Open GIS Consortium Inc.

Date: 2002-04-22

Reference number of this OpenGIS© Project Document: **OGC 02-026r1**

Version: 0.0.4d

Category: OpenGIS© OGC Interoperability Program Report

Editor: Mike Botts
University of Alabama in Huntsville

Sensor Model Language (SensorML) for In-situ and Remote Sensors

Document type: OpenGIS© Interoperability Program Report
Document subtype: Engineering Specification
Document stage: Report
Document language: English

Table of Contents

Preface

Submitting organizations

This Interoperability Program Report is being submitted to the OGC by the following organizations:

CSIRO Australia
Galdos Systems, Inc
University of Alabama in Huntsville

Submission contact point

All questions regarding this submission should be directed to the editor or the submitters:

CONTACT	COMPANY	ADDRESS	PHONE/FAX	EMAIL
Mike Botts	University of Alabama in Huntsville	ESSC / NSSTC Hunstville, AL 35899	+01-256-961-7760	mike.botts@nsstc.uah.edu
Ron Lake	Galdos Systems, Inc.	Suite 200, 115 West Pender Street, Vancouver, B.C. V6E 2P4	+01-604-484-2750	rlake@galdosinc.com
Simon Cox	CSIRO, Australia	PO Box 1130, Bentley WA 6102 Australia	+61-8-6436-8639	simon.cox@csiro.au

Revision history

Date	Release	Author	Section modified	Description
2001-07-16	001_001	meb		Original pre-OGC SensorML document
2002-04-04	0.04	meb		First “complete” DIPR version.
2002-04-09	0.04b	meb	throughout	Minor typo corrections. Incorporated edits from Carl Reed.
2002-04-09	0.04b	meb	2.2.2	Changed *ObservablePropertyType* and *ObservableProperty* to *ObservedPropertyType* and *ObservedProperty* to match the schema terms. Changed UML to match.
2002-04-16	0.04b	meb	throughout	Added suggested additions from Stefan Falke
2002-04-18	0.04a	meb	throughout	Completed incomplete sections, addd issue boxes, added references, added Annex A with excerpts from geometry sections of original SensorML design

2002-04-19	02-026 (0.0.4c)	HAN	throughout	Final OWS-1 review and edit; minor changes. Produced OGC 02-026.
2002-04-22	02-026 (0.04d)	meb	throughout	Updated Table of Contents, fixed pages where figure caption had been separated from figure; moved part 3 to Annex A (normative) and Annex A to Annex B (informative)

Recommended Changes to the OpenGIS Abstract Specification

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

Foreword

Attention is drawn to the possibility that some of the elements of this part of OGC 02-026 may be the subject of patent rights. The Open GIS Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

This specification was developed under the OWS 1.1 initiative.

Part 1. Introduction

1.1. Scope

For the purpose of this discussion, SensorML is considered as one key component of the Sensor Web Enablement. With regard to observations and measurements, we assume three fundamental components of information:

1. There are properties of physical entities and phenomena that are capable of being measured and quantified. Within the OGC Open Web Services context, properties that are capable of being measured are considered as "Observables". Each of these can be classified as an "ObservableType" and can be referenced in an "ObservablesDictionary". ObservableType definitions include, for example, properties such as temperature, count, rock type, chemical concentration, or radiation emissivity.
2. There are sensors that are capable of observing and measuring particular properties. Either by design or as a result of operational conditions, these sensors have particular response characteristics that can be used to determine the values of the measurements, as well as assess the quality of these measurements. In addition to the response characteristics, the sensor system has properties of location and orientation that allow one to associate the measured values with a particular geospatial location at a particular time. The role of the SensorML is to provide characteristics required for processing, georegistering, and assessing the quality of measurements from sensor systems.
3. Finally, there are data values that are returned by a sensor system or are derived from sensor measurements. These measurements may be accessed directly from the sensor, or from data stores that distribute and possibly process these data to various products. The processing and georegistration of these measured values require knowledge of the properties of the sensor system. Within the context of the OGC Sensor Web Enablement initiative, values returned by sensors are accounted for within the Observations and Measurements schemas.

