Open Geospatial Consortium

Date: 2011-03-21

Reference number of this document: **OGC 09-001**

OGC name of this OGC® project document: **http://www.opengis.net/doc/IS/SWES/2.0**

Version: 2.0

Category: OpenGIS® Implementation Standard

Editor: Johannes Echterhoff

OpenGIS® SWE Service Model Implementation Standard

Warning

This document is an OGC Member approved international standard. This document is available on a royalty free, non-discriminatory basis. 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: OpenGIS® Standard
Document subtype: NA
Document stage: Approved
Document language: English

Contents

Page

Figures

Page

Tables

Page

i. Abstract

This standard currently defines eight packages with data types for common use across OGC Sensor Web Enablement (SWE) services. Five of these packages define operation request and response types. The packages are: 1.) Contents – Defines data types that can be used in specific services that provide (access to) sensors; 2.) Notification – Defines the data types that support provision of metadata about the notification capabilities of a service as well as the definition and encoding of SWES events; 3.) Common - Defines data types common to other packages; 4.) Common Codes –Defines commonly used lists of codes with special semantics; 5.) DescribeSensor – Defines the request and response types of an operation used to retrieve metadata about a given sensor; 6.) UpdateSensorDescription –Defines the request and response types of an operation used to modify the description of a given sensor; 7.) InsertSensor – Defines the request and response types of an operation used to insert a new sensor instance at a service; 8.) DeleteSensor – Defines the request and response types of an operation used to remove a sensor from a service. These packages use data types specified in other standards. Those data types are normatively referenced herein, instead of being repeated in this standard.

ii. Keywords

ogcdoc, swe, swes, ows

iii. Preface

Suggested additions, changes, and comments on this standard are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

If you choose to submit suggested changes or enhancements, please use the OGC online change request application:

http://portal.opengeospatial.org/public_ogc/change_request.php

iv. Document terms and definitions

This document uses the standard terms defined in Subclause 5.3 of [OGC 06-121r3], which is based on the ISO/IEC Directives, Part 2. Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

v. Submitting organizations

The following organizations submitted this document to the Open Geospatial Consortium Inc.

a) International Geospatial Services Institute GmbH (iGSI)

b) Spot Image, S.A.

c) University of Muenster

vi. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

CONTACT	COMPANY	EMAIL
Johannes Echterhoff (editor)	iGSI	johannes.echterhoff@igsi.eu
Ingo Simonis	iGSI	ingo.simonis@igsi.eu
Alexandre Robin	Spot Image, S.A.	alexandre.robin@spotimage.fr
Arne Bröring	52°North	broering@52north.org
Christoph Stasch	University of Muenster	staschc@uni-muenster.de

Issues with standard

Any issues in this standard are captured in the following format:

Issue Name: [Issue Name goes here.] (Your Initials, Date)

Issue Description: [Issue Description.]

Resolution: [Insert Resolution Details and History.] (Your Initials, Date)]

vii. Changes to the OGC Abstract Specification

The OpenGIS® Abstract Specification does not require changes to accommodate the technical contents of this document.

viii. Future work

This document is intended to provide data types and define mechanisms that can be reused by other OGC Sensor Web Enablement (SWE) implementation standards. Once data types and mechanisms defined in this standard become usable in other OGC Web Services (OWS) in addition to the SWE standards, these types should be added to the OWS Common standard and – having been added there – be removed in a future version of this standard. The notifications package (see clause 8) is one possible candidate for such a change.

In addition, further data types currently not contained in this specification that have been identified as being useful across SWE services and that are neither contained in this specification nor of use for all OWS shall be added to future versions of this specification.

If new requirements arise that cause a modification of the functionality defined in this standard, either through extensions or profiles, then required conformance classes shall also be defined. This could for example be a conformance class to augment the existing Sensor History Provider conformance class with the requirement that sensor descriptions can only be retrieved for a specific point in time.

During the development of this specification, the OGC has changed its specification template and development policies. This specification reflects those changes as much as possible, but full compliancy to the new OGC specification model needs to be achieved in future releases.

Foreword

This is the first version of the standard defining data types that are common across SWE services. It does not replace any existing OGC standard in whole or in parts.

This standard uses data types defined by the W3C WS-Addressing and OASIS WS-Notification (set of) standards.

This document includes six annexes; Annexes A-D are normative, and Annexes E and F are informative.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Introduction

During the preparation of the second major version of the Sensor Planning Service (SPS) and Sensor Observation Service (SOS), the OGC Members recognized that certain interfaces and data types are common across SWE services. The DescribeSensor operation is the most prominent example for such an interface and according data types. Common sense was that these types should be specified in a separate document that could then be referenced by all standards that reuse the defined types. While preparing both SPS 2.0 and SOS 2.0, other data types with common use were identified, leading to the set of data type and interface definitions contained in this document.

This standard does not replace the OWS Common standard. This standard serves as the home for all data type and interface definitions that are common only to SWE services.

OpenGIS® SWE Service Model

1 Scope

This OGC® document specifies data types and interfaces common to Sensor Web services. It therefore serves as a baseline for the development of such services. Specifically this standard:

- Is applicable to all services that provide information from or about sensors;
- Is applicable for uses cases in which sensors need to be managed through service interfaces.
- Specifies how sensor descriptions can be accessed and managed;
- Specifies how historical sensor descriptions can be accessed and managed;
- Establishes the means for inserting and deleting sensors through a common service interface.
- Specifies publish/subscribe functionality for Sensor Web services – through definition of recognizable event types, their encodings and association to notification topics.
- Gives guidelines for use of identifiers;
- Provides guidelines on creating an automatic mapping of the data types relevant in a service model from their UML representation to their XML Schema encoding;
- Defines the information required in a SOAP binding to realize the specified service functionality. The SOAP binding specifies WS-Notification to realize Publish/Subscribe service functionality

2 Conformance

2.1 Overview

This standard defines data types and interfaces common to Sensor Web services.

2.2 Specification identifier

All requirements and conformance-classes described in this document are owned by the standard identified as http://www.opengis.net/spec/SWES/2.0.

2.3 Conformance Classes

The following Table 1 specifies the conformance classes defined by this standard.

Compliance with a given conformance class shall be checked using the relevant tests specified in Annex A (normative).

Table 1 — SWES Conformance Classes

Conformance class name	Conformance class identifier	Operation and/or behavior
Core	http://www.opengis.net/spec/SWES/2.0/conf/Core	The server correctly defines the core type, mechanisms and according behavior as defined by this standard.
Basic SWE Service Metadata	http://www.opengis.net/spec/SWES/2.0/conf/BasicSWEServiceMetadata	The server implements its Contents section as defined in this standard.
Sensor Provider	http://www.opengis.net/spec/SWES/2.0/conf/SensorProvider	The server implements the DescribeSensor operation.
Sensor History Provider	http://www.opengis.net/spec/SWES/2.0/conf/SensorHistoryProvider	The server implements the *Sensor Provider* conformance class and support the *validTime* option for the DescribeSensor operation.
Sensor Description Manager	http://www.opengis.net/spec/SWES/2.0/conf/SensorDescriptionManager	The server implements the UpdateSensorDescription operation.
Sensor History Manager	http://www.opengis.net/spec/SWES/2.0/conf/SensorHistoryManager	The server implements the *Sensor Description Manager* conformance class and support the *validTime* option for the UpdateSensorDescription operation.
Sensor Insertion	http://www.opengis.net/spec/SWES/2.0/conf/SensorInsertion	The server implements the InsertSensor operation.
Sensor Deletion	http://www.opengis.net/spec/SWES/2.0/conf/SensorDeletion	The server implements the DeleteSensor operation.
XML Encoding	http://www.opengis.net/spec/SWES/2.0/conf/XMLEncoding	The server encodes the data types from the conceptual model in XML as defined by this standard.
Publish Subscribe	http://www.opengis.net/spec/SWES/2.0/conf/PublishSubscribe	The server implements publish/subscribe functionality as defined in this standard.
SOAP binding	http://www.opengis.net/spec/SWES/2.0/conf/SOAPBinding	The server implements the SOAP binding as defined in this standard.

Note: As described in clause 14, the DeleteSensor operation is abstract because its semantics are undefined. This standard only defines the syntax for the operation. The conformance classes to at least check the syntax is nevertheless defined in this standard. It can thus be leveraged by conformance classes in other standards that define the missing operation semantics.

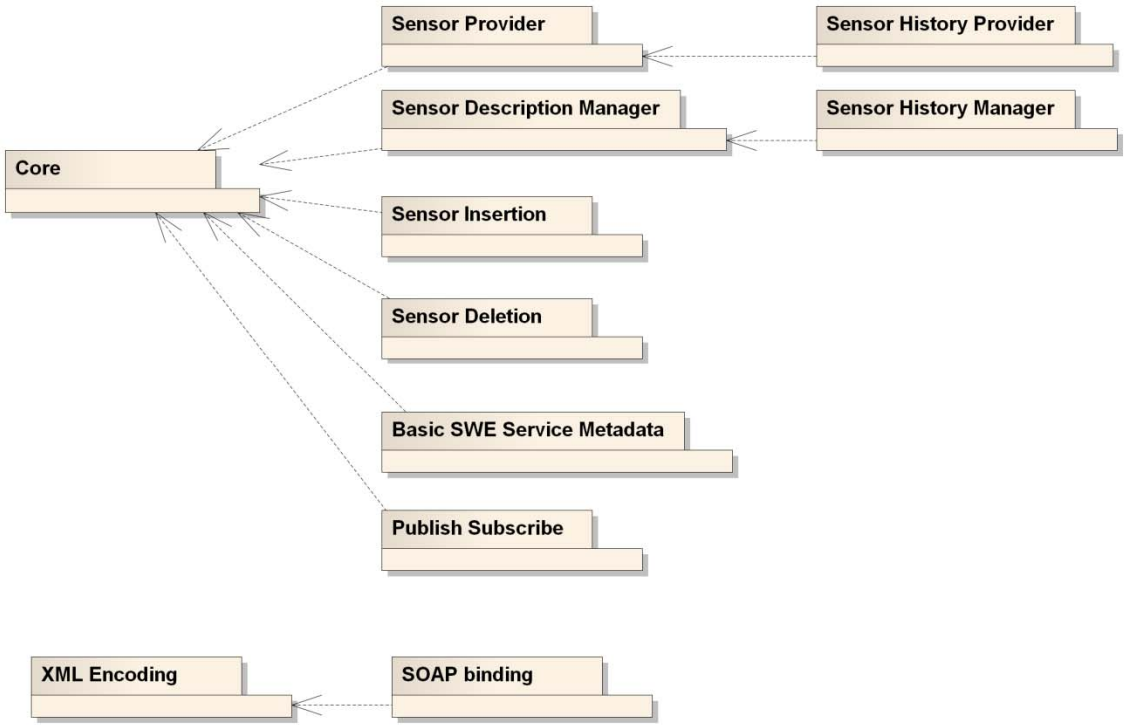

Figure 1 – SWES Conformance Classes and their interdependencies

3 Normative references

The following normative documents contain provisions that, through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

IETF RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*. (August 1998)

ISO 19105:2000, *Geographic information — Conformance and Testing*

ISO 19107:2003, *Geographic Information — Spatial schema*

ISO 19108:2002, *Geographic Information — Temporal schema*

ISO 19136:2007, *Geographic information – Geography Markup Language (GML)*

*(*see also: OpenGIS® Encoding Standard *Geography Markup Language*, v3.2.1, OGC document 07-036)

ISO 19143:2010, *Geographic information — Filter Encoding*

ISO/DIS 19156:2010, *Geographic information – Observations and measurements*

(see also: OpenGIS® Abstract Specification *Geographic Information: Observations and Measurements*, OGC document 10-004r3)

OASIS, *WS-Notification 1.3* (approved October 1st 2006)

- WS-BaseNotification
- WS-Topics
- WS-BrokeredNotification

OpenGIS® Encoding Standard *SWE Common Data Model*, v2.0, OGC document 08-094r1

OpenGIS® Implementation Specification, *Filter Encoding*, v1.1, OGC document 04-095

OpenGIS® Implementation Specification, *Web Services Common*, v1.1.0, OGC document 06-121r3

OpenGIS® Implementation Specification, *Sensor Model Language*, v1.0.0, OGC document 07-000

W3C, *Web Services Addressing 1.0 – Core* (W3C Recommendation 9 May 2006) (http://www.w3.org/TR/2006/REC-ws-addr-core-20060509)

W3C, *XML Information Set (Second Edition)* (W3C Recommendation 4 February 2004) (http://www.w3.org/TR/2004/REC-xml-infoset-20040204)

W3C, *XML Path Language (XPath) Version 1.0* (W3C Recommendation 16 November 1999) (http://www.w3.org/TR/1999/REC-xpath-19991116)

In addition to this document, this standard includes several normative XML Schema Document files as specified in clause 21.

4 Terms and definitions

For the purposes of this standard, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 06-121r3] shall apply. In addition, the following terms and definitions apply.

4.1
consumer
synonym: event sink
receiver of notifications sent from other components; role in the event architecture (see [OGC 09-032])

4.2
event
anything that happens or is contemplated as happening at an instant or over an interval of time [OGC 09-032]

4.3
event object
object that represents, encodes, or records an event, generally for the purpose of computer processing [see OGC 09-032]

4.4
notification
synonym: message
container for event objects [see OGC 09-032]

4.5
notification channel
synonyms: channel, topic, subject
representation of a set of events that will be published as event objects on the channel; usually defined as part of the notification semantics of a certain domain

4.6
notification semantics
definition of events that can be recognized, possible encodings of an event as event object and the assignment of event objects to notification channels

4.7
producer
specialization of a publisher that offers an additional subscription interface; role in the event architecture (see [OGC 09-032])

4.8
publisher
sends notifications to other components; role in the event architecture (see [OGC 09-001])

4.9
subscription
represents the relationship between consumer and producer, including any content or channel filter, along with any relevant policy and context information (compare with OASIS WS-BaseNotification)

4.10
topic expression
expression in a certain dialect to identify a set of topics

4.11
topic namespace
forest of topic trees grouped together into the same namespace for administrative purposes [OASIS WS-Topics]

4.12
topic set
collection of topics supported by a service [see OASIS WS-Topics]

4.13
topic tree
hierarchical grouping of topics [OASIS WS-Topics]

5 Conventions

5.1 Abbreviated terms

Most of the abbreviated terms listed in Subclause 5.1 of the OWS Common Implementation Standard [OGC 06-121r3] apply to this document, plus the following abbreviated terms.

OASIS	Organization for the Advancement of Structured Information Standards
SensorML	Sensor Model Language
SOS	Sensor Observation Service
SPS	Sensor Planning Service
UML	Unified Modeling Language
W3C	World Wide Web Consortium

5.2 UML notation

Diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of [OGC 06-121r3].

NOTE Packages and data types from foreign namespaces or data types from packages other than the one under consideration are shown with grey background where required to indicate which diagram components belong to the package under consideration - unless they are given only as types of attributes from classes in the model defined in this specification. Interfaces are shown with light turquoise background.

5.3 Platform-neutral and platform-specific standards

For compliance with Clause 10 of OGC Topic 12 and ISO 19119, this standard follows the pattern defined in subclause 5.4 of [OGC 06-121r3]. That is, model elements are specified in platform-neutral fashion first, using tables that serve as data dictionaries for the UML model (see clause 5.4 of this document). Platform-specific encodings of these model elements are provided in separate clauses or documents. The XML Schema encoding has automatically been generated using the rules defined in clause 24.

5.4 Data dictionary tables

The UML model data dictionary is specified herein in a series of tables. The contents of the columns in these tables are described in table 1 of [OGC 06-121r3]. The contents of these data dictionary tables are normative, including any table footnotes.

5.5 Classes imported from other specifications with predefined XML encoding

This standard uses an automatic mapping approach from the UML model to the XML Schema encoding. The approach is described in chapter 24. As shown in Figure 2, this standard uses types defined by other standards. For the mapping to XML Schema, the implementation instructions listed in table D.2 of [OGC 07-036] are used – with the

exception that ScopedName is encoded as xs:anyURI, which is explained in clause 16.3.1 - together with the instructions listed in the following tables.

For an explanation of the table columns, see clause D.2.1 in [OGC 07-036].

Table 2 — Implementation of types from ISO 19156

UML class	object element	type	property type
GFI_Feature	gml:AbstractFeature	gml:AbstractFeatureType	gml:FeaturePropertyType

Table 3 — Implementation of types from Web Services Addressing

UML class	object element	type	property type
EndpointReference	wsa:EndpointReference	wsa:EndpointReferenceType	-

Table 4 — Implementation of types from WS-Topics

UML class	object element	type	property type
TopicNamespace	wstop:TopicNamespace	wstop:TopicNamespaceType	-
Topicset	wstop:TopicSet	wstop:TopicSetType	-

Table 5 — Implementation of additional types used in this standard

UML class	object element	type	property type
Any	-	xs:anyType	xs:anyType

5.6 Namespace Conventions

This standard uses a number of namespace prefixes throughout; they are listed in Table 6. Note that the choice of any namespace prefix is arbitrary and not semantically significant.

Table 6 — Prefixes and Namespaces used in this standard

Prefix	Namespace
gml	http://www.opengis.net/gml/3.2
ows	http://www.opengis.net/ows/1.1
soap11	http://schemas.xmlsoap.org/soap/
soap12	envelopehttp://www.w3.org/2003/05/soap-envelope
swes	http://www.opengis.net/swes/2.0
wsa	http://www.w3.org/2005/08/addressing
wsn-b	http://docs.oasis-open.org/wsn/b-2
xs	http://www.w3.org/2001/XMLSchema

6 SWE Service Model Overview

This standard currently defines eight packages with data types for common use across SWE services. Five of these packages define operation request and response types. The packages are:

a) Contents – This package defines data types that can be used in specific services that provide (access to) sensors.

b) Notification – This package defines the data types that support provision of metadata about the notification capabilities of a service as well as the definition and encoding of SWES events.

c) Common – This package defines data types common to other packages.

d) Common Codes – This package defines commonly used lists of codes with special semantics.

e) DescribeSensor – This package defines the request and response types of an operation used to retrieve metadata about a given sensor.

f) UpdateSensorDescription – This package defines the request and response types of an operation used to modify the description of a given sensor.

g) InsertSensor – This package defines the request and response types of an operation used to insert a new sensor instance at a service.
NOTE: insertion metadata required by a certain SWE service type shall be defined in the according implementation specification leveraging this operation

h) DeleteSensor – This package defines the request and response types of an operation used to remove a sensor from a service.
NOTE: the semantic of deleting a sensor shall be defined by each implementation specification leveraging this operation

These packages use data types specified in other standards. Those data types are normatively referenced herein, instead of being repeated in this standard.

Figure 2 shows a UML diagram summarizing the external dependencies of the SWE Service Model.

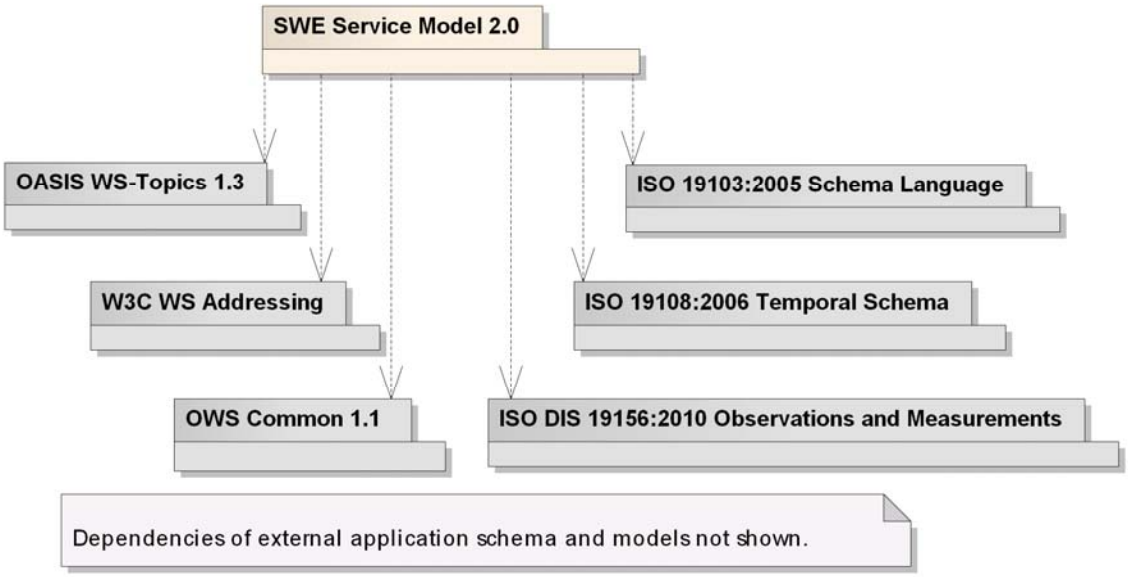

Figure 2 — SWE Service Model external dependencies

Figure 3 shows a UML diagram summarizing the package dependencies of the SWE Service Model.

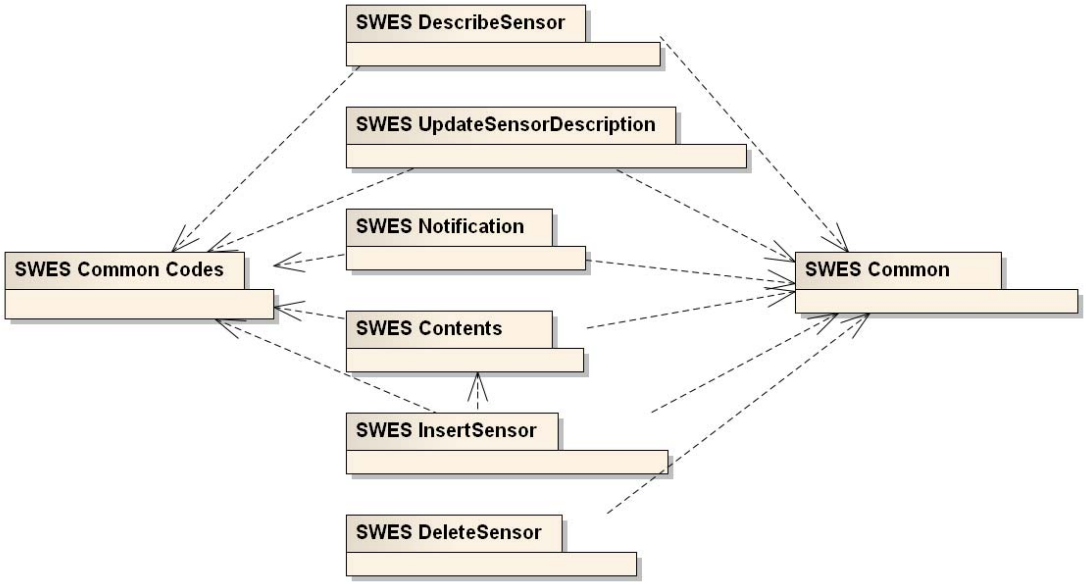

Figure 3 — SWE Service Model package dependencies

Each of the eight packages is described in the following clauses.

7 Contents

7.1 Introduction

SWE services provide access to sensors. Clients can retrieve detailed sensor descriptions and also access observations and measurements generated by sensors at SOS or task sensors at SPS.

In the contents section of these services, metadata on the sensors registered at the service needs to be provided which enables clients to perform the aforementioned functionality. This package defines the data types for providing such metadata.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/contents/property-inheritance	
REQ 1.	In order to reduce the size of the Capabilities document by reducing the amount of redundant information, the *property inheritance mechanism* shall be used (see clauses 7.2.1, 7.2.2 and 22).

7.2 Data Types

The conceptual model of the *Contents* package is shown in the following UML diagram.

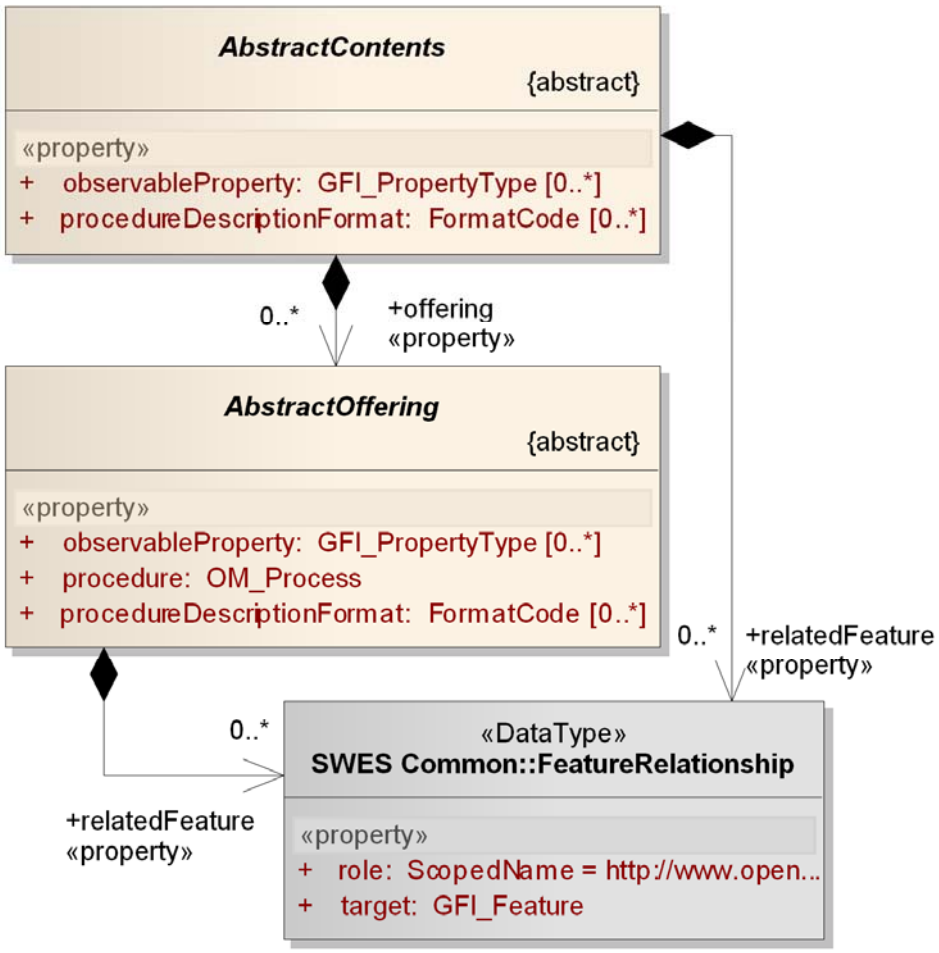

Figure 4 — Data types contained in Contents package

The details of each class contained in the package will be explained in the following subclauses.

7.2.1 AbstractContents

Each SWE service usually defines a Contents section, which provides metadata about the resources managed by the service. The *AbstractContents* can host a list of offerings – subclasses of *AbstractOffering* (see clause 7.2.2).

