegate Service Chain is considered to act on behalf of the subject invoking the service chain. A Delegate Service Chain therefore does not necessarily possess an identity; instead identity information of the invoking subject is presented to every service chain element. Thus, the service requestor has to provide identity information that has the necessary privileges for each element of the service chain.

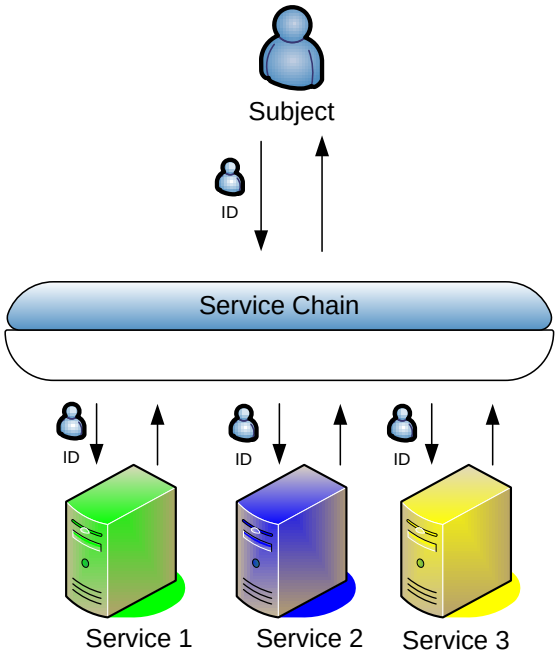


Figure 10-12: All elements accept identities (ID) from one IdP

In the case that all elements accept request from single IdP the overhead of maintaining the various policies lies in the responsibility of the respective service provider.

In the case that all elements accept requests from different (possibly their own) IdPs the overhead of acquiring the proper identities lies in the responsibility of the subject.

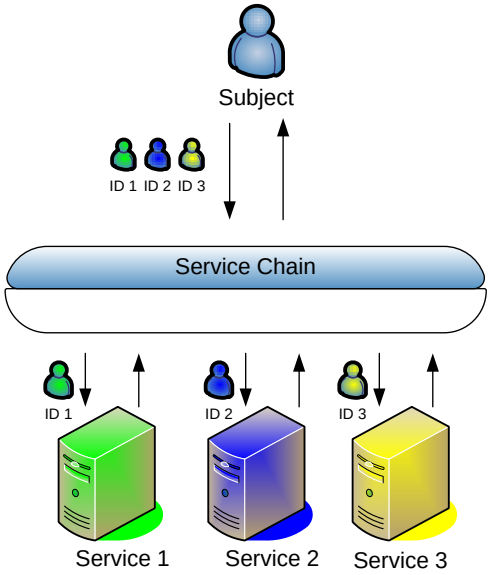


Figure 10-13: All elements accept identities from different IdPs

10.5.2.2 Identifiable Service Chain

In this approach a Service Chain is considered a subject and therefore has its own set of identity information. By using this pattern the service chain will not forward the service requestor's identity. Instead, a dedicated service chain identity is presented on every request. This approach is in line with the philosophy of the SensorSA Access Control Framework, however it implies a somewhat higher level of trust between Service Chain Elements and the Service Chain as the actual subject invoking the service chain may remain transparent to service.

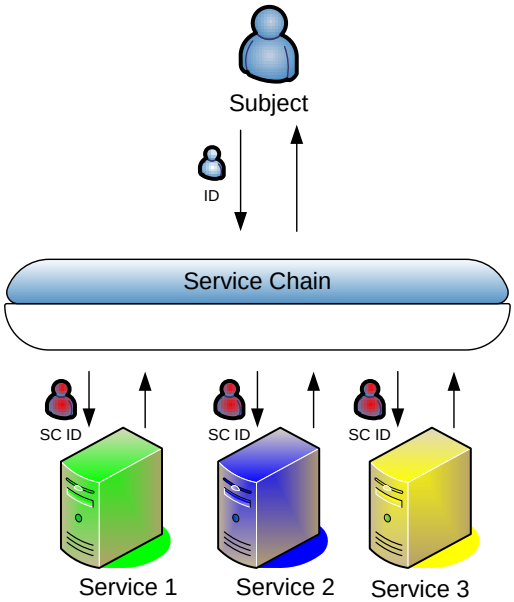


Figure 10-14: All elements accept identities from one IdP (SC ID = service chain ID)

If all elements accept requests from a single IdP, the overhead of maintaining the various policies lies in the responsibility of the service provider.

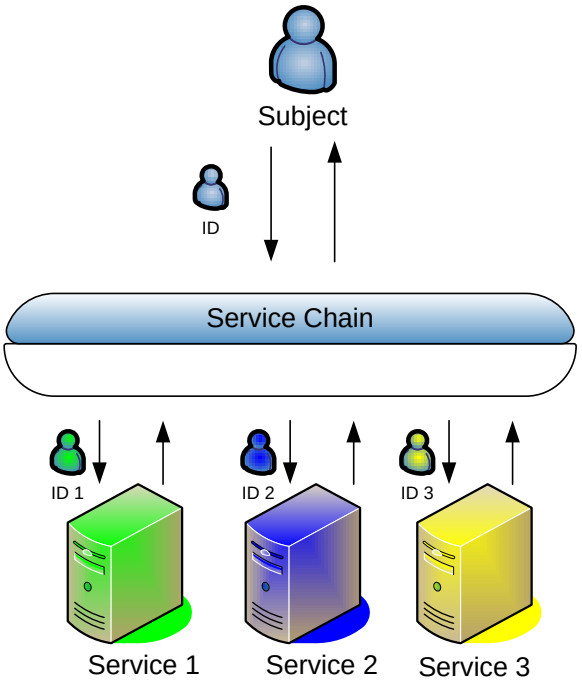


Figure 10-15: All elements accept identities from different IdPs

In the case that all elements accept request from different (possibly their own) IdPs, the overhead of acquiring the proper identities lies in the responsibility of the service chain.

10.5.2.3 Applicability for Ad Hoc Service chains

To enable on demand/ad-hoc adaptive service chaining (or composition) the approaches presented can be directly used. The general problem here is that a subject has to be enabled to obtain the proper identity information to gain access to the service chain (e.g. implemented as a BPEL process) or all elements of the service chain respectively. The on-the-fly lookup of IdPs and automatic registration and authentication however, is out of scope of the current version of SensorSA and will be subject of future work.

10.5.2.4 Conclusion

As already mentioned the selection of an authentication approach for service chains is a design decision that depends on the application requirements. With the SensorSA Access Control Framework it is even possible to combine both approaches in a single use case, if for example a Delegate Service Chain approach is necessary to support e.g. ad-hoc service chaining but for legal reasons the service requestor identity has to be presented on an element service invocation as well.

10.5.3 Patterns for Access Control in a Multi-Protocol Environment

The presented access control approach includes the extensive usage of OASIS Security standards. These standards have a focus on security enablement of SOAP messages and therefore the implemented access control architecture components cannot be directly applied to services with non SOAP bindings. As a matter of fact most OGC services and therefore a considerable number of service implementations that are part of SensorSA do not provide a SOAP binding yet. However, the available security service implementations are perfectly usable by utilising protocol adapters where necessary.

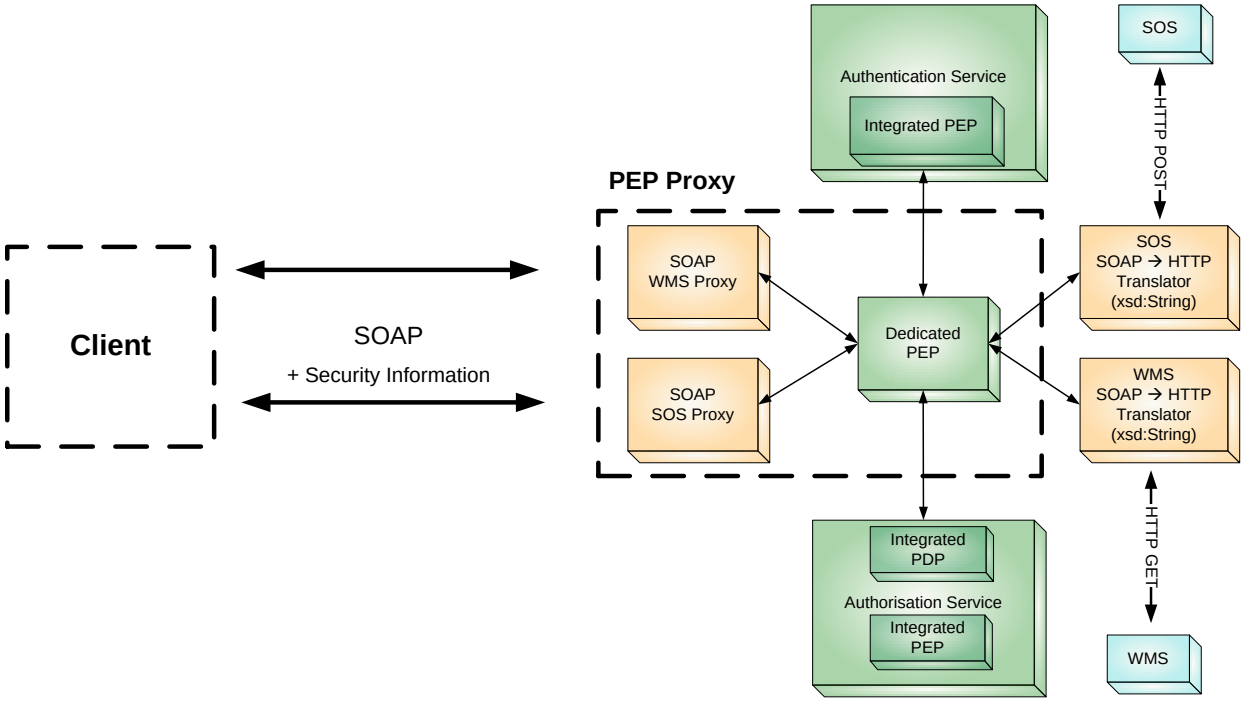


Figure 10-15: Access Control for HTTP based WMS & SOS

To use SensorSA security and access control mechanisms, service implementations with bindings according to the SensorSA OGC Web Services Platform (see section 9.2.2) have to be equipped with SOAP interfaces (protocol translators/adapters). To this end, a number of protocol translators e.g. for WMS and SOS implementations is necessary as illustrated in Figure 10-9

10.5.4 Usage of SAML

SAML assertions issued by the Identity Management & Authentication Service contain the following core elements:

- Assertion ID: unique session id generated when the assertion is issued
- Assertion IssueInstant: timestamp when the session was generated
- Issuer: contains the End Point Reference to the IdP that "issued" the assertion
- Subject NameID: contains a unique id of the identity that is managed by the issuing IdP

- Conditions: describes which conditions must be fulfilled so that the assertion can be considered valid
- SubjectConfirmation: Information on how to identify and validate session information
- AuthnStatement: contains several information about the act of authentication, especially which form of credentials were provided
- AttributeStatement: contains information about the identity's attributes