All three of these components are intimately linked within the Sensor Web Enablement concepts. While these links will be discussed within this document, only the second component is within the scope of this document.

SensorML provides an XML schema for defining the geometric, dynamic, and observational characteristics of a sensor. Sensors are devices for the measurement of physical quantities. There are a great variety of sensor types from simple visual thermometers to complex electron microscopes and earth observing satellites.

The purpose of SensorML is to:

- provide general sensor information in support of data discovery
- support the processing and analysis of the sensor measurements
- support the geolocation of the measured data.
- provide performance characteristics (e.g. accuracy, threshold, etc.)
- archive fundamental properties and assumptions regarding sensor

To this end, the information provided by SensorML includes:

- **Observation characteristics**
  - Physical properties measured (e.g. radiometry, temperature, concentration, etc.)
  - Quality characteristics (e.g. accuracy, precision)
  - Response characteristics (e.g. spectral curve, temporal response, etc.)
- **Geometry Characteristics**
  - Size, shape, spatial weight function (e.g. point spread function) of individual samples
  - Geometric and temporal characteristics of sensor and sample collections (e.g. scans or arrays) that are required for metric exploitation
- **Description and Documentation**
  - Overall information about the sensor
  - History and reference information supporting the SensorML document

The use of such a sensor model arises in several possible ways, including:

- The sensor description (model) is used to process the raw values obtained by the sensor element into a useful physical quantity and to provide support for georegistration of remotely sensed data.
- The sensor internally performs the "conversion" to physical quantities. The sensor description is used to construct the conversion equations. In some cases the sensor description might be used "post facto" to examine the data obtained from a sensor when it is behaving in an apparently "unreasonable" manner, or to assess inherent limits in the sensor's sensitivity and quality of measurements.
- The sensor does some internal processing, but additional external processing is required to take account of things that the "sensor" element and description may not be not aware of. The latter can include the measurement of other quantities, as well as the position and orientation of the sensor itself.

SensorML does not provide a detailed description of the hardware design of a sensor but rather it is a general schema for describing a functional model of the sensor. The schema is designed such that it can be used to support the processing and geolocation of data from virtually any sensor, whether mobile or dynamic, in-situ or remotely sensed, or active or passive. This allows one to develop general, yet robust, software that can process and geolocate data from a wide variety of sensors ranging from simple to complex sensor systems.

For the purpose of this discussion, we separate the description of the sensor from that of its platform. In cases where confusion is likely we will uses the term "sensor system" to refer to the sensor and its associated platform(s). The platform is the carrier for the mounted sensor and is of major concern for mobile sensors. Common platforms include ground stations, automobiles, aircraft, earth-orbiting satellites, ocean buoys, ships, and people. A deployed sensor is mounted to a static or dynamic platform (or an assembly of nested platforms).

The detailed description of the mechanical structure of the platform(s) is outside the scope of the sensor description. From our perspective, the main role of the platform is to define the (possibly dynamic) coordinate system(s) in which the sensor measurements are taken, so that these measurements can be related to some relevant external coordinate system. Thus, it is only through the association of a sensor with its platform(s), that measured values can be georegistered.

While the description of the platform is not a part of the sensor description, per se, it is of vital importance to our ability to georegister sensor measurements. Thus, an initial simple schema for defining platforms is provided in this document.

As will be discussed further, SensorML is suitable for both in-situ and remote sensor, whether mounted to static or mobile platform. The original SensorML, developed before any involvement of OGC, focused primarily on defining the geometric and dynamic properties of remote sensors. In contrast, Phase 1 of the OGC OWS activities have focused the SensorML development primarily on in-situ sensors. Because of this focus, and because of the intent to redesign the geometry and dynamics description in the next release of SensorML, this release of the document will not cover the geometric and dynamic properties of the sensor. However, the concepts surrounding sample geometry and sample collection geometry (e.g. scan properties) will be discussed briefly in the concepts section. Furthermore, "placeholders" for these properties will be left in the schema design sections, but their description is reserved for future document releases.

