Open Geospatial Consortium, Inc.

Date: 2009-07-16

Reference number of this document: OGC 09-031r1

Version: 0.3.0

Category: Public Engineering Report

Editor: Thomas Everding

OGC® OWS-6 SWE Information Model Engineering Report

Warning

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

Document type:	Public Engineering Report
Document subtype:	NA
Document stage:	Approved for public release
Document language:	English

Preface

This Engineering Report (ER) is a deliverable for the Open Geospatial Consortium (OGC) Interoperability Program Open Web Service (OWS) Testbed phase 6 (OWS-6).

This work was supported by the European Commission through the OSIRIS project, an Integrated Project, contract number 033475, Information Society and Media DG of the European Commission within the RTD activities of the Thematic Priority Information Society Technologies.

This work was supported by the European Commission through the GENESIS project, an Integrated Project, contract number 223996.

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

The changes made in this document version, relative to the previous version, are tracked by Microsoft Word, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

Forward

OWS-6 Testbed

OWS testbeds are part of OGC's Interoperability Program, a global, hands-on and collaborative prototyping program designed to rapidly develop, test and deliver Engineering Reports and Chnage Requests into the OGC Specification Program, where they are formalized for public release. In OGC's Interoperability Initiatives, international teams of technology providers work together to solve specific geoprocessing interoperability problems posed by the Initiative's sponsoring organizations. OGC Interoperability Initiatives include test beds, pilot projects, interoperability experiments and interoperability support services - all designed to encourage rapid development, testing, validation and adoption of OGC standards.

In April 2008, the OGC issued a call for sponsors for an OGC Web Services, Phase 6 (OWS-6) Testbed activity. The activity completed in June 2009. There is a series of on-line demonstrations available here: http://www.opengeospatial.org/pub/www/ows6/index.html

The OWS-6 sponsors are organizations seeking open standards for their interoperability requirements. After analyzing their requirements, the OGC Interoperability Team recommended to the sponsors that the content of the OWS-6 initiative be organized around the following threads:

1. Sensor Web Enablement (SWE)
2. Geo Processing Workflow (GPW)
3. Aeronautical Information Management (AIM)
4. Decision Support Services (DSS)
5. Compliance Testing (CITE)

The OWS-6 sponsoring organizations were:

- U.S. National Geospatial-Intelligence Agency (NGA)
- Joint Program Executive Office for Chemical and Biological Defense (JPEO-CBD)
- GeoConnections - Natural Resources Canada
- U.S. Federal Aviation Agency (FAA)
- EUROCONTROL
- EADS Defence and Communications Systems
- US Geological Survey
- Lockheed Martin

- BAE Systems
- ERDAS, Inc.

The OWS-6 participating organizations were:

52North, AM Consult, Carbon Project, Charles Roswell, Compusult, con terra, CubeWerx, ESRI, FedEx, Galdos, Geomatys, GIS.FCU, Taiwan, GMU CSISS, Hitachi Ltd., Hitachi Advanced Systems Corp, Hitachi Software Engineering Co., Ltd., iGSI, GmbH, interactive instruments, lat/lon, GmbH, LISAsoft, Luciad, Lufthansa, NOAA MDL, Northrop Grumman TASC, OSS Nokalva, PCAvionics, Snowflake, Spot Image/ESA/Spacebel, STFC, UK, UAB CREAF, Univ Bonn Karto, Univ Bonn IGG, Univ Bunderswehr, Univ Muenster IfGI, Vightel, Yumetech.

Contents

Page

Figures

Tables

Listings

OGC® OWS-6 SWE Information Model Engineering Report

1 Introduction

1.1 Scope

This OGC® document is an OGC Engineering Report for the "Harmonization of SWE Information Models" activity within the OWS-6 SWE thread.

The document discusses relations between OGC standards SensorML, SWE Common and GML and investigates solutions for increased synergy between these standards. This activity also created UML models of the data types used in SWE and GML.

This report shows how UncertML can be integrated into different SWE encodings, namely SWE Common and Observations and Measurements.

This report further discusses the integration of MathML and EML into the SWE environment with an emphasis on SensorML processes and processing.

This document does not discuss the SWE information model related aspects of catalog entries for sensor services and discovery. This topic is covered in a separate Engineering Report.

1.2 Document contributor contact points

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

Name	Organization
Thomas Everding	Institute for Geoinformatics, University of Muenster
Charles Rosswell	Individual
Matthew Williams	Aston University, Birmingham
Edzer Pebesma	Institute for Geoinformatics, University of Muenster
Jan Dürrfeld	Institute for Geoinformatics, University of Muenster
Dan Cornford	Aston University, Birmingham
Lucy Bastin	Aston University, Birmingham

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
23.10.2008	0.0.0	TE	all	Initial draft
20.03.2009	0.0.2	TE	6 and 7	Second draft
17.04.2009	1.0.0	TE	all	release version
10.07.2009	0.3.0	Carl Reed	Various	Prepare for public release

1.4 Future work

Try to realize the recommendations as described in chapter 6 for the harmonization of SWE Common and GML in future versions of these standards.

The work regarding the integration of UncertML into SWE should be extended towards the integration into the whole OGC standards suite. As the next step it is recommended to investigate the integration of UncertML in coverages and the Web Coverage Service.

2 References

The following documents are referenced in 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.

ISO 19103:2005, *Conceptual Schema Language*.

ISO 19107:2003, *Spatial Schema*

ISO 19108:2002, *Temporal Schema*

ISO 19109:2005, *Rules for Application Schema*

ISO 19111:2007, *Spatial Referencing by Coordinates*

ISO 19115:203, *Metadata*

ISO 19118:2005, *Encoding*

ISO 19123:2005, *Coverage Geometry and Functions*

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

ISO 19139:2007, *Metadata – XML Schema Implementation*

OGC 04-095, *OpenGIS® Filter Encoding Implementation Specification*

OGC 06-121r3, *OpenGIS® Web Services Common Specification*

OGC 07-000, *OpenGIS® Sensor Model Language (SensorML) Implementation Specification*

OGC 07-036, *OpenGIS® Geography Markup Language (GML) Encoding Standard*

OGC 08-132, *Event Pattern Markup Language (EML)*

W3C Recommendation 21 October 2003, *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*

W3C Working Draft 17 November 2008, *Mathematical Markup Language (MathML) Version 3.0*

3 Terms and definitions

For the purposes of this report, 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.

3.1
event
Anything that happens or is contemplated as happening at an instant or over an interval of time.

NOTE: The term event may also be used for event objects. The current meaning depends on the context.

3.2
event object
An object that represents, encodes, or records an event, generally for the purpose of computer processing.

3.3
event pattern language
event processing language
A high level computer language for defining the behavior of event processing agents.

3.4
event processing
Computing that performs operations on events, including reading, creating, transforming and deleting events.

3.5
feature
An abstraction of real world phenomena.

3.6
observation
An act of observing a property or phenomenon, with the goal of producing an estimate of the value of the property.

3.7
phenomenon
A physical property that can be observed and measured, such as temperature, gravity, chemical concentration, orientation, number-of-individuals.

A characteristic of one or more feature types, the value for which must be estimated by application of some procedure in an observation.

3.8
process
A process that takes one or more inputs, and based on parameters and methodologies, generates one or more outputs.

3.9
sensor
An entity capable of observing a phenomenon and returning an observed value.

4 Conventions

4.1 Abbreviated terms

CEP	Complex Event Processing
EML	Event Pattern Markup Language
ER	OGC Interoperability Program Engineering Report
ESP	Event Stream Processing
GML	Geography Markup Language
GPS	Global Positioning System
GUM	Guide to the Expression of Uncertainty in Measurement
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ISO/TC211	ISO Technical Committee 211
ISO/TS	ISO Technical Specification
MathML	Mathematical Markup Language

MC	Monte Carlo
NO2	Nitrogen Dioxide
O&M	Observations and Measurements
OGC	Open Geospatial Consortium
OWS-6	OGC Web Services, Phase 6
PM10	Particulate Matter 10
RELAX NG	Regular Language Description for XML New Generation
SAS	Sensor Alert Service
SensorML	Sensor Model Language
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SWE	Sensor Web Enablement
TML	Transducer Markup Language
UML	Unified Modeling Language
UncertML	Uncertainty Markup Language
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
W3C	World Wide Web Consortium
WNS	Web Notifications Service
XML	Extensible Markup Language

4.2 UML notation

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

5 SWE Information Model Harmonization overview

In chapter 6 of this Engineering Report (ER), solutions for synergy between SensorML and GML are investigated. The goal is to gain a better understanding of interoperability issues between SWE and GML. Therefore UML models were used or developed.

In the following chapter it is described how UncertML can be used in SWE and how it can be integrated into the SWE encodings.

Chapter 8 discusses different additional markup languages and their usage in SWE. These languages are mainly MathML for the representation of mathematical terms and EML for the description of complex event processing.

6 Harmonization of SWE information models

6.1 Introduction

6.1.1 Goal

The goal of this task is to develop or make use of existing UML models and application schemas to gain a better understanding of interoperability issues between SWE and GML. The Statement of Work identifies SensorML, GML, UncertML, and MathML as standards of interest in this context.

6.1.2 Sensor Model Language

Sensor Model Language (SensorML) provides general models and XML encodings for describing sensors and observation processing. It has evolved in OGC as a member of the SWE family of standards (Table 1).

Table 1 - Sensor Web Enablement (SWE) Standards

Sensor Model Language (SensorML)
Observations and Measurements (O&M)
Transducer Markup Language (TML)
Sensor Observation Service (SOS)
Sensor Planning Service (SPS)
Sensor Alert Service (SAS)
Web Notification Service(WNS)

Although SensorML serves as a component of the SWE framework, it does not depend upon other SWE components and may be used independently of those components. The current Version 1.0 of SensorML (Document 07-000) specifies a number of elements used by other components of the SWE family, which are therefore grouped under the heading of SWE Common. This section of the document is currently being revised and spun off as a separate standard (SWE Common 2.0).

6.1.3 Geography Markup Language

Geography Markup Language (GML) is an XML grammar written in XML Schema for the description of application schemas and the interchange of geographic information. GML has been developed by OGC beginning in 1999. It was submitted to ISO/TC211 in 2001 for revision and publication as an ISO International Standard under the cooperative agreement between OGC and ISO/TC211. OGC GML Version 3.2.1 is identical to ISO 19136:2007. A consequence of the revision done under the auspices of ISO TC211 is that GML Version 3.2.1 has been harmonized with and depends upon a number of other geographic information standards developed within ISO/TC211 (Table 2), some of which have been adopted as OGC Abstract Specification Topics. With the exceptions of ISO 19118 and ISO 19139, these are abstract standards; i.e., they are platform and processing language independent. GML is effectively an XML implementation of the abstract concepts specified in these International Standards.