The AbstractContents acts as the *property provider* for the *AbstractOffering* (see clauses 7.2.2 and 22).

Specifications like SOS and SPS can derive from *AbstractContents* and thereby add properties that are specific to their service domain. They may also specify that such properties are included in the property inheritance mechanism for their offerings.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/contents/AbstractContents/properties	
REQ 2.	The *AbstractContents* type shall include the properties according to Table 7.

Table 7 — Properties in the AbstractContents type

Name	Definition	Data type and values	Multiplicity and use
procedureDescriptionFormat	identifier of a specific procedure/sensor description format	FormatCode see section 10.2.2	Zero to many (optional)
observableProperty	Pointer to a property that *can* be observed by a procedure, not necessarily a property that *has* already been observed.	GFI_PropertyType [a] see ISO/DIS 19156	Zero or more (optional)
relatedFeature	feature that is directly or indirectly observed/observable by a procedure; can be any feature of which the service provider thinks the procedure can make valuable observations for	FeatureRelationship see clause 9.2.4	Zero or more (optional)
offering	contains metadata about a procedure/sensor hosted by the service	AbstractOffering see clause 7.2.2	Zero or more (optional)
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

7.2.2 AbstractOffering

An *AbstractOffering* contains metadata about a procedure/sensor hosted by the service. This includes a pointer to the procedure together with a list of formats supported by the offering for describing the procedure, properties that can be observed by the procedure and related features. These features can be any feature that the service provider thinks the procedure can make valuable observations for.

A service may create more than one offering for the same procedure, so that a 1:n ($n \geq 1$) relationship between procedures and offerings exists. This functionality supports use cases in which the service provider wants to structure offerings according to certain criteria (e.g. to segregate the observable properties). However, the recommended way of structuring the offerings is to have exactly one offering per procedure.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/contents/AbstractOffering/procedureDescriptionFormatHomogeneity	
REQ 3.	If multiple offerings exist for the same procedure, then the *procedureDescriptionFormat* property of these offerings shall have the same values to ensure consistency when retrieving the description of a procedure via *DescribeSensor* (see clause 11).

Properties to store information like a local offering identifier or metadata are automatically added to the content model of *AbstractOffering* because it is an object type (see clause 24).

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/contents/AbstractOffering/identifierUniqueness	
REQ 4.	A service shall assign an identifier value to each of its offerings that is unique for each offering contained in the contents section of the Capabilities.

Specifications like SOS and SPS can derive from *AbstractOffering* and thereby add properties that are specific to their service domain. They may also specify that such properties are included in the property inheritance mechanism for their offerings.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/contents/AbstractOffering/properties	
REQ 5.	An AbstractOffering shall include the properties according to Table 8.

Table 8 — Properties in the AbstractOffering data type

Name	Definition	Data type and values	Multiplicity and use
procedure	Pointer to the procedure/sensor associated with this offering.	OM_Process [a] see ISO/DIS 19156	One (mandatory)
procedureDescriptionFormat	identifier of a specific procedure/sensor description format	FormatCode see section 10.2.2	Zero to many (optional)
observableProperty	Pointer to a property that *can* be observed by the procedure, not necessarily a property that *has* already been observed.	GFI_PropertyType [a] see ISO/DIS 19156	Zero or more (optional)
relatedFeature	feature that is directly or indirectly observed/observable by the procedure; can be any feature which the service provider thinks the procedure can make valuable observations for	FeatureRelationship see clause 9.2.4	Zero or more (optional)
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

The *AbstractOffering* represents an inheritor of the properties contained in the *AbstractContents* (which has the role of property provider; see clause 22 for further details on the property inheritance mechanism).

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/contents/AbstractOffering/propertiesInheritance	
REQ 6.	The *AbstractOffering* shall inherit the properties defined in the content model of *AbstractOffering* according to Table 9.

Table 9 — Inheritance of AbstractOffering properties (from AbstractContents)

Property	Number	Inheritance Category
procedure	1	no
procedureDescriptionFormat	1..*	replace
observableProperty	1..*	replace
relatedFeature	0..*	replace

Thus, even though the UML model and resulting schema encoding define the *observableProperty* and *procedureDescriptionFormat* properties as optional, they are mandatory in each offering. In other words, each offering has to include at least one value for these two properties after the property inheritance mechanism was applied.

8 Notification

8.1 Introduction

Asynchronous communication is of special interest to Sensor Web applications. The Publish/Subscribe communication pattern can be applied to decouple publishers and subscribers, also allowing multiple clients to subscribe to notifications of events they might all be interested in. The WS-Notification family of standards published by OASIS provides a solid foundation to incorporate Pub/Sub functionality in service interfaces.

The data types contained in this package incorporate and extend the properties defined by WS-Notification for notification producer and broker. A service implementing WS-Notification can perform those two roles. Even without a WSDL description, the required information for Pub/Sub based upon WS-Notification can be provided by adding the metadata types contained in this package to the Capabilities document of a service.

8.2 Data Types

The conceptual model of the *Notification* package is shown in the following two UML diagrams.

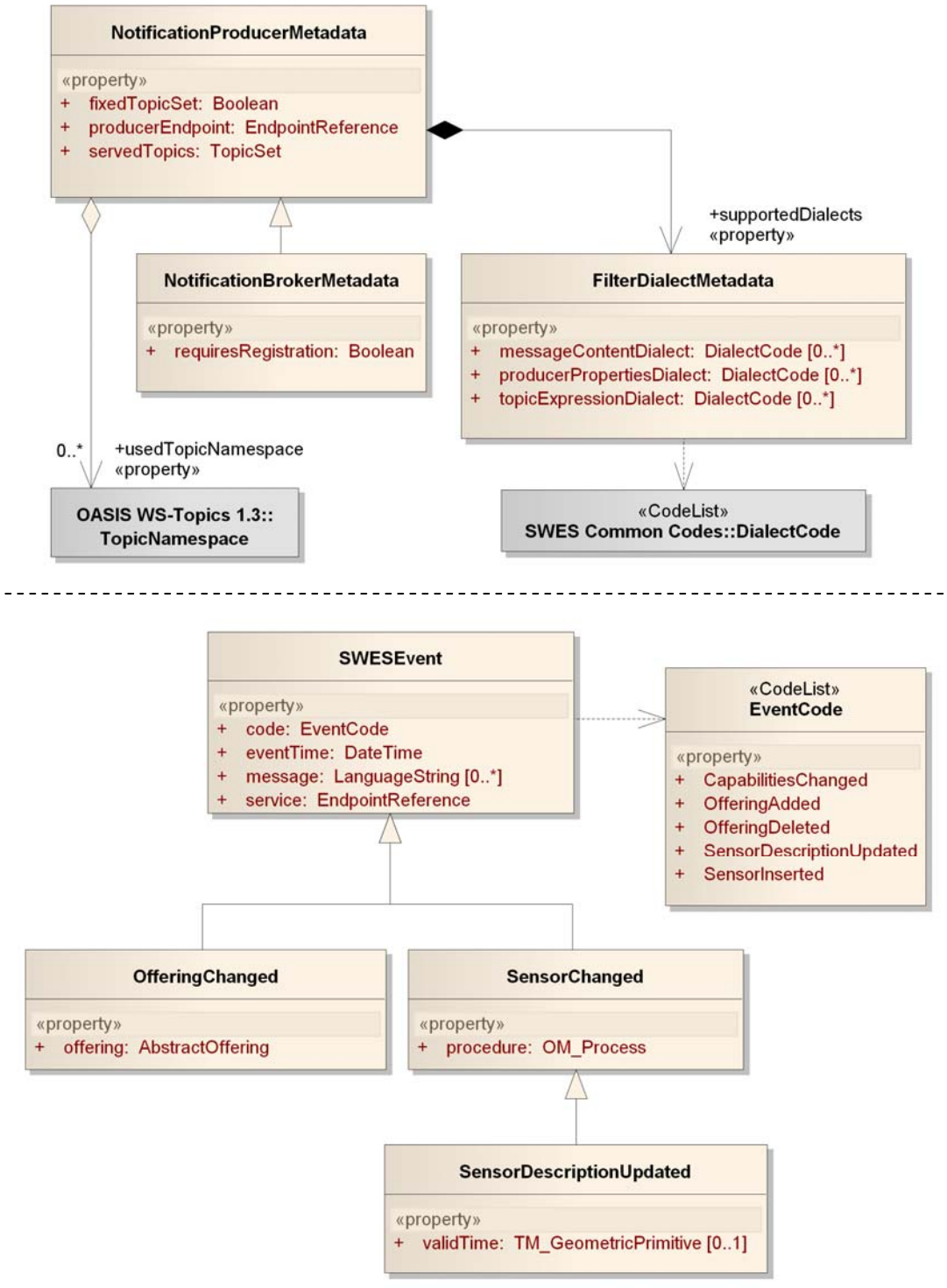

Figure 5 — Data types contained in Notification package

The classes in the upper part of the diagram are used to provide metadata about a service's publish/subscribe capabilities while those in the lower diagram are used for defining and encoding events (see clause 17.2).

The details of each class contained in the package will be explained in the following subclauses.

8.2.1 NotificationProducerMetadata

This type provides information for clients to subscribe for notifications. It defines the Web service endpoint where subscribe requests according to WS-BaseNotification shall be sent to (which can but does not have to be the URL of the service that provided the metadata). In addition the supported topics and according topic namespaces are presented. The service also indicates if the set of supported topics is fixed. The supported filter dialects are also provided.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Notification/NotificationProducerMetadata/properties	
REQ 7.	The *NotificationProducerMetadata* type shall include the properties according to Table 10.

Table 10 — Properties in the NotificationProducerMetadata data type

Name	Definition	Data type and values	Multiplicity and use
producerEndpoint	endpoint of the web service implementing the NotificationProducer interface defined by WS-BaseNotification; can but does not need to be the endpoint of the service which provided metadata	EndpointReference see WS-Addressing specification	One (mandatory)
supportedDialects	the filter dialects (used in WS-Notification Subscribe requests) supported by the service	FilterDialectMetadata see clause 8.2.2	One (mandatory)
fixedTopicSet	indicates if the set of served topics is static throughout the lifetime of the service instance	Boolean true if the topic set is static, else false	One (mandatory)
servedTopics	collection of topics supported by the service may change if the topic set is not fixed	TopicSet see WS-Topics specification	One (mandatory)
usedTopicNamespace	definition of a topic namespace used in the topic set of the service	TopicNamespace see WS-Topics specification	Zero or more (optional) required if the topic set contains one or more topics provide one topic namespace for each topic in the topic set that belongs to a distinct topic namespace other than the ad-hoc topic namespace defined by WS-Topics

8.2.2 FilterDialectMetadata

WS-Notification provides the option to restrict a subscription by defining several filter statements in a Subscribe request. These filter statements restrict the topics from which notifications shall be delivered (*topicExpressionDialect*), the content of these notifications (*messageContentDialect*) and/or the producers that generated the notification (*producerPropertiesDialect*).

In the *FilterDialectMetadata*, a service indicates general support of certain dialects.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Notification/FilterDialectMetadata/filtering	
REQ 8.	If a service does not support any filtering, the *FilterDialectMetadata* shall be empty. In that case clients can only subscribe to all events that the service publishes, without any additional sub-setting of the published events.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Notification/FilterDialectMetadata/properties	
REQ 9.	The *FilterDialectMetadata* type shall include the properties according to Table 11.

Table 11 — Properties in the FilterDialectMetadata data type

Name	Definition	Data type and values	Multiplicity and use
topicExpressionDialect	identifier of supported topic expression language	DialectCode see clause 10.2.1 applicable code value(s) as defined in Table 12	Zero or more (optional)
messageContentDialect	identifier of supported message content filter language	DialectCode see clause 10.2.1 applicable code value(s) as defined in Table 12	Zero or more (optional)
producerPropertiesDialect	identifier of supported producer properties filter language	DialectCode see clause 10.2.1 applicable code value(s) as defined in Table 12	Zero or more (optional)

Additional dialects to be used for identifying notification topics, filtering message content and filtering based upon a notification producer's properties may be defined in other specifications.

Table 12 — List of FilterDialectMetadata properties with applicable dialect codes

Property	Applicable Code Value(s)	Additional Note
topicExpressionDialect	http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple	*none*
	http://docs.oasis-open.org/wsn/t-1/TopicExpression/Concrete	*none*
	http://docs.oasis-open.org/wsn/t-1/TopicExpression/Full	*none*
	http://www.w3.org/TR/1999/REC-xpath-19991116	XPath (1.0) topic expression dialect, as defined in clause 8.4 of WS-Topics
messageContentDialect	http://www.w3.org/TR/1999/REC-xpath-19991116	XPath (1.0) message content dialect, as defined in clause 4.2 of WS-BaseNotification
	http://www.opengis.net/fes/1.1	*none*
	http://www.opengis.net/fes/2.0	*none*
producerPropertiesDialect	As of the writing of this specification, filtering upon producer properties is only defined by WS-ResourceProperties [1] in the context of Web Service Resources. Again XPath 1.0 is used. However, the query mechanism described there is not directly applicable to an OWS if it is not defined as a WS-Resource in a WSDL description. This could be defined in future versions of this or other documents, preferably in OWS Common.	

8.2.3 NotificationBrokerMetadata

The *NotificationBroker* interface as defined by WS-BrokeredNotification extends the interface of a *NotificationProducer*. As such, *NotificationBrokerMetadata* extends *NotificationProducerMetadata*.

Each entity that is going to connect a new publisher with a broker needs to know if the broker requires registration of publishers before accepting published data. The operation to register a new publisher is defined by WS-BrokeredNotification.

The *NotificationBrokerMetadata* data type is derived from the *NotificationProducerMetadata* data type specified in clause 8.2.1 and therefore inherits all the properties contained in that data type. *NotificationBrokerMetadata* does not restrict the content model of *NotificationProducerMetadata*.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Notification/NotificationBrokerMetadata/properties	
REQ 10.	*NotificationBrokerMetadata* shall contain the properties defined for *NotificationProducerMetadata*. In addition, it shall include the property according to Table 13.

Table 13 — Property in the NotificationBrokerMetadata data type

Name	Definition	Data type and values	Multiplicity and use
requiresRegistration	defines if a new publisher needs to be registered at the broker before it is allowed to send notifications	Boolean true if registration of new publishers is required, else false	One (mandatory)

8.2.4 SWESEvent

A *SWESEvent* represents the base class for encoding a SWE service event as defined by this standard. It provides the option to deliver a human readable message in multiple languages together with a code signifying the event and a reference to the service where the event actually happened.

As the type can be used in different topics of different topic namespaces, the allowed code type values to signify the exact nature of change is defined by topics using this type.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Notification/SWESEvent/properties	
REQ 11.	The *SWESEvent* type shall include the properties according to Table 14.

Table 14 — Property in the SWESEvent data type

Name	Definition	Data type and values	Multiplicity and use
eventTime	point in time the event happened	DateTime (see ISO 19103 and OGC 07-036 Table D.2)	One (mandatory)
code	signifies the event	EventCode see clause 8.2.5	One (mandatory)
message	human readable message in specific language describing the event	LanguageString see clause 10.7 in [OGC 06-121r3]	Zero to many (optional) if meaningful message is available, include one for each supported language; client may request specific languages through policies, extensions or other means
service	references the service where the event happened	EndpointReference see WS-Addressing specification	One (mandatory)

8.2.5 EventCode

The *EventCode* type is a list of codes signifying events that happen in SWE services and are identified in this standard.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Notification/SWESEventCode/properties	
REQ 12.	The *EventCode* code list shall include the properties / code values according to Table 15.

Table 15 — Properties in the *EventCode* code list

Name [a]	Definition	Value [a]
CapabilitiesChanged	A change to the Capabilities document of a service happened. Changes recognized by this definition are the addition, removal or modification of information contained in the previous version of the service's Capabilities document. If multiple changes happen at the same time (or in a time interval sufficiently small for a service to recognize them as belonging together) then they constitute one capabilities changed event.	"CapabilitiesChanged"
OfferingAdded	An offering as defined in clause 7.2.2 of OGC 09-001 was added to the contents section of a service's Capabilities document.	"OfferingAdded"
OfferingDeleted	An offering as defined in clause 7.2.2 of OGC 09-001 was removed from the contents section of a service's Capabilities document.	"OfferingDeleted"
SensorInserted	A sensor was inserted at the service. The sensor is now listed in a new offering (as defined in clause 7.2.2 of OGC 09-001) in the service's Capabilities document. The sensor may have been inserted using the *InsertSensor* operation defined in clause 13 of OGC 09-001.	"SensorInserted"
SensorDescriptionUpdated	The description of a sensor was updated. Updates recognized by this definition are the addition, removal or modification of information contained in the current and/or previous versions of a sensor's description. The description may have been updated using the *UpdateSensorDescription* operation defined in clause 12 of OGC 09-001.	"SensorDescriptionUpdated"
a Although some values listed in the column appear to contain spaces, they shall not contain spaces.		

This code list is extensible. SWES profiles/extensions or implementations may add additional event codes.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Notification/SWESEventCode/syntax	
REQ 13.	Each new code shall conform to the following regular expression: `other: [A-Za-z0-9_]{2,}` EXAMPLE: A valid new sub state would be "`other:FilterDialectAdded`".

8.2.6 OfferingChanged

The *OfferingChanged* type encodes the event of a change to one of the offerings hosted by the service.

As the type can be used in different topics of different topic namespaces, the allowed code type values to signify the exact nature of change is defined by topics using this type and not for the type in general.

The *OfferingChanged* data type derives from the *SWESEvent* data type specified in clause 8.2.4 and thus inherits all the properties contained in that data type. *OfferingChanged* does not restrict the content model of *SWESEvent*.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Notification/OfferingChanged/properties	
REQ 14.	In addition to the properties inherited from *SWESEvent*, *OfferingChanged* shall include the property according to Table 16.

Table 16 — Property in the *OfferingChanged* data type

Name	Definition	Data type and values	Multiplicity and use
offering	Pointer to the offering that changed.	AbstractOffering [a] see clause 7.2.2	One (mandatory)
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

8.2.7 SensorChanged

The *SensorChanged* type encodes the event of a change to one of the sensors hosted by the service.

As the type can be used in different topics of different topic namespaces, the allowed code type values to signify the exact nature of change is defined by topics using this type and not for the type in general.

The *SensorChanged* data type derives from the *SWESEvent* data type specified in clause 8.2.4 and thus inherits all the properties contained in that data type. *SensorChanged* does not restrict the content model of *SWESEvent*.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Notification/SensorChanged/properties	
REQ 15.	In addition to the properties inherited from *SWESEvent*, *SensorChanged* shall include the property according to Table 17.

Table 17 — Property in the SensorChanged data type

Name	Definition	Data type and values	Multiplicity and use
procedure	Pointer to the procedure that changed.	OM_Process [a] see ISO/DIS 19156	One (mandatory)
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

8.2.8 SensorDescriptionUpdated

The *SensorDescriptionUpdated* type encodes the event of an update to the description of one of the sensors hosted by the service. It conveys information not only that a sensor description was updated but also for which time the update to the description is valid. This is primarily useful if a service supports revision management of sensor descriptions (see clause 12): Clients that are only interested in updates of the current description can safely dismiss all update notifications that address updates to past descriptions.

As the type can be used in different topics of different topic namespaces, the allowed code type values to signify the exact nature of change is defined by topics using this type and not for the type in general.

The *SensorDescriptionUpdated* data type derives from the *SensorChanged* data type specified in clause 8.2.7 and thus inherits all the properties contained in that data type. *SensorDescriptionUpdated* does not restrict the content model of *SensorChanged.*

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Notification/SensorDescriptionUpdated/properties	
REQ 16.	In addition to the properties inherited from *SensorChanged*, *SensorDescriptionUpdated* shall include the property according to Table 18.

Table 18 — Property in the SensorDescriptionUpdated data type

Name	Definition	Data type and values	Multiplicity and use
validTime	time instance or time interval for which the updated sensor description is valid	TM_GeometricPrimitive see clause 5.2.3 in ISO ISO 19108 and clause 14.2.2.2 in [OGC 07-036]	Zero or one (optional) shall be omitted only if the service is not capable of managing description revisions or if the current description was updated

9 Common

9.1 Introduction

This package defines types that are commonly used in two or more other packages. It also contains the types that provide extension points for requests and responses of Web service operations.

OWS operations usually do not allow the addition of request and response parameters. However, with respect to the core & extension pattern for service specifications (where the core service functionality is defined in the base specification and extensions define further functionality) it is desirable to have a place in service requests and responses where elements defined by extensions, for example policy assertions, can be added without the XML instances becoming invalid.

This package defines two abstract data types, which provide such an extension point. Specific service requests and responses may derive from these types to inherit the extension properties.

The extension mechanism used here resembles the one which is usually encountered in WS-* specifications developed by OASIS. The intention here is to specify extension points in request and response data types of SWE service operations.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Common/honourRequestExtension	
REQ 17.	A service shall always honor the semantics of known extension properties included in a request. A service shall throw a *RequestExtensionNotSupported* exception (see clause 15) if an unknown extension was encountered in a client request.

9.2 Data Types

The conceptual model of the Operations package is shown in the following UML diagram.

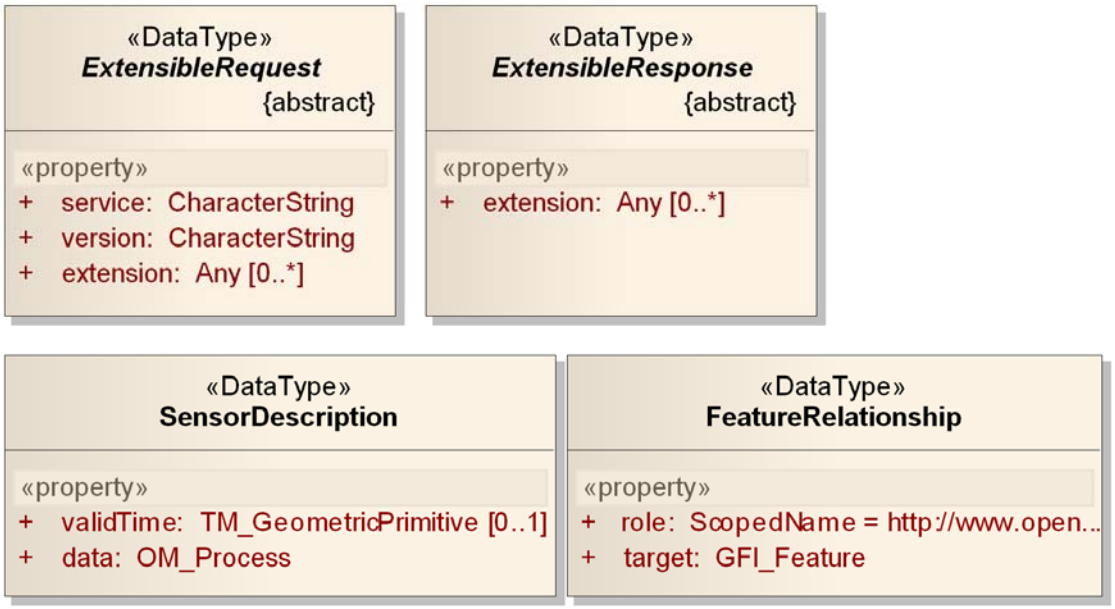

Figure 6 — Data types contained in the Common package

The details of each class contained in the package will be explained in the following subclauses.

9.2.1 ExtensibleRequest

This type serves as the superclass for all request types that implement the extension pattern introduced by this specification.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Common/ExtensibleRequest/properties	
REQ 18.	The *ExtensibleRequest* type shall include the properties according to Table 19.

Table 19 — Properties in the ExtensibleRequest data type

Name	Definition	Data type and values	Multiplicity and use
service	service type identifier	CharacterString, not empty value is OWS type abbreviation (e.g., “SOS”, “SPS”) of the implementing service	One (mandatory)
version	specification version for operation	CharacterString, not empty value is specified by service type specification implemented by service	One (mandatory)
extension	container for request parameters defined by extension	Any type value is defined by the extension specification	Zero or more (optional)

9.2.2 ExtensibleResponse

This type serves as the superclass for all response types that implement the extension pattern introduced by this specification.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Common/ExtensibleResponse/properties	
REQ 19.	The *ExtensibleResponse* type shall include the properties according to Table 20.

Table 20 — Properties in the ExtensibleResponse data type

Name	Definition	Data type and values	Multiplicity and use
extension	container for response parameters defined by extension	Any type value is defined by the extension specification	Zero or more (optional)

9.2.3 SensorDescription

The *SensorDescription* type is a container for a sensor/procedure description and may also define the time at when or the time period during which the description is/was valid. If no such time is provided, the description is currently valid, but no statements can be made about the past or future.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Common/SensorDescription/properties	
REQ 20.	The *SensorDescription* data type shall include the properties according to Table 21

Table 21 — Properties in the SensorDescription data type

Name	Definition	Data type and values	Multiplicity and use
validTime	time instant or time period for which the given sensor description is valid	TM_GeometricPrimitive see clause 5.2.3 in ISO 19108 and clause 14.2.2.2 in [OGC 07-036] if validTime is TM_Instant, the time instant shall be in the past if validTime is TM_Period, the start time of the time period shall be in the past	Zero or one (optional)
data	the sensor description	OM_Process see ISO/DIS 19156 the actual type depends upon the requested description format	One (mandatory)

9.2.4 FeatureRelationship

The *FeatureRelationship* data type allows explicit representation of the role a given feature has for some specific source type. The relationship is especially useful in situations where the exact role is not known a priori.

Example: a given AbstractOffering may have a list of features that are related to it in some way. In case of a SOS, an offering containing observations from a UAV may reference the tracks along which the UAV has gathered data. That the features associated to that offering are actually UAV tracks can be represented explicitly via the FeatureRelationship.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Common/FeatureRelationship/properties	
REQ 21.	The *FeatureRelationship* data type shall include the properties according to Table 22

Table 22 — Properties in the FeatureRelationship data type

Name	Definition	Data type and values	Multiplicity and use
role	describes the relationship of the target feature to the source type	ScopedName (see clause 16) initial/default value: http://www.opengis.net/def/nil/OGC/0/unknown	One (mandatory)
target	the feature whose role with respect to some source type is further described by this relationship	GFI_Feature see ISO/DIS 19156	One (mandatory)

10 Common Codes

10.1 Introduction

In the conceptual models of SWE services, it is often necessary to identify certain types of information. For example, in the contents section of a (SWE) service's capabilities document, the service provider needs to state which format is used to encode descriptions of procedures. It can also be necessary to identify the filter language that clients can use, not only in usual data retrieval requests but also in subscriptions (see sections 8 and 17.2).

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/CommonCodes/URI	
REQ 22.	Each code value assigned to code lists of the Common Codes package shall be of type URI.