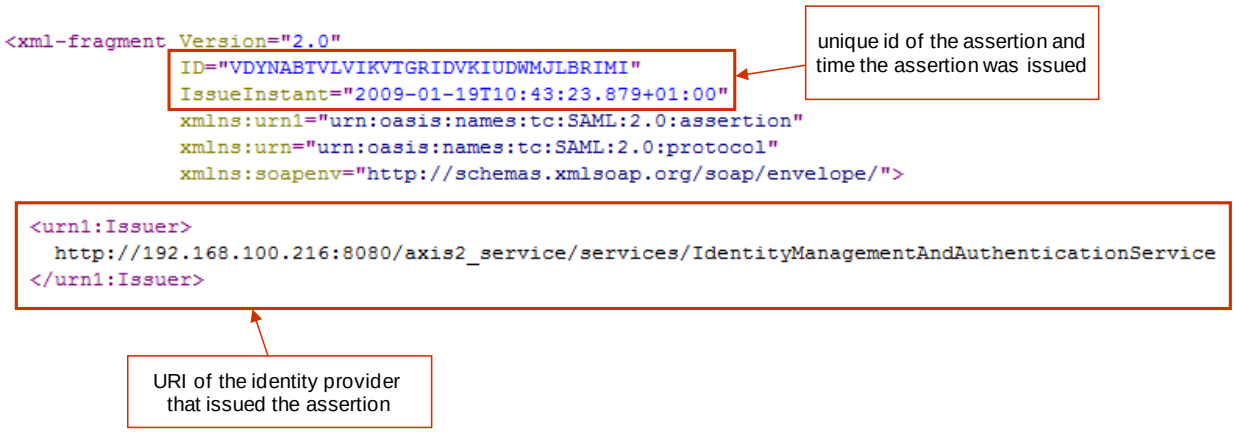


Figure 10-16: Example of a Subject NameID

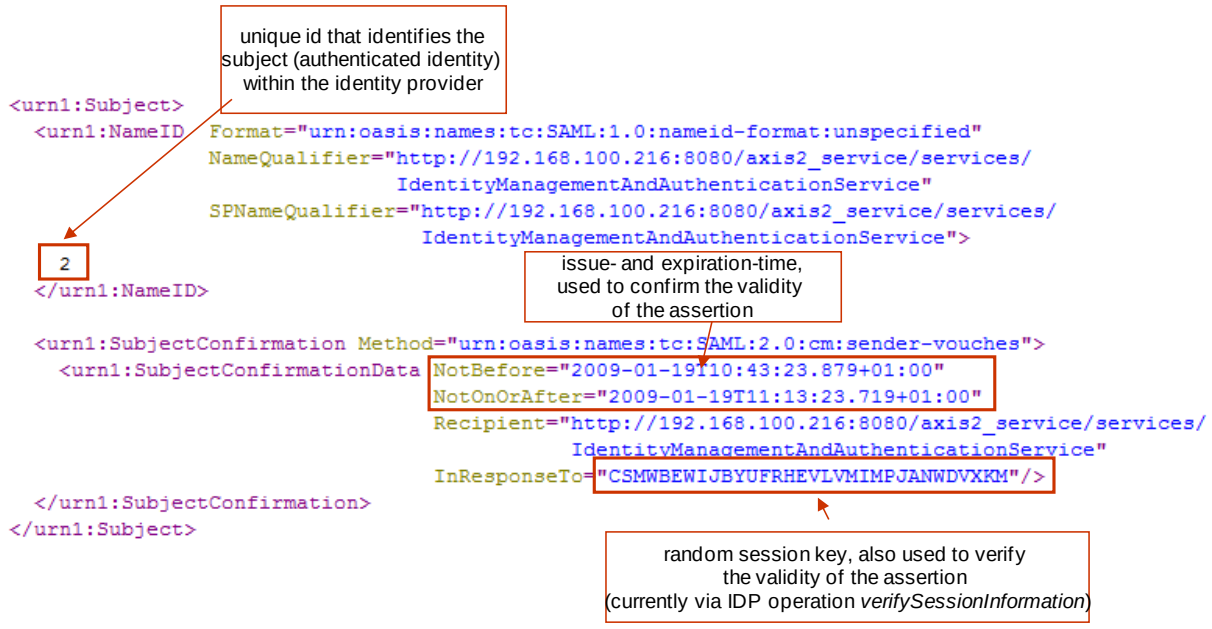


Figure 10-17: Example of a Subject Confirmation

Figure 10-18: Example of an Authentication Statement

Figure 10-19: Example of an AttributeStatement

Figure 10-20 indicates where SAML plays a role in relation to the abstract access control pattern (see 6.7.2).

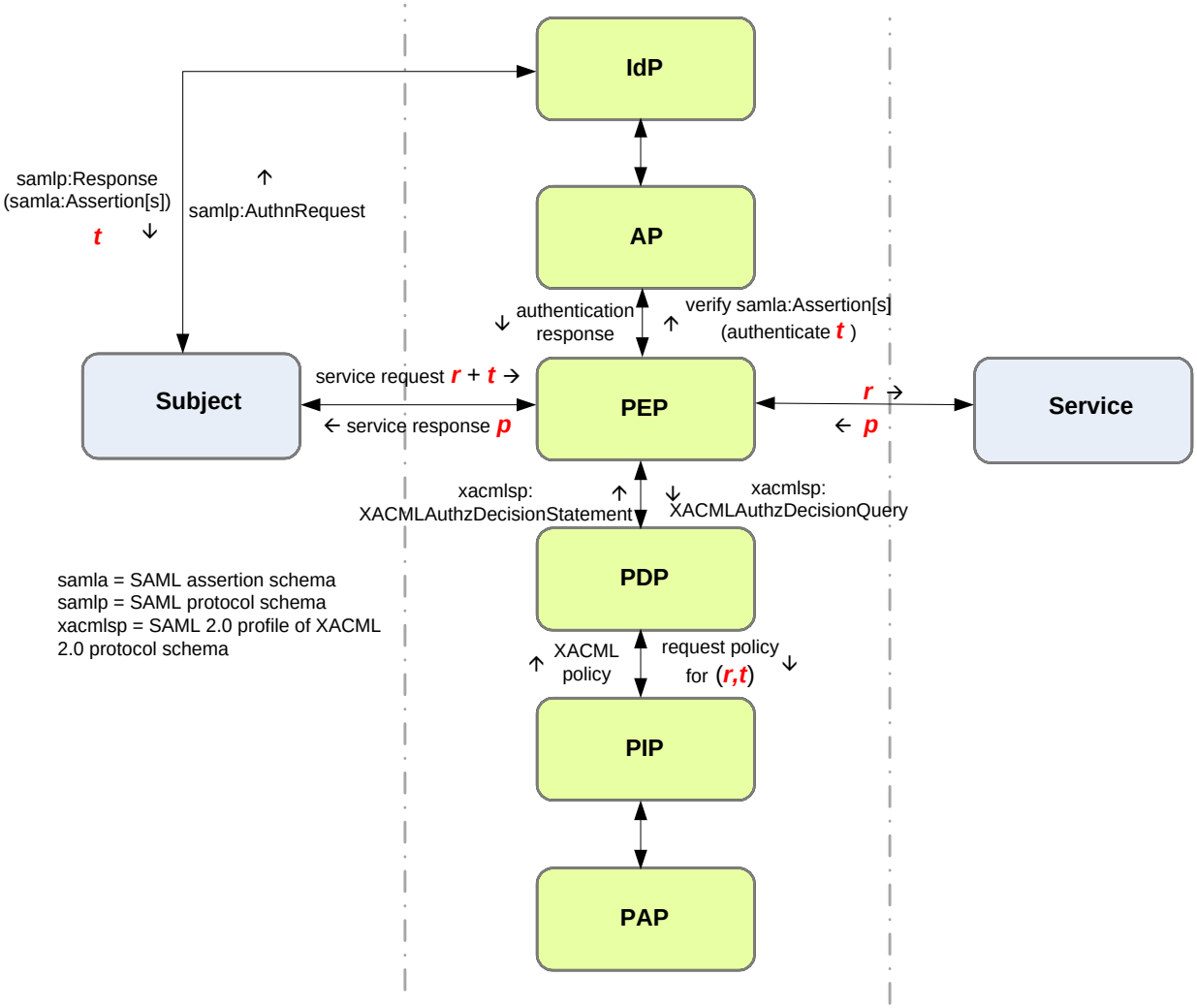


Figure 10-20: SAML in relation to the Access Control Pattern

10.5.5 Usage of XACML

This paragraph provides a simple example of how to use the basic XACML constructs described in section 7.4.6.2.

10.5.5.1 Example SOS Policy

Translated to plain English the policy below specifies that for the service *SoapBindingsSOSv3WS0*1 only those XACML subjects which are members of the *group SOS User* have read access. Further, the requested action must be *getObservation*. All of these properties are mapped in XACML policies via attributes, which are basically key value pairs. Due to this characteristic of XACML, it is possible to express arbitrary real world concepts as properties, e.g. roles, groups, company branch or department. Because there is a lot of boilerplate XML in the example code the interesting parts are marked in red boxes. For simplicity and better comprehension the … notation indicates that some parts of the policy document are omitted.

```
<Policy
  ...
        <ResourceMatch MatchId=
          "urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType=
              "http://www.w3.org/2001/XMLSchema#string">http://demoserver:808
              0/axis2_service/services/SoapBindingsSOSv3WS01
          </AttributeValue>
          <ResourceAttributeDesignator
              DataType="http://www.w3.org/2001/XMLSchema#string"
              AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
              id"/>
        </ResourceMatch>
  ...
<Rule RuleId="PermitSOSReadWS01IfUserHasGroupSOSUserSOSAdminOrRoleSOSUser"
    Effect="Permit">
    ...
          <SubjectMatch MatchId=
              "urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType=
                "http://www.w3.org/2001/XMLSchema#string">SOS User
            </AttributeValue>
            <SubjectAttributeDesignator DataType=
                "http://www.w3.org/2001/XMLSchema#string" AttributeId="name"/
                >
          </SubjectMatch>
          <SubjectMatch MatchId=
              "urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType=
                "http://www.w3.org/2001/XMLSchema#string">group
            </AttributeValue>
            <SubjectAttributeDesignator DataType=
                "http://www.w3.org/2001/XMLSchema#string" AttributeId="type"/
                >
          </SubjectMatch>
    ...
          <ActionMatch MatchId=
              "urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType=
                "http://www.w3.org/2001/XMLSchema#string">getObservation
            </AttributeValue>
            <ActionAttributeDesignator AttributeId=
                "urn:oasis:names:tc:xacml:1.0:action:action"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ActionMatch>
    ...
</Rule>
    ...
```

Figure 10-21: Example of an SOS Policy

10.5.5.2 Example SOS Authorisation Request

As in the policy example before, in plain English the request below wants to access the resource SoapBindingsSOSv3WS01. The requester is a member of the group SOS USER. Further, the action element indicates that the requester would like to access the *getObservation* operation.