Table 2 - GML Dependencies on other ISO/TC211 Standards

Standard Number	**Title**	**OGC Abstract Specification Topic Number**
ISO/TS 19103: 2005	Conceptual Schema Language	
ISO 19107:2003	Spatial schema	1
ISO 19108:2002	Temporal schema	
ISO 19109:2005	Rules for application schema	
ISO 19111:2007	Spatial referencing by coordinates	2
ISO 19115:2003	Metadata	11
ISO 19118:2005	Encoding	
ISO 19123:2005	Coverage geometry and functions	6
ISO 19139:2007	Metadata – XML schema implementation	

6.1.4 Issues

6.1.4.1 Target for harmonization

The domains of SensorML and GML are, in fact, quite different. SensorML is focused on the description of sensors and of the processes applied to sensor observations. GML is focused on the description of geographic features and their characteristics. The intersection between these two domains is to be found not in SensorML per se, but in SWE Common.

Since GML, as noted above, is dependent upon a number of other ISO/TC211 standards, harmonization with GML necessarily requires harmonization with the suite of ISO/TC211 standards. The analysis described below, therefore, involves a search for those concepts that are shared by both SWE Common and the ISO/TC211 standards.

6.1.4.2 Archictectural differences

There is a fundamental difference in the structure of GML as compared to SWE Common.

As an implementation of the ISO/TC211 abstract standards, GML follows a database organization paradigm that involves a top-down view of the data. A feature description includes a list of its attributes; an attribute description includes a characterization of the data type of its values; a value is ultimately no more than string or a number.

SWE Common, on the other hand, follows a value tagging paradigm or bottom-up view of the data, whereby a value carries information about its data type as well as information about the phenomenon it represents.

This difference has a major impact on the degree to which harmonization can be accomplished.

6.1.4.3 Platform independent models

ISO 19119, which has been adopted by OGC as Abstract Specification Topic 12, requires each service specification to contain a platform independent UML model of the service, in addition to specifications of one or more platform dependent implementations. OGC is moving toward a policy of requiring this of all OGC standards. Thus, one of the principles applied to this harmonization activity is that the SWE Common specification currently in development will contain a platform independent UML model.

One aspect of platform independence is the use of language independent data types such as those specified in ISO/IEC 11404 and represented as UML classes in ISO/TS 19103. Given its historic connection to the XML implementation specified in SensorML, the current version of SWE Common uses a number of data types specified in the XML Schema specification.

6.2 Class level mapping between GML and SWE Common

6.2.1 Introduction

There are three areas where GML and SWE Common seem to have equivalent classes: the classes of the GML valueObjects package are similar to many of the SWE Simple Data Types and Aggregate Data Types; the temporal classes from ISO 19108 that are implemented in GML resemble temporal classes specified in SWE Common; finally, the position classes from ISO 19107 that are implemented in GML are equivalent to parts of the position classes specified in SWE Common. Each of these groups is treated in a separate subclause below.

6.2.2 valueObjects and simple data types

Classes from the valueObjects package of GML seem to match many of the Simple Data Types and Generic Data Aggregates of SWE Common (07-000 Figures 8.1 and 8.2). These SWE Common data types are reused throughout SWE Common and SensorML. Table 3 lists the leaf classes from the GML valueObjects package and the classes in SWE Common that seem to be equivalent.

Table 3 - Leaf Classes from GML valueObjects Package and SWE Common Equivalents

GML 3.2.1 (ISO 19136)	SWE Common 1.0
BooleanValue	Boolean
Category	Category
Count	Count
Quantity	Quantity
CountExtent	CountRange
QuantityExtent	QuantityRange
CategoryList	tokenList
CountList	doubleList
QuantityList	doubleList
ValueArray	DataArray
Value	DataValue

It should be noted that doubleList and tokenList are not specified as UML classes in 07-000, although they are used as data types in Figure 8.1; tokenList is specified as a simpleType in the basicTypes.xsd schema of Annex B.2.

Although these classes can be aligned at the class name level, there are a number of significant differences in both properties and relationships. Compare Figure 1 to Figure 2 and Figure 3 to Figure 4.

The most obvious difference is in attribution. The simple GML value objects (Figure 3) have a only a mandatory *value* attribute, while the SWE Common simple data types (Figure 2) have up to eleven attributes (Table 4); all, including *value*, are optional. The difference seems to reflect a fundamental difference in approach between GML and SensorML. GML follows the typical ISO/TC211 pattern in which feature types and their properties are described quite specifically. The description of attribute type includes the data type of its value, usually a simple value type. The attributes of the SensorML AbstractProcess class, on the other hand, are very generic. Each of their values is a description of a more specific property possibly including its value. The differences rule out any possibility of harmonizing by simple substitution of classes from one standard into the other.

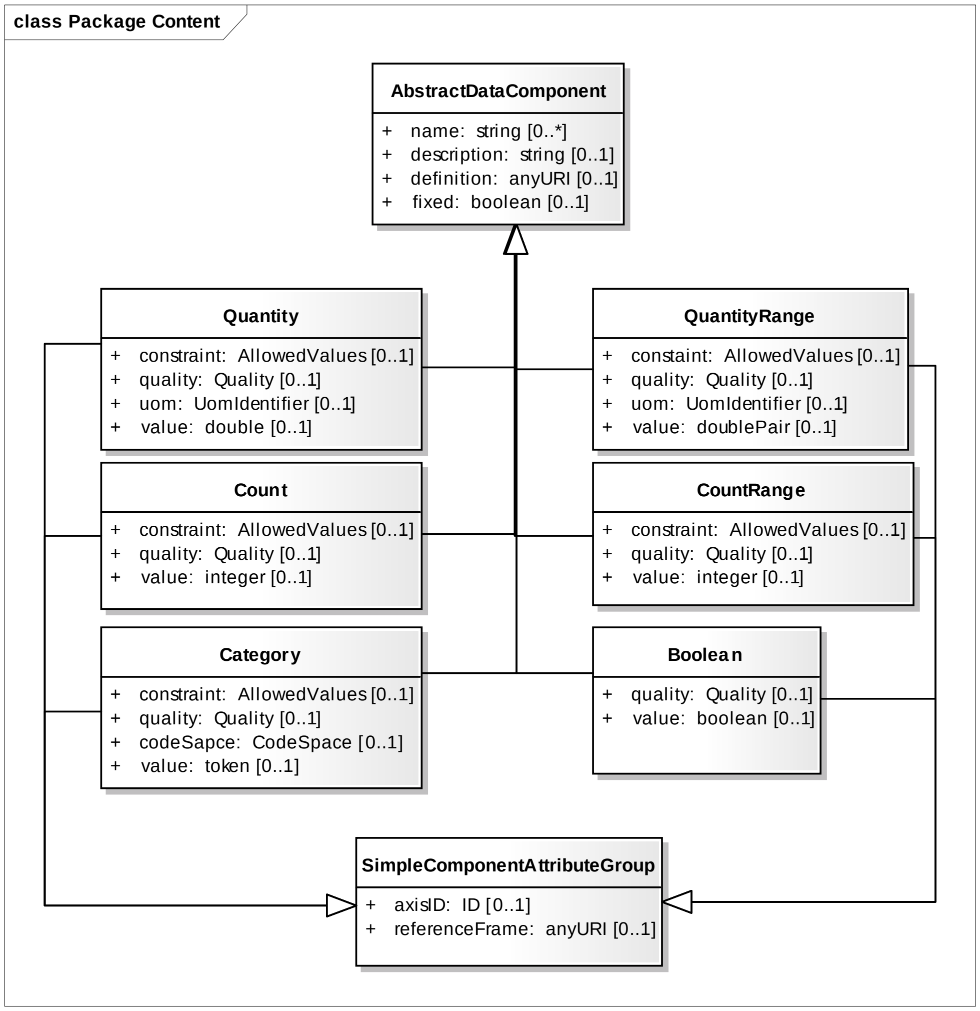
class Package Content
AbstractDataComponent
+ name: string [0..*]
+ description: string [0..1]
+ definition: anyURI [0..1]
+ fixed: boolean [0..1]
Quantity
+ constraint: AllowedValues [0..1]
+ quality: Quality [0..1]
+ uom: UomIdentifier [0..1]
+ value: double [0..1]
QuantityRange
+ constaint: AllowedValues [0..1]
+ quality: Quality [0..1]
+ uom: UomIdentifier [0..1]
+ value: doublePair [0..1]
Count
+ constraint: AllowedValues [0..1]
+ quality: Quality [0..1]
+ value: integer [0..1]
CountRange
+ constraint: AllowedValues [0..1]
+ quality: Quality [0..1]
+ value: integer [0..1]
Category
+ constraint: AllowedValues [0..1]
+ quality: Quality [0..1]
+ codeSapce: CodeSpace [0..1]
+ value: token [0..1]
Boolean
+ quality: Quality [0..1]
+ value: boolean [0..1]
SimpleComponentAttributeGroup
+ axisID: ID [0..1]
+ referenceFrame: anyURI [0..1]

Figure 1 - SWE Common Simple Data Types equivalent to GML valueObject classes

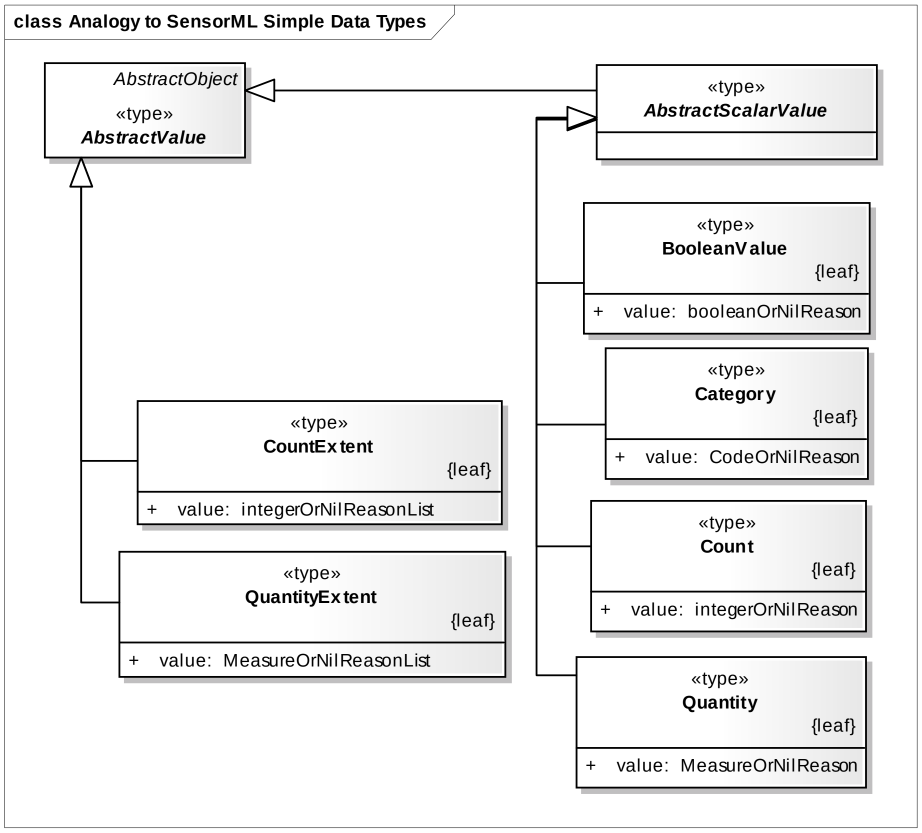

Figure 2 - GML valueObjects Equivalent to SWE Common Simple Data Types

Table 4 - Attributes of SWE Common Simple Data Types

	Boolean	Category	Count	Count Range	Quantity	Quantity Range	Text	Time	Time Range
name	X	X	X	X	X	X	X	X	X
description	X	X	X	X	X	X	X	X	X
definition	X	X	X	X	X	X	X	X	X
fixed	X	X	X	X	X	X	X	X	X
axialID	X	X	X	X	X	X			
referenceFrame	X	X	X	X	X	X			
constraint		X	X	X	X	X		X	X
quality	X	X	X	X	X	X		X	X
uom					X	X		X	X
value	X	X	X	X	X	X	X	X	X
codeSpace		X							
localFrame								X	X
referenceFrame								X	X
referenceTime								X	X

Comparison of Figure 3 and Figure 4 reveals the same problem. Although the ValueArray and Value classes of GML superficially resemble the DataArray and DataValue classes of SWE Common, their attribution is quite different.