This is necessary because in the XML encoding of SWE service models, properties that use these code lists shall be encoded as elements of type URI (see clause 24). In general, existing URIs that uniquely identify certain processing languages can be reused without modification. This can be for example the namespace used by a specification that defines an XML Schema for creating certain processing expressions. It can also be a conformance class URI. If no unique URI is available to identify a processing language, then a new URI needs to be defined and documented accordingly so that it can be used as a code value in one of the code lists defined in this package.

This package defines a number of code list types that have special semantics. These types are used in conceptual models to imply the according semantics.

10.2 Data Types

The conceptual model of the Common Codes package is shown in the following UML diagram.

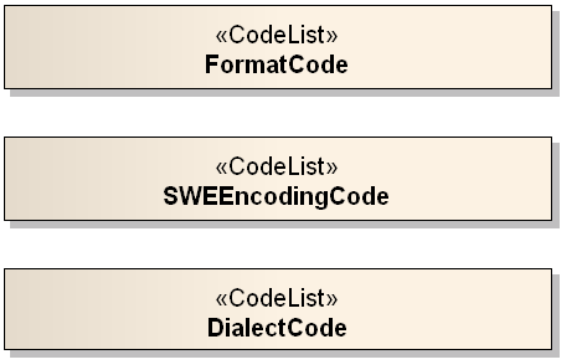

Figure 7 – Data types contained in Common Codes package

The code lists shown in the diagram are empty because they define an unlimited set of possible code values. The details of each code list contained in the package as well as some example code values will be explained in the following subclauses.

10.2.1 DialectCode

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/CommonCode/DialectCode	
REQ 23.	All values identifying language dialects used for processing data (including filtering) shall be added to the *DialectCode* code list.

The following table lists some code values for the *DialectCode* list together with their definition and meaning. A specification that defines properties of type *DialectCode* in its conceptual model should define which codes are applicable for that property. Extensions to this specification can then define additional codes to be used for that property. Such codes shall be URIs.

Table 23 – List of some code values used for identifying processing languages

Dialect code value	Definition/Meaning
http://docs.oasis-open.org/wsn/t-1/TopicExpression/Concrete	simple topic expression dialect, used to identify a set of topics from a notification producers topic set, as defined in clause 8.1 of WS-Topics
http://docs.oasis-open.org/wsn/t-1/TopicExpression/Full	concrete topic expression dialect, used to identify a set of topics from a notification producers topic set, as defined in clause 8.2 of WS-Topics
http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple	full topic expression dialect, used to identify a set of topics from a notification producers topic set, as defined in clause 8.3 of WS-Topics
http://www.opengis.net/fes/1.1	Filter according to OGC Filter Encoding version 1.1 [OGC 04-095]
http://www.opengis.net/fes/2.0	Filter according to OGC Filter Encoding version 2.0 [ISO 19143]
http://www.w3.org/TR/1999/REC-xpath-19991116	XPath (1.0), defined by W3C

10.2.2 FormatCode

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/CommonCode/FormatCode	
REQ 24.	All values identifying a format shall be added to the *FormatCode* code list. Usually, formats are used to describe a model or a sensor (e.g. SensorML for the encoding of metadata).

The following table lists some code values for the *FormatCode* list together with their definition and meaning. A specification that defines properties of type *FormatCode* in its conceptual model should define which codes are applicable for that property. Extensions to this specification can then define additional codes to be used for that property. Such codes shall be URIs.

Table 24 – List of some code values used for identifying formats

Dialect code value	Definition/Meaning
http://www.opengis.net/sensorML/1.0.1	Sensor Model Language as defined in [OGC 07-000] with schema version 1.0.1
http://www.opengis.net/om/2.0	Observations & Measurements as defined in ISO 19156
Note: at the time of writing this standard, SensorML v2.0 was not yet available – an applicable code to identify SensorML 2.0 would be the schema namespace used by that standard.	

10.2.3 SWEEncodingCode

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/CommonCode/SWEEncodingCode	
REQ 25.	All code values that identify some encoding defined by the SWE Common Data Model (or an extension thereof) shall be added to the *SWEEncodingCode* code list. Examples are the SWE Common text and binary encodings.

The following table lists some code values for the *SWEEncodingCode* list together with their definition and meaning. A specification that defines properties of type *SWEEncodingCode* in its conceptual model should define which codes are applicable for that property. Extensions to this specification can then define additional codes to be used for that property. Such codes shall be URIs.

Table 25 – List of some code values used for identifying SWE Common encodings

Dialect code value	Definition/Meaning
http://www.opengis.net/swe/2.0/TextEncoding	Encoding for arbitrarily complex data using a text based delimiter separated values (DSV) format, see [OGC 08-094r1]
http://www.opengis.net/swe/2.0/XMLEncoding	Encoding of structured data into a stream of nested XML tags, see [OGC 08-094r1]
http://www.opengis.net/swe/2.0/BinaryEncoding	Encoding of complex structured data using primitive data types encoded directly at the byte level, see [OGC 08-094r1]

11 DescribeSensor

11.1 Introduction

The *DescribeSensor* operation is used to obtain a detailed sensor/procedure description encoded in a certain format, for example SensorML. Because a SWE service might offer such descriptions in multiple formats, or different versions of the same format, clients generally need to indicate which format they request. The formats supported by a given sensor/procedure are defined in the Capabilities of the service (see clause 7).

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/current	
REQ 26.	By default, the *DescribeSensor* operation retrieves the current description of the sensor/procedure only.

However, a service may also support the retrieval of the description that was or will be valid at a certain point in time or period of time in the past or the future.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/validTime-behavior	
REQ 27.	If a client requests historic and/or future sensor/procedure descriptions by including the validTime parameter in the request and the service supports that functionality then the service shall include all the sensor descriptions in the response whose valid time is *not before and not after* the valid time given in the request.

The expected behavior for adding or not adding an existing sensor description to a response upon a DescribeSensor request with valid time parameter is illustrated in Figure 8 to Figure 11.

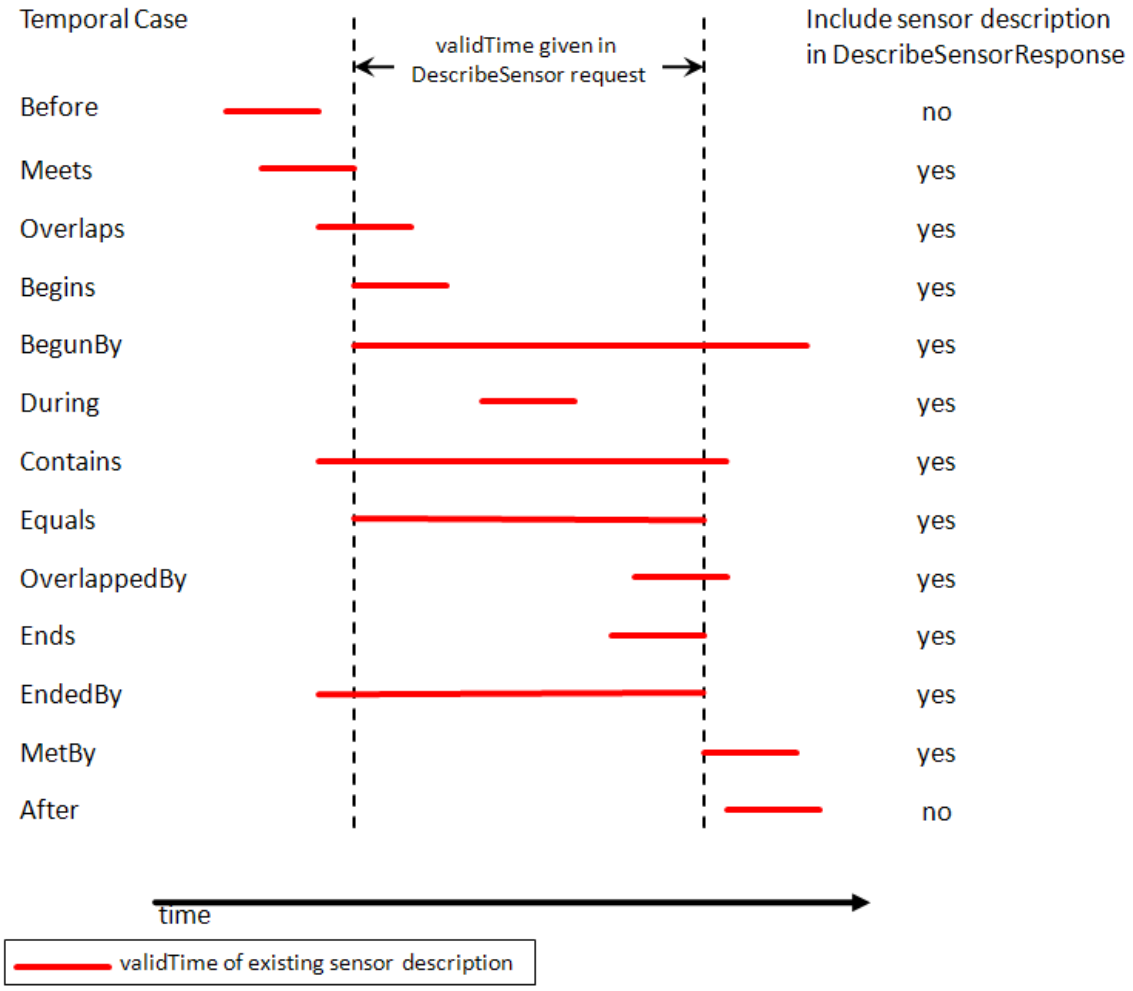

Figure 8 — temporal relationships between validTime given in DescribeSensor request (as time period) and in existing sensor description (as time period) with required server response behavior

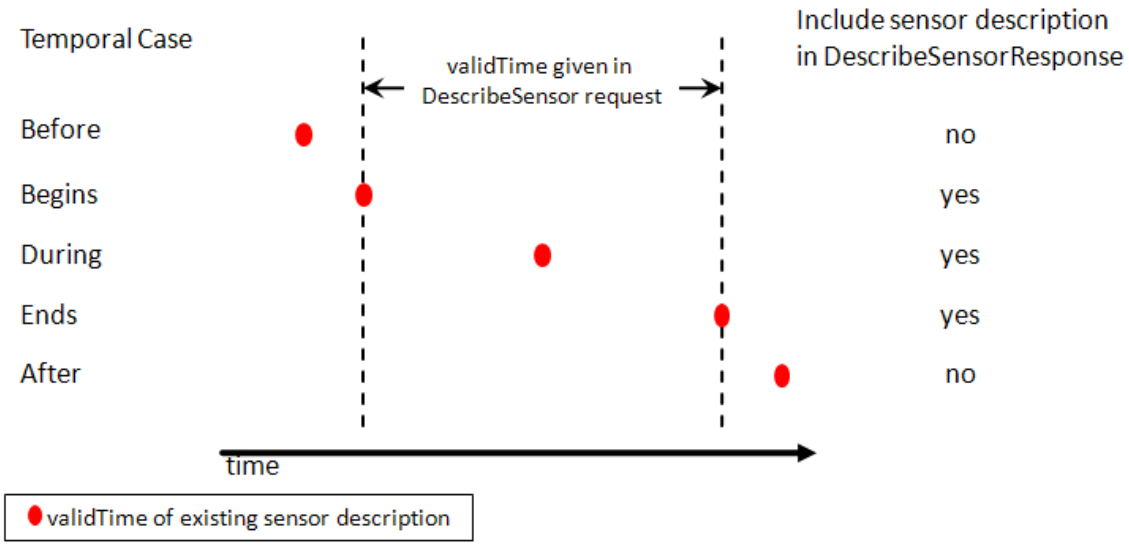

Figure 9 — temporal relationships between validTime given in DescribeSensor request (as time period) and in existing sensor description (as time instant) with required server response behavior

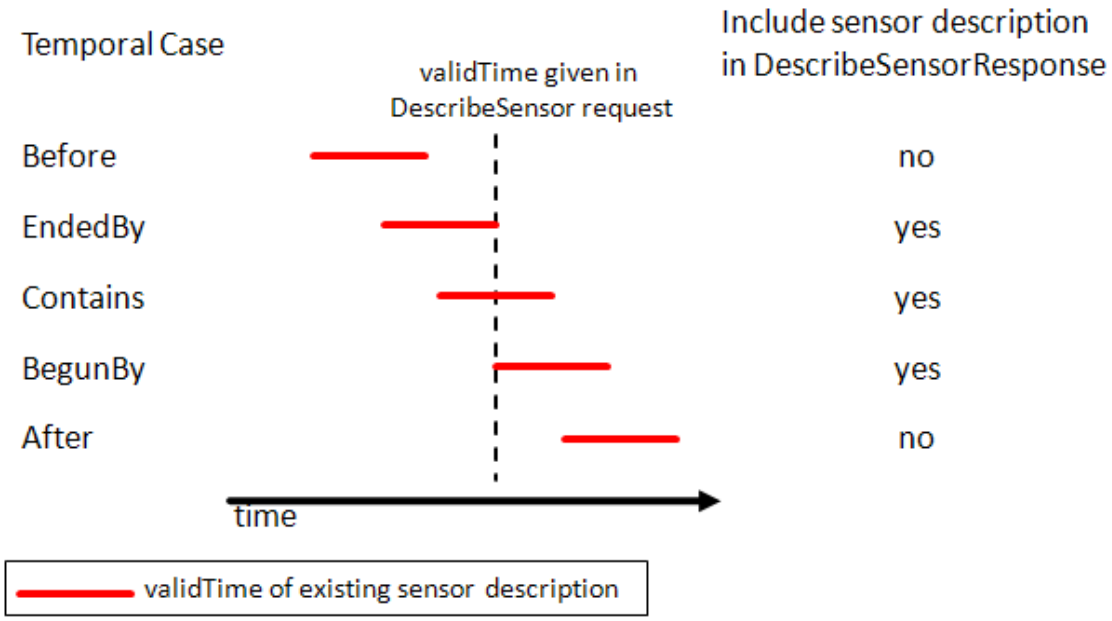

Figure 10 — temporal relationships between validTime given in DescribeSensor request (as time instant) and in existing sensor description (as time period) with required server response behavior

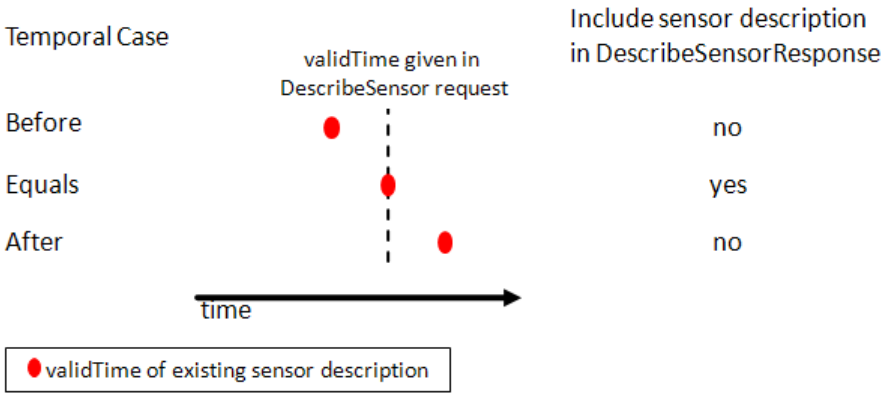

Figure 11 — temporal relationships between validTime given in DescribeSensor request (as time instant) and in existing sensor description (as time instant) with required server response behavior

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/no-matching-descriptions	
REQ 28.	If a service is generally capable of providing historic and/or future sensor/procedure descriptions but does not have a description for a particular requested point in time or time period, the service response shall not contain a sensor/procedure description.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/past-multiple	
REQ 29.	If a service is generally capable of providing historic and/or future sensor/procedure descriptions and has varying descriptions of a sensor/procedure for a particular requested time period, the service response shall contain all sensor/procedure descriptions and indicate the valid time periods. Note: Valid time periods must not overlap; see requirement http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/Response/overlaps.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/capability	
REQ 30.	If a service is generally capable of providing historic and/or future sensor/procedure descriptions, it shall advertise this feature by adding a parameter *validTime* to the metadata of the operation *DescribeSensor* in the *OperationsMetadata* section of the service's Capabilities document.

11.2 Data Types

The conceptual model of the *DescribeSensor* operation is shown in the following UML diagram.

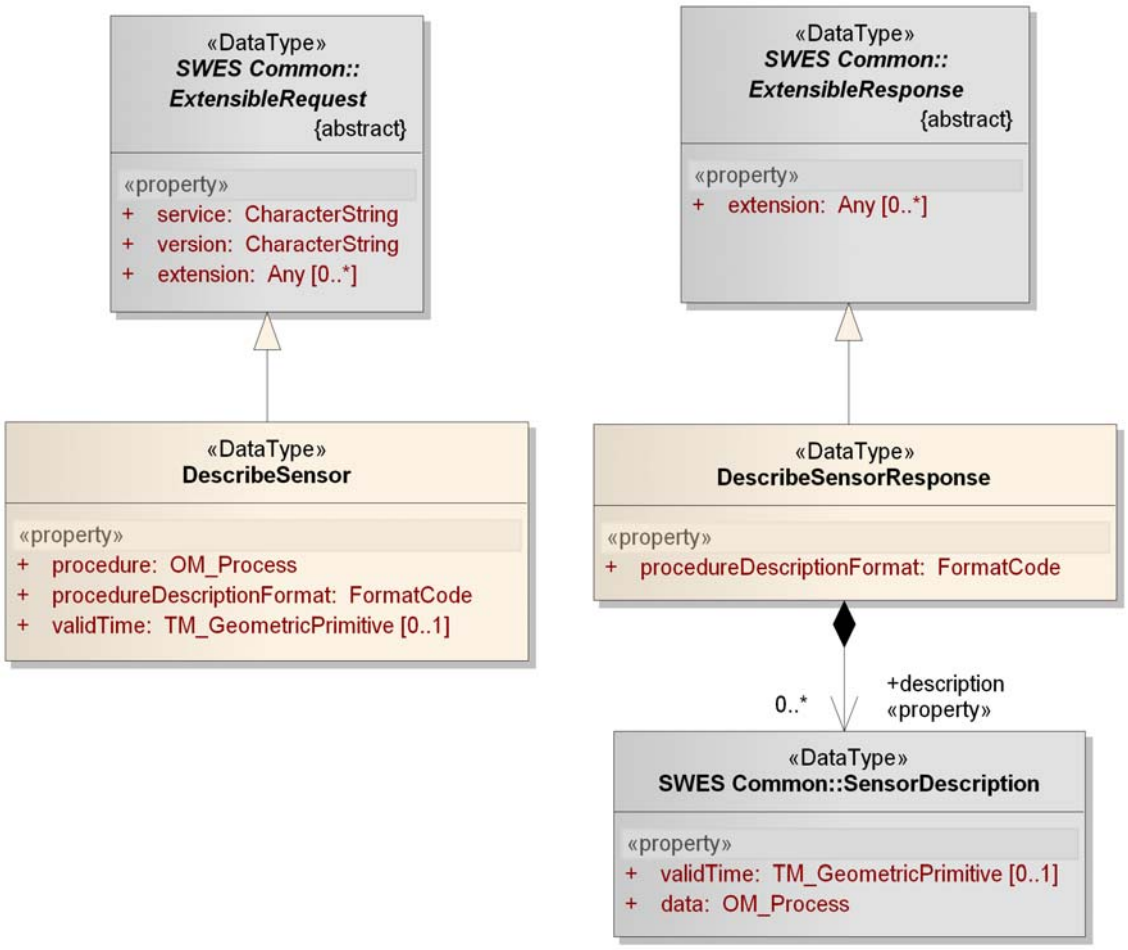

Figure 12 — Data types of the DescribeSensor operation

The details of the operation request and response are explained in the following subsections.

11.2.1 Operation Request - DescribeSensor

A *DescribeSensor* operation request is performed by sending an instance of the *DescribeSensor* data type to the service. By default, the current description is returned. If supported by the service (see clause 11.1), a client may also request the description that was/is/is going to be valid at a certain point in time – same for multiple descriptions if the client defines an interval of time in the request.

The *DescribeSensor* data type is derived from the *ExtensibleRequest* data type (see clause 9.2.1) and thus inherits all the properties contained in that data type. *DescribeSensor* does not restrict the content model of *ExtensibleRequest*. It contains all properties defined in *ExtensibleRequest*.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/Request/properties	
REQ 31.	In addition to the properties inherited from *ExtensibleRequest*, *DescribeSensor* shall contain the properties according to Table 26.

Table 26 — Properties in the DescribeSensor data type

Name	Definition	Data type and values	Multiplicity and use
procedure	Pointer to the procedure/sensor of which the description shall be returned.	OM_Process [a] see ISO/DIS 19156 value shall match one of the procedures from the AbstractOfferings listed by the service (see clause 7.2.2)	One (mandatory)
procedureDescriptionFormat	identifier of the requested procedure/sensor description format	FormatCode see section 10.2.2 value shall match one of the formats supported by the AbstractOffering(s) (see clause 7.2.2) with the given procedure	One (mandatory)
validTime	Time instant or time period for which the then valid sensor description shall be retrieved.	TM_GeometricPrimitive see clause 5.2.3 in ISO 19108 and clause 14.2.2.2 in [OGC 07-036] if validTime is TM_Instant, the time instant shall be in the past if validTime is TM_Period, the start time of the time period shall be in the past	Zero or one (optional) If end time of time period is in the future, all descriptions from start time to now - including the current description - shall be returned
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

11.2.2 Operation Response – DescribeSensorResponse

The *DescribeSensorResponse* data type represents the response to a *DescribeSensor* operation request. It contains the description that is currently valid for the requested procedure. If supported by the service and if data is available, the *DescribeSensorResponse* contains the descriptions that have been or will be valid during a particular period in time.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/Response/overlaps	
REQ 32.	Though a service may return multiple sensor descriptions (if the description has changed/will change), these descriptions shall not have overlapping validity times.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/Response/format	
REQ 33.	A service shall also ensure that all descriptions returned in the DescribeSensorResponse are encoded in the requested format.

The *DescribeSensorResponse* data type is derived from the *ExtensibleResponse* data type (see clause 9.2.2) and thus inherits all the properties contained in that data type. *DescribeSensorResponse* does not restrict the content model of *ExtensibleResponse*. It contains all properties defined in *ExtensibleResponse*.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/Response/properties	
REQ 34.	In addition to the properties inherited from *ExtensibleResponse*, *DescribeSensorResponse* shall contain the properties according to Table 27.

Table 27 — Properties in the DescribeSensorResponse data type

Name	Definition	Data type and values	Multiplicity and use
procedureDescriptionFormat	identifier of the returned procedure/sensor description format	FormatCode see section 10.2.2	One (mandatory)
description	container element that provides the description that matches the request criteria	SensorDescription see clause 9.2.3	Zero to many (optional) omit only if request contained valid time for which no description is available

11.3 Exceptions

Clause 15 defines the exception codes that apply to this operation. In addition:

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/exception/validTimeNotSupported	
REQ 35.	If a *DescribeSensor* request contains a *validTime* property but the server does not support this option (i.e., it only supports retrieval of the most recent sensor description), an exception with code *OptionNotSupported* and locator value *validTime* shall be thrown.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DescribeSensor/exception/unknownProcedureDescriptionFormat	
REQ 36.	If a *DescribeSensor* request contains a *procedureDescriptionFormat* property with unknown value, an exception with code *InvalidParameterValue* and locator value *procedureDescriptionFormat* shall be thrown.

11.4 Examples

The following two listings provide examples of a *DescribeSensor* operation request and response encoded in XML according to the XML Schema defined in Annex B.

Listing 1 – DescribeSensor operation request example

```
<swes:DescribeSensor service="SPS" version="2.0"
xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <swes:procedure>http://my.org/sensors/937239</swes:procedure>
    <swes:procedureDescriptionFormat>http://www.opengis.net/sensorML/1.0
.1</swes:procedureDescriptionFormat>
</swes:DescribeSensor>
```

Listing 2 - DescribeSensor operation response example

```
<swes:DescribeSensorResponse
xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<swes:procedureDescriptionFormat>http://www.opengis.net/sensorML/1.0.1<
/swes:procedureDescriptionFormat>
  <swes:description>
    <swes:SensorDescription>
      <swes:data>
        <sml:Component>
          <!-- details omitted for brevity -->
        </sml:Component>
      </swes:data>
    </swes:SensorDescription>
  </swes:description>
</swes:DescribeSensorResponse>
```

12 UpdateSensorDescription

12.1 Introduction

The *UpdateSensorDescription* allows updating the metadata of registered sensors. Services that implement this operation allow clients to update the current description of a sensor.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/operation/current	
REQ 37.	By default, the *UpdateSensorDescription* operation shall support the update of the current description of the sensor/procedure.

Any service may also support updating descriptions for other points in time or time periods.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/operation/time	
REQ 38.	Services supporting the update of other than the current sensor/procedure description shall advertise this by adding a parameter named *validTime* to the metadata of the operation named *UpdateSensorDescription* in the *OperationsMetadata* section of the service's Capabilities document. If the parameter is missing the option is not supported by the service.

If the *validTime* parameter is supported by the service and used by the client, the service needs to determine whether previous descriptions must be modified. The concrete action depends on the relation between the existing descriptions and the new description.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/modification/timePeriods	
REQ 39.	If a service receives an *UpdateSensorDescription* request with a new description that is valid for a time period, existing sensor descriptions shall be modified according to Figure 13 (if the existing description is valid for a time period).

As an example, if the new description addresses a time period that overlaps with an existing description, the latter one has to be shorten so that both descriptions meet

temporarily. Figure 13 shows all cases that can happen when a new description for a time period gets updated. The left column defines the temporal case, the red line indicates the temporally valid time period of the existing description, the two dashed lines indicate start and end point of the new description, and the right column defines the necessary modification the service has to apply on the existing description.