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
  <Subject>
    ...
    <Attribute AttributeId="name"
      ...
      <AttributeValue>SOS User</AttributeValue>
    </Attribute>
    <Attribute AttributeId="type"
      ...
      <AttributeValue>group</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      ...
      <AttributeValue>
        http://demoserver:8080/axis2_service/services/SoapBindingsSOSv3WS01
      </AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action"
       ...
      <AttributeValue>getObservation</AttributeValue>
    </Attribute>
  </Action>
  ...
</Request>
```

Figure 10-22: Example of an Authorisation Request for an SOS

10.5.5.3 Evaluation

Both the policy and the request form the decision basis of a XACML PDP for a particular resource. There are only four possible PDP decisions:

- Permit
- Deny
- Indeterminate
- NotApplicable

"Permit" and "Deny" are already discussed in the previous paragraphs. The value "Indeterminate" occurs if there is an exception during the evaluation of the request and no regular decision can be made. The decision "NotApplicable" indicates that there are no policies

with a matching target/rule for the given request. In the example above an XACML PDP would decide that the requester is permitted to access the resource *SoapBindingsSOSv3WS01* because a policy exists which addresses the resource in a target. Furthermore, the request has one subject which possesses the group attribute with the required value SOS User and the action attribute *getObservation*. Therefore the rule evaluates to the effect "Permit" and because of the fact that there are no further rules which could evaluate to "Deny" the resulting decision of the PDP is "Permit".

10.6. Processing of Quality Information

10.6.1 Attachment of quality information

The OGC Sensor Observation Service (see section 8.2.2) is used to access observation results. Uncertainty and quality information is relevant at two different places within this service specification:

- At the level of the observation process, uncertainty and quality information may be included in the SensorML document returned by the *describeSensor* operation. All required information about the observation process, the uncertainty of the observations as a collection and the quality assurance processes applied may be included in this document. Time dependent uncertainties, e.g. due to instrument deterioration, could be expressed in SensorML. Client applications may, however, find it more convenient to have the resulting uncertainty of observations expressed directly for individual observations as in the next item.

- At the level of the individual observations, uncertainty and quality information may be included in the Observation & Measurement documents as returned by the *getObservation* operation. Again, the observation-specific information regarding the observation process, the uncertainty of this observation value and the quality assurance process that this observation value has undergone may be included here. Figure 10-23 shows an example of a *getObservations* result with an UncertML block quantifying the uncertainty of the observations. A *href* in the result block definition provides the link between the property observations and the associated uncertainty data. There is a tacit but natural assumption that the order of the uncertainty information is the same as the observation values. The XML file can be parsed by clients not able to evaluate the uncertainty data.

A model based calculation usually makes several assumptions about the nature of the physical process under study (e.g. ground water flow in a saturated, homogeneous aquifer with uniform hydrological parameters). These assumptions shall be described in the associated SensorML of the model procedure.

It is good engineering practice to always include quality and uncertainty information.

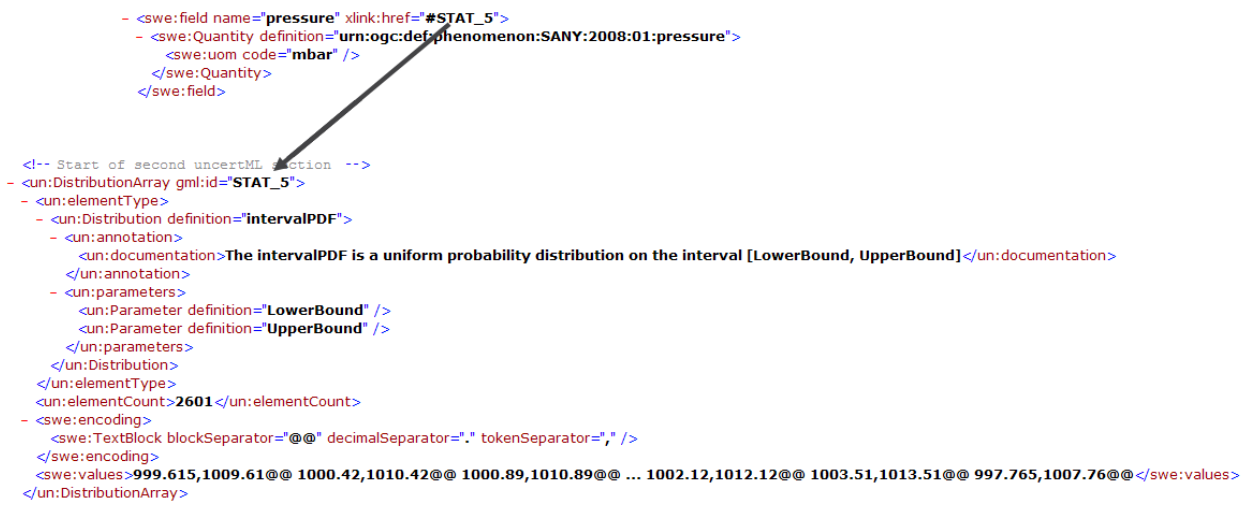


Figure 10-23: UncertML block in a *getObservations* result

10.6.2 Multi-level measurement chains

When working with quality-assurance processes, data have different "levels" of quality control information, as illustrated in Figure 10-24. For example, in the air quality domain the "raw data" sampled from the sensor undergo some automatic quality assurance process ("QC level 1"). In a second step, a manual quality control process is applied to the data ("QC level 2"). Sometimes a user may be interested in some specific level of quality controlled data (e.g. raw data). Other application scenarios require querying the "best available" data, which means that for each measurement taken, the data point with the highest level of quality control should be returned.

Within an OGC Sensor Observation Service this can be handled using different offerings with different procedures. The procedure describes the level of quality control that this specific data set has undergone. For each level a procedure has to be defined, and, if required, one (or more) additional procedure(s) defining the "best available" data can be defined.

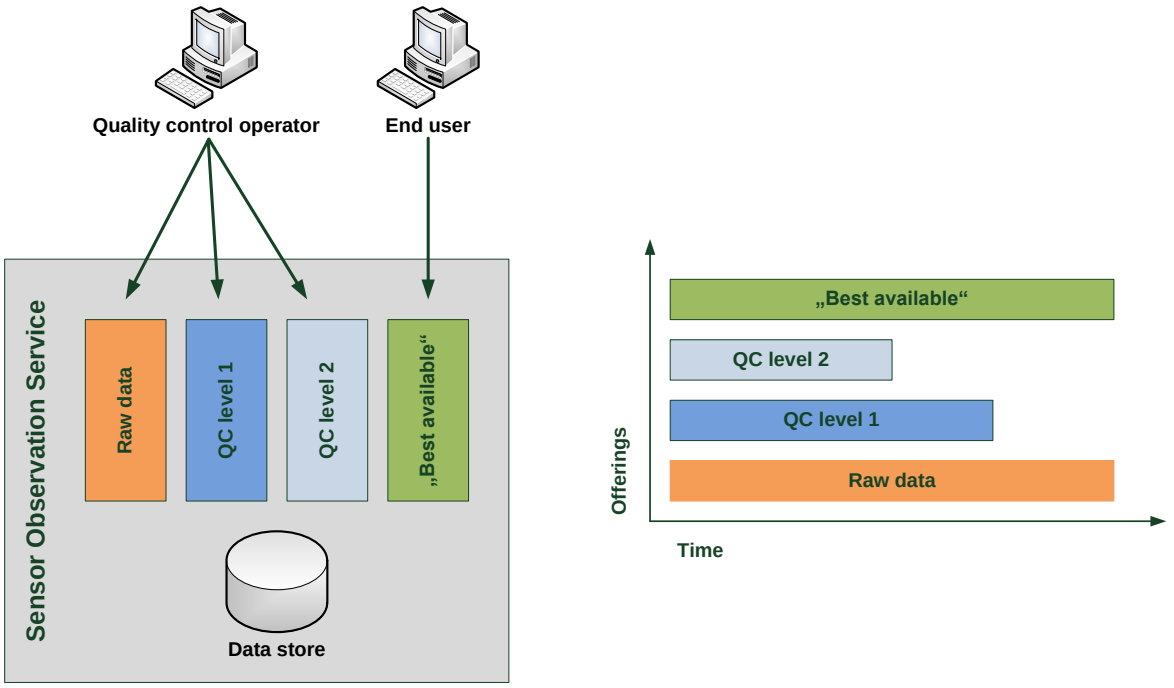


Figure 10-24: Example for a multi-level measurement chain in an SOS

10.6.3 Visualisation of Uncertainty Information

It is desirable to represent uncertainty with graphic variables on maps created with the Map and Diagram Service (see section 8.4.3). Several promising techniques have been already identified for the visualisation of uncertain information in static maps.

MacEachren (1992) promotes the use of transparency for uncertainty depiction based on a metaphor of "fog" obscuring the view proportional with the amount of uncertainty. He also includes additional modalities for uncertainty visualization such as colour saturation, crispness (contour crispness and fill clarity) and degradation of the resolution of raster images. A similar technique comes from Drecki (2002). He proposed an "opacity" display, where opaque objects are the certain ones. The last identified technique comes from Hengl (2003). His work suggests that uncertain data should appear increasingly white or “pale,” depending on the magnitude of uncertainty. Whereas the first techniques apply mainly to coverages, the last two techniques (opacity and colour bleaching) may prove to be especially important for the visualisation of discrete geographical objects such as moving sensors.

Based on the techniques presented above, it should be possible to obtain map representations that display the amount of uncertainty by varying transparency, varying colour, transparency blending, colour bleaching, use of fill patterns and adjusting resolution of geographic detail. Moreover, considering the specificity of sensor data as processed in sensor service networks (which primarily consists of interpolated measurements and coverages), the following three additional techniques have been investigated in the SANY project on a conceptual level:

- perpendicular colour and transparency variation along contour lines
- varying contour widths, and

- use of graphic filters (e.g. blurring) as a generic mechanism for localised colour manipulation.

Note: Currently, the SensorSA suggests that the use of colour, transparency and texture are the best candidates for representing uncertain information for static maps in an efficient manner. Therefore, these techniques will be further researched in the context of the Map and Diagram Service. The other techniques will remain only as concepts due to the complexity of implementing such graphically demanding techniques in the Map and Diagram Service.

10.6.4 Unit conversion

Unit conversion in a SANY sensor network can be handled in two ways:

1. Each service offers a (set of) unit(s) in which it is able to provide its values and/or perform its operations. The clients can select one of the offered units (if multiple are provided), and then must do all remaining unit conversions (if necessary) on their own.

2. All of the functionality described in (1) above, plus a dedicated Processing Service (see section 8.4.2) that can be used by client applications (and services acting as clients) to perform unit conversions that they are not able to do on their own.

Note: Unit conversion is considered to be part of the pre- or post-processing steps in a processing chain (see section 10.9.1.2).

10.7. Handling of large data sets

Data access in the SensorSA is provided mainly through instances of Sensor Observation Services. Depending on the application scenario fairly large amounts of data may be needed and thus handled by those services. When working with large amounts of data, the following scenarios can be distinguished:

- Accessing a large block of data all at once

- Accessing a smaller piece of a large data set

10.7.1 Accessing large data blocks

The Sensor Observation Service (see section 8.2.2) and its accompanying Observation and Measurement data model (see section 7.3) provide multiple mechanisms for transporting the actual observation data in response to a request, specified in the *responseMode* parameter. While some of them (e.g. inline) are only suitable for smaller sets of data, the observation data can also be transported separately from the *getObservation* result (out-of-band), which in this case only provides a description of the data and a pointer (e.g. an URL) to the data themselves.

The data themselves can be transported in any form and/or encoding that is suitable for the specific application scenario. An example would be offering the data binary encoded with

NetCDF (http://www.unidata.ucar.edu/content/software/netcdf/index.html) and transported using OPeNDAP (http://opendap.org/) as shown below (section 5.4 of Cox, 2007).