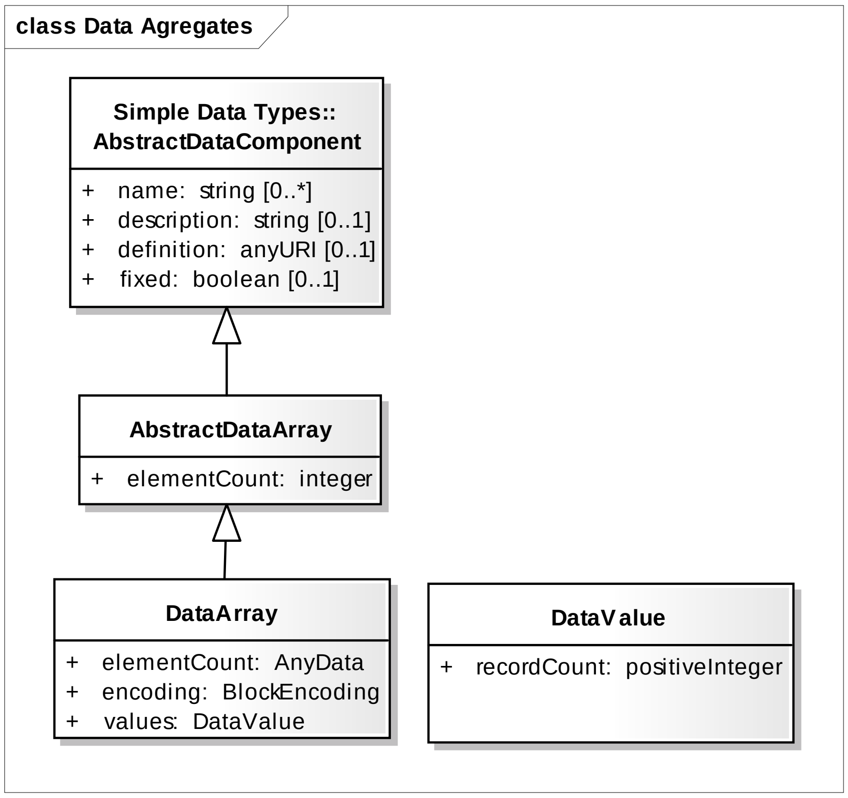

Figure 3 - SWE Common Data Aggregates equivalent to GML valueObjects

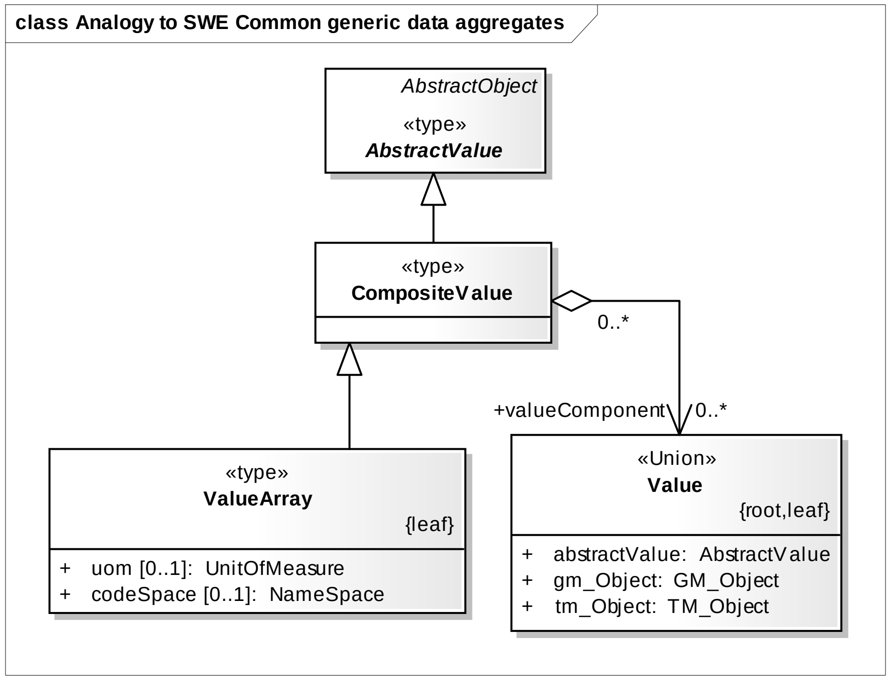

Figure 4 - GML valueObjects Classes Equivalent to SWE Common Generic Data Aggregates

In addition to differences in the numbers and names of attributes, there are differences in the basic data types used for attribute values.

Other differences are structural. SWE Common specifies atomic data types, ranges, and aggregate data types all as direct subclasses of AbstractDataComponent and uses the <<Union>> classes AnyNumerical, AnyScalar, and AnyRange to point to related groups of these data types. GML does not use the <<Union>> stereotype. It classes its atomic data types under the intermediate subclass AbstractScalarValue and its equivalents to the aggregate data types under the intermediate subclass CompositeValue. GML does not provide an equivalent to AnyRange.

The GML valueObjects package includes a set of list data types grouped under the intermediate subclass AbstractScalarValueList. Each of these specifies a list of values of one of the atomic data types. SWE Common does not specify list data types at this level, but does use doubleList, tokenList, and timePositionList as data types of attributes in its allowed values classes. As noted above, none of these are modeled as UML classes in 07-000, although tokenList is specified in the basicTypes.xsd schema of Annex B.2 and TimePositionListType is specified in the temporalAggregatges.xsd schema of Annex B.9.

There are two possibilities for partial harmonization between the GML valueObjects and the SWE Common Simple Data Types. One is to adopt a set of common basic data types. Under the concept of developing a platform independent model, these should be conceptual (language independent) data types. This is discussed in subclause 6.3. The other possibility is to consider subclassing of SWE simple data types from the valueObjects of GML 3.2. This would require SWE Common to adopt the GML concept of requiring a value or a reason for not providing a value.

6.2.3 Temporal data types

SWE Common specifies several classes for time values (Figure 5). There is already some harmonization with the ISO/TC211 standards, but more can be achieved.

Note: The discussion of harmonization in this chapter is generally based on SWE Common 1.0 (Document 07-000) as specified by the Statement of Work. However, modeling of time in SWE Common 1.0 suffers from a number of errors and omissions that have been corrected during the development of SWE Common 2.0. This subclause therefore considers the temporal data types specified in the January 2009 UML model for SWE Common 2.0.

The referenceTime attribute of AbstractTimeComponent does not appear to be necessary. ISO 8601 date and time values are referenced to the Gregorian calendar and Coordinated Universal Time (UTC), for which reference times are specified. If dates and times are provided in this format, it is not necessary to provide an additional reference time. Each of the subclasses of the TM_ReferenceSystem class that is used as the data type for the referenceFrame and localFrame attributes carries an attribute to provide a reference to Gregorian calendar and UTC.

The two <<Union>> classes TimePosition and TimeISO8601 are equivalent to the ISO 19108 classes TM_Position and TM_TemporalPosition (Figure 6). The <<Union>> class TM_Position supports all the options of TimeISO8601 with an option for alternative methods of identifying temporal position. The TM_Coordinate subclass of TM_TemporalPosition supports the byReal option of TimePosition, although it allows the value to be any Number rather than restricting it to Real. The other subclasses of TM_TemporalPosition allow referencing temporal position to an ordinal system such as the geologic time scale, or to a calendar/time system other than the Gregorian calendar and UTC. TM_Position and the subclasses of TM_TemporalPosition have been implemented in GML 3.2.