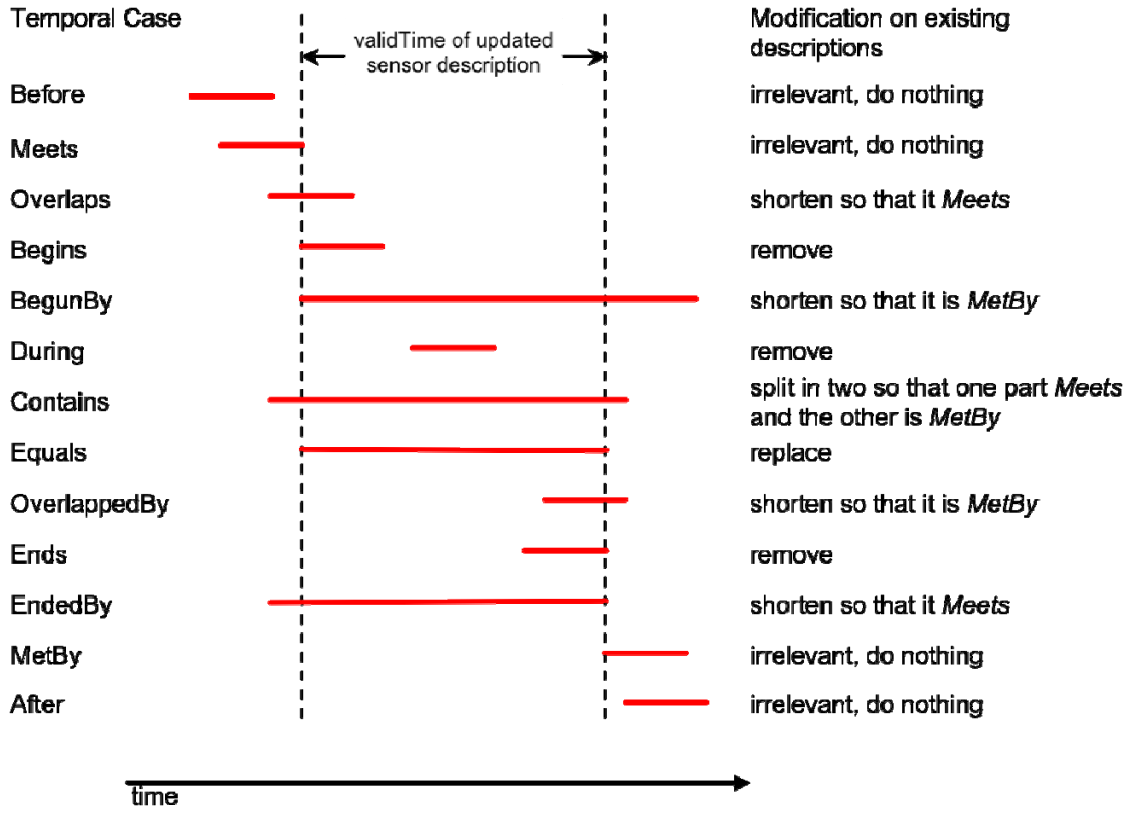

Figure 13 — temporal relationships between existing and new descriptions and required server-side modifications for time intervals

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/modification/instantInInterval	
REQ 40.	If a service receives an *UpdateSensorDescription* request with a new description that is valid for a particular point in time, existing sensor descriptions shall be modified according to Figure 14 (if the existing description is valid for a time period).

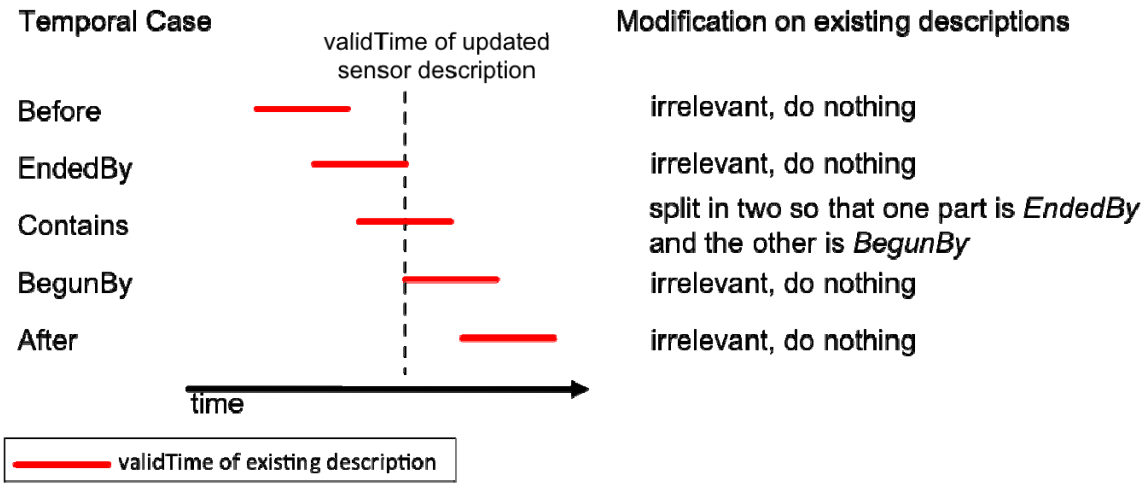

Figure 14 — temporal relationships between existing and new descriptions and required server-side modifications for the integration of time instants into time intervals

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/modification/intervalInInstant	
REQ 41.	If a service receives an *UpdateSensorDescription* request with a new description that is valid for a time period, existing sensor descriptions shall be modified according to Figure 15 (if the existing description is valid for a time instant).

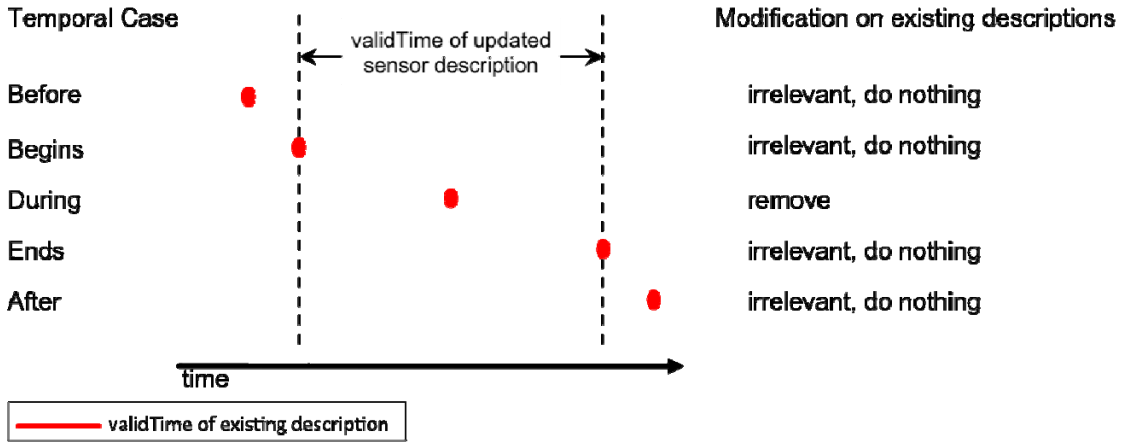

Figure 15 — temporal relationships between existing and new descriptions and required server-side modifications for the integration of time periods with time instants

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/modification/instantInInstant	
REQ 42.	If a service receives an *UpdateSensorDescription* request with a new description that is valid for a particular point in time, existing sensor descriptions shall be modified according to Figure 16 (if the existing description is valid for a time instant).

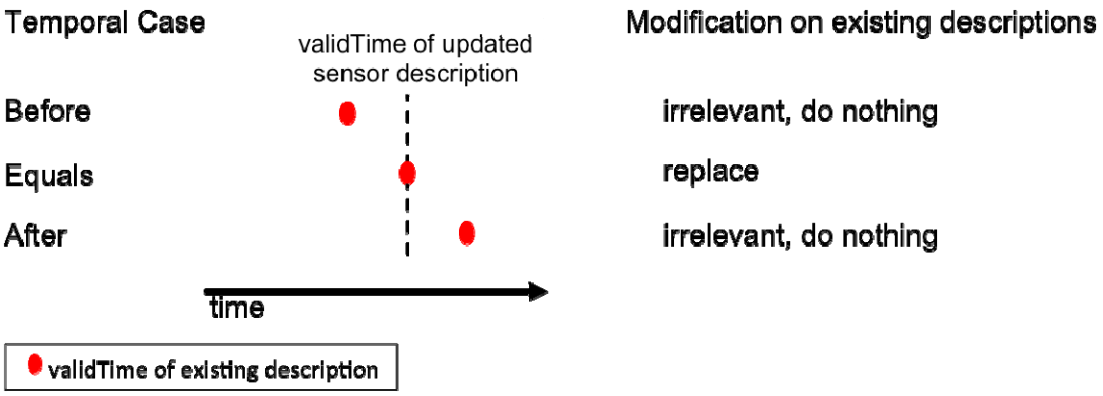

Figure 16 — temporal relationships between existing and new descriptions and required server-side modifications for time instants

This means that a client does not need to update a sensor description by providing the exact same *validTime* as existing descriptions, but the server takes appropriate measures to ensure temporal validity and correctness.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/modification/withoutValidTime	
REQ 43.	If a service receives an *UpdateSensorDescription* request that does not contain a *validTime*, then the new description is considered to be valid from the time of request reception until further requests are received. Hence all existing descriptions end at the time current time. All existing descriptions addressing the future become void shall be removed by the service.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/sideEffects	
REQ 44.	The *UpdateSensorDescription* operation shall not have any side effects other than the pure update of the sensor description. Hence, for example, a successful update of a sensor's description through this operation shall not cause a task to fail at an SPS (although the event that caused the necessity to issue an *UpdateSensorDescription* request might have caused the task to fail).

12.2 Data Types

The conceptual model of the *UpdateSensorDescription* operation is shown in the following UML diagram.

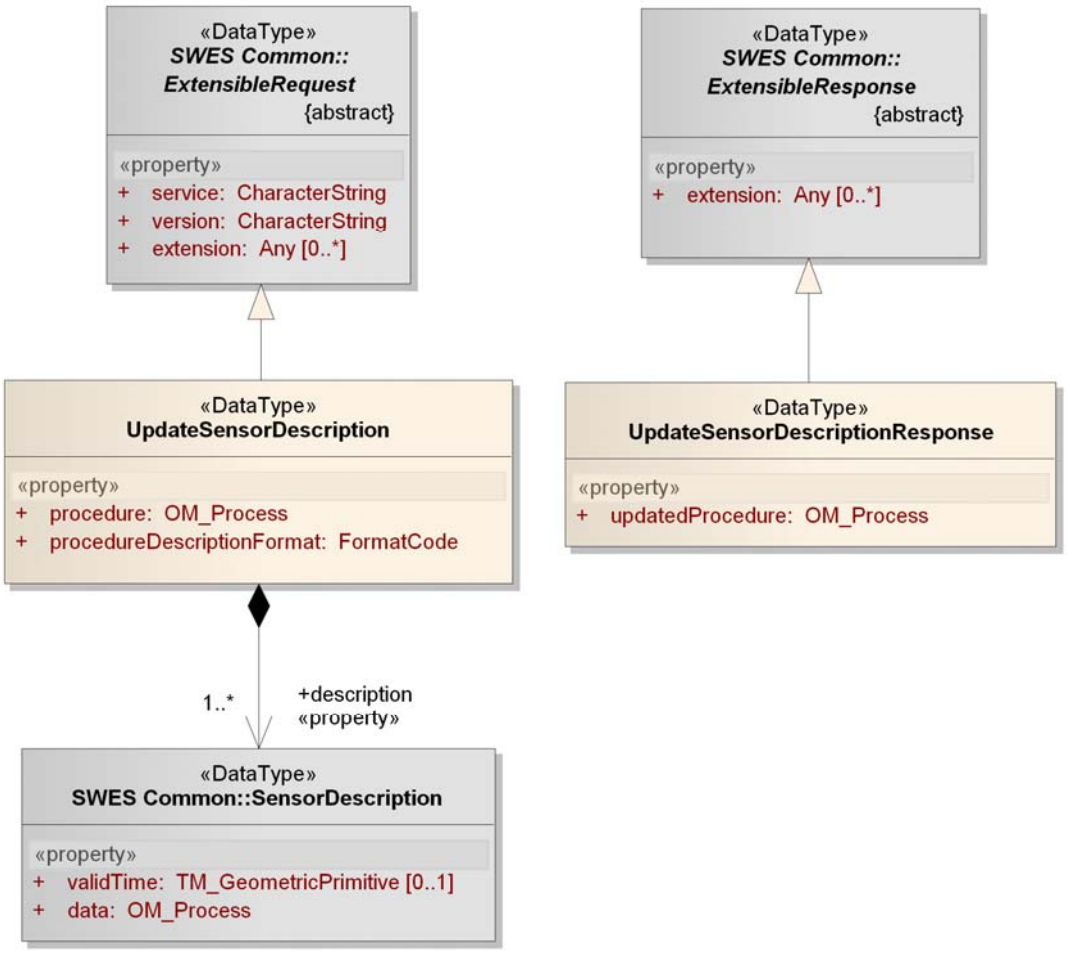

Figure 17 — Data types of the UpdateSensorDescription operation

The details of the operation request and response will be explained in the following subsections.

12.2.1 Operation Request – UpdateSensorDescription

An *UpdateSensorDescription* operation request is performed by sending an instance of the *UpdateSensorDescription* data type to the service.

The *UpdateSensorDescription* data type derives from the *ExtensibleRequest* data type (see clause 9.2.1) and thus inherits all the properties contained in that data type. *UpdateSensorDescription* does not restrict the content model of *ExtensibleRequest*.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/Request/properties	
REQ 45.	*UpdateSensorDescription* shall contain the properties defined for *ExtensibleRequest*. In addition, it shall include the properties according to Table 28.

Table 28 — Properties in the UpdateSensorDescription data type

Name	Definition	Data type and values	Multiplicity and use
procedure	Pointer to the procedure/sensor of which the description shall be updated.	OM_Process [a] see ISO/DIS 19156 value shall match one of the procedures from the AbstractOfferings listed by the service (see clause 7.2.2)	One (mandatory)
procedureDescriptionFormat	identifier of the format in which the procedure/sensor description is given in	FormatCode see section 10.2.2 value shall match one of the formats supported by the AbstractOffering(s) (see clause 7.2.2) with the given procedure	One (mandatory)
description	the updated description	SensorDescription see clause 9.2.3	One to many (mandatory)
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

12.2.2 Operation Response – UpdateSensorDescriptionResponse

The *UpdateSensorDescriptionResponse* data type represents the response to an *UpdateSensorDescription* operation request.

The *UpdateSensorDescriptionResponse* data type derives from the *ExtensibleResponse* data type (see clause 9.2.2) and thus inherits all the properties contained in that data type. *UpdateSensorDescriptionResponse* does not restrict the content model of *ExtensibleResponse*.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/Response/properties	
REQ 46.	*UpdateSensorDescriptionResponse* shall contain the properties defined for *ExtensibleResponse*. In addition, it shall include the properties according to Table 29.

Table 29 — Property in the UpdateSensorDescriptionResponse data type

Name	Definition	Data type and values	Multiplicity and use
updatedProcedure	Pointer to the procedure/sensor of which the sensor description was updated.	OM_Process [a] see ISO/DIS 19156 value shall be the one provided in the request	One (mandatory)
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

12.3 Exceptions

Clause 15 defines the exception codes that apply to this operation. In addition:

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/exception/validTimeNotSupported	
REQ 47.	If an *UpdateSensorDescription* request contains a *SensorDescription* with *validTime* property but the server does not support this option (i.e. it only supports update of the most recent sensor description), an exception with code *OptionNotSupported* and locator value *validTime* shall be thrown.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/UpdateSensorDescription/exception/unknownProcedureDescriptionFormat	
REQ 48.	If an *UpdateSensorDescription* request contains a *procedureDescriptionFormat* that is not contained in at least one of the offerings with the same value for the procedure identifier as the one given in the request, an exception with code *InvalidParameterValue* and locator value *procedureDescriptionFormat* shall be thrown.

12.4 Examples

The following two listings provide examples of an UpdateSensorDescription operation request and response encoded in XML according to the XML Schema defined in Annex B.

Listing 3 – UpdateSensorDescription operation request example

```
<swes:UpdateSensorDescription service="SPS" version="2.0"
xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <swes:procedure>http://my.org/sensors/937239</swes:procedure>
<swes:procedureDescriptionFormat>http://www.opengis.net/sensorML/1.0.1<
/swes:procedureDescriptionFormat>
  <swes:description>
    <swes:SensorDescription>
      <swes:data>
        <sml:System>
          <!-- ... -->
        </sml:System>
      </swes:data>
    </swes:SensorDescription>
  </swes:description>
</swes:UpdateSensorDescription>
```

Listing 4 - UpdateSensorDescription operation response example

```
<swes:UpdateSensorDescriptionResponse
xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <swes:updatedProcedure>http://my.org/sensors/937239</swes:updatedPro
cedure>
</swes:UpdateSensorDescriptionResponse>
```

13 InsertSensor

13.1 Introduction

This operation is used to add new sensors to the service. The parameters provided by the client provide sufficient information for the service to include the new sensor. It defines the properties required to populate an *AbstractOffering* (see clause 7.2.2) and an extension point to allow the integration of service specific metadata.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/InsertSensor/newOffering	
REQ 49.	If a service accepts an *InsertSensor* request, it shall create a new offering (see clause 7.2.2) in its contents section to host the procedure.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/InsertSensor/identifier	
REQ 50.	If any service receives and accepts an *InsertSensor* request without a procedure identifier, then the service shall assign a new one for the procedure.

The model of this operation includes an abstract property *InsertionMetadata* for which no non-abstract subclass is defined in this document. This is the extension point where other specifications may require specific metadata elements to be added.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/InsertSensor/insertionMetadata	
REQ 51.	Any service implementing the *InsertSensor* operation shall add required service specific metadata elements by implementing the *InsertionMetadata* element(s).

13.2 Data Types

The conceptual model of the *InsertSensor* operation is shown in the following UML diagram.

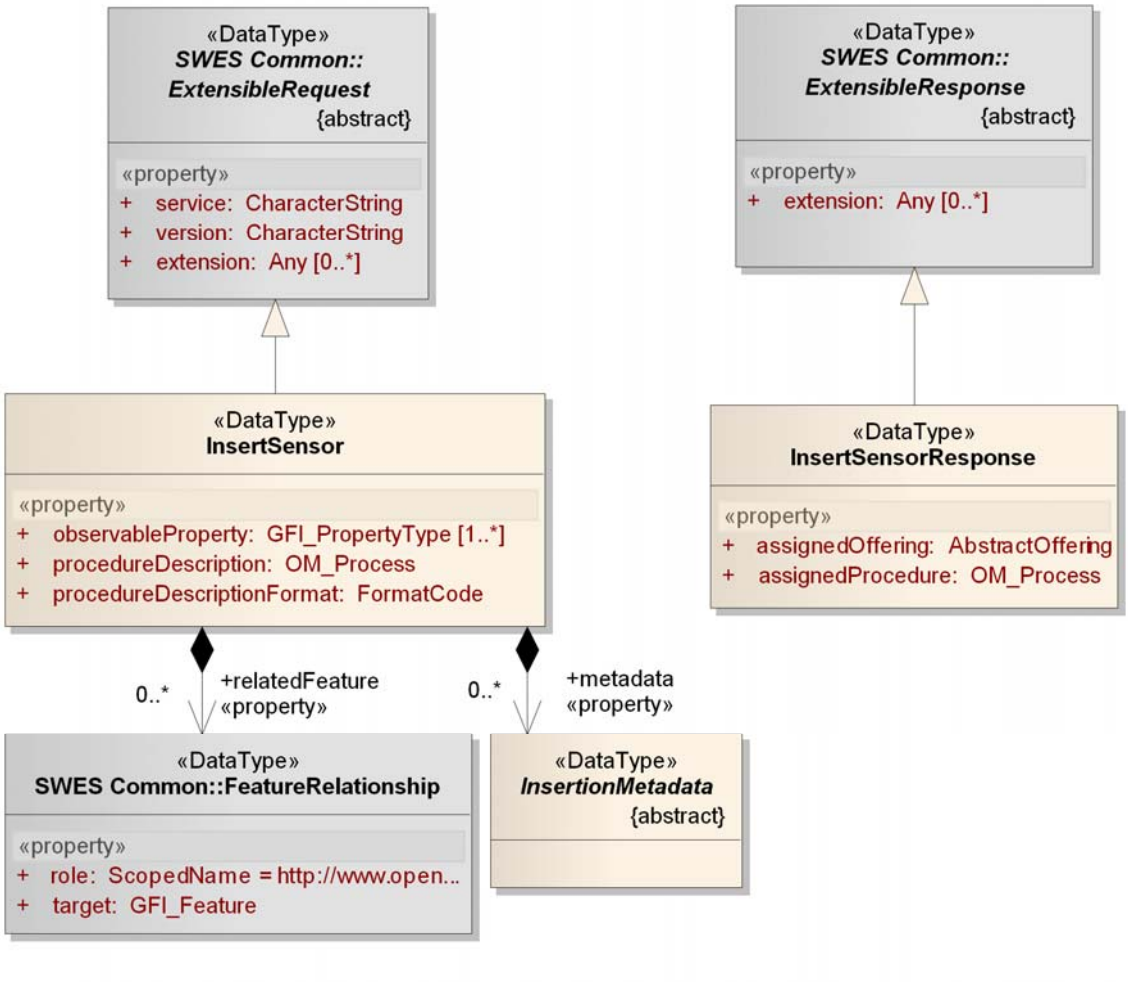

Figure 18 — Data types of the InsertSensor operation

The details of the operation request and response will be explained in the following subsections.

13.2.1 Operation Request – InsertSensor

An *InsertSensor* operation request is performed by sending an instance of the *InsertSensor* data type to the service.

The *InsertSensor* data type derives from the *ExtensibleRequest* data type (see clause 9.2.1) and thus inherits all the properties contained in that data type. *InsertSensor* does not restrict the content model of *ExtensibleRequest.*

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/InsertSensor/Request/properties	
REQ 52.	The *InsertSensor* request shall contain the properties defined for *ExtensibleRequest*. In addition, it shall include the properties according to Table 30.

Table 30 — Properties in the InsertSensor data type

Name	Definition	Data type and values	Multiplicity and use
procedureDescriptionFormat	identifier of the format in which the procedure/sensor description is given in	FormatCode see section 10.2.2 value shall match one of the formats supported by the AbstractOffering(s) (see clause 7.2.2) with the given procedure	One (mandatory)
procedureDescription	the current description of the procedure	OM_Process see ISO/DIS 19156 type shall conform to the format identified via the procedureDescriptionFormat property	One (mandatory)
observableProperty	Pointer to a property that *can* be observed by the procedure, not a property that *has* already been observed.	GFI_PropertyType [a] see ISO/DIS 19156	Zero or more (optional)
relatedFeature	feature that is directly or indirectly observed/observable by the procedure; can be any feature which the requestor thinks the procedure can make valuable observations for	FeatureRelationship see clause 9.2.4	Zero or more (optional)
metadata	additional information required for inserting the sensor at a specific service (like SOS or SPS)	InsertionMetadata see clause 13.2.2	Zero to many (optional) *abstract property – see clause 13.1 and 13.2.2*
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

13.2.2 InsertionMetadata

This abstract type represents a placeholder for all information required by SWE service implementation specifications that realize the *InsertSensor* operation. All information

required by such a specification to insert a new sensor shall be defined as property data contained in non-abstract subclasses of *InsertionMetadata*. This allows the creation of several distinct metadata types that together constitute the information required for inserting a new sensor at a specific SWE service instance.

The operation may be implemented without requiring any more metadata.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/InsertSensor/metadata	
REQ 53.	If specific metadata is required for successful insertion of a new sensor, then this information shall be provided via the *metadata* property, not via the *extension* property (as defined in the *ExtensibleRequest*).

Defining several metadata types might be necessary to support service instances that implement a specific set of functionality. Some operations supported by a service might require specific information to be included when inserting a new sensor. However, the operation itself might be optional and therefore not every service instance needs to implement it.

Note: The insertion of a new sensor at multiple service types (like SPS and SOS) in one request is not possible. An *InsertSensor* request can only target one service type (to be announced in the "service" property of the request, see clause 9.2.1). The need for multiple metadata objects in one *InsertSensor* request might nevertheless arise if metadata specific for certain operations of a service type is needed. If the evolution of the service specification did not consider all metadata upfront – then missing information can simply be added through a new metadata type.

13.2.3 Operation Response – InsertSensorResponse

The *InsertSensorResponse* data type represents the response to an *InsertSensor* operation request.

The *InsertSensorResponse* data type is derived from the *ExtensibleResponse* data type (see clause 9.2.2) and therefore inherits all the properties contained in that data type. *InsertSensorResponse* does not restrict the content model of *ExtensibleResponse*.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/InsertSensor/Response/properties	
REQ 54.	The *InsertSensorResponse* shall contain the properties defined for *ExtensibleResponse*. In addition, it shall include the properties according to Table 31.

Table 31 — Property in the *InsertSensorResponse* data type

Name	Definition	Data type and values	Multiplicity and use
assignedProcedure	Pointer created by the service for the procedure/sensor that was successfully inserted.	OM_Process [a] see ISO/DIS 19156 Value may be a pointer/an identifier provided in the sensor description of the InsertSensor request; however, the service is free to assign a new value if it cannot find one in the description or if that value is already in use.	One (mandatory)
assignedOffering	Pointer to the offering created by the service to host the new procedure.	AbstractOffering [a] see clause 7.2.2	One (mandatory)
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

13.3 Exceptions

Clause 15 defines the exception codes that apply to this operation. In addition:

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/InsertSensor/exception/unknownProcedureDescriptionFormat	
REQ 55.	If an *InsertSensor* request contains a description property that is not compliant to the *procedureDescriptionFormat* provided in the request, an exception with code *InvalidParameterValue* and locator value *procedureDescriptionFormat* shall be thrown.

13.4 Examples

This standard does not define all the information required to create a meaningful *InsertSensor* request. The necessary information and further operation semantics need to be defined by standards leveraging SWES. As such, this specification does not provide concrete examples for *InsertSensor* operation request and response.

14 DeleteSensor

14.1 Introduction

This operation is used to delete a sensor at a SWE service.

Note: this operation is abstract, even though the operation request and response data types are non-abstract! The operation can only be used in combination with a given SWE service if the specification of that service type clearly defines the semantics of this operation. These are quite diverse and at the time of writing this specification, were not clear for existing services. Different behavior is possible. For example, if a sensor was deleted at SPS, then one behavior could be to immediately fail all currently running tasks for the sensor and not accepting any new, while another behavior could be to not accept new tasks but let all currently running tasks finish as expected. At SOS, one behavior could be to completely remove the procedure from the service capabilities and delete all observations associated to that sensor, while another one could be to keep that information but to not accept insertion of any new data for that sensor. Another aspect to consider is that a procedure may belong to more than one offering.

The diversity of possible behavior cannot be foreseen here and therefore only the operation syntax is defined in this specification, not the complete semantics.

14.2 Data Types

The conceptual model of the *DeleteSensor* operation is shown in the following UML diagram.