```
<?xml version="1.0" encoding="UTF-8"?>
<om:Observation gml:id="timeSeries1" ... references omitted ...>
 <gml:description>Observation test instance - time series</gml:description>
 <gml:name>Time series 1</gml:name>
 <om:samplingTime>
  <gml:TimePeriod gml:id="ts1t">
   <gml:beginPosition>2005-06-17T09:00:00+08:00</gml:beginPosition>
   <gml:endPosition>2005-06-21T09:00:00+08:00</gml:endPosition>
  </gml:TimePeriod>
 </om:samplingTime>
 <om:procedure xlink:href="urn:ogc:object:feature:Sensor:BOM:t_2a"/>
 <om:observedProperty
xlink:href="http://sweet.jpl.nasa.gov/ontology/property.owl#Temperature"/>
 <om:featureOfInterest xlink:role="urn:ogc:def:featureType:OGC:Station"
xlink:href="http://my.big.org/feature?type=station%26name=st1"/>
 <om:parameter>
  <swe:Quantity
definition="http://sweet.jpl.nasa.gov/ontology/property.owl#Elevation">
   <swe:uom xlink:href="urn:ogc:def:uom:UCUM:m"/>
   <swe:value>3.45</swe:value>
  </swe:Quantity>
 </om:parameter>
 <om:result xlink:href="http://www.flakey.org/opendap/378.cdf"/>
</om:Observation>
```

10.7.2 Accessing small pieces of a large data set

When a Sensor Observation Service is based on a large data set, clients can still choose to select only pieces of that data set for each operation. However, using the current version of the Sensor Observation Service specification, it is not easy for the client to determine or control the amount of data that will be returned in response to a request. Information about the amount of data that a sensor produces (or has produced) can be encoded in the description of the sensor (the SensorML document returned in response to a *describeSensor* operation request), but this is implementation specific and problems with interoperability of different applications may arise.

Because of this the Sensor Observation Service itself should support mechanisms for efficiently accessing large data sets. There are different solution approaches:

- Allow the client to determine the amount of data a given request would produce in advance.
- Allow the client to limit the amount of data that should be transferred in one response.
- Allow the client to select between alternative data transfer mechanisms and result models (e.g. file transfer with files of a standard format) depending on statements of the server about the amount of data to be transferred.

10.8. Cascading Sensor Observation Services

Sensor Observation Services (SOS) as specified in section 8.2.2 can be used at many different places in applications built according to the SensorSA. Cascading of sensor observation services may sometimes be an effective approach for fulfilling certain requirements. When SOS instances are cascaded, an SOS acts as the data source for an intermediate cascading SOS, which itself provides a SOS interface to its clients. From an architectural point of view, a number of scenarios of using a cascading SOS are of interest and thus described in the following.

10.8.1 Data flow optimization

While on a conceptual level data is directly accessible from the service provider or data source, when engineering real-world applications some obstacles can hinder efficient direct usage of an SOS by a client. Problems that may occur include:

- Network performance problems
- Limited resources on SOS servers
- Support of different versions or feature sets of the SOS protocol in the client and server applications

To mitigate those negative effects without the need of changing servers and/or clients a cascading SOS can be used as an intermediate service for optimizing the data flow from the server to the client. Depending on the problem that should be solved the cascading SOS has to provide different functions, which are described in the following.

For decoupling the data flow from the server to the client, caching can be implemented on the intermediate SOS service. This reduces negative effects of limited network bandwidth, unreliable network connections, unreliable servers, or limited performance of the original SOS service.

When the implementation details of the SOS protocol (e.g. use of different SOS versions,) impose a problem on the interoperability between a specific client and a server, the most straightforward solution would be to change either the client or the server application (or both). If that is not possible or feasible for economic or other reasons, an intermediate SOS service can be used to translate between the unmodified client and server, and make interoperability between these systems possible.

From a network point of view the intermediate SOS can be placed near the original SOS server(s) or near the client(s) that are supposed to use it. The most effective location depends on the issues that are about to be solved in this specific application scenario and thus cannot be defined on a generic level.

10.8.2 Providing alternative views to data

When implementing applications according to the SensorSA situations may arise where using the data directly in the form provided by the data source may not be feasible or possible. Examples for such scenarios are:

- Different data providers may implement different data models for their Sensor Observation Services because of differences in their requirements and/or intentions. While this is still compliant with the SensorSA, it may impose an additional burden on applications that have to use those different data sources.

- Data models used by organizations internally may not be feasible or appropriate for publishing them or making them available for a specific purpose.

- Organisations may need to provide an aggregated view of data collected by different providers, e.g. for implementing federated data pools

Cascading SOS can be used to facilitate cleaner and more robust implementation architectures in these cases. The intermediate SOS server provides a single interface to all the underlying data sources. This results in a clean distinction between the data access and processing on the client side, and the aggregation, transformation and/or filtering of the data that is necessary for a specific purpose in the intermediate SOS.

10.8.3 Data (pre-)processing

In a sensor network data processing occurs on various occasions. The classical use case is pulling a data set from a service, processing it as required for the application scenario, and probably storing the result somewhere. This use case is described in detail in section 10.9. However, for some common, more lightweight data processing tasks the application scenario could be optimized by processing the data on the fly when they are accessed.

In such a scenario a cascading SOS acts as a service providing access to derived data without the need to first fetch all of the source data and applying the calculations. While not feasible for all types of data processing operations (e.g. lengthy calculations), it simplifies application architectures where it can be applied.

A typical scenario would be the calculation of mean values for time series data. While the measured data may be available with, for example, half-hour mean values from the sensors, an application may require daily mean values for its operation. This can be solved by using a cascading SOS that calculates the daily mean values on the fly using the half-hour mean values as the data source.

10.8.4 Multi-level sensor data storage

Some of the scenarios described in section 4.5 include SOS interfaces provided directly by the sensors, or by data loggers connected to the sensors. These devices typically are physically located in remote locations near the place where the observations are taken, and not in a typical data centre environment. When applications (e.g. GUI clients, data processing applications) would access theses devices directly, it would be very hard to meet requirements of those applications regarding availability, fault tolerance, performance, etc. In addition, those devices usually have tight constraints regarding storage space, which imposes problems for long time storage of observations.

To remedy these problems a cascading SOS can be used as illustrated in Figure 10-25. It fetches and stores the data provided by the sensors or data loggers, and all client applications

access this service instead of accessing the sensors directly. The cascading SOS can be located in a data centre where it is much easier to meet availability and performance requirements. Long term data storage is also easier to implement in that scenario.

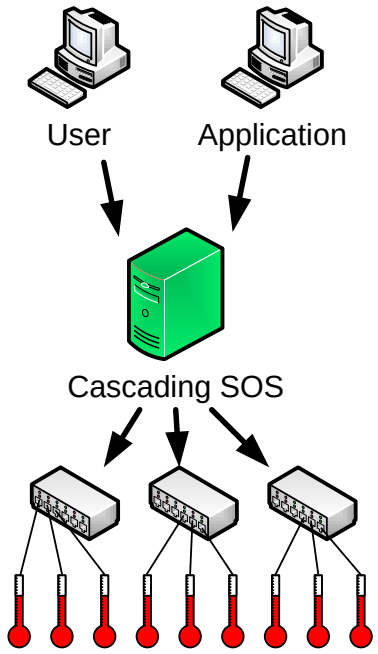


Figure 10-25: Multi-level sensor data storage

10.8.5 Caching of data

In most of the previously described scenarios, caching of the data from the source SOS at the cascading SOS is either a primary aspect or at least a "nice to have" feature. For this caching process different approaches can be taken. Depending on the requirements of a specific application every approach has its benefits and weaknesses, or may not be applicable at all. An approach for caching can be broken down into a few different aspects of its operation, which are described in the following.

The first distinction can be made on the source of the event that triggers the (re-)fetching of the data from the source SOS:

- Data retrieval from the source SOS can be triggered by the request that the client makes to the cascading SOS. At this moment, the cascading SOS has to decide whether the data that is available in his cache is valid. If it is invalid, the data has to be updated by reloading it from the source SOS

- The trigger of the (re-)fetching can be the source SOS itself. By using event-based interaction patterns (see section 6.3.3), it can notify the cascading SOS of new or updated data. The new data values can be included in the event notification itself, or the cascading SOS may fetch data from the source SOS using conventional SOS operations in response to the event.

- The (re-)fetching of the data can be triggered by events not depending on either the source or the client. An example would be a time schedule that controls when data is fetched from source SOS servers.

Another classification can be made on how to determine if data in the cache is still current, i.e. if the cache is still valid:

- The cascading SOS can query the source SOS if the data is still current. Currently the SOS specification does not foresee operations or metadata to support this approach in a generic way, but using the O&M and SOS specifications it can be realised if both the source SOS and the cascading SOS agree on a common way of implementing it.

- The data in the source SOS can contain information if the data is current. Depending on the granularity required, this can be either encoded in the SensorML description of a procedure if it remains the same for all observations made using this procedure, or it can be encoded using O&M together with the data values if each observation can have different constraints to determine if it is still current.

- It may be determined at the level of a cascading SOS implementation. If the source SOS does not support any information about how long its observations are current at all, it may be possible (depending on the application scenario) to define this at the level of the cascading SOS itself.

For updating data in its cache, a cascading SOS has to identify each observation. Since the current O&M and SOS specifications do not provide a generic identifier that can be used for this purpose, a work-around solution has to be implemented currently. The implementation approach depends again on the application scenario. An example of such a solution would be to use an artificial unique key to identify an observation, e.g. consisting of the result time, the sampling time, the identifiers of the feature of interest, observed property and procedure. In many sensor network scenarios this may be sufficient to identify an observation for caching purposes.

Another important aspect that has to be handled is the deletion of data. Although in many scientific applications data is not deleted but instead archived and "logically" replaced by "newer" values, there may be applications that require the ability to delete an observation. The current SOS specification does not support this, and thus again work-around solutions have to be implemented.

10.8.6 Event-based interaction in cascaded scenarios