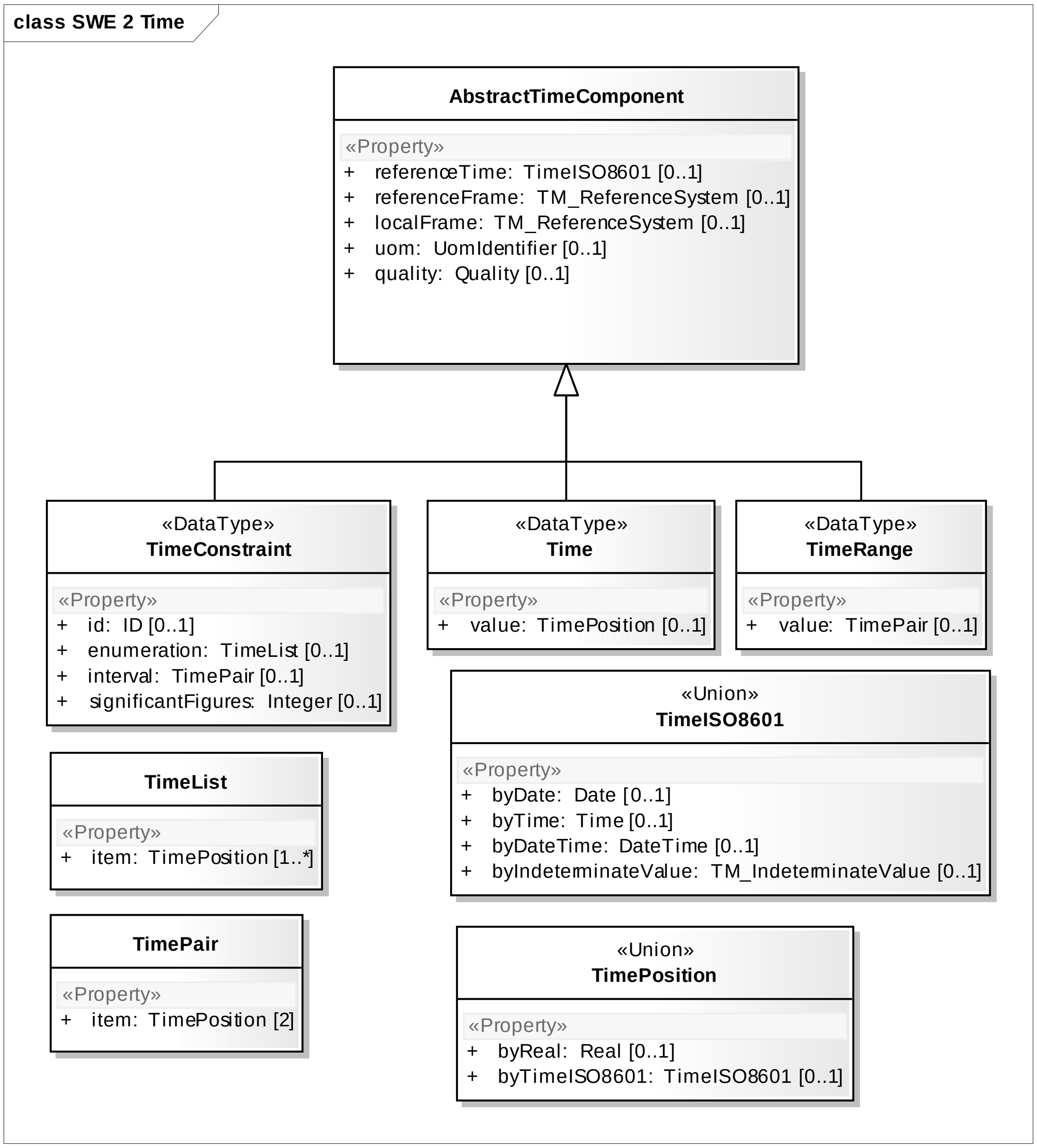
class SWE 2 Time
AbstractTimeComponent
«Property»
+ referenceTime: TimeISO8601 [0..1]
+ referenceFrame: TM_ReferenceSystem [0..1]
+ localFrame: TM_ReferenceSystem [0..1]
+ uom: UomIdentifier [0..1]
+ quality: Quality [0..1]
«DataType» TimeConstraint
«Property»
+ id: ID [0..1]
+ enumeration: TimeList [0..1]
+ interval: TimePair [0..1]
+ significantFigures: Integer [0..1]
«DataType» Time
«Property»
+ value: TimePosition [0..1]
«DataType» TimeRange
«Property»
+ value: TimePair [0..1]
TimeList
«Property»
+ item: TimePosition [1..*]
TimePair
«Property»
+ item: TimePosition [2]
«Union» TimeISO8601
«Property»
+ byDate: Date [0..1]
+ byTime: Time [0..1]
+ byDateTime: DateTime [0..1]
+ byIndeterminateValue: TM_IndeterminateValue [0..1]
«Union» TimePosition
«Property»
+ byReal: Real [0..1]
+ byTimeISO8601: TimeISO8601 [0..1]

Figure 5 - Temporal data types from SWE Common

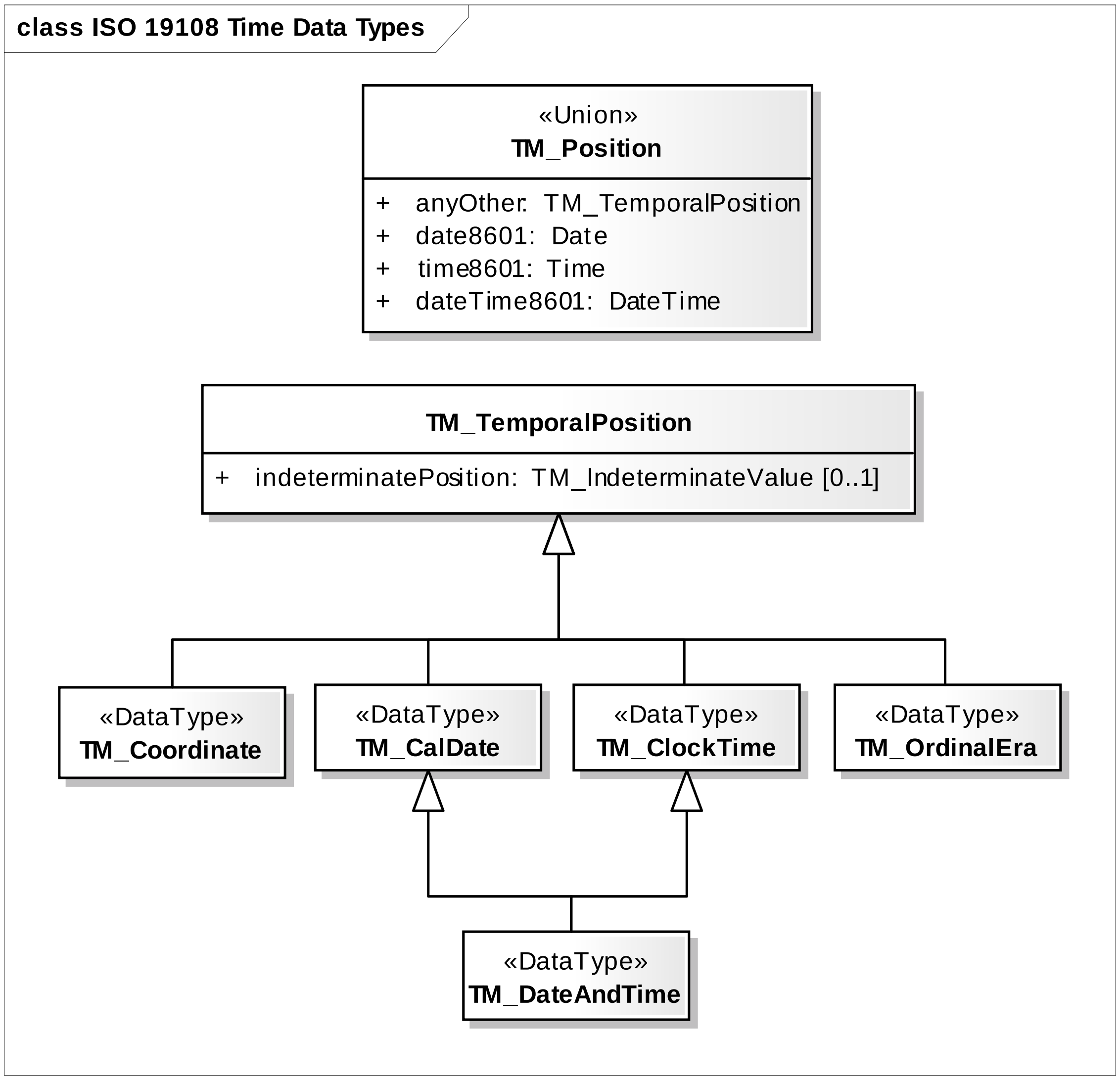

Figure 6 - Temporal data types from ISO 19108

Using TM_Position and TM_TemporalPosition in place of TimePosition and TimeISO8601 results in replacing TimePosition as the data type of the ‘value’ attribute of the SWE Common Time class and the ‘item’ attributes of TimePair and TimeList with TM_Position.

This report recommends that the SWE Common temporal classes shown in Figure 5 be replaced with the modified classes show in Figure 7 that make use of the two classes TM_Position and TM_TemporalPosition from ISO 19108.

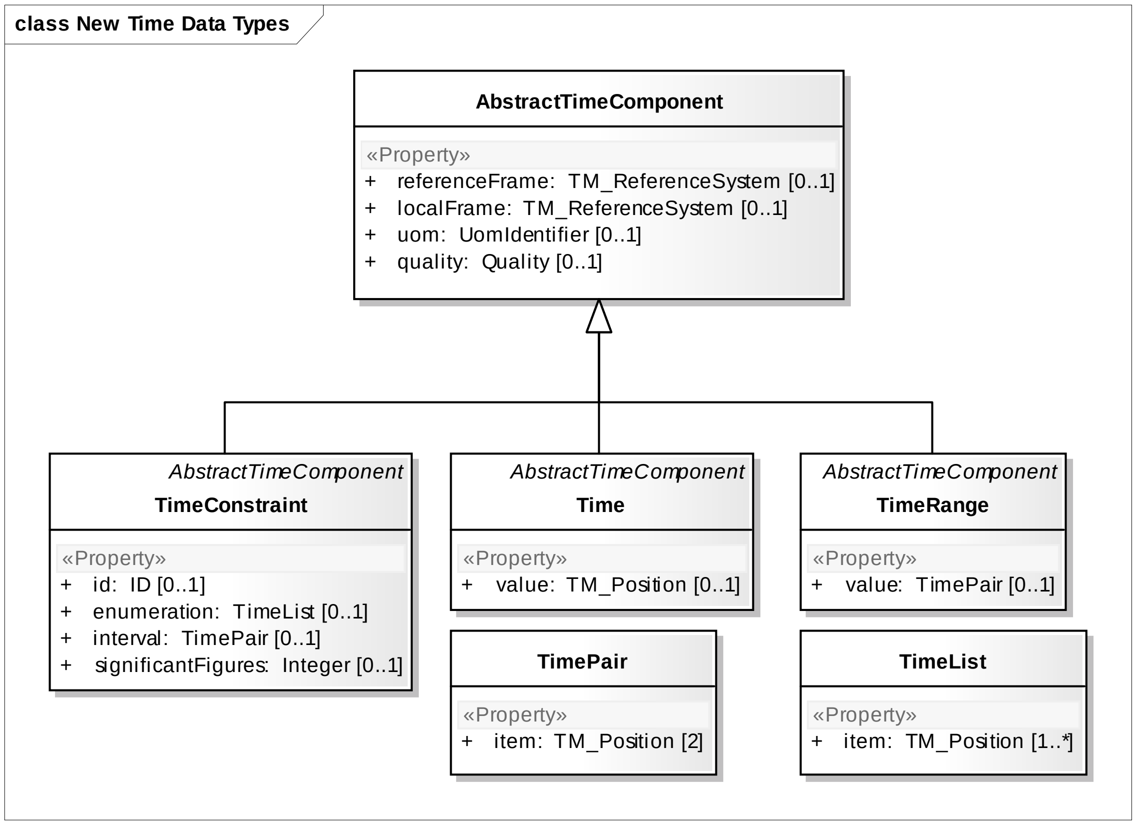
class New Time Data Types
AbstractTimeComponent
«Property»
+ referenceFrame: TM_ReferenceSystem [0..1]
+ localFrame: TM_ReferenceSystem [0..1]
+ uom: UomIdentifier [0..1]
+ quality: Quality [0..1]
AbstractTimeComponent
TimeConstraint
«Property»
+ id: ID [0..1]
+ enumeration: TimeList [0..1]
+ interval: TimePair [0..1]
+ significantFigures: Integer [0..1]
AbstractTimeComponent
Time
«Property»
+ value: TM_Position [0..1]
AbstractTimeComponent
TimeRange
«Property»
+ value: TimePair [0..1]
TimePair
«Property»
+ item: TM_Position [2]
TimeList
«Property»
+ item: TM_Position [1..*]

Figure 7 - Modified temporal data types for SWE Common

6.2.4 Positional data types

The positional data types specified in the UML model of SWE Common (Figure 8) are similar to two of the data types specified in ISO 19107:2003 (Figure 9) and implemented in GML 3.2. However, the SWE Common Position class has a much broader scope than the DirectPosition class of ISO 19107. DirectPosition identifies only spatial or temporal position relative to a coordinate reference system. Position includes an additional capability to identify temporal position in ISO 8601 format, as well as the capability to carry a number of position-related characteristics of an object that occupies a spatio-temporal position. Two elements of Position can be harmonized with GML. First, the data type of the ‘time’ attribute could be changed from Time as specified in SWE Common to TM_Position as specified in ISO 19108. TM_Position does not carry the descriptive attributes that Time inherits from AbstractDataComponent, but, given the fact that Position does carry these attributes, there does not seem to be a need for the value of its ‘time’ attribute to carry them as well. Likewise, the data type of the ‘location’ attribute could be changed from Vector as specified in SWE Common to DirectPosition specified in ISO 19107. DirectPosition, like TM_Position, does not carry the descriptive attributes that Vector inherits from AbstractDataComponent, but, given the fact that Position does carry these attributes, there does not seem to be a need for the value of its ‘location’ attribute to carry them as well.