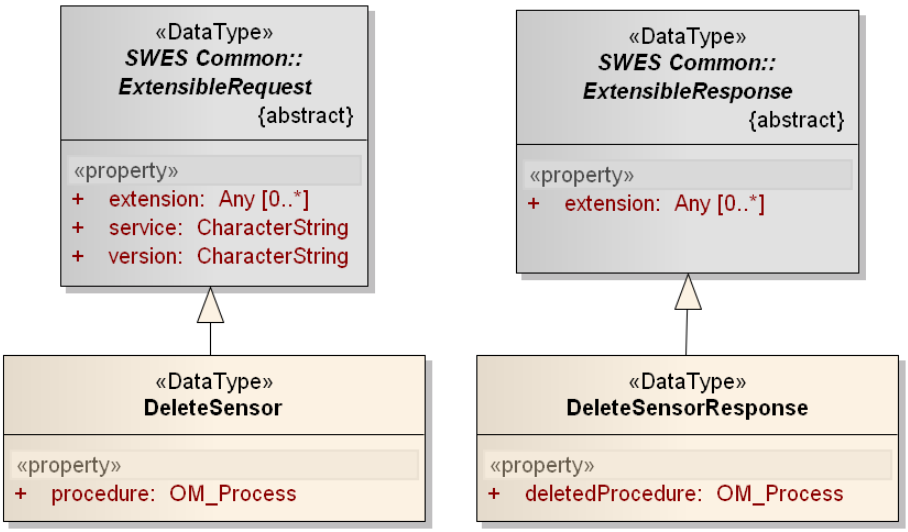

Figure 19 — Data types of the DeleteSensor operation

The details of the operation request and response will be explained in the following subsections.

14.2.1 Operation Request – DeleteSensor

A *DeleteSensor* operation request is performed by sending an instance of the *DeleteSensor* data type to the service.

The DeleteSensor data type is derived from the ExtensibleRequest data type (see clause 9.2.1) and therefore inherits all the properties contained in that data type. DeleteSensor does not restrict the content model of ExtensibleRequest.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DeleteSensor/Request/properties	
REQ 56.	The *DeleteSensor* request shall contain the properties defined for *ExtensibleRequest*. In addition, it shall include the properties according to Table 32.

Table 32 — Property in the *DeleteSensor* data type

Name	Definition	Data type and values	Multiplicity and use
procedure	Pointer to the procedure/sensor that shall be deleted. operation semantics are defined by specification of targeted SWE service type	OM_Process [a] see ISO/DIS 19156 value shall match one of the procedures from the AbstractOfferings listed by the service (see clause 7.2.2)	One (mandatory)
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

14.2.2 Operation Response – DeleteSensorResponse

The *DeleteSensorResponse* data type represents the response to a *DeleteSensor* operation request.

The *DeleteSensorResponse* data type is derived from the *ExtensibleResponse* data type (see clause 9.2.2) and therefore inherits all the properties contained in that data type. *DeleteSensorResponse* does not restrict the content model of *ExtensibleResponse*.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/DeleteSensor/Response/properties	
REQ 57.	The *DeleteSensorResponse* shall contain the properties defined for *ExtensibleResponse*. In addition, it shall include the properties according to Table 33.

Table 33 — Property in the DeleteSensorResponse data type

Name	Definition	Data type and values	Multiplicity and use
deletedProcedure	Pointer used to reference the procedure that has been deleted by the service.	OM_Process [a] see ISO/DIS 19156	One (mandatory)
a) Note: the primary use of this property is to provide a pointer/identifier – see clause 16.3.1 for further details			

14.3 Exceptions

Clause 15 defines the exception codes that apply to this operation.

14.4 Examples

The *DeleteSensor* operation is defined by SWES as an abstract operation. The missing operation semantics need to be defined by standards leveraging SWES. Examples for the *DeleteSensor* operation requests and responses should be provided by such standards.

15 SWE Service Model Exceptions

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Exceptions/general	
REQ 58.	When a server encounters an error while performing one of the operations defined in this standard, it shall return an exception message as specified in chapter 8 of [OGC 06-121r3].

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Exceptions/codes	
REQ 59.	The allowed standard exception codes are those defined in table 25 of OGC 06-121r3 and those defined in Table 34 in this standard. For each listed *exceptionCode*, the contents of the *locator* parameter value shall be used as specified in the right column of the according table.

Table 34 — Exception (code) defined by SWES

exceptionCode value	Meaning of code	"locator" value
InvalidRequest	Request does not conform to its XML Schema definition.	Exception message generated by validator
RequestExtensionNotSupported [a]	One or more of the extensions used in the operation request are unknown to or not supported by the service.	None, omit "locator" parameter
a A service shall add the complete unsupported request extension into the ExtensionText property of the Exception. In the XML encoding, use CDATA to surround markup. If the encoded extension itself contains the string "]]>", use two CDATA tags following this pattern: <![CDATA[*content]]*]]><![CDATA[>*content*]]> (see Listing 5).		

NOTE: Each SWES operation may define additional requirements with respect to exception handling for that operation.

An example for the *RequestExtensionNotSupported* code is given in the following listing.

Listing 5 – Example Exception with code RequestExtensionNotSupported

```
<ows:Exception exceptionCode="RequestExtensionNotSupported"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:abc="http://www.example.org/abc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ows:ExceptionText>
  <![CDATA[
    <abc:MyExtParameter>
      <abc:PropertyA>...</abc:PropertyA>
      <abc:PropertyB>
        <!CDATA[some other markup]]
  ]]>
  <![CDATA[
        >
      </abc:PropertyB>
    </abc:MyExtParameter>
  ]]>
  </ows:ExceptionText>
</ows:Exception>
```

Figure 20 presents which exceptions are applicable to which SWES operation.

Operation \ exceptionCode	OperationNotSupported	MissingParameterValue	InvalidParameterValue	OptionNotSupported	NoApplicableCode	InvalidRequest	RequestExtension NotSupported
DescribeSensor	x	x	x	x	x	x	x
UpdateSensorDescription	x	x	x	x	x	x	x
InsertSensor	x	x	x	x	x	x	x
DeleteSensor	x	x	x	x	x	x	x
exception code defined by:	[OGC 06-121r3]					this standard	

Figure 20 — SWES operations with applicable exceptionCodes

NOTE: Because the operations defined in this standard are not required to be implemented by a standard leveraging the SWE service model, the OperationNotSupported exception is flagged for all operations. Another standard may use one or more of these operations and may make one or more of them required – in that case the OperationNotSupported exception would no longer be applicable.

16 Identifier Usage

16.1 Introduction

Being able to identify something is critical both in everyday life and IT systems. Each object – a real world object or one used in software – has identity.

The identity of an object can be implicit in which case we just know of the object being a unique entity that exists. If it is explicit, then we know the object by name. That name is unique in a given namespace. If an object is known in more than one domain, it may have different names assigned in the namespaces of the domains. A domain may even assign multiple names to the same object. In general, the domain that owns that namespace governs naming of objects.

The following chapter explains the implementation and usage of identifiers according to ISO/GML. How identifiers are used in the SWE Service Model is explained afterwards. Handling of identifiers is slightly but significantly different in the two approaches, this is the reason why additional documentation was deemed necessary.

16.2 Implementation in ISO 19103 and ISO 19136

ISO 19103 defines three classes to support the concept of explicit identity: GenericName, LocalName and ScopedName – see Figure 21.

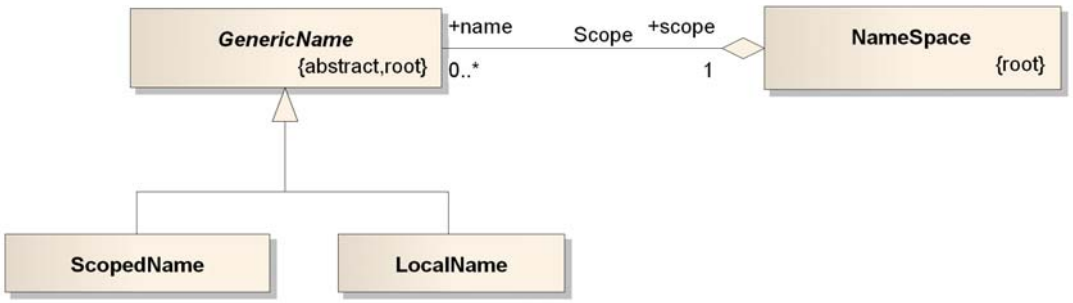

Figure 21 — overview of name types defined by ISO 19103

Without going into too much detail here, a *GenericName* is bound to a *NameSpace* and is bound to a certain object and thus identified in the namespace. *GenericName* itself is abstract and therefore needs to be substituted by a non-abstract subclass, in this case *LocalName* or *ScopedName*. ISO 19136/OGC 07-036 defines the mapping of both types to XML encodings (see table D.2 in OGC 07-036).

Both *LocalName* and *ScopedName* belong to a namespace. The difference in the XML encoding defined by GML is that for a *ScopedName* the namespace is explicitly provided in the *codeSpace* attribute while for a *LocalName* the namespace is usually implicit - but may also be provided via the (optional) *codeSpace* attribute.

The XML encoding of *LocalName* according to OGC 07-036 Table D.2 is *gml:CodeType* while the encoding of *ScopedName* is *gml:CodeWithAuthorityType* – see following listing.

Listing 6 – GML CodeType and CodeWithAuthorityType

```
  <complexType name="CodeType">
    <annotation>
      <documentation>
            gml:CodeType is a generalized type to be used for a
         term, keyword or name. It adds a XML attribute codeSpace
         to a term, where the value of the codeSpace attribute
         (if present) shall indicate a dictionary, thesaurus,
         classification scheme, authority, or pattern for the
         term.
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string">
        <attribute name="codeSpace" type="anyURI"/>
      </extension>
    </simpleContent>
  </complexType>

  <complexType name="CodeWithAuthorityType">
    <annotation>
      <documentation>
      gml:CodeWithAuthorityType requires that the codeSpace
      attribute is provided in an instance.
      </documentation>
    </annotation>
    <simpleContent>
      <restriction base="gml:CodeType">
        <attribute name="codeSpace" type="anyURI"
use="required"/>
      </restriction>
    </simpleContent>
  </complexType>
```

A namespace/*codeSpace* may be implemented as a dictionary, thesaurus or classification scheme. It can also be a register (available or virtual) of any authority, e.g. standardization body, governmental organization etc.

The key point is that – following GML encoding rules – the *codeSpac/*namespace has to be provided for the XML encoding of a *ScopedName* while it is optional in the encoding of a *LocalName*.

Whenever *GenericName* is used in a UML model, either *LocalName* or *ScopedName* replaces it in the implementation. In the XML encoding following GML rules, this is possible because the XML encoding of a *GenericName* is also *gml:CodeType*.

Whether a name is unique or uniquely identifies an object in a given *codeSpace*/namespace is up to the authority responsible for that *codeSpace*/namespace. However, as one name is bound to one object only in one namespace, in the XML encoding of *GenericName* (and thus *ScopedName* and *LocalName*) only the combination of the value and the *codeSpace* (provided explicitly or implicitly) make an identifier unique.

Example: an identifier with value “observation-1” can be assigned to different sensor outputs. Only in combination with a codeSpace - like “http://some.organization.org/sensor1/” does the identifier reference one unique object.

16.3 Implementation in SWE Service Model

In a service environment, concepts, objects (real, simulated or virtual) and representations of these objects need to be identified – for example:

- Procedure/process/sensor
- Observation
- Phenomenon/(feature) property
- mimeType
- Filter dialect
- Format of a procedure description (e.g. SensorML)
- Extension/a conformance class/conformance profile

How these are modeled, encoded and used in SWE services is explained in the following sub clauses.

16.3.1 Modeling identifier properties

This section explains the approach that was chosen to model properties whose primary purpose is to identify a particular object. The implications and advantages over the GML approach described before are discussed. In addition, we will emphasize some critical aspects and explain why this approach needs to be applied carefully.

Oftentimes service models use indirect identifiers for referring to objects that are of interest. This is depicted in Figure 22.

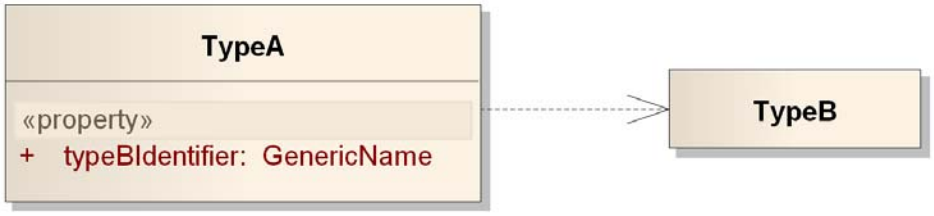

Figure 22 – Using indirect identifiers for modeling identified object relationships

In this example, objects of *TypeA* have a property *typeBIdentifier* of type *GenericName*. *GenericName* is just a name that does not provide information about the type of the referenced object (e.g. urn:foo:object:39). Though the type might be indicated by the *GenericName* (e.g. urn:foo:object:**CategoryObservation**:22), it usually remains vague.

The *propertyName* might provide more information about the type of the identified object. A dependency relationship as illustrated in Figure 22 might provide more information as well. In any case, the relationship between *TypeA* and *TypeB* remains weak.

Example: The following example shows a weak relationship between the object *TemperatureSensor* and the referenced object, which is named *observationStation*. The reference name does not provide any information about the type of the referenced object, though it indicates that it is an observation station in principle.

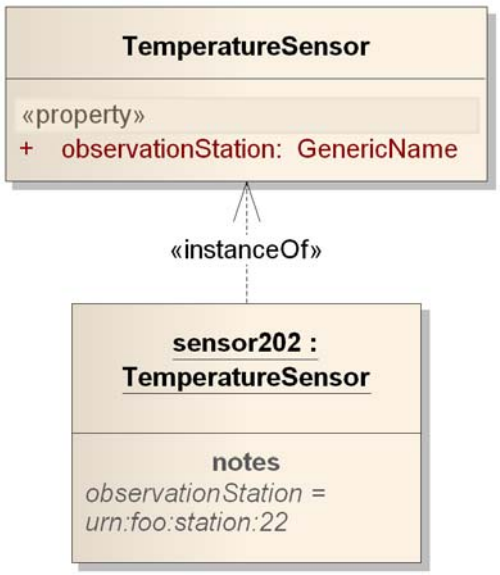

Figure 23: Example of a weak relationship

To understand the relationship between the two types, further explanation has to be provided in a separate way. Another issue is that in order to actually get the identified object, a client would need to know at which registry service it can have the identifier value resolved to the actual (*TypeB*) object.

Figure 24 shows a different approach in which a strong relationship between *TypeA* and *TypeB* exists.

Figure 24 – Using direct pointer for modeling identified object relationship

Here, the directed association from *TypeA* to *TypeB* provides the information about the type of the referenced object. In addition, the model allows providing more detailed information about the referenced object by selecting an appropriate role name.

Example: The following example shows a strong relationship between the object *TemperatureSensor* and the referenced object, which is of type *WeatherStation*. The role provides further information about the relationship. Here, it defines that the *TemperatureSensor is_part_of* the *WeatherStation.*

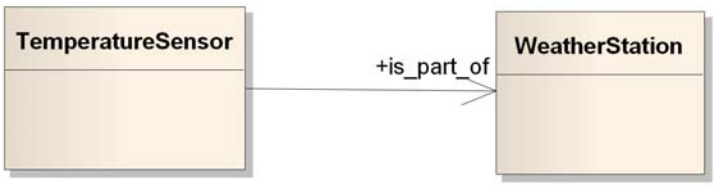

Figure 25: Example of a strong relationship

The two modeling approaches have been discussed during the development of the SWE Service Model. The Standards Working Group eventually decided to follow the second approach (depicted in Figure 24) for modeling identified object relationships in SWE service models. The implications of this choice are described in the following.

Whenever the purpose of a given type's property is to point to or to identify another object, the documentation of this property should highlight this characteristic.

Note: Examples can be found throughout this specification, e.g. the *procedure* property in the *AbstractOffering* type (clause 7.2.2), which provides a "*Pointer to* the procedure/sensor associated with this offering" and has an according table footnote.

Whenever a SWE service requires that object types used in the conceptual model are identifiable (e.g. *offerings* in a SOS instance) then they need to be identifiable objects. This means that they are either SWES objects or GML objects. SWES objects always inherit from *AbstractSWES* (see Figure 29), GML objects always inherit from *AbstractGML*.

All SWES and GML objects provide an optional *identifier* property. Whenever a service provider wants to ensure identity for its objects, this property shall become mandatory.

Example: A SOS instance lists a number of *offerings* in the *Contents* section of its *capabilities* document. A SOS offering is modeled as a SWES object. By default, the *identifier* property would be optional. If

omitted, clients could not use a reference to any offering in a service call. To support this functionality, the SOS specification has to explicitly state that the *identifier* property is mandatory for all SOS offerings. In contrast, SPS uses offerings as well, but the core SPS does not require offerings being identifiable. Clients simply do not use offering identifiers as part of their service calls. This behavior would change if a SPS provider announces its offerings by pub&sub advertisements; more specifically the SPS instance implements the OfferingAdded or OfferingDeleted events (see clause 17.2.2). Here, offerings get referenced, i.e. need to be identifiable and the property *identifier* becomes mandatory again.

The strong relationship model has another advantage over the weak relationship model. The advantage of the chosen approach is that in specific realizations of the conceptual model such a property may identify a specific object and provide the mechanism to resolve the identifier directly (e.g. if the identifier is a URL, the resolving mechanism is that of http). An additional proprietary resolving mechanism would become redundant.

Example: The identifier http://www.foo.org/observation37 can be directly resolved using http techniques, whereas the identifier urn:foo:observation:37 cannot be directly resolved. The same applies to combinations of *codeSpace* and *value* tupels. Even if both are modeled as URLs, it would not be clear how to resolve the object itself. Similar problems apply for URNs. As there is no standard mechanism to resolve URNs, the client would even need to know how to interact with the resolver.

Identifiers of SWES objects are encoded in XML Schema as elements of type xs:anyURI (see clause 24.2.4.1) (in contrast to GML objects that use *codeSpace* and *value*). Properties that point to objects are also encoded as elements of type xs:anyURI, matching the model of SWES object identifiers.

Example: The *OfferingAdded* event points to the offering that was added. This pointer is of type xs:anyURI in the XML serialization.

The decision to encode the *identifier* property in AbstractSWES as xs:anyURI was deliberate. In the conceptual model of *AbstractSWES* the type of the property is *ScopedName* (which is always bound to a namespace as described above (16.2)) just like the identifier in *AbstractGML*, but serialized differently. In GML, the *ScopedName* is serialized as *gml:CodeWithAuthorityType* (that uses *codeSpace* and *value*). In the SWE Service Model, the XML Schema encoding of *ScopedName* is xs:anyURI. Thus SWES objects are identified via URIs. The authority of URIs is either explicit (for example in case that the URI is an HTTP or FTP URI) or implicit (for example if the URI is a URN). URIs therefore satisfy the requirement that they are bound to a namespace.

Care needs to be taken when applying this approach to point to or to identify GML objects. Here the identifier property provided by the *AbstractGML* base type is encoded as type *gml:CodeWithAuthorityType*. This implies that not a single URI is used to identify an object but rather a combination of two distinct values. When the conceptual model is realized using XML Schema, SWE services leveraging this standard should ignore the *codeSpace* value of GML objects (e.g. the *OM_Process* in the procedure property of an *AbstractOffering*, see clause 7.2.2). Instead, they should use the URI provided in the value of the *gml:identifier* element.

Note: Extensions to this standard can define ways to combine codeSpace and value into one URI.

16.3.2 Identity of objects and their representations

Objects – whether they exist in the real world or are part of a computer program – have identity. In distributed computing, the identity of an object in general and the identity of a representation created of that object at a certain time should be different. This is particularly important if the representation itself needs to be traceable.

An example shall help to state this fact: A sensor performs an in-situ measurement of air temperature. It provides this measurement – with according phenomenon time and result value – as an instance of an O&M *Observation*. The observation is a representation of the observation event. The observation is stored in a Sensor Observation Service. The SOS implements publish/subscribe functionality and sends a notification message including the observation to interested consumers.

Two cases need to be differentiated:

1. The serialization of the observation contains an explicit identifier

2. The serialization of the observation does not contain such an identifier

The subscribers of the SOS receive the observation. At a later stage, a validation process recognizes that the result of the observation is incorrect and therefore sends out another notification to inform interested consumers. If the observation can no longer be identified because the serialization of the observation did not contain an explicit identifier then the consumer cannot update its internal data store or computations that may already have been performed on the original observation.

NOTE: At this point, one could say that the combination of some object properties already uniquely identifies this object. However, this may not necessarily be true in all cases. Furthermore, comparing identity of two objects by comparing their property values can be quite time-consuming for complex properties. Therefore, it is better to provide explicit identity to all object representations that might need to be referenced in the system.

Such a computation can for example include the distribution of the observation to other services. With explicit identifiers included in the serialization of object representations, these representations can be traced in a network of distributed services.

An object in the real world can also be represented in different ways. It is important to understand the difference between identifiers of these representations and a label for the real-world object. Suppose that a tree is represented through two objects that belong to different classes defined in (a) GML application schema. The classes can have different properties and the amount of information can vary considerably. An authority can assign a label for the tree. Would that label be the identifier used for the two objects? No, each object has its own object identifier, its own identity. However, because the objects both represent the same tree, they can carry the same label. In GML features, such labels can be encoded using the *gml:name* property. Likewise, SWES objects can also be labeled using the *swes:name* property (see Figure 29). If that mechanism is not applicable for any reason, then a dedicated property for identifying the real-world object needs to be added to the types used for representing the object.

Note: Providing explicit revision tags to SWES objects is not defined in this standard. A future revision or extension can define this functionality.

16.3.3 Identifiers in GML Application Schema and SWE Service Model

In a GML application schema, all object types have identity. If serialized to XML, the objects are identified using the *gml:id* attribute, which is mandatory for all subtypes of type *AbstractGML*. Services may impose additional constraints – for example, WFS requires that the *gml:id* is unique within a WFS instance, thus making the *gml:id* a *LocalName* (where the implied namespace is the service itself). To identify a GML object outside the local scope (XML instance document or WFS service), the *gml:identifier* element shall be used.

NOTE GML mentions global uniqueness with respect to the gml:identifier property. However, uniqueness of an identifier can only be guaranteed in the scope of the system that is controlled by the authority responsible for assigning the identifier.

The *gml:identifier* element is the XML encoding of an optional identifier property in *AbstractGML* of type *ScopedName*. Therefore, whenever non-local identity of a GML object is required, the identifier value needs to be accompanied by a *codeSpace*. The combination of identifier *value* and *codeSpace* make the identifier unique.

In the SWE Service Model, an object type also has identity. However, unlike in GML a SWES object does not necessarily have an XML attribute that provides identity within an XML instance document (see clause 24.2.4.1). In SWES, it is the responsibility of the service to assign these attributes to all elements that are referenced (using *xlink:href*) from other elements. Like in GML, SWES does not impose any further constraints on the uniqueness of the *swes:id* attribute. As the type of *swes:id* is *xs:ID*, it only needs to be unique within an XML document. Services that implement this standard may impose additional constraints.

Furthermore - as explained in clause 16.3.1 - SWES objects have a property that is similar to the *gml:identifier* property but has some important differences:

- Although in the UML model (see Figure 29) the property is also of type *ScopedName*, in the XML Schema it is encoded as type *xs:anyURI*

- SWES objects can (therefore) uniquely be identified via a single URI, NOT a combination of *codeSpace* and separate *value*

- The optional *identifier* property of all SWES object types belongs to the *AbstractSWES* type and is therefore in the namespace of SWES, not GML

16.3.4 Referencing SWES objects

Objects encoded according to SWES encoding rules (see clause 20) or GML encoding rules can be given inline and/or by reference – depending on the model design. This is explained in more detail in clause 7.2.3.4 of OGC 07-036.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Identifier/inlineAndByReference	
REQ 60.	If a property is given by reference and also inline, then the value retrieved by following the reference shall be used. If resolving the reference did not succeed then the inline value shall be used as a fallback – this follows the mechanism defined in OGC 07-036 clause 7.2.3.4.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Identifier/xLink	
REQ 61.	The mechanism for providing properties by reference shall be using XLink/XPointer as defined in OGC 07-036 clause 8.1.

NOTE To reference SWES/GML objects using their identifier, it is important to understand the implications of using QNames in query strings of XLinks/XPointers. It is not necessarily safe to assume that the mapping of an XML namespace to its prefix is kept by XML processors/applications. Therefore, the XML namespace should be provided explicitly.

EXAMPLE – Referencing a SWES Object:
xlink:href=http://my.server.com/anyPath/document.xml#xmlns(swes=http://www.opengis.net/swes/2.0)
xpointer(//*[swes:identifier = "http://www.mydomain.com/service/offering/X7KE76"])

NOTE The XML encoding of SWES objects is different compared to GML objects. One change is that SWES objects do not have a mandatory id attribute as described further in clauses 16.3.3 and 24.2.4.1.

17 Communication Patterns

17.1 Asynchronous Request/Response

Traditional client-server communication typically involves an exchange of request and response messages. The response can be transmitted synchronously or asynchronously. OGC 09-032 clause 8.2 explains this in further detail and documents potential realizations of asynchronous responses in different bindings.

17.2 Publish/Subscribe

17.2.1 Introduction

In OGC, one of the less prominent messaging patterns is the Publish/Subscribe pattern. However, it has great value for use cases in which notification of events need to be sent to interested consumers as soon as possible and the frequency with which these events occur is unknown. The pattern is described in more detail in OGC 09-032 clause 8.1.3.

One important aspect of the pattern is the subscription model. According to OGC 09-032, a subscription model characterizes how the producer matches notifications against subscriptions. Various models exist. While this standard does not restrict the set of allowed models, it defines requirements for the usage of the *Channel* and *Filter* subscription models (see OGC 09-032 clauses 8.1.3.1.1 and 8.1.3.1.3 for further details on these two models). Channel based subscription is also called *Topic* based subscription.

Publish/Subscribe implementations can combine various subscription models per binding. This allows clients for example to use both channel and filter criteria in one subscription.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Communication/PubSub	
REQ 62.	Specifications making use of this SWES publish/subscribe standard shall define all functionality to enable publish-subscribe on a per-binding base.

NOTE The realization in the SOAP binding is documented in clause 19.

17.2.2 Events and their encoding

The events recognized by this standard and their encoding are defined in Table 35. A SWE service that implements publish/subscribe functionality is free to publish each type of event.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Communication/PubSub/EventEncoding	
REQ 63.	SWES events published by a service shall be encoded as defined in Table 35.

Table 35 — SWES Events and their encoding

Event name [a]	Event definition	Encoding of the event	Use at SWE service that implements Publish/Subscribe
CapabilitiesChanged	see Table 15 (in the according row)	SWESEvent (see clause 8.2.4) code value shall be "CapabilitiesChanged"	optional
OfferingAdded		OfferingChanged (see clause 8.2.6) code value shall be "OfferingAdded"	optional
OfferingDeleted		OfferingChanged (see clause 8.2.6) code value shall be "OfferingDeleted"	optional
SensorInserted		SensorChanged (see clause 8.2.7) code value shall be "SensorInserted"	optional
SensorDescriptionUpdated		SensorDescriptionUpdated (see clause 8.2.8) code value shall be "SensorDescriptionUpdated"	optional

a Although some values listed in the column appear to contain spaces, they shall not contain spaces.