The event-based architectural style as described in section 6.4 can also be applied to cascaded scenarios. The client accesses the cascading SOS in the same way as any other SOS service. All event-based interactions can thus be implemented in the same way. Similar to this, the cascading SOS is a client to its source SOSes, which means it can also interact with them using events as any other SOS client does.

Some of the approaches for the replication of data in a cascading SOS depend on the usage of event-based interaction patterns, but also other types of events as described in the event taxonomy (see section 6.4.4) are important when SOS services are cascaded.

- Sensor Available Event: depending on the implementation of a cascading SOS it may automatically include new sensors when they become available. In this case the cascading SOS has to act on the "Sensor Available" event in the sensor network and add the new sensor to it's configuration.

- Sensor Unavailable Event: some of the scenarios for cascading SOSes require that data in the cascading SOS is still available even if the source SOS is no longer active.

- Sensor Properties Changed: this requires the cascading SOS to update it's metadata accordingly

- New sensor data: depending on the replication strategy implemented in a cascading SOS (as described before), this event may force an update of the data in the cache of a cascading SOS.

10.9. Processing and Fusion Support

10.9.1 Processing Chains

10.9.1.1 Introduction

Processing in general and fusion in particular often follows a multi-step pattern. First the input data to be processed or fused must be discovered using meta-information that characterises these data and that is compatible with the processing algorithm to be used. Then the input data must be fetched from different places using the appropriate access methods and protocols. Next, the fetched input data must often be pre-processed to deal with unit and format conversion needed to match the inputs expected by the processing algorithm. At this point, the processing per se can be performed and outputs are produced. Those outputs must often be post-processed to again deal with unit and format conversion before storing the processing results. Then, the converted output data must be stored in various places using the appropriate access methods and protocols. Finally, data rendering could be performed in preparation for (later) visualisation.

In the SensorSA, this multi-step processing pattern is supported by a service processing chain. This processing chain is itself exposed as a service.

10.9.1.2 Processing Chain Service

Referring to the processing flow illustrated in Figure 10-26, the main process has three inputs and two outputs. The input data are fetched from three Sensor Observation Servers (SOS) and the processing results are stored in two SOS servers. All of the processing (i.e. pre-processing, main processing, and post-processing) is performed by instances of the Processing Service (see section 8.4.2).

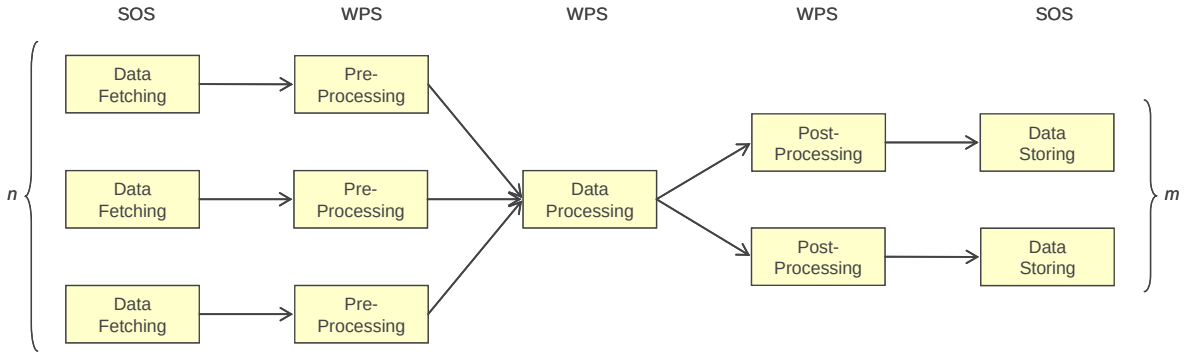


Figure 10-26: Processing Flow

The multi-step processing pattern described above can be implemented by an instance of a Processing Service (PS) called processing chain in Figure 10-27. To a client the processing chain exhibits a PS interface (front-end interface). As a back-end interface it uses a number of other services in order to execute the processing chain:

- The discovery of input data can be accomplished using a catalogue service (see section 8.2).
- The input data fetching and output data storing can be accomplished using a Sensor Observation Service (see section 8.2.2), a Feature Access Service (see Table 8-11) or an FTP service.
- The input data pre-processing and output data post-processing can be done using a Processing Service (see section 8.4.2).
- Finally, the data rendering could be achieved using a Map and Diagram Service (see section 8.4.2), e.g. for the generation of isolines/contours.

The processing chain is opaque (i.e. not modifiable by the user) and is likely to be implemented using BPEL. Whenever possible, i.e. mainly for data pre-processing and data post-processing, parallel execution is performed using the BPEL <flow> activity. This approach is expected to cover a wide range of processing needs with only moderate modifications to the BPEL source code.

All the inputs needed to access the individual services composing the chain must be provided as input to the processing chain. Temporary storage (e.g. an FTP server) is needed in order to store intermediate results that are passed (by reference) from one service to another. If each PS instance has its own FTP server to store its outputs, then the number of data transfers across the Internet can be reduced to its minimum (but cannot be eliminated). Nevertheless, the PCS must provide its own FTP server to store the outputs of the Processing Services that do not support stored outputs and to store its *execute* response which can be updated to provide process execution status information (e.g. percentage completion). Storing the *execute* response is the WPS mechanism to implement asynchronous process execution. To avoid running out of file storage space, some form of garbage collection must be implemented on the FTP server of the Processing Service instances underneath. For example, all output files older than a pre-defined time (e.g. 1 day) could be removed on a regular basis.

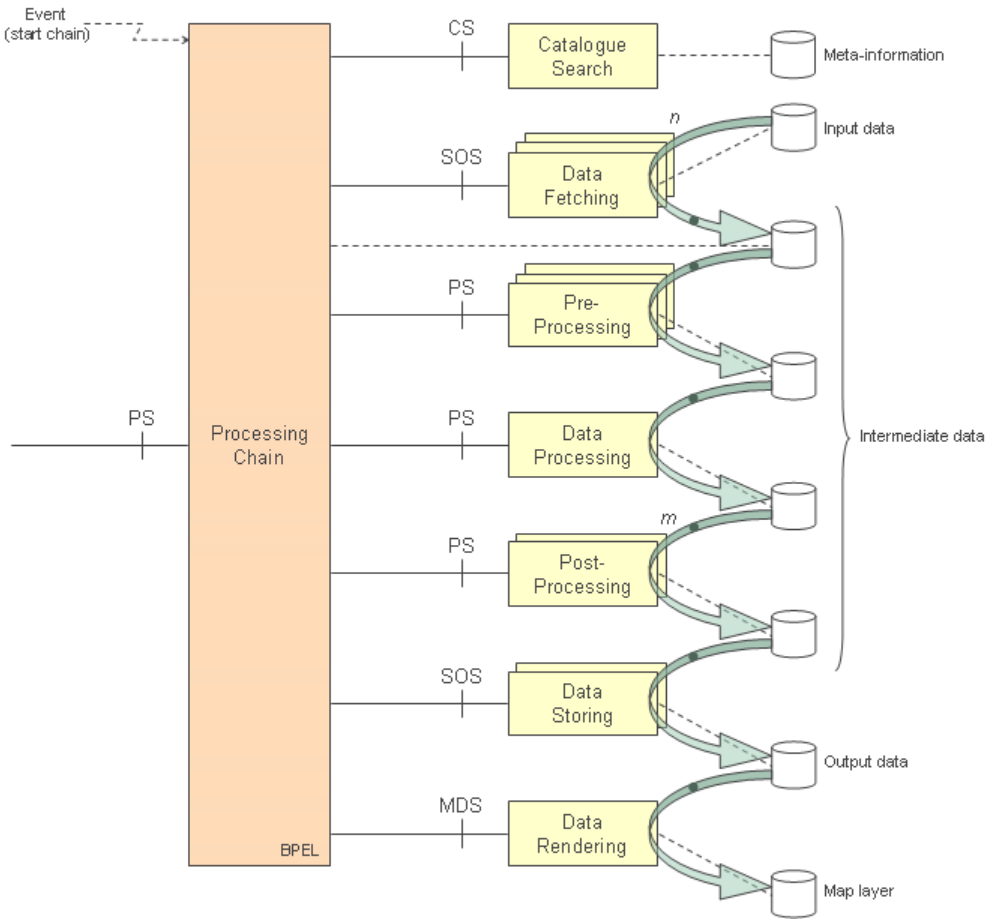


Figure 10-27: Processing Chain as an Instance of a Processing Service

10.9.1.3 Advanced Topics for Processing Chains

10.9.1.3.1. Continuous Feeding

One particular case of a processing chain arises when the input is a continuous flow of data (e.g. temporal fusion). In this case, the data fetching step must be repeated on a regular basis and the complete chain must be executed each time, producing new (incremental) results. This cyclic execution of the processing steps can be handled by the processing chain itself but the cycle period and stop condition (e.g. total number of cycles or total duration) must be provided in the processing chain *Execute* request as additional input parameters.

The cyclic execution of the processing chain assumes that all the services in the chain are able to operate incrementally i.e. using only the data fetched in the current processing cycle. It also assumes that all the processing can be completed during the cycle period i.e. before a new cycle begins.

However, there are cases where the main data processing step is stateful or simply requires data that was acquired in previous cycles. The main data processing step may then have to be designed to support incremental execution. This means that the service hosting this data processing must be able to create, save, and restore the context (algorithm state, data cache, etc)

needed to relate successive executions. A context identifier must therefore be assigned by the data processing service in the first cycle and be provided by the processing chain as an additional parameter in the *execute* request of the following cycles. Also, in order to initiate the data processing, the first cycle may require a much larger amount of data fetching (e.g. historical data). Finally, it is up to the data processing service to decide if it must cache data provided in previous processing cycles (e.g. for algorithm tuning or retraining).

Regarding the data fetching step, the SOS specification defines an optional *GetResult* operation (see section 8.2.2) that could be of interest in this continuous feeding use case (if supported by the SOS instances providing the data). It could be used in all processing cycles except probably the first one.

10.9.1.3.2. Event-Triggered Processing

It could be of interest to trigger the execution of the processing chain upon reception of a particular event. The processing chain is armed by the *Execute* operation but only really starts when the event is actually received. Depending on the option chosen, once the execution is complete the processing chain could automatically re-arm itself or require a new *Execute* operation. The information needed to define the triggering event (e.g. topic) and the stop condition (e.g. event count or event topic) must be provided in the processing chain *Execute* request as additional input parameters.

Although it is easy to imagine such an event-triggered processing chain, it is actually not straight forward to implement it. The natural and most efficient approach would be to have the processing chain passively waiting for the event to be pushed by the event producer. This means that the event must be addressed and delivered to a particular instance of the processing chain which cannot be done without support from the BPEL environment hosting the processing chain. The BPEL engine may support the invocation of an asynchronous service where the service is able to call back the instance of the BPEL workflow that made the service invocation. In this case, using WS-Addressing information in the SOAP header of the service request (e.g. ReplyTo and MessageId elements) and of the call back request (e.g. RelatesTo element), the BPEL engine is able to find the target workflow instance. However, if for example WS-Notification is used by the processing chain to receive event notifications, the Notification Producer or Notification Broker will not provide the correlation information needed by the BPEL engine to find the particular instance of the processing chain.

A workable but less efficient approach would be to have the processing chain actively polling for the availability of events and pull the event from a pull point as illustrated in the figure below. The polling interval could be specified in the processing chain *Execute* request as an additional input parameter.

The processing chain first requests the creation of a pull point from a PullPoint Factory. Then, the processing chain subscribes to a Notification Producer or Notification Broker and provides the topic(s) of interest as well as the end point reference of the newly crated pull point. Next, the notifications generated by the Notification Producer or Notification Broker are pushed to the pull point and can be retrieved (pulled) by the processing chain at polling time. By specifying the number of notification messages in the *GetMessages* operation, the processing chain may decided to pull one notification at a time (i.e. execute the complete processing chain for each notification). Alternatively, by not specifying the number of notification messages in the

GetMessages operation, the processing chain may flush all the notification messages accumulated by the pull point during the polling period.

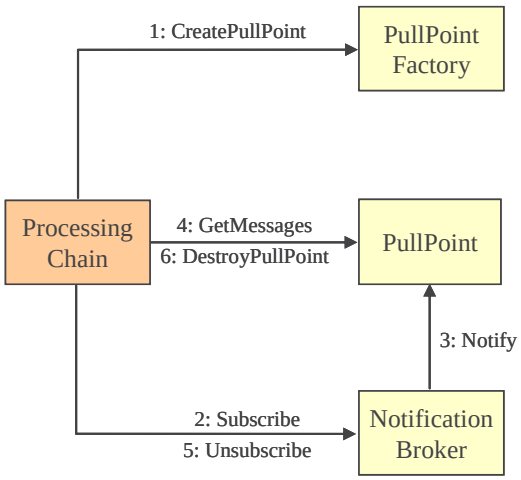


Figure 10-28: Reception of Notifications by Processing Chain Instance

When the processing chain stop condition has been reached (e.g. event count or particular event topic), the processing chain must unsubscribe to the Notification Producer or Notification Broker and then destroy the pull point.

10.9.1.3.3. Discovery

Meta-information is needed to discover input data that can be used for a processing chain, i.e. that are compatible with the processing algorithm used in the chain. Another approach could be to discover the Web Processing Service (WPS) to call as part of the processing chain to match available input data. This requires that appropriate meta-information about the Processing service be available in some catalogue. One possible way of providing information about the processing service is to describe it using SensorML (Botts, 2005).

The catalogue service (section 8.4.1) offers a broker mechanism that could be used to link processes/tasks of fusion services to data sets. These links could be established manually or automatically (during harvesting). The processing chain could use this mechanism to discover (from service to data set or from data set to service) the compatibility between fusion services and fusion data sets.

10.9.2 Uncertainty Handling in Processing Chains

The degree of uncertainty of the input data has a major influence on the reliability of the output data. Thus, within a process chain, information about the uncertainty of the data at each position in the chain has to be handled.