1.2. Conformance

Conformance and Interoperability Testing for this OGC Interoperability Program Report may be checked using all the relevant tests specified in Annex A (normative). The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in ISO 19105: Geographic information — Conformance and Testing.

1.3. Normative references

The following normative documents contain provisions, which through reference in this text, constitute provisions of this part of OGC 02-026. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of OGC 02-026 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

1.4. Terms and definitions

For the purposes of this document, the following terms and definitions apply /the terms and definitions given in … and the following apply.

Observable
A phenomenon to that can be observed and measured, such as temperature, gravity, chemical concentration, orientation, number-of-individuals. The equivalent term is **measurand** when the values are determined by measurement.

Observed Value
A value describing a natural phenomenon, which may use one of a variety of scales including nominal, ordinal, ratio and interval. The term is used regardless of whether the value is due to an instrumental observation, a subjective assignment or some other method of estimation or assignment.

Sensor
An entity capable of observing a phenomenon and returning an observed value. A sensor can be an instrument or a living organism (e.g. a person), but herein we concern ourselves primarily with modeling instruments, not people.

Measurement
An instance of a procedure to estimate of the value of a natural phenomenon, typically involving an instrument or sensor. This is implemented as a dynamic feature type, which has a property containing the result of the measurement. The measurement feature also has a location, time, and reference to the method used to determine the value. A measurement feature effectively binds a value to a location and to a method or instrument.

(Sensor) Platform
An entity to which can be attached sensors or other platforms. A platform has an asoociated local coordinate frame that can be refernced to an external coordinate frame and to which the frames of attched sensors and platforms can be referenced.

1.5. Conventions

1.5.1. Symbols (and abbreviated terms)

The following symbols and abbreviated terms are used in this document.

API Application Program Interface

CEOS	Committee for Earth Observation Sensors
COTS	Commercial Off The Shelf
GML	Geographic Markup Language
ISO	International Organization for Standardization
JPL	Jet Propulsion Laboratory
ODM	Observation Dynamics Model
OGC	Open GIS Consortium
OWS	Open Web Services
UML	Unified Modeling Language
SensorML	Sensor Model Language
WGS84	World Geodetic System 84
XML	eXtended Markup Language
xs:*	type definitions within XMLSchema.2002
1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional

1.5.2. UML Notation

The diagrams that appear in this document are presented using the Unified Modeling Language (UML) static structure diagram. The UML notations used in this document are described in the diagram below.