NOTE The insertion of a sensor causes the addition of an offering and thus also of a change to the Capabilities. As such, depending upon which of the events defined in the table a service actually recognizes and publishes, up to three notifications may be triggered when a sensor was inserted. The number of triggered notifications can get even higher if a sensor insertion causes events defined by other standards.

17.2.3 Content based filtering of notifications

Filters are used to identify the notifications from the set of all notifications generated by a producer (see OGC 09-032 clause 6.6.1.3) that are of interest to a consumer (see OGC 09-032 clause 6.6.1.1). Only those notifications will be sent to the consumer.

The complexity of the filter criteria depends on the filter language, also called *dialect*. Content based filter dialects like XPath only work on XML encodings of notifications. Object based filter dialects like the Filter Encoding Specification (FES) use the object model underlying the notification content, potentially performing conversion of reference systems on the fly.

A filter is usually applied on the actual event or message, not on a possible container type (that may be added by the technology realizing the publish/subscribe pattern in a specific binding before the message is actually sent out to a consumer). The context for filter execution should be defined for a given dialect.

This standard recognizes XPath 1.0 and FES 1.1 as potential content based filter dialects (also see clause 8.2.2).

17.2.3.1 Content based filtering using XPath 1.0

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Communication/PubSub/XPathFiltering	
REQ 64.	When using XPath 1.0, the value of a filter statement in a subscription shall be an XPath 1.0 predicate expression (*PredicateExpr*). Its context is: • Context Node: the root element of the event/message XML – NOT of any container element that wraps the actual event/message • Context Position: 1 & Context Size: 1 – because the filter applies to exactly one event/message that has exactly one root element • Variable Bindings: None. • Function Libraries: XPath 1.0 Core Function Library. • Namespace Declarations: The (in-scope namespaces) property (see W3C XML Information Set) of the XML element that contains the XPath filter statement.

NOTE A service that accepts a subscription with XPath 1.0 filter dialect should store the in-scope namespaces and their prefix mappings from the subscription request. The service should use those mappings when evaluating the XPath expression for a given event/message. This avoids conflicts when the prefix mapping used in the subscription request does not match with the prefix mapping in the event/message.

Consider the following example of a SWESEvent:

Listing 7 – SWESEvent that signifies a capabilities change happened

```
<swes:SWESEvent xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
<swes:identifier>http://www.example.org/events/a87s9c76s</swes:identifi
er>
  <swes:eventTime>2010-08-03T12:14:43Z</swes:eventTime>
  <swes:code>CapabilitiesChanged</swes:code>
  <swes:service>
    <wsa:EndpointReference>
      <wsa:Address>http://my.swe-service.com/path</wsa:Address>
    </wsa:EndpointReference>
  </swes:service>
</swes:SWESEvent>
```

To find out whether the code in a SWESEvent matches a specific value, the following XPath expression can be used:

```
swes:code='CapabilitiesChanged'
```

This expression would be evaluated to true.

17.2.3.2 Content based filtering using FES 1.1

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Communication/PubSub/FESFiltering	
REQ 65.	When using FES 1.1, the value of a filter statement in a subscription shall be an FES 1.1 filter, the context of which is: • Context Node: the root element of the event/message XML – NOT of any container element that wraps the actual event/message • Filter Capabilities (operators, operands, functions): as indicated by the service in its Capabilities. • Namespace Declarations (may be used in path expressions of property names): The (in-scope namespaces) property (see W3C XML Information Set) of the XML element that contains the XPath expression in an FES filter statement.

17.2.4 Channel based filtering/SWES notification topics

When using channel based filtering, it is imperative to define which channels can be used and which events are published on each channel. The definitions of events recognized by this standard are listed in Table 35. Each event is given by its name and definition. These events may be detected and subsequently published by a service.

The OASIS WS-Topics standard defines the *TopicNamespace* type as a means to group and describe channels/topics that belong to a specific (target) namespace. The topic namespace of this standard is defined through Listing 8 and Table 36.

Listing 8 – SWES Topic Namespace

```
<wstop:TopicNamespace xmlns:wstop="http://docs.oasis-open.org/wsn/t-1"
xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="SWE-
Service-Model-Topic-Namespace"
targetNamespace="http://www.opengis.net/swes/2.0" final="true">
  <wstop:Topic name="CapabilitiesChange" messageTypes="swes:SWESEvent">
    <wstop:Topic name="OfferingAddition"
messageTypes="swes:OfferingChanged"/>
    <wstop:Topic name="OfferingDeletion"
messageTypes="swes:OfferingChanged"/>
  </wstop:Topic>
  <wstop:Topic name="SensorInsertion"
messageTypes="swes:SensorChanged"/>
  <wstop:Topic name="SensorDescriptionUpdate"
messageTypes="swes:SensorDescriptionUpdated"/>
</wstop:TopicNamespace>
```

The following table defines which events are published on which topics. The events and their encoding are defined in Table 35.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Communication/PubSub/ChannelBased	
REQ 66.	A SWE service that supports channel-based filtering/notification shall publish events on topics according to Table 36.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Communication/PubSub/ChannelBased/TopicSet	
REQ 67.	If a service supports channel based subscriptions (see clause 17.2.4), it shall state the supported topics and corresponding event types in the topic set contained in its notification metadata (see clause 8.2.1).

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Communication/PubSub/ChannelBased/EventsOnTopics	
REQ 68.	Such a service shall publish only those SWES events that belong to topics listed in the topic set of the service (see Table 10). Events from a different topic namespace may be published by the service.

Note: This is to ensure that a SWE service publishes SWES events according to its advertised topic set.

Table 36 — Topics and the events posted on them

Topic name	Parent topic name	Name of event(s) posted on topic	Use at SWE service that realizes channel based filtering/notification topics
CapabilitiesChange	-	CapabilitiesChanged	optional
OfferingAddition	Capabilities Change	OfferingAdded	optional
OfferingDeletion	Capabilities Change	OfferingDeleted	optional
SensorInsertion	-	SensorInserted	optional
SensorDescriptionUpdate	-	SensorDescriptionUpdated	optional

Apparently, one event can trigger notifications in multiple topics. It depends on the topic and event design whether the sets of events published on two topics are disjoint or not.

18 Using SWES Extensions Points

The SWE Service Model is intended to be used as a baseline for SWE services. There are several ways that another standard can extend the functionality defined in this standard. These will be described in the following sections.

18.1 Extending service model elements

Most parts of the service model defined in this standard cannot be used standalone because of abstract data types in the models. For example, the *InsertSensor* operation defined in chapter 13 has an abstract element in its request model, named *InsertionMetadata*. Services that intend to leverage *InsertSensor* functionality have to derive a non-abstract type to implement *InsertionMetadata*. This can be done as shown in the following figure.

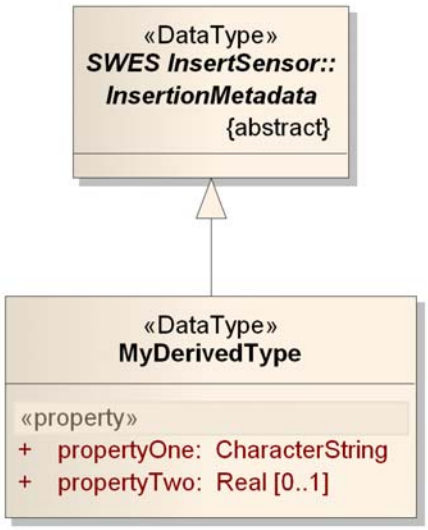

Figure 26 — extending a model type through derivation (informative)

The type derivation and subsequent encoding should follow the rules defined in chapter 20.

All non-final types defined in the SWE Service Model can be extended through derivation. However, extensions that are applicable in more than one type, for example in operation requests, shall be created as standalone types that can leverage the extension points defined in the service model as explained below.

18.2 Overloading operations

As described in the previous section, the functionality of the service model can be extended. When it comes to extending operation requests and responses, the preferred way of doing so is to overload the operation by adding parameters defined in an extension. The core request and response types remain unchanged. Furthermore, the

extension parameters can be reused easily and are not tied to a specific namespace and version of the core model.

The SWE Service Model defines a request and response type for service operations that provide the necessary extension properties. These types are specified in clause 9. This standard requires a service to throw an exception if it encounters an unsupported request extension (see clause 15). This can be done as shown in the following listings.

Listing 9 – Example of operation request with extensions

```
<swes:DescribeSensor service="SPS" version="2.0"
xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:abc="http://www.example.org/abc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <swes:extension>
    <abc:MyExtensionElement>value</abc:MyExtensionElement>
  </swes:extension>
  <swes:procedure>http://my.org/sensors/937239</swes:procedure>
<swes:procedureDescriptionFormat>http://www.opengis.net/sensorML/1.0.1<
/swes:procedureDescriptionFormat>
</swes:DescribeSensor>
```

Listing 10 – Example of RequestExtensionNotSupported exception

```
<ows:Exception exceptionCode="RequestExtensionNotSupported"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:abc="http://www.example.org/abc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ows:ExceptionText>
  <![CDATA[
    <abc:MyExtensionElement>value</abc:MyExtensionElement>
  ]]>
  </ows:ExceptionText>
</ows:Exception>
```

Extensions themselves may provide points for further extension.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Extensions/RequestExtensionNotSupported	
REQ 69.	If a service encounters a sub-extension in an extension property of an operation request and it does not support it, then it shall throw a *RequestExtensionNotSupported* exception.

18.3 Adding notification topics

Services implementing this standard may define their own topic namespace to define notification semantics. They may also extend topics defined in the topic namespace of this standard according to the rules defined by the OASIS WS-Topics standard.

An extension topic defines its own notification semantics and does not change the semantics of its parent topic (which in this case is defined by this standard). Each topic defines the sending trigger and content of an event. In addition, it usually defines the information encoding of the notification.

Extension topics define the way topics are included in a given *TopicSet* of a service instance (they have to be a child of the parent topic – see WS-Topics). See Listing 11 for an example and WS-Topics chapter 3 for further details.

For example, this specification defines a *CapabilitiesChange* topic. A notification is sent whenever the Capabilities document of the referenced service has changed. Another standard can define an extension of this topic named *ServiceProviderChange.* Here, a notification will be sent whenever a change to the *ServiceProvider* section of the Capabilities document occurs. Notifications sent on this new topic could provide specific information which parts of the *ServiceProvider* section has changed. Clients interested only in changes to the *ServiceProvider* section then subscribe to this new topic. Note that changes to the *ServiceProvider* section also trigger a *CapabilitiesChanged* event and hence result in a notification on the parent topic (*CapabilitiesChange*). Consumers subscribed to that topic and all child topics would then receive two notifications even though originally only one event has happened.

18.4 Object type extensions

Each type defined in this specification has either no stereotype or has the stereotype <<Type>>. Hence it has automatically a well-defined extension point – see clause 24.2.4.1. This is the place where other specifications may add additional properties. For example, the *NotificationProducerMetadata* type has such an extension point. A possible extension could for example add information about potential publication of events to web feeds. This aspect, which is a candidate for future versions of this standard, is not covered herein.

There is a key difference between object type extensions and request/response extensions as described in clause 18.2. If a service encounters an unsupported request/response extension, it throws an exception. In contrast:

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/Extensions/UnsupportedObjectTypeExtension	
REQ 70.	If a service encounters an unsupported object type extension in a place other than a request extension, it shall ignore it.

19 SOAP binding

19.1 Introduction

This section defines the realization of functionality defined in this standard for a service using SOAP. This standard does not prescribe usage of either SOAP 1.1 or SOAP 1.2. It also does not prescribe WSDL 1.1 or WSDL 2.0.

This standard does not define any specific policy statements to be included in a WSDL document or in service requests and responses for defining specific established, available or desired behavior. If the need for such policies arises in the future, necessary policy statements can be included in the standard or in extensions of it.

19.2 Exceptions

19.2.1 Introduction

The operations defined in this standard use exception codes defined by OWS Common [OGC 06-121r3] chapter 8. The encoding of those exceptions follows the definitions given in the following paragraphs.

The encoding of a SOAP 1.1 fault is slightly different compared to a SOAP 1.2 fault. Each of the exceptions recognized by this standard (see section 19.2.4) is therefore defined by specifying values for the following abstract fault properties which are mapped to the properties of SOAP 1.1/1.2 faults (see sections 19.2.2 and 19.2.3):

- [Code] The fault code (of type *QName*)
- [Subcode] The fault subcode.
- [Reason] Human readable text explaining the exception. It is recommended to use this text but service implementations may use other text (e.g. in other languages)
- [Details] The detail elements to be added to the SOAP fault.

NOTE Sub subcodes (which can be defined in SOAP 1.2 faults) are not relevant for the exceptions defined in this standard.

19.2.2 SOAP 1.1 Fault Binding

The SOAP 1.1 fault is slightly less expressive than the SOAP 1.2 fault. It maps only [Code] together with [Reason] and [Details].

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/FaultBinding1.1	
REQ 71.	The abstract fault properties as defined in section 19.2.1 bind to a SOAP 1.1 fault as follows: • The value of the [Code] property is bound as the value of the SOAP faults soap11:Fault/faultcode[1] element. • The value of the [Reason] property is bound as the value of the SOAP faults soap11:Fault/faultstring element[2]. • The value of the [Details] property is bound as child elements of the SOAP faults soap11:Fault/detail element information item.

[1] Note that the elementFormDefault in the SOAP 1.1 schema is not set to qualified and therefore the faultcode child element of a soap11:Fault element is unqualified – also see WS-I Basic Profile 1.1 section 3.3.3

[2] The faultstring in a SOAP 1.1 fault does not provide the functionality to define the language of the contained textual description.

19.2.3 SOAP 1.2 Fault Binding

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/FaultBinding1.2	
REQ 72.	The abstract fault properties as defined in section 19.2.1 bind to a SOAP 1.2 fault as follows: • The value of the [Code] property is bound as the value of the SOAP faults soap12:Fault/soap12:Code/soap12:Value element information item. • The value of the [Subcode] property is bound as the value of the (optional) SOAP faults soap12:Fault/soap12:Code/soap12:Subcode/soap12:Value element information item. • The value of the [Reason] property is bound as the value of the SOAP faults soap12:Fault/soap12:Reason/soap12:Text element information item (with appropriate language indicator depending upon the chosen language). • The value of the [Details] property is bound as child elements of the SOAP faults soap12:Fault/soap12:Detail element information item.

19.2.4 Exceptions recognized by SWES as SOAP faults

NOTE Each operation defined in this standard may have additional requirements with respect to the implementation of the ows:Exception element to be used in the [Details] property of faults generated while performing that operation. These requirements are stated in the according clauses of each operation.

19.2.4.1 OperationNotSupported exception

The meaning of this exception (code) is defined in table 25 of [OGC 06-121r3].

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/OperationNotSupportedException	
REQ 73.	The abstract fault properties for this exception shall be as follows: • [Code] The *QName* soap11:Client (SOAP 1.1) or soap12:Sender (SOAP 1.2) • [Subcode] The *QName ows:OperationNotSupported* • [Reason] the string: "The requested operation is not supported by this server." • [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3]

19.2.4.2 MissingParameterValue exception

The meaning of this exception (code) is defined in table 25 of [OGC 06-121r3].

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/MissingParameterValueException	
REQ 74.	The abstract fault properties for this exception shall be as follows: • [Code] The *QName* soap11:Client (SOAP 1.1) or soap12:Sender (SOAP 1.2) • [Subcode] The *QName ows:MissingParameterValue* • [Reason] the string: "The request did not include a value for a required parameter and this server does not declare a default value for it." • [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3]

19.2.4.3 InvalidParameterValue exception

The meaning of this exception (code) is defined in table 25 of [OGC 06-121r3].

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/InvalidParameterValueException	
REQ 75.	The abstract fault properties for this exception shall be as follows: • [Code] The *QName* soap11:Client (SOAP 1.1) or soap12:Sender (SOAP 1.2) • [Subcode] The *QName ows:InvalidParameterValue* • [Reason] the string: "The request contained an invalid parameter value." • [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3]

19.2.4.4 VersionNegotiationFailed exception

The meaning of this exception (code) is defined in table 25 of [OGC 06-121r3].

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/VersionNegotiationFailedException	
REQ 76.	The abstract fault properties for this exception shall be as follows: • [Code] The *QName* soap11:Client (SOAP 1.1) or soap12:Sender (SOAP 1.2) • [Subcode] The *QName ows:VersionNegotiationFailed* • [Reason] the string: "The list of versions in the 'AcceptVersions' parameter value of the GetCapabilities operation request did not include any version supported by this server." • [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3]

19.2.4.5 InvalidUpdateSequence exception

The meaning of this exception (code) is defined in table 25 of [OGC 06-121r3].

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/InvalidUpdateSequenceException	
REQ 77.	The abstract fault properties for this exception shall be as follows: • [Code] The *QName* soap11:Client (SOAP 1.1) or soap12:Sender (SOAP 1.2) • [Subcode] The *QName ows:InvalidUpdateSequence* • [Reason] the string: "The value of the updateSequence parameter in the GetCapabilities operation request was greater than the current value of the service metadata updateSequence number." • [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3]

19.2.4.6 OptionNotSupported exception

The meaning of this exception (code) is defined in table 25 of [OGC 06-121r3].

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/OptionNotSupportedException	
REQ 78.	The abstract fault properties for this exception shall be as follows: • [Code] The *QName* soap11:Client (SOAP 1.1) or soap12:Sender (SOAP 1.2) • [Subcode] The *QName ows:OptionNotSupported* • [Reason] the string: "The request included/targeted an option that is not supported by this server." • [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3]

19.2.4.7 NoApplicableCode exception

The meaning of this exception (code) is defined in table 25 of [OGC 06-121r3].

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/NoApplicableCodeException	
REQ 79.	The abstract fault properties for this exception shall be as follows: • [Code] The *QName* soap11:Server (SOAP 1.1) or soap12:Receiver (SOAP 1.2) • [Subcode] The *QName ows:NoApplicableCode* • [Reason] the string: "A server exception was encountered." • [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3]

19.2.4.8 InvalidRequest exception

The meaning of this exception (code) is defined in clause 15 of this standard.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/InvalidRequestException	
REQ 80.	The abstract fault properties for this exception shall be as follows: • [Code] The *QName* soap11:Client (SOAP 1.1) or soap12:Sender (SOAP 1.2) • [Subcode] The *QName swes:InvalidRequest* • [Reason] the string: "The request did not conform to its XML Schema definition." • [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3]

19.2.4.9 RequestExtensionNotSupported exception

The meaning of this exception (code) is defined in clause 15 of this standard.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Fault/RequestExtensionNotSupportedException	
REQ 81.	The abstract fault properties for this exception shall be as follows: • [Code] The *QName* soap11:Client (SOAP 1.1) or soap12:Sender (SOAP 1.2) • [Subcode] The *QName swes:RequestExtensionNotSupported* • [Reason] the string: "The request included an extension that is not supported by this server." • [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3] which should contain the unsupported extension as defined in clause 15 of this standard.

19.3 Action URIs

For the SOAP binding, a standard needs to define action URIs for the following features:

- SOAPAction HTTP header field of a SOAP 1.1 request
- Action parameter in a SOAP 1.2 request (SOAP 1.2 feature: "http://www.w3.org/2003/05/soap/features/action/")
- WS-Addressing [action] message addressing property

NOTE If and how a service instance makes use of one or more of these features depends upon the chosen SOAP and WSDL version as well as on the requirements of the service instance.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/ActionURI	
REQ 82.	Table 37 and Table 38 list the action URIs that shall be used for the various message facets (requests and responses of operations) defined by SWES and also for various exception/fault message types.

Table 37 — Action URIs for SWES message facets

Message Facet [a]	Action URI [a]	applicable in feature (Y=yes, N=no)		
		SOAP 1.1 SOAPAction	SOAP 1.2 action	WS-Addressing [action]
DescribeSensor request	http://www.opengis.net/swes/2.0/DescribeSensor	Y	Y	Y
DescribeSensor response	http://www.opengis.net/swes/2.0/DescribeSensorResponse	N	N	Y
UpdateSensorDescription request	http://www.opengis.net/swes/2.0/UpdateSensorDescription	Y	Y	Y
UpdateSensorDescription response	http://www.opengis.net/swes/2.0/UpdateSensorDescriptionResponse	N	N	Y
InsertSensor request	http://www.opengis.net/swes/2.0/InsertSensor	Y	Y	Y
InsertSensor response	http://www.opengis.net/swes/2.0/InsertSensorResponse	N	N	Y
DeleteSensor request	http://www.opengis.net/swes/2.0/DeleteSensor	Y	Y	Y
DeleteSensor response	http://www.opengis.net/swes/2.0/DeleteSensorResponse	N	N	Y

a Although some values listed in the column appear to contain spaces, they shall not contain spaces.

NOTE The action URIs for the messages defined by WS-Notification are not listed here – they can be found in the according paragraphs of WS-Notification.

Table 38 — Action URIs for various exceptions/fault types

Exception/Fault type	WS-Addressing [action] message addressing property value
Exception defined by SWES	http://www.opengis.net/swes/2.0/Exception
Exception defined by OWS Common [OGC 06-121r3]	http://www.opengis.net/ows/1.1/Exception
SOAP defined faults (including version mismatch, must understand, and data encoding unknown – see WS-Addressing SOAP binding clause 6[3])	http://www.w3.org/2005/08/addressing/soap/fault
WS-Addressing specific fault (see WS-Addressing SOAP binding clause 6)	http://www.w3.org/2005/08/addressing/fault
WS-Notification specific fault (see WS-BaseNotification and WS-BrokeredNotification clause 1.4)	http://docs.oasis-open.org/wsn/fault

Issue Name: General purpose operations in OWS (JE, Nov. 16th 09)

Issue Description: There appears to be a slight mismatch between operation/interface handling in web services defined by OGC and those that are referred to as WS-* web services. The issue is that OWS standards are designed with the intention of creating one complete set of operations for a web service type, rather than for defining functionality that can easily be integrated in one web service instance (for example defined using WSDL).

For example, WS-Notification defines interfaces that can be added to the set of interfaces implemented by an existing service. With the current approach in OWS, e.g. to derive a service type specific GetCapabilities operation, this idea of interface/functionality combination is not easily doable.

With OWS, one would have multiple GetCapabilities operations in one WSDL or would have multiple WSDLs, one for each implemented service type. While the latter is doable, this approach obviously requires duplication of functionality. A better approach would be to design a general purpose GetCapabilities operation where the service specific elements can be plugged in (e.g. via abstract elements, so that for example new sections can be added in these extension points).

[3] also see http://www.ws-i.org/Profiles/BasicProfile-2_0%28WGD%29.html#SOAP_Defined_Faults_Action_URI and http://www.ws-i.org/Profiles/BasicProfile-1_2%28WGAD%29.html#SOAP_Defined_Faults_Action_URI

Operations like those defined in this standard include mandatory service and version parameters in support of current OGC practice. These parameters would no longer be needed (at least in SOAP bindings) if we followed a web service design that supports general purpose interfaces rather than service type specific operation sets.

OWS Common would need careful revision to support interface composition which could subsequently be adopted by each OGC standard that specifies service functionality. Such a revision should take compatibility with WSDL 1.1 and WSDL 2.0 mechanisms into account.

Resolution:

Issue Name: Referencing WSDL from Capabilities (JE, Nov. 17th 09)

Issue Description: In version 1.1 of OWS Common (OGC 06-131r3), no element is defined which would allow to reference the WSDL description of a service from the Capabilities document of that service. Clause 6.2.2 in OGC 08-009r1 mentions such a mechanism and OGC 06-121r8 describes it in some more detail in (the informative) Annex E clause E.2 ("Relationship to OGC service descriptions").

However, the XML Schema from OGC 06-121r8 for the CapabilitiesBaseType and otherwise do not contain such an element, though it would be quite helpful.

The question is whether such functionality would be needed in a SOAP binding or if the WSDL document of a service itself should be the entry point for service discovery. If it was required, then the functionality to reference WSDL from a service's Capabilities needs to be defined either in OWS Common or in this specification. This could be done simply by creating another – optional – section that can be queried by clients and would be listed by a service as supported section.

Resolution:

19.4 Realization of Asynchronous Request / Response

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/Asynch/WS-Adressing	
REQ 83.	A service leveraging this standard shall use WS-Addressing to enable asynchronous request-response in its SOAP binding.

19.5 Realization of Publish/Subscribe

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/PubSub/WS-Notification	
REQ 84.	A service leveraging this standard shall use WS-Notification to enable Publish/Subscribe functionality (see clause 17.2) in its SOAP binding.

> *"The purpose of the Web Services Notification (WSN) TC is to define a set of specifications that standardize the way Web services interact using 'Notifications' or 'Events'. They form the foundation for Event Driven Architectures built using Web services.*
>
> *These specifications provide a standardized way for a Web service, or other entity, to disseminate information to a set of other Web services, without having to have prior knowledge of these other Web Services. They can be thought of as defining 'Publish/Subscribe for Web services'."* (OASIS - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn)

OASIS WS-BaseNotification defines the interfaces for performing basic publish/subscribe. Together with WS-Topics and WS-BrokeredNotification, a broad set of functionality is available to support more sophisticated use cases. A tutorial on WS-Notification is provided in OGC 09-032 Annex A[4]. A detailed introduction to WS-Notification is therefore not provided in this document.

[4] Be aware that OGC 09-032 Annex A includes examples of topic namespaces and type definitions that use the same target namespace as the one used in this standard. These examples are informative only – the normative definitions are given in this standard.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/PubSub/WS-Topics	
REQ 85.	A service that enables Publish/Subscribe functionality in its SOAP binding shall use WS-Topics to indicate to clients which topics (see Table 36) it supports.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/PubSub/WS-Topics/NotificationMetadata	
REQ 86.	A service that enables Publish/Subscribe functionality in its SOAP binding shall populate its notification metadata with the according topic set information (see clause 8).

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/SOAP/PubSub/WS-Topics/Dialects	
REQ 87.	A service that enables Publish/Subscribe functionality in its SOAP binding shall support at least one of the topic expression dialects listed in Table 12 and list it in its filter dialect metadata (see section 8.2.2).

The information required to enable channel based subscription models in WS-Notification (where *topic* is used as synonym for *channel*) is defined in clause 17.2.4. The topic namespace defined there is modeled according to the WS-Topics standard. A SWE service may include one or more of the topics defined in the topic namespace in its topic set and client subscriptions may target these topics in subscribe requests sent to the service. The dialects to encode topic expressions for identifying topics supported in a service's topic set are defined in WS-Topics and Table 12 of this standard. WS-Notification allows a combination of topic- and content-based filters through multiple filter expressions in one *subscribe* request. Various expression dialects can be used, a list of known dialects and their identifiers is given in Table 12. The dialects supported by a service shall be listed in the *FilterDialectMetadata* that is part of the *NotificationProducerMetadata* type (see clause 8.2.1), a section of the Capabilities document that shall be supported by each service implementing publish/subscribe functionality in its SOAP binding as defined in this standard.