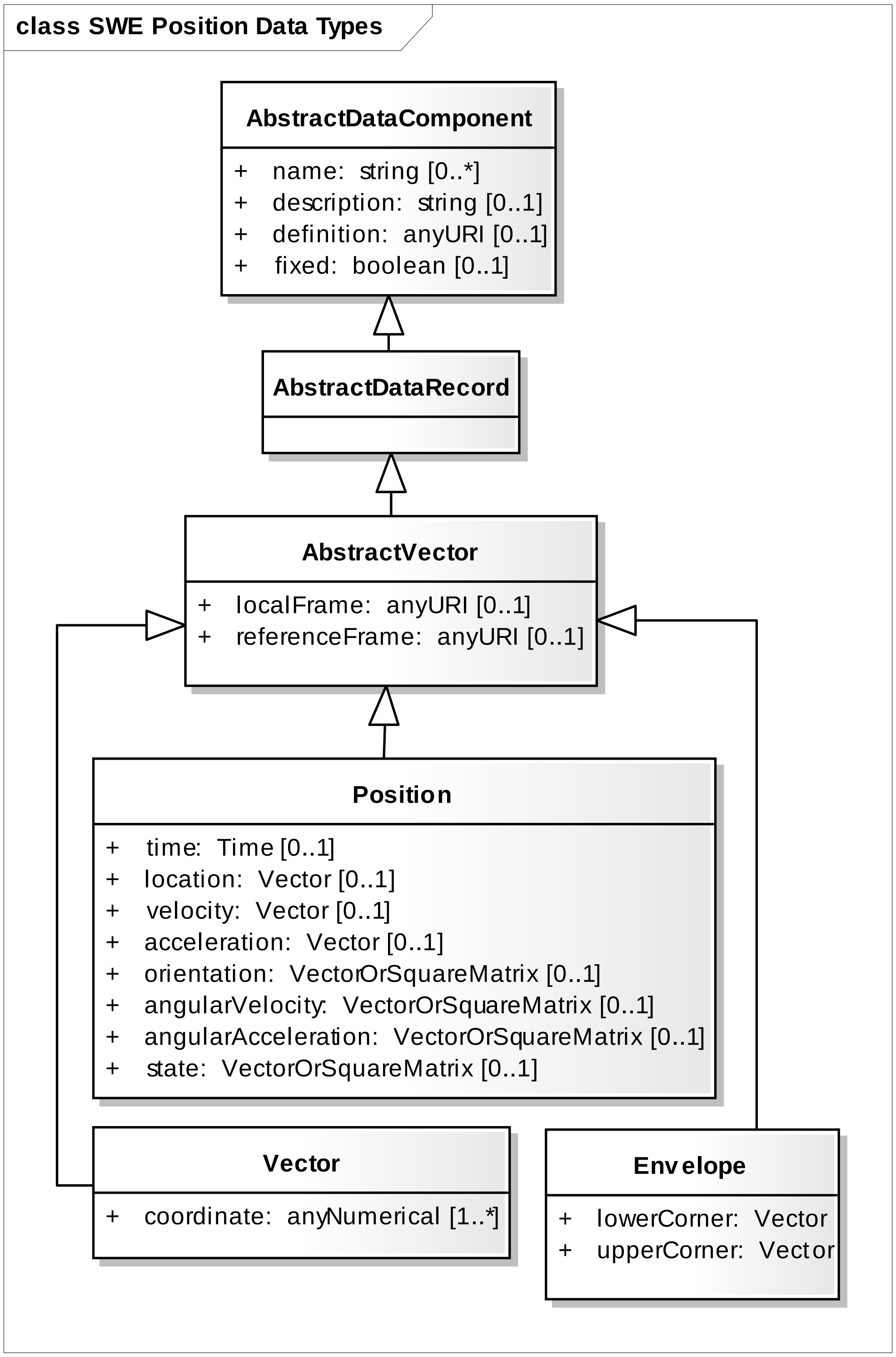
class SWE Position Data Types
AbstractDataComponent
+ name: string [0..*]
+ description: string [0..1]
+ definition: anyURI [0..1]
+ fixed: boolean [0..1]
AbstractDataRecord
AbstractVector
+ localFrame: anyURI [0..1]
+ referenceFrame: anyURI [0..1]
Position
+ time: Time [0..1]
+ location: Vector [0..1]
+ velocity: Vector [0..1]
+ acceleration: Vector [0..1]
+ orientation: VectorOrSquareMatrix [0..1]
+ angularVelocity: VectorOrSquareMatrix [0..1]
+ angularAcceleration: VectorOrSquareMatrix [0..1]
+ state: VectorOrSquareMatrix [0..1]
Vector
+ coordinate: anyNumerical [1..*]
Envelope
+ lowerCorner: Vector
+ upperCorner: Vector

Figure 8 - Positional data types from SWE Common

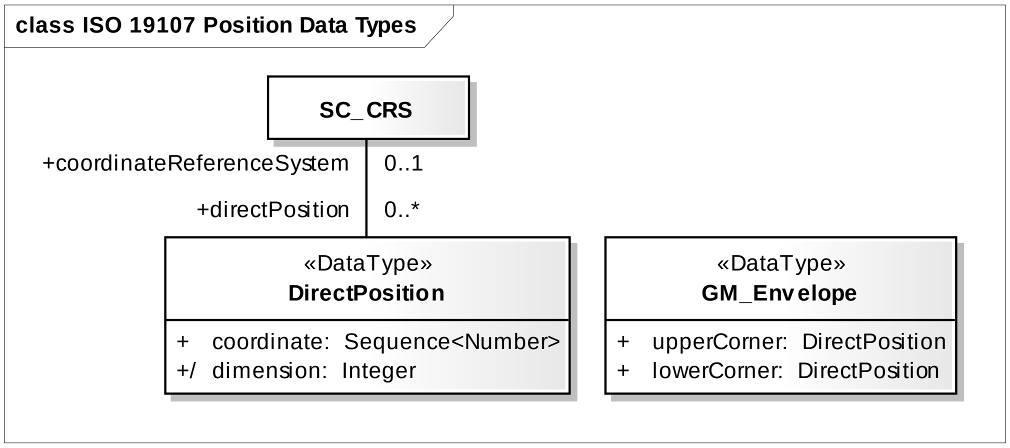
class ISO 19107 Position Data Types
SC_CRS
+coordinateReferenceSystem
0..1
+directPosition
0..*
«DataType»
DirectPosition
+ coordinate: Sequence<Number>
+/ dimension: Integer
«DataType»
GM_Envelope
+ upperCorner: DirectPosition
+ lowerCorner: DirectPosition

Figure 9 - Positional data types from ISO 19107

The Envelope data type specified in SWE Common 1.0 is like GM_Envelope of ISO 19107, except that the data type of its attributes upperCorner and lowerCorner is Vector rather than DirectPosition. It is understood that Envelope has been deleted from the draft of SWE Common 2.0 on the grounds that there is no requirement for it. If a requirement is found in the future, SWE Common should use GM_Envelope, which is implemented by the gml:Envelope specified in GML 3.2.

6.3 Mapping of SWE Common basic data types to ISO/TC211 data types

6.3.1 Introduction

Examination of the basic data types of the attributes specified in the UML model for SWE Common reveals that several of them are conceptual data types drawn from the ISO/TC211 standards and that many of the remainder are mappable or potentially mappable to conceptual data types specified in the ISO TC211 standards. Such mapping is the subject of this section of this report.

The data types under consideration fall into two groups: data types that are identical to conceptual data types from the ISO/TC211 standards and may have been drawn from those standards (6.3.2) and data types from the XML Schema specification that are obvious implementations of ISO/TC211 conceptual data types (6.3.3).

The first column of the tables below contains the names of mappable data types from version 1.0 of the SWE Common specification included in document 07-000. The second column in each row lists the classes and attributes of the UML models that use that data type. Column 3 of these tables identifies the ISO/TC211 conceptual data type to which the SWE Common data type is or could be mapped.

6.3.2 Data types from ISO/TC211 standards

Six of the data types used in the UML model of SWE Common 1.0 are identical in name and description to conceptual data types specified in ISO/TC211 standards (Table 5).

Table 5 - Data types in ISO/TC211 standards

Data Type	Class/Attribute where used	TC211 Source
Any	TypedValue.value	ISO 19103
CharacterString	ConstrainedPhenomenon.otherConstraint PhenomenonSeries.otherConstraint	ISO 19103
ScopedName	TypedValue.property	ISO 19103
TM_Duration	TM_Grid.offset TM_Grid.duration TM_IntervalGrid.windowDuration	ISO 19108
TM_Position	TM_Instant.position	ISO 19108
UomIdentifier	Quantity.uom Time.uom QuantityRange.uom TimeRange.uom	ISO 19136[1]
Notes: 1. UomIdentifier is specified in the GML "Xlinks and basic types" schema, but is not in the UML package "basicTypes."		

6.3.3 Data types from XML Schema

Nine of the data types used in the UML model of SWE Common 1.0 appear to be primitive or derived data types from the XML Schema specification. All of these can be mapped to conceptual data types specified in the ISO/TC211 standards (Table 6).