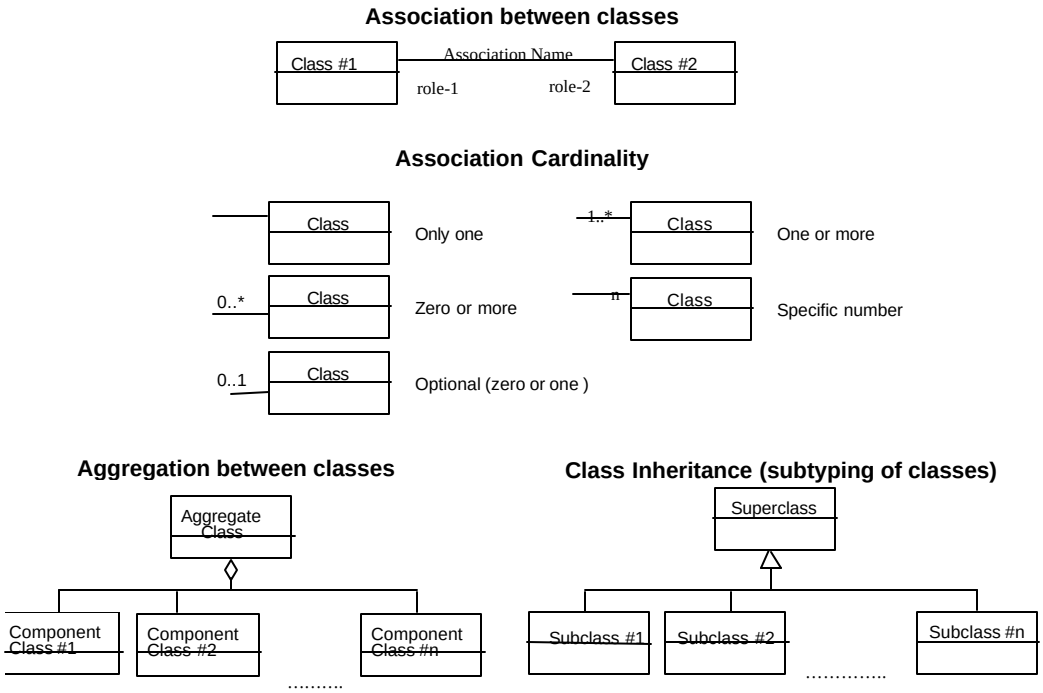

Figure 1.1 — UML notation

In this diagram, the following three stereotypes of UML classes are used:

a) <<Interface>> A definition of a set of operations that is supported by objects having this interface. An Interface class cannot contain any attributes.

b) <<DataType>> A descriptor of a set of values that lack identity (independent existence and the possibility of side effects). A DataType is a class with no operations whose primary purpose is to hold the information.

c) <<CodeList>> is a flexible enumeration that uses string values for expressing a list of potential values.

In this document, the following standard data types are used:

a) CharacterString – A sequence of characters

b) Integer – An integer number

c) Double – A double precision floating point number

d) Float – A single precision floating point number

1.6. Background

1.6.1. Motivation

The importance of long-term monitoring of the Earth's environment and the development of improved data processing techniques, has raised awareness of the need for preserving low-level sensor data and the information required for reprocessing this data. Unfortunately, such information is often lost or difficult to find five to ten years after completion of a sensor's original mission life. The proposed SensorML is one step toward preserving part of the vital information required for geolocation and processing of sensor data for both real-time and archival observations.

Web-enabled sensors provide the technology to achieve rapid access to accurate environmental information from the field. Streaming sensor data in standard formats faciliates integration, analysis, and creation of various data "views" that are more meaningful to the end user and avoids the drawbacks of locating and accessing incompatible archived data. This provides a significant advantage in that it reduces the time lag between making measurements and applying those measurements in decision-making. Time savings are particularly noticable in the management of time critical events such as emergency response, advanced warning systems, and forecasting. A second benefit is in the routine use of data for everyday decision-making. Together, these developments will advance the realization of an integrated, yet distributed, monitoring and assessment system used by government, researchers, businesses, and the public in improving decision making based on high quality, near real time data and information.

Furthermore, recent research and development activities have demonstrated several significant benefits of providing on-demand sensor geolocation within desktop or on-board tools. These include:

(1) Significant reduction of distributed data from Earth observation sensors; large data volumes resulting from the distribution and storage of per-pixel latitudes and longitudes, as well as other preprocessed geometric relationships can be replaced with the calculation of these values on-demand

(2) Improved capabilities for visually integrating and analytically comparing multi-sensor data

(3) The ability to more easily correct geolocation errors from within the end-user tools and to redistribute these corrections to the user community

(4) The ability to take advantage of several adaptive methods in computer graphics for improving interactivity within visualization tools

(5) Greatly improved capabilities for search and query of spatial-temporal sensor data without the need to request and perhaps store large data sets.

Traditionally, the geolocation of low-level sensor data has required writing or utilizing software specifically designed for that sensor system. The availability of a standard model language for describing platform position and rotation, as well as instrument geometry and dynamics, allows for the development of generic multi-purpose software

that can provide geolocation for potentially all remotely sensed data. The availability of such software, herein referred to as an Observation Dynamics Model (ODM), in turn provides a simple, single Application Programming Interface (API) for tool developers to incorporate sensor geolocation and processing into their application software.