The remainder of this section contains examples to describe certain aspects of doing publish/subscribe for SWE services using WS-Notification.

Assume that a service provides publish/subscribe functionality with the topic set shown in Listing 11.

Listing 11 – example topic set

```
<wstop:TopicSet xmlns:wstop="http://docs.oasis-open.org/wsn/t-1"
xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <swes:CapabilitiesChange>
    <swes:OfferingAddition wstop:topic="true"/>
    <swes:OfferingDeletion wstop:topic="true"/>
  </swes:CapabilitiesChange>
  <swes:SensorInsertion wstop:topic="true"/>
  <swes:SensorDescriptionUpdate wstop:topic="true"/>
</wstop:TopicSet>
```

A simple subscribe request to listen on the SensorInsertion topic could look like shown in Listing 12.

Listing 12 – example Subscribe request using XPath topic expression (SOAP envelope omitted)

```
<wsn-b:Subscribe xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:wsn-b="http://docs.oasis-open.org/wsn/b-2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsn-b:ConsumerReference>
    <wsa:Address>http://www.example.org/consumer</wsa:Address>
  </wsn-b:ConsumerReference>
  <wsn-b:Filter>
    <wsn-b:TopicExpression Dialect="http://www.w3.org/TR/1999/REC-
xpath-19991116">//swes:SensorInsertion</wsn-b:TopicExpression>
  </wsn-b:Filter>
</wsn-b:Subscribe>
```

To avoid possible issues with changes of namespace prefix mappings (prefix being changed globally but not in attribute values or element content) while the subscribe request is delivered to the service, the following XPath expression can be used:

```
//*[namespace-uri()='http://www.opengis.net/swes/2.0' and local-
name()='SensorInsertion']
```

A *subscribe* request with simple topic expression as defined by WS-Topics that also subscribes to the *SensorInsertion* topic would look like shown in Listing 13.

Listing 13 - example Subscribe request using simple topic expression (SOAP envelope omitted)

```
<wsn-b:Subscribe xmlns:tns="http://www.opengis.net/swes/2.0"
xmlns:wsn-b="http://docs.oasis-open.org/wsn/b-2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsn-b:ConsumerReference>
    <wsa:Address>http://www.example.org/consumer</wsa:Address>
  </wsn-b:ConsumerReference>
  <wsn-b:Filter>
    <wsn-b:TopicExpression Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Simple">tns:SensorInsertion</wsn-b:TopicExpression>
  </wsn-b:Filter>
</wsn-b:Subscribe>
```

A client may also subscribe multiple topics, this can for example be done using more than one topic expression in the subscribe request, as shown in Listing 14.

Listing 14 - example Subscribe request using two XPath topic expressions (SOAP envelope omitted)

```
<wsn-b:Subscribe xmlns:tns="http://www.opengis.net/swes/2.0" xmlns:wsn-
b="http://docs.oasis-open.org/wsn/b-2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <wsn-b:ConsumerReference>
    <wsa:Address>http://www.example.org/consumer</wsa:Address>
  </wsn-b:ConsumerReference>
  <wsn-b:Filter>
    <wsn-b:TopicExpression Dialect="http://www.w3.org/TR/1999/REC-
xpath-19991116">//tns:SensorInsertion</wsn-b:TopicExpression>
    <wsn-b:TopicExpression Dialect="http://www.w3.org/TR/1999/REC-
xpath-19991116">//tns:OfferingAddition</wsn-b:TopicExpression>
  </wsn-b:Filter>
</wsn-b:Subscribe>
```

This is one example where the consumer can receive two notifications once a sensor was inserted, one on the *SensorInsertion* and one on the *OfferingAddition* topic. To avoid too much unnecessary traffic, a service should send the two event objects in one notification message (see Listing 15) to the consumer, because they originate from the same event and are thus related to each other.

Listing 15 – notify message with two notifications (SOAP envelope omitted)

```
<wsnt:Notify xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <wsnt:NotificationMessage>
    <wsnt:Message>
      <swes:SensorChanged>
        <swes:eventTime>2010-08-04T07:24:00Z</swes:eventTime>
        <swes:code>SensorInserted</swes:code>
        <swes:service>
          <wsa:EndpointReference>
            <wsa:Address>http://my.swe-service.com/path</wsa:Address>
          </wsa:EndpointReference>
        </swes:service>
        <swes:procedure>http://my.nspce.com/sid018aed6</swes:procedure>
      </swes:SensorChanged>
    </wsnt:Message>
  </wsnt:NotificationMessage>
  <wsnt:NotificationMessage>
    <wsnt:Message>
      <swes:OfferingChanged>
        <swes:eventTime>2010-08-04T07:24:00Z</swes:eventTime>
        <swes:code>OfferingAdded</swes:code>
        <swes:service>
          <wsa:EndpointReference>
            <wsa:Address>http://my.swe-service.com/path</wsa:Address>
          </wsa:EndpointReference>
        </swes:service>
        <swes:offering>http://my.nspce.com/off0187asd76</swes:offering>
      </swes:OfferingChanged>
    </wsnt:Message>
  </wsnt:NotificationMessage>
</wsnt:Notify>
```

The event encodings defined in this standard only provide a small part of the information that belongs to the event. Consumers interested in more information will have to retrieve more information directly from the service.

20 Annex A - Abstract test suite (normative)

Specific conformance tests for the SWE Service Model need to be defined on the concrete service level in order to ensure full interoperability. Thus, the abstract test suite defined herein only ensures general interoperability between client and server.

An implementation of a service based on the SWE Service Model shall satisfy the following system characteristics to be minimally conformant with this specification:

20.1 Common Test Elements

20.1.1 Exception Reporting

20.1.1.1 Exception validity

http://www.opengis.net/spec/SWES/2.0/conf/Core/ExceptionReporting/Validity

a) Test Purpose: To verify that the exceptions the server generates validate according to the schema defined in Clause 8 of 06-121r3.

b) Test Method: Devise and execute a request that generates an error. Verify that the exception that the server generates is valid.

c) Reference: 15

d) Test Type: Basic

20.1.1.2 Exception appropriateness

http://www.opengis.net/spec/SWES/2.0/conf/Core/ExceptionReporting/Appropriateness

a) Test Purpose: Verify that the server generates an appropriate exception by setting the value of the code and locator parameters to an appropriate value.

b) Test Method: Devise a series of requests that generate an error for each error code applicable for the according operation (for error codes applicable to SWES operations, see Figure 20). Ensure that server generates an appropriate exception for each case by verifying that the code and locator parameters have been set to the expected value.

c) Reference: 15

d) Test Type: Basic

20.1.2 Common Request/Response Parameters

20.1.2.1 Service and version appropriateness

http://www.opengis.net/spec/SWES/2.0/conf/Core/RequestResponse/ServiceAndVersion

a) Test Purpose: To verify that the server recognizes incorrect values for service and version parameters in request.

b) Test Method: Devise and execute a request with incorrect value for the service (type) request parameter and another request with incorrect value for the version parameter. Verify that the exception that the server generates is valid.

c) Reference: 9.2.1

d) Test Type: Basic

20.1.2.2 Extension Handling

http://www.opengis.net/spec/SWES/2.0/conf/Core/RequestResponse/ExtensionHandling

a) Test Purpose: To verify that the service handles request and object extensions correctly.

b) Test Method: Devise and execute a request that contains a request extension property which is known to not be supported by the service. Verify that the service throws a valid RequestExtensionNotSupported exception.

   If the service supports a request extension and if that extension has further extension points, devise and execute a valid request that contains a value for this sub-extension that is unknown to the service. Verify that the service throws a valid RequestExtensionNotSupported exception.

   Verify further that the service honors the semantics of supported request extensions by executing valid requests with those extensions and ensuring that they do not result in a RequestExtensionNotSupported exception.

   Devise and execute a valid request whose model allows the inclusion of an object type extension in a place other than the extension property that may be derived from the *ExtensibleRequest* type. Set that object extension to an unknown value. Ensure that the service ignores the unknown extension value by verifying that the response to the request is the same that was expected when the request was performed without that extension.

c) Reference: 9.1, 15, 18.2, 18.4

d) Test Type: Basic

20.1.3 Common Codes

http://www.opengis.net/spec/SWES/2.0/conf/Core/CommonCodes

a) Test Purpose: To verify that the values of properties that are of one of the code list types defined in the Common Codes package (see clause 10) are URIs.

b) Test Method: Devise and execute a valid request whose according response is known to contain properties that are of the code list types defined in the Common Codes package. Ensure that the values of these properties are valid URIs.

c) Reference: 10.1

d) Test Type: Basic

20.1.4 Feature Provisioning

http://www.opengis.net/spec/SWES/2.0/conf/Core/FeatureProvisioning

a) Test Purpose: To verify that a related feature is either given inline or through a resolvable reference and that the given object is a valid GML 3.2 feature.

b) Test Method: If a related feature is given in a request or response, resolve its target if it is given by reference and verify that the feature object is a valid substitution for the gml:AbstractFeature as defined in OGC 07-036 clause 9.3.

c) Reference: 7.2.1, 7.2.2, 9.2.4, 13.2.1 and OGC 07-036 clause 9.3

d) Test Type: Basic

20.1.5 Sensor Description

http://www.opengis.net/spec/SWES/2.0/conf/Core/SensorDescriptionStructure

a) Test Purpose: To verify that a sensor description is well structured.

b) Test Method: If a sensor description is given in a request or response, verify that the validTime property – if set – is a TM_Instant as defined in ISO 19108 with time in the past or a TM_Period as defined in ISO 19108 with start time in the past.

c) Reference: 9.2.3

d) Test Type: Basic

20.1.6 Property Inheritance

http://www.opengis.net/spec/SWES/2.0/conf/Core/PropertyInheritanceMechanics

a) Test Purpose: To verify that the values of a given property for which the *property inheritance mechanism* applies (that property being an inheritor as defined by that mechanism) according to the inheritance category that is defined for that property.

b) Test Method: For each property that is an inheritor and defined in the conceptual model of one of the operations supported by the service, devise and execute a valid request that either contains that property or that results in a response which

includes that property. Verify that the response is as expected when the value inheritance according to the category defined for that property was applied by the service correctly.

c) Reference: 22

d) Test Type: Basic

20.1.7 Service Metadata

20.1.7.1 Property Inheritance

20.1.7.1.1 Number of property values for (abstract) offering

http://www.opengis.net/spec/SWES/2.0/conf/BasicSWEServiceMetadata/CorrectOfferingPropertyCardinalityWithInheritance

a) Test Purpose: To verify that the server has the correct number of values for the properties contained in the AbstractOffering in each of its offerings listed in its contents section.

b) Test Method: Devise and execute a GetCapabilities request that requests the contents section. Verify that the number of values for the procedure, procedure description format, observable property and related feature properties in each offering after applying the property inheritance mechanism (clause 22) are as defined in Table 9.

c) Reference: 7.2.2, 22, Table 9

d) Test Type: Basic

20.1.7.1.2 Adherence to property inheritance mechanism

http://www.opengis.net/spec/SWES/2.0/conf/BasicSWEServiceMetadata/AdherenceToPropertyInheritanceMechanism

a) Test Purpose: To verify that the service adheres to the rules of property inheritance.

b) Test Method: Devise and execute requests that test each of the values for the procedure, observable property and procedure description format properties that an offering has when applying the property inheritance mechanism as defined in clause 22. Verify that the cardinality of the property is as defined in Table 9.

c) Reference: 7.2.2, 22, Table 9

d) Test Type: Basic

20.1.7.2 Indication of support for validTime option in DescribeSensor

http://www.opengis.net/spec/SWES/2.0/conf/SensorHistoryProvider/ValidTimeOption

a) Test Purpose: To verify that the service indicates support for the validTime option of the DescribeSensor operation in its service metadata.

b) Test Method: Devise and execute DescribeSensor requests that test the validTime option defined in this standard (clause 11.1). Verify that the service handles the request as described in the according clauses of the operation if it supports the validTime option for that operation or throws a valid exception with code OptionNotSupported otherwise.

c) Reference: 11, 15

d) Test Type: Basic

20.1.7.3 Indication of support for validTime option in UpdateSensorDescription

http://www.opengis.net/spec/SWES/2.0/conf/SensorHistoryManager/ValidTimeOption

a) Test Purpose: To verify that the service indicates support for the validTime option of the UpdateSensorDescription operation in its service metadata.

b) Test Method: Devise and execute UpdateSensorDescription requests that test the validTime option defined in this standard (clause 12.1). Verify that the service handles the request as described in the according clauses of the operation if it supports the validTime option for that operation or throws a valid exception with code OptionNotSupported otherwise.

c) Reference: 12, 15

d) Test Type: Basic

20.1.8 Sensor Deletion

http://www.opengis.net/spec/SWES/2.0/conf/SensorDeletion/OperationBehavior

a) Test Purpose: To verify that the service identifies the correct procedure in the response.

b) Test Method: Create and send a request to delete a specific sensor. Verify that the response identifies the same procedure if the response is not an exception.

c) Reference: 14.2.2

d) Test Type: Basic

NOTE: further tests on the behavior and effect of deleting a sensor cannot be defined by this standard because the according operation is abstract in this specification. A service leveraging the DeleteSensor operation has to specify and test the specific deletion behavior.

20.1.9 Sensor Description

20.1.9.1 Basic Retrieval

http://www.opengis.net/spec/SWES/2.0/conf/SensorProvider/BasicRetrieval

a) Test Purpose: To verify that the service returns the current description of a sensor as requested.

b) Test Method: Create and send a request to retrieve the current description of a specific sensor. Verify that the response contains the same value for the procedure description format as contained in the request. Verify that the response contains a valid sensor description (test 20.1.5) in the requested format.

c) Reference: 10

d) Test Type: Basic

20.1.9.2 History Retrieval

http://www.opengis.net/spec/SWES/2.0/conf/SensorHistoryProvider/HistoryRetrieval

a) Test Purpose: To verify that the service handles a request to retrieve historic sensor description correctly.

b) Test Method: Create and send a request to retrieve the description of a specific sensor, including an indication for which time instant or time period valid descriptions shall be retrieved. Verify that the response contains the same value for the procedure description format as contained in the request. Verify that the response contains only valid sensor descriptions (test 20.1.5) in the requested format.
Verify that no description is provided if the request contains a valid time for which no description of the procedure is available – this can be done by choosing a valid time in the request that is sufficiently back in the past.
Verify that the valid time of the description(s) returned in the response is not before and not after the valid time given in the request.
Verify that at most one description is returned if the valid time in the request was a time instant. If multiple descriptions are returned, verify that these descriptions do not overlap each other.

c) Reference: 10

d) Test Type: Basic

20.1.10 Sensor Insertion

http://www.opengis.net/spec/SWES/2.0/conf/SensorInsertion/OperationBehavior

a) Test Purpose: To verify that the service handles a request to insert a sensor correctly.

b) Test Method: Create and send a request to insert a sensor. Verify that:

- a new offering was created in the service's metadata, with same identifier as the one provided in the response;
- the new offering has a procedure property with the same value as the assignedProcedure provided in the response;
- the new offering has the same observable properties as provided in the request (when the property inheritance mechanism is applied);
- the new offering has the same related features as provided in the request (when the property inheritance mechanism is applied);
- the new offering has at least the procedure description format as provided in the request (when the property inheritance mechanism is applied);
- a procedure description is available (not necessarily the one provided in the request because it may already have been updated) and can be retrieved for the inserted sensor (identifier as assigned in the response)

c) Reference: 13

d) Test Type: Basic

20.1.11 Sensor Description Update

20.1.11.1 Basic Update

http://www.opengis.net/spec/SWES/2.0/conf/SensorDescriptionManager/BasicUpdate

a) Test Purpose: To verify that the service updates the current description of a sensor as requested.

b) Test Method: Create and send a request to update the current description of a specific sensor. Verify that:

- the response contains the same value for the updated procedure identifier property as the procedure identifier property contained in the request;

- the current description has been updated with the new one, by retrieving the current description and ensuring that it is the same as the one used in the update request.

c) Reference: 12

d) Test Type: Basic

20.1.11.2 Update of historic descriptions

http://www.opengis.net/spec/SWES/2.0/conf/SensorHistoryManager/UpdatingHistoricDescriptions

a) Test Purpose: To verify that the service updates the historic descriptions of a sensor as requested.

b) Test Method: Create and send a request to update the description of a specific sensor, including an indication for which time instant or time period the given description is valid. Verify that:

- the response contains the same value for the updated procedure identifier property as the procedure identifier property contained in the request;
- any existing sensor description whose valid time interacts with the valid time from the updated description are handled as defined in clause 12.1;
- the sensor description has been updated with the new one, by retrieving the description that has a valid time as the one used in the update request and ensuring that it is the same description as the one used in the update request

c) Reference: 12

d) Test Type: Basic

20.1.12 XML Encoding

20.1.12.1 XML Encoding Validity

http://www.opengis.net/spec/SWES/2.0/conf/XMLEncoding/Validity

a) Test Purpose: Verify that XML implementations of the conceptual types defined in the specification are valid according to their XML Schema implementation.

b) Test Method: For all XML instance documents received from the service that are in the namespace http://www.opengis.net/swes/2.0, verify that they are valid according to their XML Schema definition listed in Table 39.

Note: the swes.xsd can be used for validating any such XML instance against its schema definition.

c) Reference: 21

d) Test Type: Basic

20.1.12.2 XML Validation Exception Reporting

http://www.opengis.net/spec/SWES/2.0/conf/XMLEncoding/ValidationExceptionReporting

a) Test Purpose: Verify that the service sends an exception with appropriate code if it received an invalid request.

b) Test Method: For all SWES operations supported by the service, create an XML request instance that is invalid according to its schema definition outlined in Table 39 and send it to the service. Verify that the service returns an exception with code *InvalidRequest*.

c) Reference: 15, 21

d) Test Type: Basic

20.1.13 Publish/Subscribe

20.1.13.1 Event Publication

http://www.opengis.net/spec/SWES/2.0/conf/PublishSubscribe/EventPublication

a) Test Purpose: To verify that the service publishes events as advertised in its topic set.

b) Test Method: Subscribe for the topics where the event (see Table 35) of interest is published. Devise and send a request that will trigger this event. Verify that:

- the service sends a notification on each topic where it shall publish the notification on according to Table 36
- the event is encoded as defined in Table 35

c) Reference: 17.2.4

d) Test Type: Basic

20.1.13.2 Topic Namespace

http://www.opengis.net/spec/SWES/2.0/conf/PublishSubscribe/TopicNamespace

a) Test Purpose: To verify that the service only uses topics in its topic set for which it also provides information about the according topic namespace.

b) Test Method: Retrieve the notifications section of the service's metadata. Verify that the section contains topic namespace information for each topic listed in the topic set of the service – except for the ad-hoc topic namespace (as defined by WS-Topics).

c) Reference: 8.2.1

d) Test Type: Basic

20.1.13.3 Filter Dialects

http://www.opengis.net/spec/SWES/2.0/conf/PublishSubscribe/FilterDialects

a) Test Purpose: To verify that the service supports all filter dialects as advertised in its Capabilities.

b) Test Method: Retrieve the notifications section of the service's metadata. Create and send a Subscribe for request for each filter dialect listed in the notifications section – ensure that the filter you create is valid. Verify that the service does not return an exception/fault that indicates the filter dialect is invalid or not supported.

c) Reference: 8.2.2

d) Test Type: Basic

20.1.14 SOAP binding

20.1.14.1 Exception encoding

http://www.opengis.net/spec/SWES/2.0/conf/SOAPBinding/ExceptionEncoding

a) Test Purpose: To verify that service exceptions are encoded according to the requirements of this standard.

b) Test Method: Devise and execute a request that generates an exception. Verify that the exception that the server generates is valid according to clause 19.2 (with correct exception/fault binding – code, subcode, reason, details).

c) Reference: 19.2

d) Test Type: Basic

20.1.14.2 Action URIs

20.1.14.2.1 Operation Actions

http://www.opengis.net/spec/SWES/2.0/conf/SOAPBinding/ActionURIs/OperationActions

a) Test Purpose: To verify that the service recognizes and uses correct action URIs for operation requests and responses as well as notifications as defined in this standard.

b) Test Method: Depending upon the SOAP binding available at the service, execute a request for each operation defined by this service. Verify that the service uses the correct SOAP action as defined in Table 37 or uses an empty action in its response. If WS-Addressing is used, verify that the service uses the correct WS-Addressing action URIs as defined in Table 37.

c) Reference: 19.3

d) Test Type: Basic

20.1.14.2.2 Exception Actions

http://www.opengis.net/spec/SWES/2.0/conf/SOAPBinding/ActionURIs/ExceptionActions

a) Test Purpose: To verify that the service uses correct action URIs for exceptions/faults if WS-Addressing is used.

b) Test Method: Depending upon the SOAP binding available at the service, devise and execute a request that results in an exception for each exception relevant to the SOAP binding (see Table 38). Verify that the service uses the correct WS-Addressing action as defined in Table 38 for the exception types listed there.

c) Reference: 19.3

d) Test Type: Basic

20.1.14.3 Publish/Subscribe

http://www.opengis.net/spec/SWES/2.0/conf/SOAPBinding/PublishSubscribe

a) Test Purpose: To verify that the service uses WS-Notification for realizing the publish/subscribe functionality.

b) Test Method: Verify that the producer endpoint provided in the notifications section of the service's metadata implements the NotificationProducer interface as defined by WS-BaseNotification. Verify that the Subscribe operation defined by that interface is executed correctly. Verify that the service supports at least one

topic expression dialect, that it accepts valid subscriptions using this topic dialect to target the topics stated in its topic set and that it publishes the events that are expected to be published on these topics.

c) Reference: 8.2.1, 19.5

d) Test Type: Basic

20.2 Conformance Classes

20.2.1 Core

http://www.opengis.net/spec/SWES/2.0/conf/Core

a) Test Purpose: Verify that the server implements the *Core* conformance class.

b) Test Method: Verify the following list of conformance tests: 20.1.1.1, 20.1.1.2, 20.1.2.1, 20.1.2.2, 20.1.3, 20.1.6.

c) Reference: 9, 10, 15, 18, 22

d) Test Type: Capability

20.2.2 Basic SWE Service Metadata

http://www.opengis.net/spec/SWES/2.0/conf/BasicSWEServiceMetadata

a) Test Purpose: Verify that the server implements the *Basic SWE Service Metadata* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the following list of conformance tests: 20.1.4, 20.1.7.1.1, 20.1.7.1.2.

c) Reference: 2, 7, 22

d) Test Type: Capability

20.2.3 Sensor Provider

http://www.opengis.net/spec/SWES/2.0/conf/SensorProvider

a) Test Purpose: Verify that the server implements the *Sensor Provider* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the following list of conformance tests: 20.1.5, 20.1.9.1

c) Reference: 9.2.3, 11

d) Test Type: Capability

20.2.4 Sensor History Provider

http://www.opengis.net/spec/SWES/2.0/conf/SensorHistoryProvider

a) Test Purpose: Verify that the server implements the *Sensor History Provider* conformance class.

b) Test Method: Verify that the server implements the *Sensor Provider* conformance class. Verify the following list of conformance tests: 20.1.7.2, 20.1.9.2

c) Reference: 9.2.3, 11

d) Test Type: Capability

20.2.5 Sensor Description Manager

http://www.opengis.net/spec/SWES/2.0/conf/SensorDescriptionManager

a) Test Purpose: Verify that the server implements the *Sensor Description Manager* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the following list of conformance tests: 20.1.5, 20.1.11.1

c) Reference: 9.2.3, 12

d) Test Type: Capability

20.2.6 Sensor History Manager

http://www.opengis.net/spec/SWES/2.0/conf/SensorHistoryManager

a) Test Purpose: Verify that the server implements the *Sensor History Manager* conformance class.

b) Test Method: Verify that the server implements the *Sensor Description Manager* conformance class. Verify the following list of conformance tests: 20.1.7.3, 20.1.11.2

c) Reference: 9.2.3, 12

d) Test Type: Capability

20.2.7 Sensor Insertion

http://www.opengis.net/spec/SWES/2.0/conf/SensorInsertion

a) Test Purpose: Verify that the server implements the *Sensor Insertion* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the following list of conformance tests: 20.1.4, 20.1.10

c) Reference: 13 and OGC 07-036 clause 9.3

d) Test Type: Capability

20.2.8 XML Encoding

http://www.opengis.net/spec/SWES/2.0/conf/XMLEncoding

a) Test Purpose: Verify that the server implements the *XML Encoding* conformance class.

b) Test Method: Verify the following list of conformance tests: 20.1.12.1, 20.1.12.2

c) Reference: 15, 21

d) Test Type: Capability

20.2.9 Sensor Deletion

http://www.opengis.net/spec/SWES/2.0/conf/SensorDeletion

e) Test Purpose: Verify that the server implements the *Sensor Deletion* conformance class.

f) Test Method: Verify that the server implements the *Core* conformance class. Verify the following list of conformance tests: 20.1.7.3

g) Reference: 14

h) Test Type: Capability

20.2.10 Publish Subscribe

http://www.opengis.net/spec/SWES/2.0/conf/PublishSubscribe

a) Test Purpose: Verify that the server implements the *Publish Subscribe* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the following list of conformance tests: 20.1.13.1, 20.1.13.2, 20.1.13.3

c) Reference: 8, 17.2

d) Test Type: Capability

20.2.11 SOAP binding

http://www.opengis.net/spec/SWES/2.0/conf/SOAPBinding

a) Test Purpose: Verify that the server implements the *SOAP binding* conformance class.

b) Test Method: Verify that the server implements the *XML Encoding* conformance class.

   If the server implements one or more of the *Sensor Provider*, *Sensor History Provider*, *Sensor Description Manager*, *Sensor History Manager*, *Sensor Insertion* or *Sensor Deletion* conformance classes, verify the following list of conformance tests as applicable for the implemented operations: 20.1.14.1, 20.1.14.2.1, 20.1.14.2.2.

   If the server implements the *Publish Subscribe* conformance class, verify the conformance test 20.1.14.3.

c) Reference: 19

d) Test Type: Capability

21 Annex B - XML Schema Documents (normative)

In addition to this document, this standard includes several normative XML Schema Documents. These XML Schema Documents are bundled in a zip file with the present document. After OGC acceptance of this standard, these XML Schema Documents will also be posted online at the URL http://schemas.opengis.net/swes/2.0. In the event of a discrepancy between the bundled and online versions of the XML Schema Documents, the online files shall be considered authoritative.