The propagation of uncertainty through the chain has to be estimated. If $y=f(x_1,..,x_n)$ is a multi-dimensional value (typically a state vector) computed as a function f of vectors $x_1,..,x_n$ (e.g. as an indirect measurement), then uncertainty in the input values leads to an uncertainty in

y. For linear functions f the variance of y can be calculated from the covariance matrix of the x_i. For non-linear functions it is estimated using a Taylor expansion of f (e.g. Barry N. Taylor and Chris E. Kuyatt, 1994).

Another similar approach for which efficient algorithms exist for non-linear f is to use an Unscented Kalman Filter to estimate the mean and covariance of y from the mean and covariance of the input vectors (Wan, E. and Van Der Merwe, R. 2001). The distribution of the input data is represented by a small number of so-called sigma particles and the processing function f is applied to these particles to compute the propagated sigma particles. The mean and covariance of the output y are weighted functions of the mean and covariance of the propagated sigma particles.

For more complicated relationships – e.g. when y is the output of a model based computation or algorithm – a specific sensitivity analysis will be required and the assumptions made will have to be documented or referenced in SensorML for evaluation by an expert.

10.9.3 Combining Earth Observation and In-situ data

10.9.3.1 Introduction

Combining (fusing) Earth Observation (EO) data with in-situ data is attractive, especially when they relate to the same phenomenon, because the two types of data present different but complementary inherent properties. In-situ data is typically of high quality and temporally rich (higher acquisition frequency) but spatially poor (limited number of sensors) whereas EO data is typically spatially rich (images covering a wide area) but temporally poor (lower acquisition frequency) and of lesser quality.

For example, in the geo-hazard domain (monitoring of soil and building displacements), vertical displacements obtained through interferometry processing of Advanced Synthetic Aperture Radar (ASAR) satellite images can be combined with vertical displacements measured in-situ using theodolite based monitoring systems. In this case, the acquisition frequency of in-situ displacement data is typically of once every 30 minutes as opposed to once every 35 days for satellites images. The accuracy of the in-situ displacement data is typically of ± 1 mm versus ± 3 mm to ± 5 mm for EO derived displacement data. However, interferometry processing of ASAR images may lead to an average density of about 500 points per Km^2 (in urban areas) as opposed to only about 150 per Km^2 for in-situ monitoring stations.

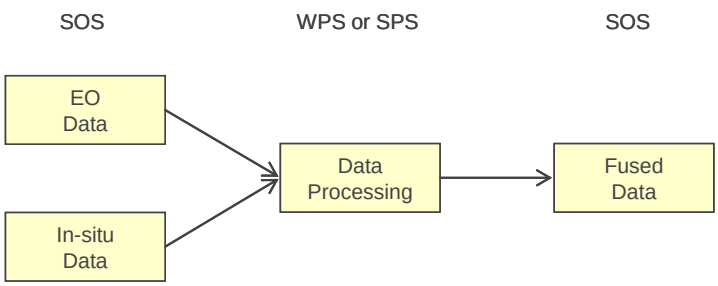


Figure 10-29: Combining Earth Observation and In-situ Data

In the context of SensorSA, the combination of EO and in-situ data is best performed using a Web Processing Service (WPS) (see section 8.4) or Sensor Planning Service (SPS) (see section 8.2.3) to access EO and in-situ data stored in Sensor Observation Services (SOS) and store the processed (fused) results also in a SOS as illustrated in Figure 10-29.

10.9.3.2 Approach

The typical steps needed to deploy a SOS containing EO derived data are as follows:

1. First, EO images (EO data products) covering the area of interest and period of interest have to be ordered/purchased from a satellite (or air-borne) image provider. These images are in a certain format and are the result of instrument specific processing (performed by the EO image provider) to reformat, time re-order, calibrate, and geo-locate the raw data.

2. The EO images are then submitted to thematic processing (manual or semi-automatic) to extract the observations related to the desired phenomenon with the appropriate level of quality.

3. Finally, these observations are stored in an SOS server along with the associated quality meta-information and SensorML descriptions.

*In the geo-hazard example considered above, **high resolution** ASAR images produced by the ENVISAT satellite and covering the area of interest (e.g. city of Barcelona) can be purchased from Spot Image. Using rather complex interferometry processing, ground displacements maps (interferograms) can be generated by comparing all the images to one of those images chosen as a reference and by making topographic adjustments using a digital elevation model (DEM). Stable reflector points (permanent scatterers) presenting a good signal/noise ratio (reflectivity) can then be extracted and their displacements can be stored (along with the associated quality metada and SensorML description resulting from all the above processing) in a database that is directly accessed by a SOS server.*

The typical steps needed to deploy a SOS containing in-situ data are as follows:

- Acquire in-situ data in the area of interest for the period of interest and, after possible sensor specific processing, store this data in a database along with its quality parameters. This data is geo-located either directly or indirectly through the location of the sensors.

- Publish the in-situ data with the associated quality meta-information and SensorML descriptions in a SOS server by accessing the acquisition database directly or by feeding/inserting the data into the SOS data store.

In the geo-hazard example considered above, automated monitoring systems, combining a theodolite (measuring angles) and a distance measuring device aiming at prismatic optical targets attached to structures (e.g. buildings) to be monitored, can be used to measure the movement (in X, Y, and Z directions) of the targets on a cyclic basis (e.g. every 30 minutes). After filtering for night and day structure breathing and vibrations due to traffic, the acquired target displacement data can be stored in a database along with the location of these targets. This database can be accessed directly by a SOS server supporting two result models: one for temporal coverages and one for spatial point coverages.

Once the EO and in-situ data is available from SOS services, a WPS or SPS based data processing (e.g. fusion) service can be used to implement the processing chain shown above in Figure 10-29. The processing service will typically use the quality meta-information found in both EO and in-situ SOS services to judiciously combine their observations and produce new observations with better spatial and/or temporal coverage and quality. These new observations can be stored in an SOS (could be one of the input SOS services) along with uncertainty information which is essential for decision making.

In the geo-hazard example considered above, inverse distance interpolation (spatial fusion) limited to a given radius (e.g. 30 m) can be applied to improve the accuracy of the EO derived vertical displacement observations at the selected stable reflector points. Statistical information (uncertainty) related to the interpolated displacements can also be generated and published along with the updated observations in a Fused SOS server with the same structure as the EO and in-situ SOS servers.

10.10. Integration of Mobile Sensors

Of the sensor network scenarios shown in Table 4-1 in section 4.5, all but one involve mobile sensors:

no. 2. Mobile sensors and fixed or mobile data logger
no. 3. Mobile sensors moving in different service networks
no. 4. Mobile sensor cluster on vehicles (e.g. on ships) - block data transfer on demand
no. 5. Mobile earth observation sensors (satellite, airborne)
no. 6. Mobile sensors with their own IP address

For a mobile sensor the location of the sensor is time dependent. In addition, the associated sampled feature and / or sampling point of a feature of interest are normally time dependent.

For scenarios 2-4, observation data is transferred from the sensor to a “data logger”. For simplicity, it is assumed that each data logger has exactly one associated SOS instance where the observations are published. The protocol between the sensors and data logger is proprietary and outside the scope of SensorSA. The data logger shall register its SOS instance with a catalogue service. The result of a SOS *getCapabilities* request to the data logger provides a list of sensors associated with the data logger and for what sampling or result times observations are available. The catalogue service can subsequently use *describeSensor* operations to acquire information about the sensors associated with a data logger. The data logger may have no or only partial knowledge about which sensors are still alive. A catalogue service may compile information as to the location of sensors in order to support network management.

Scenario 3 differs from scenario 2 in that a sensor may transfer its observation data to several different data loggers, and hence to several SOS instances. If the observations of a given sensor relate to the same feature of interest, then there are two approaches to dealing with this in applications requiring all data for this feature:

- A cascading SOS may be employed to merge the observations from the different SOS instances of the data loggers. Thus applications may need to access only the cascading SOS. This assumes that the cascading SOS has the necessary knowledge of relevant underlying SOS instances.

- A catalogue has knowledge about available observations for a feature of interest including references to the pertinent SOS instances. A client application first queries the catalogue for the SOS instances and subsequently the individual SOS instances with *getObservations*.

A special case in scenario 3 arises when a sensor transfers the same observation data to several data loggers – this may be done intentionally for reasons of redundancy and reliability, or may happen by accident depending on the underlying protocols. A cascading SOS or client application can detect and remove duplicate observations based on the combined reference to the procedure (sensor), feature and sampling time.

Scenario 4 is actually a particular case of scenario 3, whereby the time required to make observations available in an SOS instances can be considerably longer. This has consequences for applications as they may need to wait for a certain period before data processing is started or can be completed.

In scenarios 4-6 especially, the usage of a SPS is recommended to task the sensor deployment and to obtain information about what observations are feasible and when they will be available (cf. section 8.2.3).

In scenario 5, the mobile sensor platform may have its own integrated SOS instances, or it may communicate with a base station where the SOS instance is located. There is an obvious parallel to scenario 3 in that observations may become available at a later time.

In scenario 6, it is assumed that the sensor has its own SOS instance. Such sensors therefore play a similar role in SensorSA as the data loggers described above.

10.11. Event Handling

OGC defines the Sensor Alert Service interface (see its description in section 8.2.4). The Sensor Alert Service (SAS) is a service that combines the most important aspects described in sections 6.3.3 and 6.4. The SAS specification allows the setup of simple alert services that inform clients about interesting events. Those simple services can then be chained to more complex scenarios. Figure 10-30 illustrates such a possible SAS orchestration. Sensors publish observation data, and SAS instances process the data, generating new alerts that will be used by SAS instances further down the processing/orchestration chain. The events generated by the last SAS instances might integrate data from various sources. The path back to the original sensors does not have to be traceable. To make the functionality of SAS more transparent and to reflect the recent requirements identified in SANY, this version of the architecture specification will focus on non-orchestrated alert services.

Though providing most of the features of event-based interaction models (see section 6.3.3), the SAS is a Web Service interface specification. The notification transport is limited to the eXtensible Messaging and Presence Protocol (XMPP), exclusively. SAS is not a fully featured event system, though it supports a number of basic requirements. The OGC event notification system consists of event producers, notification services, notification brokers that match incoming data with event subscriptions, and notification consumers. The SAS implements both the notification service as well as notification broker functionality. In combination with the Web Notification Service (WNS) (see its description in section 8.2.5), OGC provides a system