Table 6 - Data types from XML Schema

Data Type	Class/Attribute where used	TC211 Equivalent
anyURI	AbstractDataComponent.definition SimpleComponentAttributeGroup.referenceFrame TRSAttributeGroup.localFrame TRSAttributeGroup.referenceFrame TRSAttributeGroup.referenceTime AbstractVector.localFrame AbstractVector.referenceFrame Abstractmatrix.referenceFrame AbstractMatrix. localFrame MultiplexedStreamFormat.type Block.encryption Block.compression Component.encryption	ISO 19136 URI?
boolean	AbstractDataComponent.fixed Boolean.value	19103 Boolean
dateTime	Time.value	ISO 19103 DateTime[1]

double	Quantity.value	ISO 19103 Real
ID	AllowedValues.id AllowedTokens.id AllowedTimes.id AbstractEncoding.ID	ISO 19115 MD_Identifier
integer	Count.value abstractDataArray.elementCount TM_GridEnvelope.high TW_GridEnvelope.low CompoundPhenomenon.dimension	ISO 19103 Integer
positiveInteger	DataValue.recordCount BinaryBlock.byteLength Block.byteLength Block.paddingBits-before Block.paddingBites-after Component.significantBits Component.bitLength Component.paddingBits-before Component.paddingBit-after	ISO 19103 Integer
string	AbstractDataComponent.name AbstractdataComponent.description Text.value MultiplexedStreamFormat.version	19103 CharacterString
token	Category.value SimpleComponentAttributeGroup.axisID GeolocationArea.name TextBlock.tokenSeparator TextBlock.tupleSeparator TextBlock.decimalSeparator Block.ref Component.ref	ISO 19103 ScopedName
Notes: 1. Both XML dateTime and ISO 19103 DateTime are based on ISO 8601 representation of Gregorian Calendar & UTC with time zone offsets. This is a string representation, not a number.		

Note that the XML implementation data type ‘string’ is currently used four times in the UML model although its ISO/TC211 conceptual equivalent ‘CharacterString’ is used twice. In all cases, the ISO/TC211 conceptual data types should be used in the UML model rather than the XML implementation data types, which should be used only in the XML schemas.

6.4 Conclusion

This chapter has identified issues involved in the harmonization of GML with SWE Common. It includes recommendations for limited harmonization of GML valueObjects with SWE Common Simple Data Types, for harmonization of Temporal Data Types of the two specifications, and for harmonization of spatial position elements of the two specifications.

7 Using UncertML in SWE

This section explains why UncertML is useful to express error information in sensor data. UncertML is generic, and can be used to describe any form of uncertainty which can be represented within a probabilistic framework. In the sensor context, two sources of error can be distinguished: positional errors (i.e. uncertainty in the location of the sensor) and attribute error (i.e. uncertainty in the measured/sensed value). Positional errors are commonly characterized using probability distributions, via indices such as the Root Mean Squared Error, which summarizes the locational errors observed at a set of sample sites, assuming a symmetrical Gaussian distribution of spatial uncertainty. Attribute errors are also commonly represented in terms of variation around the 'truth', by error bars which represent significance or tolerance limits. Since UncertML represents Gaussian distributions using the standard parameters of mean and variance, it can be straightforwardly employed to convey these familiar error measures. However, it can also be used to represent subtler and more complex forms of error, such as biased, skewed and bounded distributions, and error which varies in space or time, or otherwise across a dataset. These 'non-standard' uncertainty measures may be represented as user-defined distributions, histograms, covariance matrices or even (for stochastic modeling purposes) as a set of raw data samples, or realizations from a hypothetical distribution. Specific uncertainty measures (e.g., 'Attribute value uncertainty at 95 percent significance level') may also be defined by users through data dictionaries, and some examples of these statistics may be seen at http://dictionary.uncertml.org/statistics.As a self-contained and specific language for representing pure numerical uncertainty, UncertML can be combined with other XML schemata which convey spatial, physical or other information about the measurement and the instrument. It is in this context that we envisage UncertML combining with SWE, to expand the current quality element available in SWE.

7.1 Individual observations

When a sensor senses a phenomenon, there will inevitably be a difference between the true value of that phenomenon, and the reported value from that sensor. One reason for this disparity is limited precision in the instrument; for example, a temperature could be reported as 12.1 whereas the real temperature is 12.083151298. The operation of the measurement device itself may influence accuracy, and this effect will not always be consistent – for example, the sensor may have a self-effect, may generate heat over the course of operation or may adapt slowly. Experimental calibration or factory specifications can give detailed information on expected sensor error for the various ranges of the measured phenomenon. This is highly valuable information, especially when we consider how errors propagate as sensor data is combined and further processed for decision-making applications., Whether this information on sensor error is reported routinely, or supplied only on requested, it should be as rich and representative as possible..

For many simple devices, (e.g. temperature sensors) errors may be relatively small and the need to quantify these errors does not seem compelling unless the context calls for great precision. However, when we extend the interpretation of what "sensed data" is to highly-manipulated datasets, the errors can become substantial. Satellite imagery, for example, may be recalibrated and classified to generate estimated coverages showing total column NO2 or soil moisture. Similarly, complex models may be applied to spatially-distributed networks of sensor measurements, combining them with the sensor data such as meteorological sources, to generate, for example, estimates of PM10 or interpolated gamma dose rates. In these contexts, input sensor error must be tackled and communicated openly and clearly, in order to gauge the reliability of the outputs and make properly-informed decisions.

7.1.1 Full description of uncertainty

Ideally, it would be possible to characterize the uncertainty of a sensor's measurements fully by specifying a pattern with strict numerical parameters: for example: "the mean value is 12.1, the standard deviation is 1, and the error distribution is normal". If we assume that the fit of the observed uncertainty to a normal distribution was indeed close, then this information enables many useful predictions to be made; for example, 'what is the probability that, given a certain real value, the value of the sensor measurement will exceed a certain threshold, or fall within a certain range?' More importantly, we can begin to address issues such as sensitivity and specificity, for example by considering the likelihood of false negatives (a radiation sensor reports a tolerable value, but the real value exceeds the danger level). Commonly-used distributions (e.g. Poisson and Gaussian) are embedded within UncertML via a dictionary which defines their parameters and their formulae. For example, a normal (Gaussian) distribution is defined by two parameters: mean and variance – knowing these values and the nature of the distribution allows the sort of inference described above about the meaning of observed values. The normal distribution is commonly used to represent uncertainty in environmental data, since its symmetrical shape approximates many observed patterns, and non-normal data can often be easily transformed to satisfactorily fit a normal distribution. For this reason, it is one of the distributions which is already defined in the UncertML dictionary. However, uncertainty in many measured values (rainfall, for example) will be bounded, non-normal or non-negative. UncertML therefore has been designed to be extensible in that it allows users to characterize their own distributions, either by creating a dictionary entry which defined the mathematical characteristics of the distribution, or by using a histogram to represent its specific shape.

7.1.2 Description by sampling

In many contexts, one is not certain how exactly error is distributed, but one is able to provide a sample from it, e.g. by giving 100 alternative values for a particular point in space and/or time. These 100 numbers might be real samples, or could be realizations output from a Monte Carlo experiment, e.g. ensemble weather forecasts. While this description is not complete, if the sample is large enough, one can derive valuable information experimentally about the underlying pattern. Even for small samples, the range is of interest in addition to the mean value. UncertML has elements specifically designed to store and represent such sets of samples or realizations, and as with all UncertML types, these can be embedded within other XML objects which convey full information about the spatial, temporal and physical context.

7.2 Collections of observations and dependent errors

7.2.1 Partial description by a number of samples

As mentioned in 7.1.2, the output from a large MC experiment could provide spatially-distributed sets of samples of the field “sensed”. From an analysis of these samples, it should be possible to identify spatial structure, autocorrelation and dependency in the errors observed, and to make useful predictions about the likely error in novel locations.

7.2.2 Full description by a multivariate Gaussian distribution

One multivariate distribution that can easily be parameterized is the Gaussian distribution; its parameters are the mean vector (mean values for all sensor locations) and covariance matrix. The covariance matrix is the matrix with all variances of sensors and all covariances of pairs of sensors. If the number of sensors is large (1000) then the number of covariances is equal to the number of pairs (roughly 1000 x 1000/2). Storing this matrix may become prohibitive for problems of a certain size, but UncertML allows you to do so.

7.3 Positional error: a GPS example

Listing 1 shows one conceivable example using UncertML for positional errors is the description of a position measured by a GPS Sensor.

Listing 1 - Exemplary use of UncertML in O&M

```
<?xml version="1.0" encoding="UTF-8"?>
<om:ObservationCollection
xmlns:om="http://www.opengis.net/om/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:un="http://www.uncertml.org"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:sa="http://www.opengis.net/sampling/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/om/1.0
http://schemas.opengis.net/om/1.0.0/observation.xsd
http://www.uncertml.org
http://schemas.uncertml.org/1.0.0/UncertML.xsd
http://www.opengis.net/sampling/1.0
http://schemas.opengis.net/sampling/1.0.0/sampling.xsd">
   <om:member>
      <om:Observation gml:id="OBSERVATION1">
         <om:samplingTime/>
         <om:procedure/>
         <om:resultQuality>
            <un:Statistic definition ="
http://dictionary.uncertml.org/statistics/covariance_matrix">
               <un:value>9.2 2.7 2.7 9.2</un:value>
            </un:Statistic >
         </om:resultQuality>
         <om:observedProperty><!--...--></om:observedProperty>
         <om:featureOfInterest><!--...--></om:featureOfInterest>
         <om:result>
            <gml:Point>
               <gml:pos>51.9408968 7.6095461</gml:pos>
            </gml:Point>
         </om:result>
      </om:Observation>
   </om:member>
   <om:member>
      <om:Observation gml:id="OBSERVATION2">
         <om:samplingTime/>
         <om:procedure/>
         <om:resultQuality>
            <un:Statistic definition ="
http://dictionary.uncertml.org/statistics/covariance_matrix">
               <un:value >9.4 2.8 2.8 9.4</un:value>
            </un:Statistic >
         </om:resultQuality>
         <om:observedProperty><!--...--></om:observedProperty>
         <om:featureOfInterest><!--...--></om:featureOfInterest>
         <om:result>
            <gml:Point>
               <gml:pos>51.9408968 7.6095461</gml:pos>
            </gml:Point>
         </om:result>
      </om:Observation>
   </om:member>
</om:ObservationCollection>
```

7.4 Integration in the SWE information models

UncertML concentrates on how to encode and implement uncertain information into existing standards. It does not instruct on how to encode supporting information such as units of measure and spatial domains. In geospatial contexts, it is anticipated that UncertML may be used in a 3-tier architecture as demonstrated in Figure 10, with each supporting layer in this architecture adding an extra level of detail. Delegating appropriate responsibilities to supporting schemata decouples UncertML from any one existing standard and ensures that it may be integrated into a wide range of domains.