The data types specified in this standard are contained in seven packages which themselves are children of the SWE Service Model package (see clause 6).

The UML model has been mapped to its XML Schema encoding using the rules described in clause 24), resulting in the following XML Schema documents:

swes.xsd (includes the other schema through xs:include statements)

swesCommon.xsd

swesContents.xsd

swesDeleteSensor.xsd

swesDescribeSensor.xsd

swesInsertSensor.xsd

swesNotification.xsd

swesUpdateSensorDescription.xsd

Note: as the types defined in the Common Codes package are not intended to be encoded as XML elements, an XML Schema file for that package is not needed and thus not available.

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/XML/GeneralEncodingRule	
REQ 88.	The XML encoding of the conceptual types defined in this standard shall be as defined by the XML Schema files listed and referenced in clause 21. More specifically, the XML encoding of each conceptual type shall be valid against the XML Schema definition of the according mapping as defined in Table 39.

The following table provides an overview how each of the conceptual model types defined by this standard has been realized in the XML Schema implementation.

Table 39 — XML Schema implementation of types defined by the SWES conceptual model

UML class	object element	type	property type
SWES Common Package			
AbstractSWES	swes:AbstractSWES	swes:AbstractSWESType	swes:AbstractSWESPropertyType
ExtensibleRequest	swes:ExtensibleRequest	swes:ExtensibleRequestType	swes:ExtensibleRequestPropertyType
ExtensibleResponse	swes:ExtensibleResponse	swes:ExtensibleResponseType	swes:ExtensibleResponsePropertyType
SensorDescription	swes:SensorDescription	swes:SensorDescriptionType	swes:SensorDescriptionPropertyType
FeatureRelationship	swes:FeatureRelationship	swes:FeatureRelationshipType	swes:FeatureRelationshipPropertyType
SWES Contents Package			
AbstractContents	swes:AbstractContents	swes:AbstractContentsType	swes:AbstractContentsPropertyType
AbstractOffering	swes:AbstractOffering	swes:AbstractOfferingType	swes:AbstractOfferingPropertyType
SWES DeleteSensor Package			
DeleteSensor	swes:DeleteSensor	swes:DeleteSensorType	swes:DeleteSensorPropertyType
DeleteSensorResponse	swes:DeleteSensorResponse	swes:DeleteSensorResponseType	swes:DeleteSensorResponsePropertyType
SWES DescribeSensor Package			
DescribeSensor	swes:DescribeSensor	swes:DescribeSensorType	swes:DescribeSensorPropertyType
DescribeSensorResponse	swes:DescribeSensorResponse	swes:DescribeSensorResponseType	swes:DescribeSensorResponsePropertyType
SWES InsertSensor Package			
InsertionMetadata	swes:InsertionMetadata	swes:InsertionMetadataType	swes:InsertionMetadataPropertyType
InsertSensor	swes:InsertSensor	swes:InsertSensorType	swes:InsertSensorPropertyType
InsertSensorResponse	swes:InsertSensorResponse	swes:InsertSensorResponseType	swes:InsertSensorResponsePropertyType

UML class	object element	type	property type
SWES Notification Package			
EventCode	-	swes:EventCodeType	-
FilterDialectMetadata	swes:FilterDialectMetadata	swes:FilterDialectMetadataType	swes:FilterDialectMetadataPropertyType
NotificationBrokerMetadata	swes:NotificationBrokerMetadata	swes:NotificationBrokerMetadataType	swes:NotificationBrokerMetadataPropertyType
NotificationProducerMetadata	swes:NotificationProducerMetadata	swes:NotificationProducerMetadataType	swes:NotificationProducerMetadataPropertyType
OfferingChanged	swes:OfferingChanged	swes:OfferingChangedType	swes:OfferingChangedPropertyType
SensorChanged	swes:SensorChanged	swes:SensorChangedType	swes:SensorChangedPropertyType
SensorDescriptionUpdated	swes:SensorDescriptionUpdated	swes:SensorDescriptionUpdatedType	swes:SensorDescriptionUpdatedPropertyType
SWESEvent	swes:SWESEvent	swes:SWESEventType	swes:SWESEventPropertyType
SWES UpdateSensorDescription Package			
UpdateSensorDescription	swes:UpdateSensorDescription	swes:UpdateSensorDescriptionType	swes:UpdateSensorDescriptionPropertyType
UpdateSensorDescriptionResponse	swes:UpdateSensorDescriptionResponse	swes:UpdateSensorDescriptionResponseType	swes:UpdateSensorDescriptionResponsePropertyType

22 Annex C - Property Inheritance Mechanism (normative)

The inheritance mechanism was introduced to reduce the amount of redundant information in the Capabilities of a service. The Web Mapping Service specification [2] uses a quite similar mechanism to reduce the size of its Capabilities.

The idea is that properties – rather, their values – can be inherited by a so-called *inheritor* type from a so called *provider* type. In the WMS, these are both "Layer" types. In this specification, these are the *AbstractOffering* and *AbstractContents* type.

Properties can be inherited from the provider in different ways:

- *arbitrary* – another element that shall be identified by the specification using this mechanism (can be of the same or another UML type)

- *parent* - the parent element (in a hierarchy of elements of the same type; example: layer list in WMS)

Requirement	
http://www.opengis.net/spec/SWES/2.0/req/PropInheritance/Style	
REQ 89.	A given property (in the conceptual model of the inheritor type) shall be inherited from the according provider property according to one of the following inheritance categories defined for that property: • *no* – the property shall not be inherited at all • *replace* – value shall be inherited from provider if omitted by inheritor, but if specified by inheritor then the provider value shall be ignored. Note: no difference is made between properties with cardinality one and those with higher cardinality, i.e. even if the provider defines multiple values for one property, all of them will be replaced if the inheritor defines its own value(s) for the property. • *add* – Inheritor inherits any value(s) supplied by provider and adds any value(s) of its own to the list. Any duplicated value by the inheritor shall be ignored. Note: this category is only applicable to properties that have a cardinality of more than one.

A specification shall clearly state for which of its data types this mechanism applies. For each identified inheritor type, the according provider type (*arbitrary* or *parent*) shall be specified.

In addition, the inheritance style needs to be documented for all properties of the inheritor. This can be done by adding a table like the following to the paragraph that specifies the inheritor type in the specification.

Table 40 — Inheritance of TypeXXX properties (from TypeXXX)

Property	Number	Inheritance Category
property1	0..*	no
property2	0..1	replace
…	…	…

The first column (“Property”) lists the name of each inheritor property exactly as it is written in the UML model of the inheritor.

The second column (“Number”) indicates the number of times each element may appear in the inheritor element, either explicitly or through inheritance:

- n: appears exactly the specified n number of times (n >= 0)
- n..m: appears n to m times (n < m, n >= 0); example: 0..1, 0..4
- n..*: appears n or more times (n >= 0); example: 0..*, 1..*

Note that this cardinality can be more restrictive than constraints enforced by the UML model of the inheritor element.

Finally, the third column (“Inheritance”) indicates whether or how the property is inherited (*no*, *replace*, *add*).

Example:

In this specification, the AbstractOffering represents an inheritor of the properties contained in the AbstractContents (the provider). The following table shows which of the properties defined in the content model of AbstractOffering can be inherited and which cardinality is expected after the inheritance mechanism has been applied.

Table 41 — Inheritance of AbstractOffering properties (from AbstractContents)

Property	Number	Inheritance
procedure	1	no
procedureDescriptionFormat	1..*	replace
observableProperty	1..*	replace
relatedFeature	0..*	replace

Figure 27 shows an exemplary structure of a service's Contents section with four associated offerings.

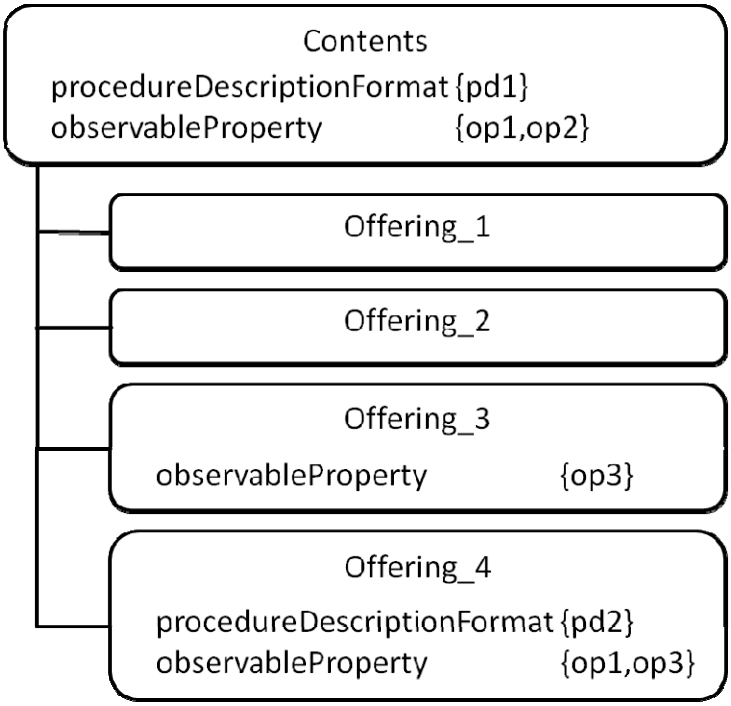

Figure 27 — Property Inheritance Mechanism, exemplary Contents structure

The following table shows which property values each offering has after applying the inheritance mechanism.

Table 42 — Property Inheritance Mechanism, resulting offering properties

Property	Offering_1	Offering_2	Offering_3	Offering_4
procedureDescriptionFormat	{pd1}	{pd1}	{pd1}	{pd2}
observableProperty	{op1,op2}	{op1,op2}	{op3}	{op1,op3}

As you can see, Offering_1, Offering_2 and Offering_3 inherit property values from the Contents element.

By intelligent structuring of the contents section, SWE services can reduce the amount of redundant information considerably.

23 Annex D – Additional UML diagrams (normative)

23.1 Introduction

This clause provides additional UML diagrams that further define the conceptual model specified by this standard.

23.2 Interfaces

The SWE Service Model defines several operations which are assigned to interfaces as shown in Figure 28. A standard leveraging this standard may require implementation of one or more of these interfaces.

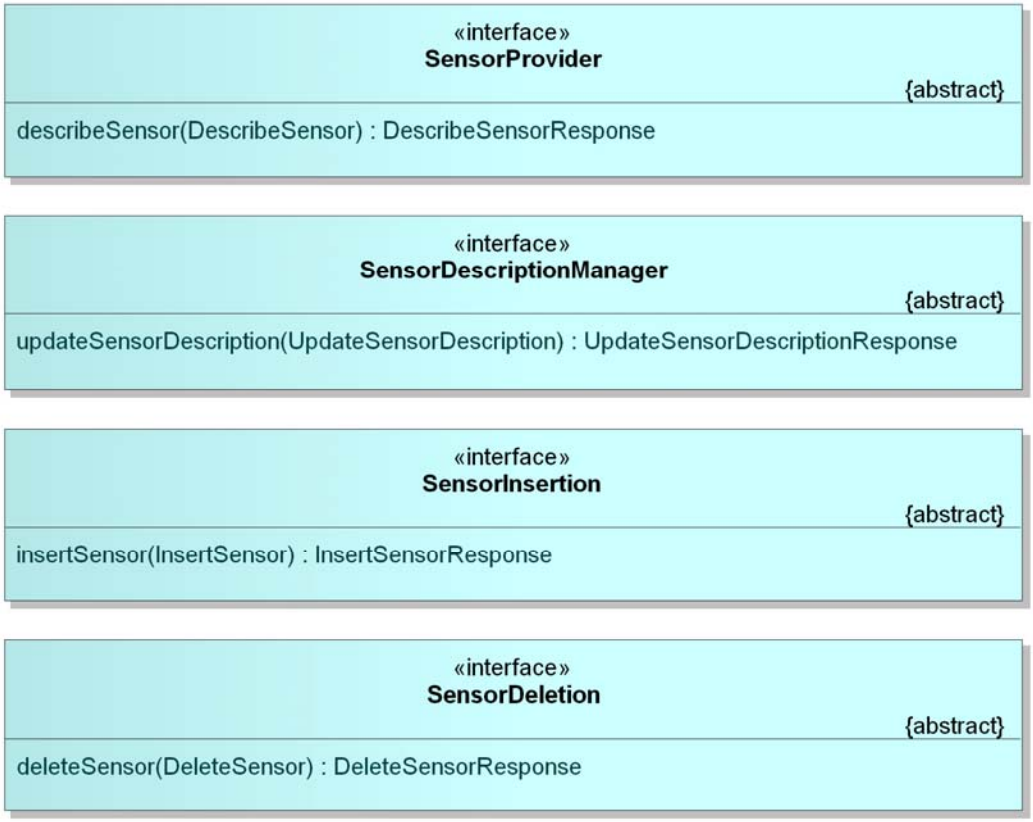

Figure 28 — SWES interfaces

24 Annex E - UML to XML Service Model Encoding Rules (informative)

24.1 Introduction

The UML model of SWE Service Model was designed following the concepts and rules of designing GML Application Schema (see [OGC 07-036]) - with some modifications and extensions. The idea was to generate the XML Schema (XSD) encoding of a service model automatically rather than handcrafting it. The XSD is created using a set of rules which define an automatic mapping from UML to XML Schema. This leads to a cleaner modeling approach both for the UML model as well as XSD encoding.

The rules used in SWES follow the rules defined in OGC 07-036 Annex E. However, because some modifications were made to these rules, the resulting model is NOT a GML Application Schema.

24.2 Encoding Rules

In the following, we only document the changes and extensions that were applied to the rules defined in OGC 07-036 Annex E.

24.2.1 General Encoding Requirements

24.2.1.1 Application schemas

24.2.1.1.1 General (application schema, packages)

Clause E.2.1.1.1 of OGC 07-036 applies without changes.

24.2.1.1.2 Classes

Clause E.2.1.1.2 of OGC 07-036 applies without changes but with one amendment.

In the SWE Service Model, object types have identity – however, a difference is made between explicit and implicit identity in the XSD encoding of an object type.

Explicit identity of an object type instance encoded in XML is only required if it is given by reference rather than inline. The entity (service or client) that uses the byReference pattern for an object type instance has to ensure that its identity is explicitly provided (also see section 24.2.4.1).

24.2.1.1.3 Attributes

Clause E.2.1.1.3 of OGC 07-036 applies without changes.

24.2.1.1.4 Associations and association ends

Clause E.2.1.1.4 of OGC 07-036 applies without changes.

24.2.1.1.5 Predefined types

Clause E.2.1.1.5 of OGC 07-036 applies without changes.

24.2.1.1.6 OCL constraints

Clause E.2.1.1.6 of OGC 07-036 applies without changes.

24.2.1.1.7 Other information

Clause E.2.1.1.7 of OGC 07-036 applies without changes.

24.2.1.2 Character repertoire and languages

Clause E.2.1.2 of OGC 07-036 applies without changes.

24.2.1.3 Exchange metadata

Clause E.2.1.3 of OGC 07-036 applies without changes.

24.2.1.4 Dataset and object identification

Clause E.2.1.4 of OGC 07-036 applies without changes.

24.2.1.5 Update mechanism

Clause E.2.1.5 of OGC 07-036 applies without changes.

24.2.2 Input data structure

Clause E.2.2 of OGC 07-036 applies without changes.

24.2.3 Output data structure

Clause E.2.3 of OGC 07-036 applies without changes.

24.2.4 Conversion rules

24.2.4.1 General concepts

Clause E.2.4.1 of OGC 07-036 applies with some changes to table E.1 and table E.2 (marked in yellow in Table 43 and Table 45 of this document).

Table 43 — Schema encoding overview

Table: UML → Service schema overview	
UML application schema	**Service schema**
Package	One XML Schema document per package (default mapping)
<<Application Schema>>	XML Schema document
<<DataType>>	Global element, whose content model is a globally scoped XML Schema complexType, property type
<<Enumeration>>	Restriction of xsd:string with enumeration values
<<CodeList>>	Union of an enumeration and a pattern (default mapping, an alternative mapping is a reference to a dictionary)
<<Union>>	Choice group whose members are objects or GML features, or objects corresponding to DataTypes
<<FeatureType>>	Global element, whose content model is a globally scoped XML Schema type derived by direct/indirect extension of gml:AbstractFeatureType, property type
No stereotype or <<Type>>	Global element, whose content model is a globally scoped XML Schema type derived by direct/indirect extension of swes:AbstractSWESType, property type
Operations	Not encoded
Attribute	local xsd:element, the type is either a property type (if the type is a complex type) or a simple type.
Association role	local xsd:element, the type is always a property type (only named and navigable roles)
General OCL constraints	Not encoded

The restriction that only GML objects can be included in a <<Union>> is removed. Thus, also object types according to this specification are allowed in a <<Union>>.

Classes without stereotype or with stereotype <<Type>> are derived in the service model by direct/indirect extension from AbstractSWES, not from AbstractGML. The model of AbstractSWES is shown in Figure 29 and further defined in Table 44.

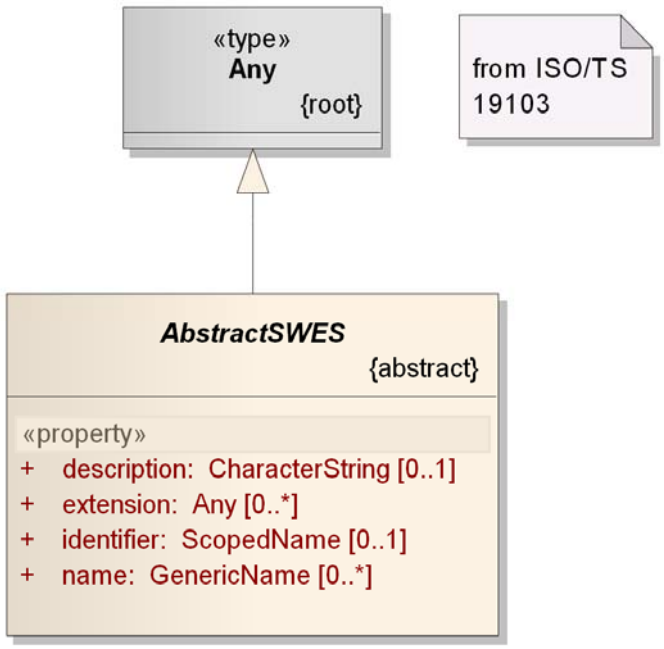

Figure 29 — AbstractSWES, base type for SWES object types

Table 44 — Properties in the AbstractSWES base type

Name	Definition	Data type and values	Multiplicity and use
description	textual description of the object	CharacterString, not empty	Zero or one (optional)
identifier	identifier of the object	ScopedName (see clause 16)	Zero or one (optional)
name	label/name of the object	GenericName (see clause 16)	Zero or more (optional)
extension	place where other specifications may insert additional information – a service shall ignore values it does not support	Any type the actual type depends upon the extension specification	Zero or more (optional)

Note: identity within an XML instance document can be provided via the swes:id attribute available in the XML encoding of SWES objects. This attribute should primarily be used for referencing the according objects via the xlink:href mechanism.

When a SWE service receives a request that contains SWES objects that have their extension property populated with an unknown value, then it shall ignore that extension value. All request extensions that request specific behavior from the service shall be stated through an according extension value in the ExtensibleRequest type (see clause 9) or a derivation thereof. This ensures that a service that does not support requested extension behavior can easily determine that situation and raise a

RequestExtensionNotSupported exception. An extension that is indicated in the ExtensibleRequest's extension property may of course involve additional population of the extension properties from SWES objects that are contained in the request and/or response (or in events).

The XML Schema encoding of AbstractSWES shall be as defined in the following listing.

Listing 16 – XML Schema encoding of the AbstractSWES base type

```
  <element name="AbstractSWES" type="swes:AbstractSWESType"
abstract="true"/>
  <complexType name="AbstractSWESType" abstract="true">
    <sequence>
      <element name="description" type="string"
minOccurs="0"/>
      <element name="identifier" type="anyURI"
minOccurs="0"/>
      <element name="name" type="gml:CodeType"
minOccurs="0" maxOccurs="unbounded"/>
      <element name="extension" type="anyType"
minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="swes:id"/>
  </complexType>
  <attribute name="id" type="ID">
    <annotation>
      <documentation>Supports provision of a handle for the
XML element representing a SWES Object. It is of XML type
ID, so is constrained to be unique in the XML document
within which it occurs.</documentation>
    </annotation>
  </attribute>
```

Table 45 — Tagged values

UML model element	Associated tagged values
Package	• documentation • xsdDocument • targetNamespace (only <<Application Schema>>) • xmlns (only <<Application Schema>>) • version (only <<Application Schema>>) • gmlProfileSchema (only <<Application Schema>>)
Class	• documentation • noPropertyType • byValuePropertyType • isCollection • asDictionary (only <<CodeList>>) • xmlSchemaType (only <<Type>>)
Attribute and association end	• documentation • sequenceNumber • inlineOrByReference • isMetadata • asXMLAttribute (see section 24.2.4.11) • purpose (see section 24.2.4.11)

Note: the tagged value “gmlProfileSchema” was not renamed because profiling a service schema is not in the scope of this standard. A future version of this standard can define rules for service model profiles and their encoding to XSD.

24.2.4.2 UML packages

Clause E.2.4.2 of OGC 07-036 applies with one change. The gmlProfileSchema tagged value used on a package with stereotype <<Application Schema>> shall be ignored.

24.2.4.3 UML classes (general rules)

Clause E.2.4.3 of OGC 07-036 applies without changes.

24.2.4.4 UML classes (basic types)

Clause E.2.4.4 of OGC 07-036 applies without changes.

24.2.4.5 UML classes (data types)

Clause E.2.4.5 of OGC 07-036 applies with the following changes:

- The supertype of a class with stereotype <<DataType>> shall not derive (directly or indirectly) from `swes:AbstractSWESType`.

- If the class has no superclass then the substitution group of the global XML element shall be empty – a <<DataType>> in the service model shall not be in the substitution group of `gml:AbstractObject`.

24.2.4.6 UML classes (feature types)

Clause E.2.4.6 of OGC 07-036 applies without changes.

24.2.4.7 UML classes (object types)

Clause E.2.4.7 of OGC 07-036 applies with the following changes:

- UML classes with no stereotype or stereotype <<Type>> derive directly or indirectly from `swes:AbstractSWESType`, not from `gml:AbstractGMLType`.

- If the class is a class without supertype, it directly extends `swes:AbstractSWESType`, otherwise it extends its supertype which shall be derived from `swes:AbstractSWESType` (again, directly or indirectly)

- The substitution group of the global XML element shall be the name of the supertype or `swes:AbstractSWES` for a class without supertype.

24.2.4.8 UML classes (enumerations)

Clause E.2.4.8 of OGC 07-036 applies without changes.

24.2.4.9 UML classes (code lists)

Clause E.2.4.9 of OGC 07-036 applies with one change:

- The mapping of a UML class with stereotype <<CodeList>> without tagged value "asDictionary" equals "true" shall have the facet "`<pattern value='other: [A-Za-z0-9_]{2,}'/>`".

Note: The regular expression used here is more explicit than the one from 07-036. Some tools – like XMLSpy – have problems recognizing the underscore, therefore the switch to the more explicit form.

24.2.4.10 UML classes (unions)

Clause E.2.4.10 of OGC 07-036 applies without changes.

24.2.4.11 UML attributes and association roles

Clause E.2.4.11 of OGC 07-036 applies with the following changes/amendments:

- If the type of the a property is of simple content, the tagged value "asXMLAttribute" of the property is "true" and the cardinality is 0..1 or 1 then the property shall be encoded as an attribute with the according type.

  Example: <attribute name="updatable" type="boolean" use="required"/>

- If the target end of a composition is a UML class with stereotype <<DataType>> then only the inline pattern shall be used.

- All attributes and association roles represent properties of the application schema and therefore may be marked with the stereotype <<property>>.

- If the tagged value "soft-typed" on a property is set to "true" then a "name" attribute of type xs:NCName shall be added in the XML encoding of that property and the attribute shall not be empty. Soft-typed properties are usually of a SWE Common type (see OGC 08-094r1).

- If the tagged value "inlineOrByReference" on a property (attribute or association end) is "byReference" and the property has a tagged value "purpose" with value "identifier" then in the XML Schema encoding the type of that property shall be xs:anyURI instead of gml:ReferenceType. This applies also to the encoding of properties that have stereotype <<CodeList>> or <<Enumeration>>. The appinfo annotation element gml:targetElement shall have the value "unknown" in case that no XML Schema element type is known or available for the target type of such a property[5].

- The gml:OwnershipAttributeGroup is not used. Therefore also no schema constraint regarding this attribute group is used.

24.2.4.12 UML classes (unions)

Clause E.2.4.12 of OGC 07-036 applies without changes.

24.2.4.13 Classes imported from the ISO 19100 series of International Standards

Clause E.2.4.13 of OGC 07-036 applies with one change.

[5] This is for example the case with the *GFI_PropertyType* type defined by O&M.

In the SWE Service Model, the ScopedName type is encoded as xs:anyURI. Standards leveraging this specification may use the mapping defined in table D.2 of OGC 07-036.

24.2.4.14 Classes imported from other conceptual models with a predefined XML encoding

Clause E.2.4.14 of OGC 07-036 applies without changes.

25 Annex F - Revision history

Date	Release	Editor	Primary clauses modified	Description
08-12-16	0.1.0	Johannes Echterhoff	throughout	initial draft
09-09-22	0.2.0	Johannes Echterhoff	throughout	some modifications
09-11-20	1.0.0	Johannes Echterhoff	throughout	first draft version for RFC
09-12-17	1.0.0	Johannes Echterhoff	throughout	RFC version
10-07-28	1.0.0	Johannes Echterhoff	throughout	first draft version for adoption vote
10-08-10	1.0.0	Johannes Echterhoff	throughout	second draft version for adoption vote
10-08-12	1.0.1	Ingo Simonis	throughout	requirements added
10-09-10	1.0.2	Johannes Echterhoff	DescribeSensor (11)	defined response behavior for requests with validTime parameter in more detail
10-09-30	1.0.3	Johannes Echterhoff	throughout	changed type of SensorDescription/data and InsertSensor/procedureDescription to OM_Process
11-01-17	1.0.4	Johannes Echterhoff	throughout	changed usage of AbstractFeature to GFI_Feature, added FeatureRelationship type to enhance semantics of relatedFeature properties

Bibliography

[1] OASIS (2006), *Web Services Resource Properties 1.2* (OASIS Standard April 1st 2006)

[2] OpenGIS® Implementation Specification, *Web Map Server*, OGC document 06-042

[3] Public Engineering Report, *OGC® OWS-6 SWE Event Architecture Engineering Report*, OGC document 09-032