One intent of a standard SensorML is to allow the development of software libraries that can parse these files and calculate required look angles and timing for each sensor pixel. Other efforts are establishing standards for storage and transmission of sensor platform location and rotation in order to insure that such formats are also maintained, available, and readable by similar APIs [CCSDS, 1999].

1.6.2. Importance to archival needs

A standard description format for sensors is important for the long-term definition of the sensor model's fundamental characteristics and assumptions for use in future reprocessing and refining of sensor data.

We are currently entering an era of Earth observation in which we have realized the importance of long-term observation of the Earth's environment. Thus, archiving the raw or low-level sensor data for future reprocessing has taken on greater importance. Equally important is that we preserve the characteristic metadata and assumptions required to reprocess the sensor data. The characteristic data include what is needed for geolocation, calibration, and radiometric processing of the remotely sensed data. Simply archiving the latitude and longitude values will not only be expensive, but will also prove to be highly inadequate. It is anticipated that further efforts within organizations such as the CEOS Data Subgroup, ISO TC211, and the OpenGIS Consortium, will be directed toward insuring proper standardization and archiving of other required data, such as platform position and rotation, and target or planet models. This current paper is directed specifically at the adequate description and standardization of fundamental geometric, dynamics, and measurement characteristics of the sensor.

As an example, sensor look angles have traditionally been either pre-calculated and stored within a data array structure, or are calculated as needed within software systems developed specifically for that sensor. With time, unfortunately, the actual parameter values for the geometric and dynamic characteristics of the sensor are often lost as contract reports and software become obscure, and as look angle arrays and hardwired software prove difficult to deconvolve into the characteristic sensor parameters. Once the initial mission has been completed and the processing teams dispersed, reprocessing, correction, or refinement of the sensor data thus become very difficult, if not impossible. For example, this was the case for reprocessing of the archived Optical Line Scanner (OLS) data (Ken Knowles/University of Colorado, personal communication), as well as for the 20 year old data from the Viking Mission to Mars (Bill Taber/JPL, personal communication).

1.6.3. Importance to software support

The standardization of a Sensor Model Language (SensorML) and the availability of SensorML documents for all Earth observing sensors will allow for significant

opportunities for software systems to support the processing, analysis, and visual fusion of multiple sensors.

Traditionally software that supported multiple sensors has been forced to deal with proprietary software designed for each individual sensor. When such software systems still exist and can be located, the software developer is often faced with trying to merge incompatible software architectures and development languages, or with rewriting the software to meet the requirements of his or her software. Even then, the addition of each new sensor system that the developer wishes to support requires the development of individual software modules specific to that sensor, often resulting in redundant code for manipulating and transforming the data.

In contrast, the availability of standard SensorML files allows for the development of general navigation software capable of geolocating and transforming any sensor data for which a SensorML file exists. Referred to as an Observation Dynamics Model (ODM), this concept is built around the availability of separate description files for providing sensor-system specific information regarding platform position and rotation, instrument geometry and dynamics, target planet shape and position, and perhaps other time-tagged information, such as data dropouts, instrument modes of operation, or spacecraft clock adjustments.

The second part of the Observation Dynamics Model concept consists of a generic software library for parsing these files, and calculating the transformations required to geolocate and perhaps process the sensor data. A significant advantage of the ODM concept for the developer and ultimately the end user, is that it provides a single source, single API, for the geolocation and processing of any sensor system, rather than requiring the developer to locate and implement proprietary software for each sensor system.

In addition, the ODM allows for tools that can provide all the benefits of on-demand geolocation and mapping. As ODM capable application software becomes more common, there will be less need to store and distribute volumetrically costly latitude, longitude, altitude, and incident angles values per pixel. Furthermore, with an ODM, correction of sensor geolocation requires only the redistribution of much smaller description files, rather than the redistribution of large collections of reprocessed sensor data. In fact, correction or refinement of geolocation can be conducted by the end user as necessary, rather than relying strictly on the instrument team. Finally, the ability to provide spatial-temporal knowledge of the sensor's coverage, independent of the on-line presence of sensor data, allows much needed search and query capabilities for determining sensor coverage for given a location or time, or for determining coincident sampling between two or more sensors.