that allows message delivery to any form of communication endpoint, but lacks a number of security and maintenance features.

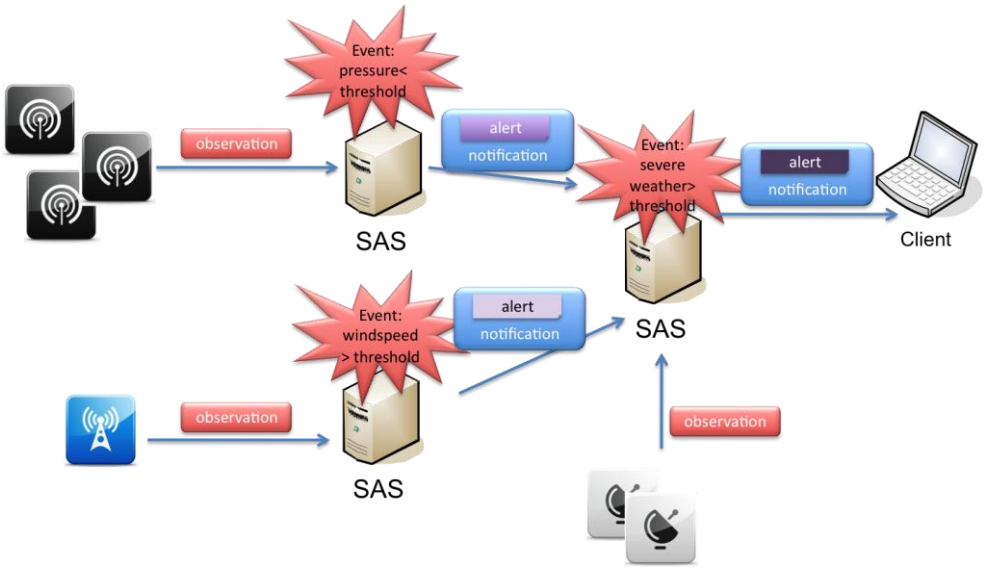


Figure 10-30: Event handling using OGC Sensor Alert Services

In the following, we will illustrate the principle modes of operation of SAS as used in SANY. Events can be discovered at various levels, for example by sensors themselves, or during processing of reported observation data by the Sensor Alert Service. The following figures illustrate the various cases.

10.11.1 Definition and Subscription of Events

Initially, two scenarios must be differentiated, as illustrated in Figure 10-31 and Figure 10-32. Figure 10-31 illustrates the simplest scenario. Sensors define events and advertise them to an SAS instance. Those event-types will then be advertised by the SAS. Clients can subscribe to those events exclusively. As an example, the sensor triggers events if the temperature exceeds 10°C. The clients can subscribe to "temperature exceeds 10°C" exclusively. Other commonly used examples of predefined events are "battery low" or "observation failure".

.

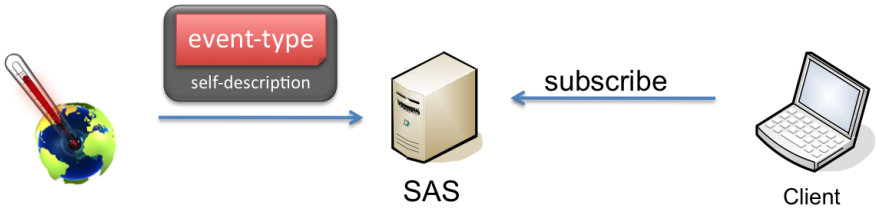


Figure 10-31: Clients subscribe to sensor-defined event types

Figure 10-13 illustrates the second scenario. Sensors don't define event-types, but advertise observation data to the SAS. SAS will advertise these data to clients. Clients are now free to define their own events based on the observation data. As an example, a sensor offers

temperature observations at a certain location in degree Celsius. The client may define an event as an observation with a result greater than 20°Celsius.

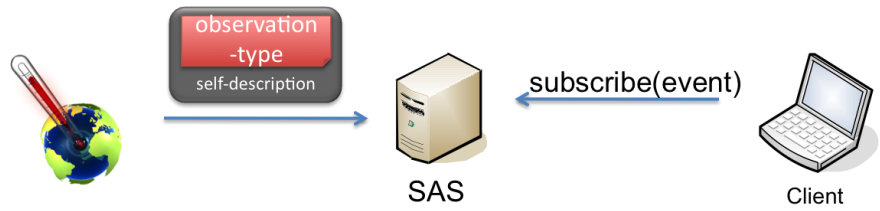


Figure 10-32: Clients define events based on observation offerings by sensors or SAS respectively

The next scenario, illustrated in Figure 10-33, describes a combination of the two base types described above. Here, a single sensor or any number of sensors push data to the SAS instance. Independently of the type of incoming data (either observation results or event notifications), the SAS instance may define and advertise new event-types. Clients can then register those event-types. The SAS will process all incoming data to detect the type of event it advertises. An example would be an SAS that advertises "storm warning" events. The "storm warnings" are detected based on data coming from a number of meteorological sensors.

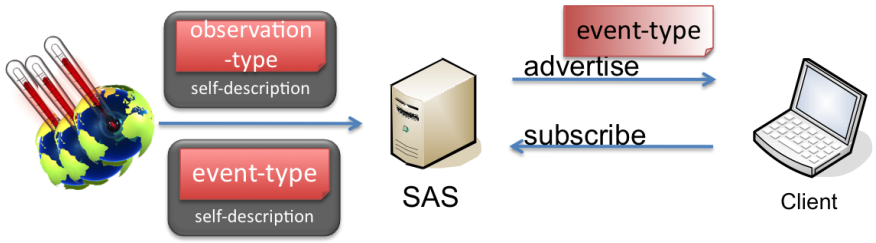


Figure 10-33: SAS defines event types based on various incoming data sets. Clients subscribe to those events

10.11.2 Generation and Dispatching of Alerts

Alerts are generated and dispatched either directly at the sensor (Figure 10-34), or based on incoming observation data from the sensors at SAS (Figure 10-35).

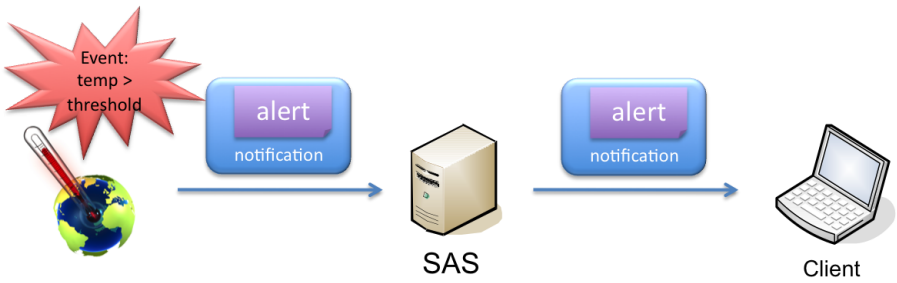


Figure 10-34: Event detection and alert dispatching at sensor

The detection of events directly at the sensor requires sensors that allow the processing of observation results. Often, sensors simply observe physical properties and report the observation data to a SAS instance, as illustrated in Figure 10-35. This situation has the advantage that the

SAS does all processing of incoming data (leading to simpler sensors with less processing power and corresponding energy consumption) and SAS can offer all data in an arbitrary format to clients, e.g. mean values in an area where individual sensors don't even know of each other.

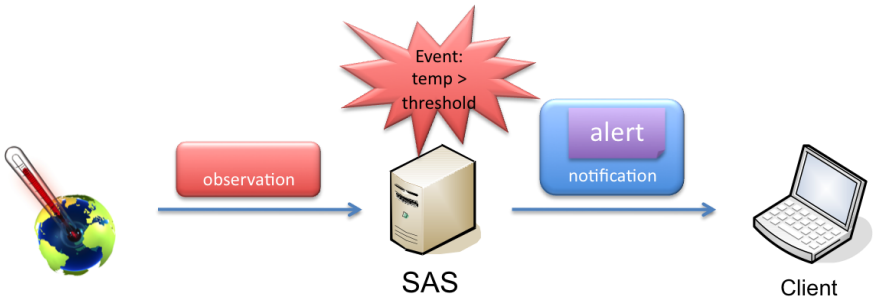


Figure 10-35: Event detection and alert dispatching by SAS

Figure 10-36 illustrates the flexibility of SAS. Even though events are detected at sensor level and dispatched to SAS, it is up to the SAS provider to offer any other kind of alert. The SAS alert might take other sensor data into account, or the SAS queries other services for additional data and generates alerts based on provider-specific algorithms. Theoretically, the alerts dispatched by an SAS instance can be based on any type of incoming data, internal calculation, modelling etc.

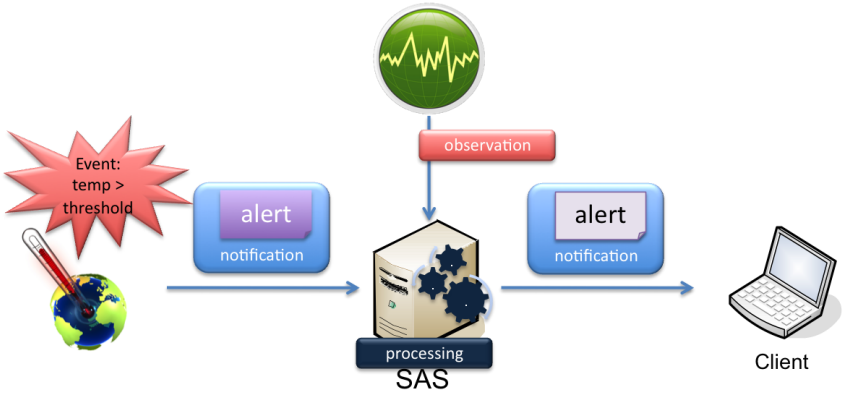


Figure 10-36: Event detection at the sensor level with conversion of alerts at SAS

10.12. Plug-and-measure Support

"Plug and Measure" refers to the degree of capability to plug a sensor into an operating station computer, begin measurements and access its observations through services without additional manual intervention, e.g. a restart of the computer. Therefore, plug-and-measure functionality must be embedded in all functional domains. When looking at the sensor scenarios described in section 4.5 the necessity and advantages of the plug-and-measure functionality become obvious. Within all scenarios, regardless of whether in-situ or mobile sensors are involved, the main advantage is easy deployment and seamless integration of additional sensors in existing networks. Independent of the sensor connection technology (wired or wireless) all of these scenarios have a measurement station (central data acquisition point).

In the following, some of the major steps in supporting plug-and-measure functionality are described. They are illustrated in Figure 10-37.

10.12.1 Sensor Plug In

The concepts and mechanisms described in the following concentrate on the deployment of a new smart sensor in a sensor network controlled by a station computer, where "new smart sensor" means a new instance of a sensor type initially unknown to (i.e. not registered with) the measurement station. A smart sensor is considered to be a sensor that provides a certain amount of processing and storage ability that can be connected to the station, meaning that both sensor and station computer have to support the same communication technology (e.g. USB, ZigBee, IEEE1451, CAN-BUS) and the necessary hardware and software layers have to exist in order to enable simple byte stream communication between the two.

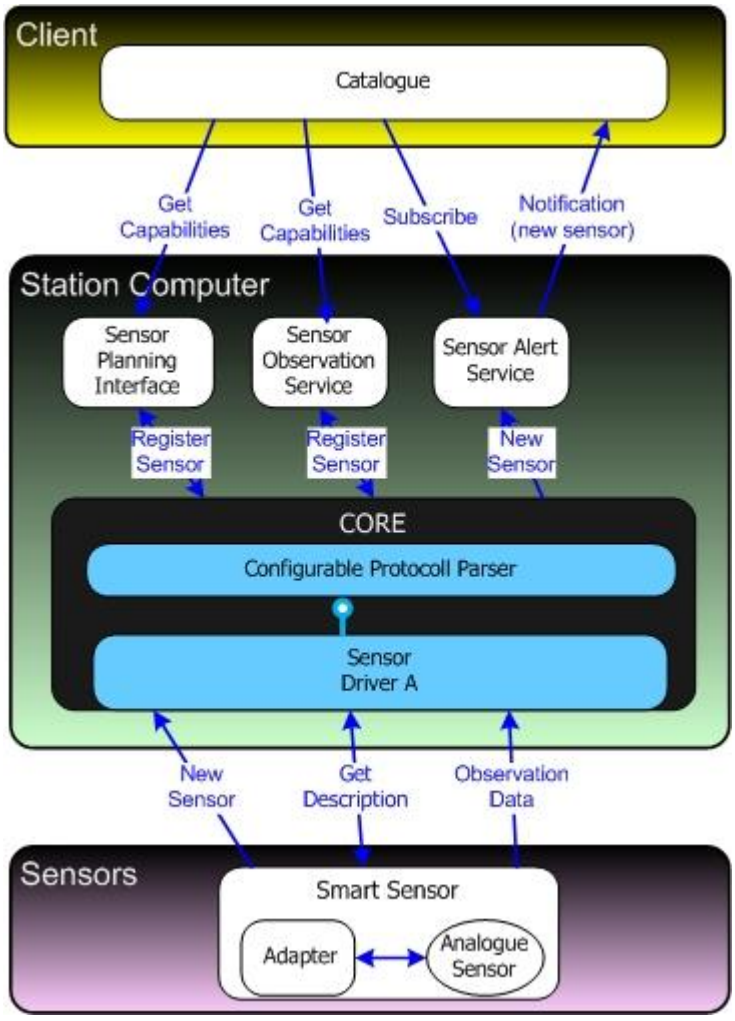


Figure 10-37: Plug-and-measure Component Interaction

Upon plug-in (connection on the hardware level) the application layer on the measurement station has to become aware of the new sensor (e.g. through a notification mechanism). Considering the rather large number of existing technologies at this level it is impossible to provide a generic approach to how this should be accomplished. Two prominent patterns can be identified though:

1. The “sensor triggered plug-and-measure” pattern implies that the station computer passively listens for new sensors and is notified when a new sensor is connected. To illustrate this consider the case of a USB sensor connecting to the USB subsystem of the station. Upon connection the USB subsystem fires an interrupt that is propagated through various software layers, eventually translated into a notification and caught by the station computer sub-process handling new sensors.

2. The “station triggered plug-and-measure” pattern implies that the station actively polls for new attached devices. This can easily be imagined for the case of simple bus technologies like CAN-BUS.

10.12.2 Sensor recognition and connection establishment

To enable the station to communicate with the sensor, the sensor must be able to provide type information. We make no assumptions here about how this is realised or about the format of the sensor type information. For example, USB devices are providing complex meta-information describing the device (e.g. device identifier, manufacturer name, interfaces etc.). The station needs knowledge about the sensor type in order to be able to load and use the appropriate software component that enables application layer level communication (byte stream) between the station and sensor. As soon as application layer level communication is possible further sensor information can be retrieved by an appropriate software component.

The sensor description can be stored on the sensor (much like the IEEE1451 TEDS) or on a local or remote sensor description repository. The sensor description shall be encoded as a SensorML (Botts, 2005) document and contain the information necessary to enable the station computer to configure the sensor, initiate measurement tasks and retrieve observation data. The information encompassed in a sensor description is dependent on the use-case and the specific station computer and sensor implementation. Such information might include a description of the software protocol by which the sensor and the station communicate. The protocol description can be interpreted by a so called generic de-serializer component running on the station computer and enabling packet based communication with the sensor over the existing byte stream. Moreover, parts of the SensorML description are process descriptions. An example of a process is the conversion from raw observation values into engineering units, taking into account sensor calibration and decoration of observation data with units of measurement or annotation.

10.12.3 Sensor Adapters

Simple sensors with analogue inputs and outputs usually do not fulfil the abovementioned requirements on interface, processing and storage. This kind of sensor requires a “Smart Sensor Adapter” device that enables the plug and measure functionality. An example of such a device is shown in Figure 10-38. It interfaces a simple sensor with analogue I/O (right side) with the station computer (left side). It enables plug and measure capability by providing a digital interface with the station computer (e.g. USB) and providing, on demand, a SensorML description of the sensor. A composite of a simple sensor and a plug-and-measure adapter is treated as an entity when described in SensorML. The SensorML description is initially transferred to the adapter and deployed together with the sensor.

Figure 10-38: Smart Sensor Adapter

10.12.4 Sensor Access through Service Interfaces

The SensorSA recommends to expose a new sensor through sensor planning (SPS), sensor observation (SOS) and sensor alert service (SAS) interfaces as defined in the OGC Sensor Web Enablement (see section 8.2). Every measurement starts by submitting a task via the SPS interface. In order to be able to task a measurement the sensor must be registered with an SPS instance. This means on one hand that the sensor must be assigned a unique ID within the scope of the station. On the other hand two documents have to be provided and mapped to the sensor ID: the sensor tasking description (an XML document returned upon invocation of SPS *DescribeTasking* operation describing the parameters needed to task a specific sensor) and a SensorML description of the sensor. Further, if the station exposes observation data through an SOS interface the sensor description shall be also registered with the SOS instance.

10.12.5 Publish plug-and-measure related information

The information about the new sensor can also be propagated to other systems, for example to a catalogue service. There are two main ways to accomplish this depending on the interaction model that is being applied (see section 6.3). The first solution, applying the request/reply interaction model, is to have the catalogue initiate harvesting of station capabilities and discovery of new sensors. The second solution, applying an event-based interaction model, is to arrange for the catalogue to subscribe to notifications from the station when new sensors are registered. In more detail, this can be accomplished by implementing the SAS interface on the station and offering notifications about events concerning new sensors, having the catalogue acting as an SAS client and subscribing to these notifications. Depending on the implementation, the notification might contain a new sensor description or just information that a sensor has been added to the station. Depending of the notification's content, the catalogue might start a harvesting process on the station or just propagate the new sensor information provided in the notification.

11. References

The following references are used as background documents. They are categorised as normative references (i.e. ISO Standards or respective drafts) or other technical or scientific documents and books.