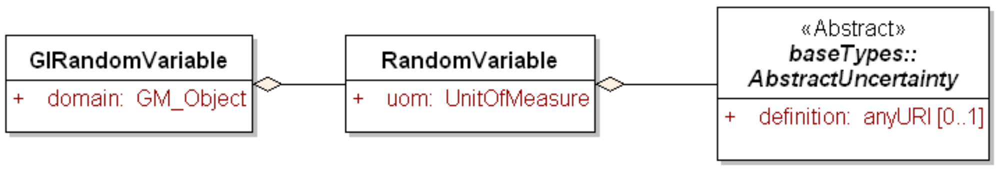

Figure 10 - UncertML in a 3-tier architecture

UncertML does not provide a mechanism for describing units of measure, geospatial domains or any other such properties. The removal of such constraints allows UncertML to be utilized in a variety of different domains.

Given the wide applicability of UncertML, this tiered arrangement may consist of as many levels as are necessary to clearly characterize a measurement, using the most appropriate schemata for that problem domain. Sections 7.4.1- 7.4.2 discuss how UncertML may be integrated into existing SWE standards.

7.4.1 Integration into SWE Common & SensorML

The SWE Common standard provides a neat framework for describing primitive data types, aggregations of these data types and related semantics. Each of the 'simple data types' within SWE Common (Quantity, Count, Boolean etc) contain a property for attaching quality information. However, there is no defined mechanism for quantifying the quality. With a simple extension it would be possible to allow UncertML to reside within the SWE Common quality property.

Listing 2 – UncertML used within the quality property of the SWE Common simple data type "Quantity"

```
<swe:Quantity definition="urn:ogc:def:phenomenon:Temperature">
   <swe:uom>Cel</swe:uom>
   <swe:quality>
      <un:Statistic definition="standard_deviation">
         <un:value>3.4</un:value>
      </un:Statistic>
   </swe:quality>
   <swe:value>12.6</swe:value>
</swe:Quantity>
```

The example in Listing 2 describes a temperature quantity of 12.6 degrees Celsius with a standard deviation of 3.4. While this typical use case of UncertML within SWE Common provides some description of uncertainty, a more complete description is often necessary. In such circumstances it is useful to regard quantities as 'random', i.e. the value is not known with certainty. Extending SWE Common to allow for the addition of a 'RandomQuantity' whose value property is any UncertML type allows a variable to be described completely by its parametric distribution (Listing 3).

Listing 3 - Extending SWE Common to add a RandomQuantity type whose value is any UncertML

```
<swe:RandomQuantity
definition="urn:ogc:def:phenomenon:Temperature">
   <swe:uom>Cel</swe:uom>
   <swe:value>
      <un:Distribution definition="gaussian_distribution">
         <un:parameters>
            <un:Parameter definition="mean">
               <un:value>12.6</un:value>
            </un:Parameter>
            <un:Parameter definition="variance">
               <un:value>11.56</un:value>
            </un:Parameter>
         </un:parameters>
      </un:Distribution>
   </swe:value>
</swe:RandomQuantity>
```

SensorML uses the SWE Common data types as inputs and outputs of process methods. Extending SWE Common to allow UncertML to be integrated, automatically propagates the benefits to SensorML.

7.4.2 Integration into O&M

Listing 4 demonstrates how to encode a sensor noise model in an Observations & Measurements (O&M) document. The ‘sa’ and ‘gml’ namespaces are used, in keeping with common practice, to encode the results of a measurement sampled by the sensor in question, and the point location of the sensor.

Listing 4 - UncertML used to encode a noise model for a particular sensor in conjunction with the O&M standard

```
<om:Observation>
   <om:samplingTime>
      <gml:TimeInstant>
         <gml:timePosition>2008-07-07T13:59</gml:timePosition>
      </gml:TimeInstant>
   </om:samplingTime>
   <om:procedure
xlink:href="http://www.mydomain.com/sensor_models/temperature"/>
   <om:resultQuality>
      <un:Distribution
definition="http://dictionary.uncertml.org/distributions/gaussian
">
         <un:parameters>
            <un:Parameter
definition="http://dictionary.uncertml.org/distributions/gaussian
/parameters/mean">
               <un:value>0.0</un:value>
            </un:Parameter>
            <un:Parameter
definition="http://dictionary.uncertml.org/distributions/gaussian
/parameters/variance">
               <un:value>3.6</un:value>
            </un:Parameter>
         </un:parameters>
      </un:Distribution>
   </om:resultQuality>
   <om:observedProperty xlink:href="urn:x-
ogc:def:phenomenon:OGC:AirTemperature"/>
   <om:featureOfInterest>
      <sa:SamplingPoint>
         <sa:sampledFeature
xlink:href="http://www.mydomain.com/sampling_stations/ws-04231"/>
         <sa:position>
            <gml:Point>
               <gml:pos srsName="urn:ogc:def:crs:EPSG:4326">
                  -1.89538836479 52.4773635864
               </gml:pos>
            </gml:Point>
         </sa:position>
      </sa:SamplingPoint>
   </om:featureOfInterest>
   <om:result xsi:type="gml:MeasureType"
uom="urn:ogc:def:uom:OGC:degC">19.4</om:result>
</om:Observation>
```

The result property contains a temperature measurement of 19.4° Celsius. However, the resultQuality property indicates (through the use of an encapsulated UncertML Distribution) that this temperature has an associated variance of 3.6 around the measured value with no bias, as indicated by the mean value of 0.0. An assumption is made that the units of measure within UncertML are the same as those specified within the result property (degrees Celsius). The Observations & Measurements schema accommodates any XML type inside the resultQuality-property, therefore, no extensions to the standard are necessary to allow the use of UncertML.

7.5 UncertML and MathML

UncertML seeks to provide a simple mechanism for describing complex concepts such as parametric distributions. In order to achieve this goal a decision was made to exclude any mathematical functions from within the uncertainty data types (Statistic, Distribution etc). Instead these mathematical functions are described within the UncertML dictionary using presentation MathML (Figure 11). It can be argued that allowing functions to be described in-line, using content MathML, could act as an interoperable mechanism for automatically exchanging any distribution or statistic. However, in a context where users are working with previously unknown distributions or statistics, it would be necessary to spend time developing the required mathematics for processing, negating the need for content MathML. Delegating the description of these functions into an accompanying dictionary is in line with the vision of the ISO/IEC guide to the expression of uncertainty in measurement (GUM), in which they facilitate the need for a worldwide consensus on the evaluation and expression of uncertainty in measurement, not dissimilar to the International System of Units.

Figure 11 - Example of presentation MathML within the UncertML dictionary

8 Integration of other Markup Languages in SensorML

This chapter discusses how different markup languages can be integrated into SensorML for the description of sensor process methods. In SensorML every sensor is described as a process. These processes are divided in physical and non-physical processes where the former have a relation to space and time.

All of these process descriptions contain of parts for the description of process inputs, outputs, parameters and methods. The method part is where different markup languages can be integrated and is encoded as a ProcessMethodType (see Figure 12). Besides standard GML attributes and metadata it contains sections for rules, algorithms and implementations.

The rules for a method can be encoded using RELAX NG or Schematron. They can for instance be utilized to constrain the inputs of a process to a specific set. The implementation section points to implementations of the given process. This may be a SensorML process chain or program code in source or binary form. The algorithm description of a process may contain a MathML document as a link or embedded. All of these sections may contain a textual description.

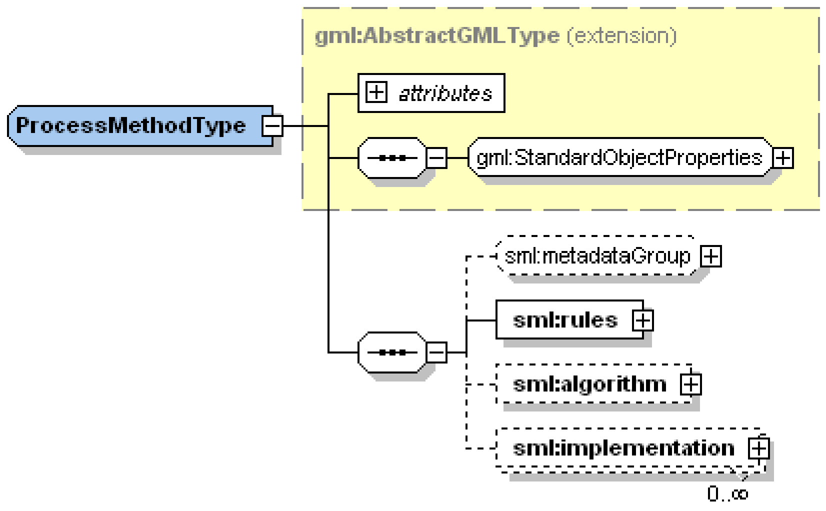

Figure 12 - SensorML ProcessMethodType

8.1 Integration of MathML

The Mathematical Markup Language (MathML) is an XML application developed by the World Wide Web Consortium (W3C). It is available in the version 2.0 (Second Edition) whereas version 3.0 is under progress. MathML can be used to describe mathematical terms and enables the visualization, exchange and automated execution.

The specification describes two markups which can be used separate or in a combined form. The first one is the presentation markup (see Listing 5). It is used to “describe the layout structure of mathematical notation” [W3C 2003]. Therefore about 30 possible XML elements are defined. The semantics of these elements are defined for rendering and visualization of mathematical terms. If the presentation markup is used it is therefore difficult to interpret and process them. The following listing gives an example of the following formula in presentation markup:

$$Tc = (Tf - 32)/1.8 \qquad \text{(Transformation from °F to °C)}$$

Listing 5 - MathML Presentation Markup example

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
   <mrow>
      <msub>
         <mi>T</mi>
         <mi>C</mi>
      </msub>
      <mo stretchy="false">=</mo>
      <mfrac>
         <mrow>
            <msub>
               <mi>T</mi>
               <mi>F</mi>
            </msub>
            <mo stretchy="false">-</mo>
            <mn>32</mn>
         </mrow>
         <mn>1.8</mn>
      </mfrac>
   </mrow>
</math>
```

The content markup of MathML is used to describe the “underlying structure” of a mathematical expression. It provides an encoding with clearly defined semantics for each of the about 120 elements. Hence automated interpreting and processing is easier and less error-prone. The drawback of this is that not every term can be represented.

The following areas are supported to some degree [W3C 2003]:

– Arithmetic, algebra, logic and relations

– Calculus and vector calculus

– Set theory

– Sequences and series

– Elementary classical functions

– Statistics

– Linear algebra

8.1.1 MathML in SensorML

In a SensorML sensor (or process) description MathML can be used to describe the mathematic background. It can be used to give information about a calculation that takes place inside of a sensor (for instance mapping of a measured voltage to a temperature value) or to define further processing instructions that can be applied when needed. In the latter it is necessary, that the provided mathematics can be interpreted and executed automatically. Therefore only the content markup of MathML is allowed for integration in SensorML [OGC 07-000].

Figure 13 shows the element that shall be used to integrate MathML documents in SensorML process descriptions:

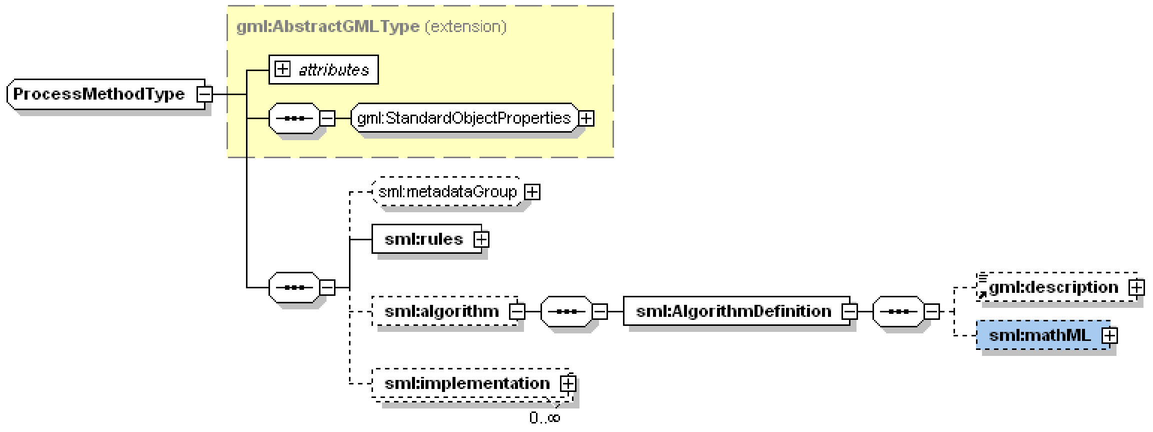

Figure 13 - MathML element in SensorML process methods

Besides the embedded MathML document a processor has to know how to map the inputs, outputs and parameters of the SensorML process to the mathematical expression. Input, output and parameter descriptions in SensorML provide a name field which shall be used for the mapping (see Figure 14). This name shall also be used for the variables in the MathML document that are related to process inputs or parameters.

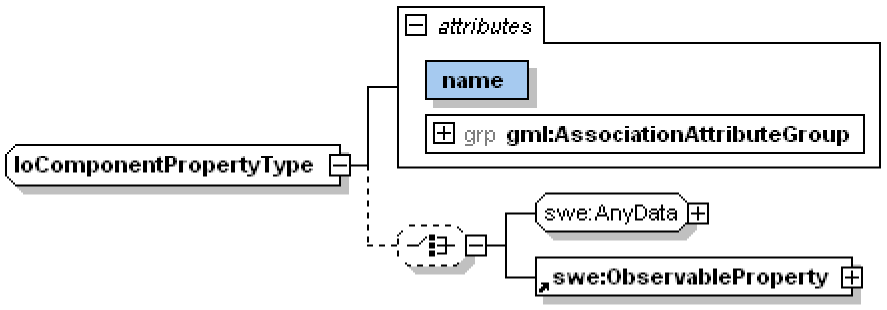

Figure 14 - Name field in SensorML input descriptions

In order to use the names of the process outputs as well for the connection between SensorML and MathML a function has to be defined for every output. Such an assignment of a value (or calculation result) to a variable cannot be done easily with MathML because it aims on encoding mathematic terms and not on algorithms using mathematics. The *equals*-operator for instance is only used for comparison.

The definition of functions in MathML is done by the use of the declare- and lambda-operators. The first parameter of the declare-operator is the name of the function to declare. In the case of the MathML integration in SensorML this could be the name of the process output. The second declare parameter is a lambda operator defining the function itself.

The lambda part takes two groups of parameters. At first all variables (for instance names of the process inputs and parameters) are listed following by an apply block describing the calculation. Listing 6 shows the MathML content markup version of Listing 5:

Listing 6 - MathML Content Markup example

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
   <declare type="function">
      <ci>Tc</ci>
      <lambda>
         <bvar><ci>Tf</ci></bvar>
         <apply>
            <divide/>
            <apply>
               <minus/>
               <ci>Tf</ci>
               <cn>32</cn>
            </apply>
            <cn>1.8</cn>
         </apply>
      </lambda>
   </declare>
</math>
```

In detail the MathML document consists of a single *math*-block. Inside of it a function is declared (`<declare type="function">`). The output operator is defined as the first parameter (`<ci>Tc</ci>`) followed by a *lambda* block. This block at first lists all input variables (here only `<bvar><ci>Tf</ci></bvar>` for the temperature measured in °Fahrenheit) and an *apply* block defining how the results are calculated.

A disadvantage of this solution is that a declaration does not mean the same as an assignment. Once a declaration is made it is valid forever (in its declaration scope), while values could be reassigned. This means that it is possible to emulate declarations via assignments, but not the other way around.

The way described above is valid for version 2.0 of MathML (October 21st, 2003). In the meantime there is work on version 3.0 ongoing and available as a draft specification (dated on November 17th, 2008). In terms of the integration of MathML in SensorML version 3.0 contains one major change: the *declare* element is deprecated. Because of that it should not be used or introduced as standard practice.

To obtain this another possible way to encode assignments in MathML has to be used. This is to define a new assignment operator via the *csymbol* element which is available in version 2.0 as well as 3.0 of MathML [W3C 2003, W3C 2008]. Listing 7 gives an example how the previous formula can be encoded.

Listing 7 - MathML example using the csymbol element

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
   <apply>
      <csymbol definitionURL=”TBD”>assign</csymbol>
      <ci>Tc</ci>
      <apply>
         <divide/>
         <apply>
            <minus/>
            <ci>Tf</ci>
            <cn>32</cn>
         </apply>
         <cn>1.8</cn>
      </apply>
   </apply>
</math>
```

Note that the *csymbol* element contains a *definitionURL* attribute that is used to point to a definition of the newly defined operator. This URL could point to a public available version of this document or any other public available document containing the following section (8.1.2).

8.1.2 Definition of an assignment operator for MathML

The operator is encoded using the *csymbol* element of MathML. The value of the *definitionURL* attribute has to point to a public available document containing this definition. The value of the *csymbol* element itself is “assign”.

The first following element shall be a *ci* element defining the variable where a value is assigned to.

The second following element shall be an *apply* element containing a mathematical expression that is used to calculate a result value that is assigned to the prior given variable.

8.1.3 Execution of MathML

The execution of MathML is a large task. There are many operators with multiple parameters to interpret. Although there are programs and program libraries to solve MathML expressions, not all of them cover the full extent. Therefore a service that is capable of executing SensorML processes with embedded MathML should provide a kind of MathML capabilities. These would announce the supported operators rather like the filter capabilities of the OGC filter encoding [OGC 04-095].

8.1.4 Comparison of MathML and other options for process descriptions

There are several different ways to describe and define the method that is performed by a process. In this clause some of them are discussed and compared with a focus on the representation of mathematical expressions.

In addition to MathML there are other possibilities given in SensorML. Thus, process method descriptions may contain textual descriptions, SensorML process chains, source, or compiled binary code and rules described using RELAX NG or Schematron. Furthermore one could think of including mathematical expressions as a simple character string or to utilize other (markup) languages like Formula 3 used in the S@ny project[1] or EML (see clause 8.2).

Textual descriptions can be the easiest way to provide information to a human reader if they are well written. They do not provide useful information to machines because it is nearly impossible to parse their content.

SensorML process chains can also be used to describe process methods. But they again contain process methods that finally rely on other techniques to define mathematical expressions or algorithms.

Compiled binary code can be used to provide implementations of the process method that can be read and executed by a machine. Human beings are in most cases not able to read and understand binary code. If the implementation is provided as source code it is at least possible for experts to read and understand it. If the code cannot be executed by an interpreter it has to be compiled before execution which can be too complex for an application. In both cases it is necessary that the framework for the execution is known to the developer in advance.

RELAX NG and Schematron can be included into SensorML process methods to define restrictions to the process instead of describing the method itself. They can for instance be used in addition to compiled binary code to verify that the inputs and outputs of the process are correctly described for the given implementation.

Using character strings one can define mathematical expressions in a similar manner as one does it using a pen and paper. Simple expressions can be easily written and read by humans and they are also readable and executable by machines. When describing more complex expressions, this can become more difficult for various reasons. First it is difficult to write constructs in one row that usually take multiple rows like matrices. This is the same for operators with a complex representation like the sigma sign. It is possible to encode these constructs in character strings but it results in a worse readable representation. Also the semantics of all allowed constructs have to be clearly defined. But due to cultural differences in the representation of mathematical this also comes with a reduced readability at least for members of some cultures. Furthermore some abbreviations in common use with mathematical expressions can cause problems such as

[1] http://www.sany-ip.eu/

juxtaposition in multiplication. In addition a string based mathematical expression can in SensorML only be integrated as a textual description without a kind of a schema definition or other rule definition.

Formula 3 is more a time sequence algebra than an encoding for mathematical expressions. Though it is possible to encode such expressions via Formula 3 it comes with a lot overhead for this purpose. It is recommended to use mainly in a time sequence domain.

The content markup of MathML as introduced earlier can be used to encode a large set of mathematical expressions. The semantics are clear for each such expression and it is possible to read and execute them by machines. It is also human readable to some degree. This can be improved when using a renderer software and if necessary a combination with the presentation markup of MathML. In this case the cultural context of the rendering of mathematical expressions could also be respected. The major disadvantage of MathML is that it can be complex even for simple expressions.

It is recommended to use the content markup of MathML or compiled binary code if the expressions have to be executed. Use MathML if possible to describe the process methods that should be read or may be reused complete or in parts. Here the focus is on clearly defined process methods. Use compiled code for problems that cannot be solved with MathML or where MathML is too complex. Add precise textual descriptions in these cases. Here the focus is on easy executable process methods using a known framework. Special languages like Formula 3 or EML can be used for special purposes like calculations on time sequences or complex event processing.

If the focus is on readability MathML, string based expressions (if unambiguous), textual descriptions or a combination of them can be used. Meaningful textual descriptions should always be used.

8.2 Integration of EML

Besides MathML there can be use of integrating other markup languages for the description of process methods. One of them is the Event Pattern Markup Language (EML). It is used to define patterns to perform Complex Event Processing (CEP) and Event Stream Processing (ESP). These techniques can be useful when dealing with large amounts of input data to detect patterns and derive information of higher value.

The main obstacle for embedding EML patterns into SensorML is that there is no element where to put it right now. One solution could be to introduce a new element to SensorML. On the other hand there is an element for MathML descriptions of process methods. This element accepts any content so it can be used for EML as well. In order to prevent confusion with EML patterns found in the MathML element it should be renamed. This element could then be used to embed process method description in whatever (markup) language is most suitable. There also would not be the need to add a new element for further markup languages. This matter is committed in a change request to the SensorML standards working group (document number OGC 08-192r1).

To map inputs and outputs to EML patterns the names can be utilized just like when using MathML. Dynamic parameters are not supported in the current version of the EML and should therefore not be used. Another possibility is to handle parameters as additional inputs to the process. Methods to access properties of data collections in EML processes are described in the EML specification. [OGC 08-132]