In addition to the significant benefits discussed above, on-demand processing of geolocation has been shown to be as fast or faster than reading in and processing pre-calculated location values. With CPU power increasing faster than I/O rates, the improvement in the calculation of geolocation information will increase even further in the future, with the added benefit of not wasting valuable RAM capacity with storage of blocks of latitude and longitude values.

1.6.4. Importance to Sensor Web Enablement

Issue Name: [**INCOMPLETE**. (meb, 2002-04-16)]

Issue Description: Provide text regarding:

- Sensor autonomy and communication between sensors
- Web friendly access to information required to find, process, and geolocate sensor data
- Ability to bypass time consuming process of collecting data at a data centers, geolocating and processing all data to given data level, and then aggregating data into a data product that meets general needs of data community. Instead can enable immediate access to subset of the data of interest to the user and allow processing and mapping to custom desires.

Resolution:

1.6.5. History

The concepts proposed in this paper have been successfully tested and implemented in limited studies, and have been shown to provide significant benefits to both the developer and user of sensor data systems.

The SensorML was originally conceived as a sensor description language that would aid in the processing and geolocation of multiple sensors. Being driven by the remote sensing community, the original focus was on providing properties to assist in the geolocation of individual pixels within scanners and cameras.

The concept for the standardized description files for all aspects of sensor geolocation was first proposed by the planetary science community and was implemented in the SPICE software system, that was developed and maintained by the Navigation and Ancillary Information Facility (NAIF) at NASA JPL. Written in FORTRAN and utilized by the planetary science community for nearly every mission since Galileo, SPICE has proved beneficial by reducing redundant programming for each mission, and by providing additional benefits not previous experienced before the implementation of SPICE.

In 1993, the University of Alabama in Huntsville (UAH) in cooperation with NASA JPL, implemented and tested the SPICE concepts for application within the Earth observation community. Termed the NASA EOS Interuse Experiment, this research effort focused on improving the integration of multiple sensor data by avoiding the need to distribute data as incompatible map projection grids. The results of the Interuse Experiment proved that the ODM concept for Earth observation sensors was well within our abilities, and provided perhaps even more significant benefits to the Earth observation than had been experienced within the planetary community.

With subsequent funding, the UAH VisAnalysis System Technology (VAST) team has been implementing a more lightweight ODM directed principally for the Earth observation community and developed in Java.

In April 1998, the Global Mapping Task Team (GMTT) within Committee for Earth Observation Satellites (CEOS) released the following recommendation:

> **April 1998 - Recommendation to CEOS: Interoperability of Multiple Space Agency Sensor Data from the Global Mapping Task Team – Bernried, Germany**
>
> **Definition of Problem:** There is an increasing realization by Earth observation scientists that data from space-borne sensors are not adequately nor easily georeferenced to meet their requirements. The consequence of this is that it is currently extremely difficult or impossible to combine data from different space-borne sensors or ground-based data. The first impediment is the lack of adequate, publicly available data on the spatial-temporal extents of data from space-borne sensors.
>
> **Recommendation:** We, the CEOS Global Mapping Task Team, recommend that the space agencies seriously consider the production, storage, public access, and interoperability of adequate data for describing the dynamics and geometry of the sensor system. These data might include satellite position (ephemeris), satellite rotations (attitude), sensor model (dynamics, geometry, and calibration), relevant planet models, and spacecraft clock model. This data should be made available in real-time. There should also be an effort to provide publicly available software to ingest the above data using a common API. Any recommendations for the most appropriate propagation model should be adequate documented or algorithms provided.

In September, 1998, Mike Botts introduced the beginnings of a "Sensor Description Format" to the CEOS GMTT and received recommendations to consider XML as a description framework. In September 1999, the initial XML-based version of the SensorML was introduced to the CEOS GMTT by Dr. Botts. In March 2000, he was awarded a contract through the NASA AIST program to complete, implement, and test the SensorML (funding was actually received in December 2000). Initial focus of the SensorML was focused on the geolocation and description of remote sensing instruments, with minor attention given to measurement characteristics such as radiometry [Botts, 2001].