11.1. Normative references

ISO/IEC 7498-4:1989	Information processing systems -- Open Systems Interconnection -- Basic Reference Model -- Part 4: Management framework
ISO/IEC 10746-1:1998	Information technology - Open Distributed Processing – Reference model
ISO/IEC 10746-2:1996	Information technology - Open Distributed Processing – Foundations
ISO/IEC 18023:2006	Information technology -- SEDRIS
ISO 19101:2004	Geographic information -- Reference model
ISO/TS 19103:2005	Geographic information -- Conceptual schema language
ISO 19107:2004	Geographic information -- Spatial schema
ISO 19108:2004	Geographic information -- Temporal schema
ISO 19109:2005	Geographic information -- Rules for application schema
ISO 19111:2003	Geographic information -- Spatial referencing by coordinates
ISO 19112:2003	Geographic information -- Spatial referencing by geographic identifiers
ISO 19115:2004	Geographic Information -- Metadata
ISO 19119:2005	Geographic Information -- Services
ISO 19123:2005	Geographic Information -- Schema for coverage geometry and functions
ISO 19136: 2007	Geographic Information -- Geography Markup Language (GML)
ISO/IEC 27002:2005	Information technology -- Security techniques -- Code of practice for information security management

11.2. Documents and Books

Allen, E., Edwards, G. and Bédard, Y. , 1995

Spatial Information Theory. A Theoretical Basis for GIS", Springer Berlin / Heidelberg: Lecture Notes in Computer Science, pp. 397-412, 1995.

Botts M., Percivall, G., Reed, C., Davidson, J., 2006

OGC White Paper: OGC® Sensor Web Enablement: Overview And High Level Architecture. Version 2.0. OGC 06-050r2. 2006-07-19

Botts, M., 2005 Sensor Model Language Version 1.0.0. OGC Document 07-000, http://portal.opengeospatial.org/files/?artifact_id=21273, 2007

CAFÉ, 2008 CAFE, Clean Air for Europe, http://ec.europa.eu/environment/air/index_en.htm, http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2008:152:0001:0044:EN:PDF, 2008.

Chandy, M., Schulte, R., 2007

'What is Event Driven Architecture (EDA) and Why Does it Matter?', Technical report, California Institute of Technology and Gartner Inc.

Cox, S., 2007 (Ed.) Observation & Measurements - Part 1: Observation Schema. OGC Document 07-022r1, approved as OpenGIS® specification, http://portal.opengeospatial.org/files/artifact_id=22466&version=2 , December 2007

Dictionary, 2004 Dictionary of the English Language, Fourth Edition. Houghton Mifflin Company, 2004. 05 Feb. 2007 at Dictionary.com

Drecki, I, 2002 Visualisation of uncertainty in geographical data. In: Spatial data quality. London: Taylor & Francis. pp. 140-159.

ESA SSE, 2007 Service Support Environment - Architecture, Model and Standards. White Paper of the European Space Agency, December 2004, http://earth.esa.int/rtd/Documents/SSE_Whitepaper_2.pdf

Everding and Echterhoff (Eds.), 2009

Everding, T. and Echterhoff, J. (Eds.). OWS-6 SWE Event Architecture Engineering Report, Version 1.0.0, OGC Document No 09-032, 2009.

Fielding, T. R., 2000 Architectural Styles and the Design of Network-based Software Architectures, Dissertation University of California, Irvine, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

GEOSS 1000R, 2005 GEOSS 10-Year Implementation Plan, Reference Document, GEO 1000R / ESA SP-1284, February 2005, Editor: Bruce Battrick, http://www.earthobservations.org/documents/10-Year%20Plan%20Reference%20Document.pdf

GEOSS 100S, 2007 The First 100 Steps to GEOSS, 30 November 2007, Editor: GEO Secretariat,http://www.earthobservations.org/documents/the_first_100_steps_to_geoss.pdf

GEOSS CAIR, 2007 GEOSS Core Architecture Implementation Report, GEO Architecture and Data Committee, 6 November 2007, Editors: George Percivall & Ingo Simonis, http://portal.opengeospatial.org/files/?artifact_id=24315

GEOSS AIP CFP, 2008 Architecture Implementation Pilot (AIP), Phase 2: IOC Augmentation - Call for Participation (CFP), GEO Task Team AR-07-02, 26 June 2008, http://www.earthobservations.org/documents/aip/20080626_geo_aip2_call_for_participation.pdf

GMES FR, 2004 Global Monitoring for Environment and Security (GMES): Final Report for the GMES Initial Period (2001-2003) http://www.gmes.info/action_plan/index.html

GMES, 2005 GMES reflection paper on Data Integration and Information Management Capacity, DG-INFSO, Draft 6, July 2005

Hadley, M. J., 2006 Web Application Description Language (WADL), Sun Microsystems, https://wadl.dev.java.net/wadl20061109.pdf, November 2006

Hengl, T. 2003 Visualisation of uncertainty using the HIS colour model: Computations with colours. In: Proceedings of the 7th International Conference on GeoComputation, Southampton, United Kingdom.

Hilbring and Schleidt, 2009

Hilbring, D. and Schleidt, K. Automatic creation of INSPIRE related metadata from Sensor Web Enablement Services. AGILE 2009 Pre-Conference Workshop "Challenges in Geospatial Data Harmonization", http://www.esdi-humboldt.eu/events/agile2009.html#papers, 2009.

ISO GUM 1993 BIPM, IEC, IFCC, ISO, IUPAC, IUPAP, OIML. Guide to the Expression of Uncertainty in Measurement. International Organisation for Standardization, Geneva, Switzerland. ISBN 92-67-10188-9, First Edition 1993.

JOW, 2007 Joint Operability Workshop Report "Towards a single information space for Environment in Europe", Frascati, 3 April 2007

Kanneganti R. and Chodavarapu P., 2008

SOA Security, Manning Publications Co., 2008.

Langran, G., 1992 Time in Geographic Information Systems, Taylor & Francis Ltd., London.

Luckham and Schulte (Eds.), 2008

Luckham, D., Schulte, R. (Eds.). Event Processing Glossary - Version 1.1. Event Processing Technical Society, http://www.epts.com/component/option,com_docman/task,doc_download/gid,66/Itemid,84/ . July 2008. Downloaded on July 5, 2009.

MacEachren, A., 1992 Visualizing uncertain information. In: Cartographic Perspective, Number 13, Fall, pp. 10-19, 1992

Muehl, G., Fiege, L., Pietzuch, P.R., 2006

Distributed Event-Based Systems. Springer Verlag, Berlin. ISBN-10: 3540326510. Juli 2006.

Naveen, S., Wagner, D., 2004

Security Considerations for IEEE 802.15.4 Networks, University of California, Berkeley.http://www.cs.berkeley.edu/~nks/papers/15.4-wise04.pdf

OASIS 2003 Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1, http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf, 2003

OASIS 2004 XACML Profile for Role Based Access Control (RBAC), Committee Draft 01, 13 February 2004, http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf, 2004

OASIS 2005 eXtensible Access Control Markup Language (XACML) Version 2.0, OASIS Standard, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, 2005-02-01

OASIS 2006 Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), OASIS Standard Specification, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 2006

OASIS 2006b Security Assertion Markup Language (SAML) V2.0 Technical Overview Working Draft, http://www.oasis-open.org/committees/download.php/20645/sstc-saml-tech-overview-2%200-draft-10.pdf 10, 9 October 2006,

OGC 03-040 Open Geospatial Consortium Doc. No. 03-040. OGC Reference Model, V0.1.2 , http://portal.OpenGIS.org/files/?artifact_id=3836, 2003-03-04

OGC 06-023r1 Open Geospatial Consortium Doc. No. 06-023r1. Definition identifier URNs in OGC namespace. OGC™ Best Practices Paper, Version 1.0.0, Editor: Arliss Whiteside, 2006-08-08

OGC 06-004r4 Vowles, G. (Ed.). Open Geospatial Consortium Abstract Specification 06-004r4: The OpenGIS® Abstract Specification Topic 18: Geospatial Digital Rights Management Reference Model (GeoDRM RM). Version: 1.0.0. 2006-12-29

OGC 06-103r3 Open Geospatial Consortium Inc.(OGC 06-103r3): OpenGIS® Implementation Specification for Geographic information – Simple feature access – Part 1: Common architecture, Version: 1.2.0, Date: 2006-10-05, http://portal.opengeospatial.org/files/?artifact_id=18241

OGC 07-026r2 Open Geospatial Consortium Inc. (OGC 07-026r2): Geospatial eXtensible Access Control Markup Language (GeoXACML), Version 1.0, Date: 2008-02-20, http://portal.opengeospatial.org/files/?artifact_id=25218

OGC 07-098r1 Open Geospatial Consortium Inc. (OGC 07-098r1): Geospatial eXtensible Access Control Markup Language (GeoXACML) – Extension A – GML2 Encoding, Version 1.0, Date: 2007-11-16, http://portal.opengeospatial.org/files/?artifact_id=25219

OGC 07-099r1 Open Geospatial Consortium Inc. (OGC 07-099r1): Geospatial eXtensible Access Control Markup Language (GeoXACML) – Extension B – GML3 Encoding, Version 1.0, Date: 2007-11-16, http://portal.opengeospatial.org/files/?artifact_id=252120

OGC 08-009r1 Open Geospatial Consortium Inc. (OGC 08-009r1): OWS 5 SOAP/WSDL Common Engineering Report, Version 0.1.0, Date: 2008-01-16, http://portal.opengeospatial.org/files/?artifact_id=26521

OMG, 2008 Alert Management Service, Revised Submission. Answering OMG RFP c4i/04-11-13. OMG document c4i/2008-02-01

ORCH-AbstrServ, 2007 WP3.4 OA Service Abstract Specifications. Deliverables D3.4.x Integrated Project 511678 ORCHESTRA. Editor: Environmental Informatics Group (EIG). http://www.eu-orchestra.org/publications.shtml#OAspecs, October 2007

ORCH-ImplServ, 2007 WP3.6 OA Service Implementation Specifications. Deliverables D3.6.x. Integrated Project 511678 ORCHESTRA. Editor: Environmental Informatics Group (EIG). http://www.eu-orchestra.org/publications.shtml#OAspecs, October 2007

Ricker, S., Havens, J., 2005

Sensor Fusion. Theory and Application. Technical Report. Unpublished Material.

Richardson, L., Ruby, S., 2007

RESTful Web Services. O'Reilly Media, Inc.. ISBN-10: 0-596-52926-0. 2007

RM-OA, 2007 Usländer, T. (Ed.) Reference Model for the ORCHESTRA Architecture Version 2 (Rev. 2.1). OGC Best Practices Document 07-097. http://portal.opengeospatial.org/files/?artifact_id=23286, October 2007

Schimak and Watson (eds.), 2008

Schimak, G. and Watson, K. (eds.). SANY Technical Requirements. SANY deliverable D1.3.1.1, http://sany-ip.eu, 2008.

Simonis, I., 2007 (Ed.) OGC® Sensor Alert Service Implementation Specification V0.9, OGC Engineering Specification (status: pending) 06-028r5, http://portal.opengeospatial.org/files/?artifact_id=24780&version=1, October 2007

Simonis, I., 2008 (Ed.) Sensor Web Enablement Architecture, OGC Engineering report 06-021r2, http://portal.opengeospatial.org/files/?artifact_id=27775&version=1, 2008.

SOA-RA, 2008 OASIS Reference Architecture for Service Oriented Architecture Version 1.0 Public Review Draft 1, 23 April 2008 http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf

SOA-RM, 2006 OASIS Reference Model for Service Oriented Architecture 1.0. Committee Specification 1, 2 August 2006. http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

Taylor, B.N. and Kuyatt, C.E., 1994

NIST Technical Note 1297, 1994 Edition, Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results. Physics Laboratory of the National Institute of Standards and Technology (U.S.), http://physics.nist.gov/cuu/Uncertainty/bibliography.html

UKAS, 2007 M3003, The Expression of Uncertainty and Confidence in Measurement, Edition 2, http://www.ukas.com/Library/downloads/publications/M3003.pdf

Usländer, T., 2008 Integration of Resource-Oriented Architecture Concepts into the OGC Reference Model, OGC Document 07-156r1

Wan, E. and Van Der Merwe, R. 2001

Chapter 7: The unscented Kalman filter, in Kalman Filtering and Neural Networks, pp. 221-280, Wiley.

Watson, K., Kunz, S., 2007 (Eds.)

SANY Project Deliverable D2.2.2 Sensor Scenarios and Requirements Version 1.2, October 2007

WfXML-R, 2008 A RESTful Protocol for Run-Time Integration of Process Engines, Draft 0.4, http://wfxml.googlegroups.com/web/WfXML-R-0.4.pdf, 24 January 2008,

Williams, M., Cornford, D., Bastin, L., Pebesma, E., 2007

Uncertainty Markup Language (UncertML) Discussion Paper: Overview and High Level Architecture, Version 1.0.0, 18-08-2008, European FP6 project INTAMAP, http://www.intamap.org/pub/UncertML.pdf

Wilson, T., 2006 OWS-4 CSW ebRIM Modelling Guidelines IPR. Annex A: OWS-4 Sensor Web Enablement Catalogue Resource Profile. OGC Discussion Paper 06-155, 2007-03-12

W3C, 2001 Web Services Description Language (WSDL) 1.1, W3C Note 15, http://www.w3.org/TR/wsdl, March 2001

W3C, 2004 Web Services Architecture. W3C Working Group Note 11 February 2004. http://www.w3.org/TR/ws-arch/

W3C, 2007 Web Services Description Language (WSDL) 2.0, W3C Note 15, http://www.w3.org/TR/wsdl20/, June 2007