In Fall 2000, Liping Di (a CEOS GMTT member) proposed to the ISO TC211 Committee to establish a standard sensor and data model framework. This was approved in December 2000 and Dr. Botts was asked to serve as a team member. The first meeting was scheduled for June 2001. It is expected that the SensorML will both influence and be influenced by the ISO activities. The intent is that the SensorML will be a compliant implementation of the ISO standard.

In March 2001, Dr. Botts was accepted to participate in the OpenGIS Consortium (OGC) Military Pilot Project (MPP-1) with an emphasis on introducing the concepts of the SensorML and ODM into the OGC SensorWeb initiative.

In September 2001, the OGC initiated the Open Web Services (OWS) project with one of three threads focused on the initial design and testing of SensorWeb concepts. Within that initiative, the SensorML provides a key component for describing the characteristics and

capabilities of any sensor, whether designed for in-situ or remote observation. The result of the initial phase of this activity has been to drive the development of the SensorML into several directions, including:

- Generalizing the structure and grammar to support both in-situ and remote sensors
- Providing more generalized and more robust support for describing measurement characteristics, regardless of whether the sensor measures radiation, chemical concentration, velocity, temperature, or any other physical phenomena
- Further migration toward an XML schema, with inheritance from other schema, such as OGC GML
- Providing more robust support for non-scanning sensors, such as profilers and frame cameras

Part 2. Design and Specification

2.1. Design Criteria and Assumptions

2.1.1. Basic definition of a sensor

Sensors are devices for the measurement of physical quantities. There are a great variety of sensor types from simple visual thermometers to complex electron microscopes and radiometers on-board earth orbiting satellites. In some cases, sensing may be accomplished by a person rather than a device, and the result of the "measurement" may be a category rather than a numeric quantity.

Typically, sensors fall into one of two basic types. In-situ sensors measure a physical property within the area immediately surrounding the sensor, while remote sensors measure physical properties at some distance from the sensor, generally by measuring radiation reflected or emitted from an observed object. Regardless, any geometric properties described within the SensorML schema are defined within the sensor's local coordinate frame and are only related to the geospatial domain through it frame's association with the platform, mount, and their association with some geospatial reference frame. For example, to fully describe a wind profiler's wind speed and direction measurements, the height of the sensor needs to be known as that sensor could be situated on the roof of a building, mounted to a 10-meter tower, or sitting at ground-level.

A SensorML document can be considered a "living" description of a sensor. The SensorML document begins as a template document, which is initially created using the sensor model design and is then appended or altered during the manufacturing, calibration, deployment, maintenance, and ultimately the removal of the sensor from service. Much of the specification of a sensor is shared by all sensor instances of the same model-number from the same manufacturer. This will typically include a description of measured properties, sample geometry, and the geometry and dynamics of any internal sampling arrays (such as scan patterns or frame camera properties. This initial template may include in addition some calibration parameters.

However, a SensorML document describing a particular sensor instance will acquire additional information that will distinguish it from other instances of the same model. In particular it may acquire unique identifiers such as ID and serial number. It will further be attached to some platform that will provide it with location and orientation within a known geospatial-temporal frame. In many cases the sensor instance will also have (for example) additional calibration information specific to the instance. As a sensor progresses through these stages, it's document will not only gain additional property information, but it will also record the changes to the sensor and the document itself through the inclusion of a history description.

Issue Name: [**COMMENTS – living document** (sc, 2001-12-16)]

Issue Description:

In this object-oriented view of a sensor much of the descriptive information is attached to the sensor class. In a strict translation of such a view to a serialized encoding, this means that the values of the parameters from the catalog are attached to the validation schema, and the additional parameters are added in the valid instance document, which inherits the catalog values from the class (schema). The schema is associated with the sensor blueprint, while the instance document is associated with a built sensor. The information common to all members of the same class is primarily available from the schema. (sc)

Template document is a pragmatic implementation of this approach …PSVI issue … solution strategy … SensorML as living document also records history, etc. (sc)

In lieu of the template and living document approach, an alternative way of modeling the sensor, is that an instance has an association with a particular sensor type. The information associated with a sensor from a particular manufacturer and model is recorded in a document describing the type, while the additional information associated with the sensor instance is recorded in a document for the instance. Since (in the usual case) there will be many instances of the same type, you would expect to store the information about the type in one place and re-use it by reference. Information about the sensor which is common to the type is then available to an instance by traversing the association. (sc) We chose the other means because …

Resolution:

Issue Name: [**COMMENTS – living document** . (meb, 2002-02-16)]

Issue Description:

The discussion above assumes we provide a schema for each sensor model. While this could be done and would probably have some benefits, I question whether it should be a requirement.

Resolution:

2.1.2. Sensor Collection Concepts

Within SensorML, a sensor collection can itself be defined as a sensor. A sensor collection can be of two types, herein referred to as “sensor package” and “sensor array”.

A sensor package is composed of multiple sensors that operate together to provide a collective observation or related group of observations. For example, a collection of temperature sensors (i.e. thermistors) can be used in a combined fashion to create a sensor that measures wind velocity and direction. Similarly, a group of individual sensors that measure different chemical species can be grouped as one sensor that provides “water quality”. Sensor package produces either a single observation or a composite observation of multiple related properties.

A sensor array is a set of sensors of the same type at different locations. These locations may be within a single sensor frame, a different location on a single mount, or on different platforms. A sensor array produces observations that are used to build a spatial coverage.

2.1.3. Relationship of the sensor to a platform

A sensor system is composed of three main elements as shown in the UML diagram in Figure 2.1. Only the sensor element is viewed as being able to measure physical quantities. A platform such as an aircraft (carrying a frame camera) may be able to determine its own instantaneous orientation and position, and in such a case, these measurements would be obtained by other sensors attached to the platform.

The main importance of the mount is that it defines the position and orientation of the measured quantity for each carried sensor in a mount defined frame of reference. Within the design of SensorML, a mount is considered a special case of a platform (in particular, an *AttachedPlatform*). The position and orientation of the mount are then defined with respect to another mount-defined frame of reference and ultimately to the platform-defined frame of reference. All of the frames of reference can in general be considered to be moving. Figure 2.2 illustrates the relationship of a sensor's frame (in pink) that is fixed but has been translated and rotated relative to the moving spacecraft frame (in black). The concept of frames is introduced in Section 2.1.4.

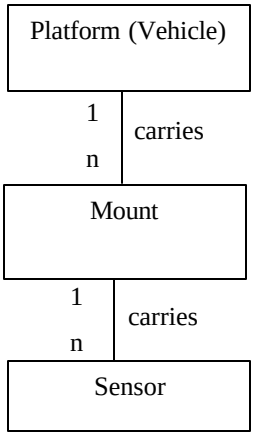

Figure 2.1 Diagram showing basic relationships between sensor, mount, and platform.

Based on the type of sensor and on the characteristics of the platform, a sensor system can be classified according to Table 2.1. Based on the dynamics of the platform, a sensor system may be fixed (stationary) or mobile (dynamic). Based on the sensor characteristics, a sensor system may measure either in-situ (in place) or remotely. Thus, a remote sensing atmospheric profiler might be fixed to the ground (fixed remote) or attached to an aircraft (mobile remote). Similarly, an in-situ water quality sensor might be attached to a fixed station (fixed in-situ) or to a boat (mobile in-situ).

Measures / Mobility	In-Situ	Remote
Fixed	Stationary O2 Probe	Doppler Radar station
Mobile	"Diving" Salinity probe	Airborne LIDAR

Table 2.1. Relationships between in-situ and remote sensors and dynamic and fixed platforms.

As previously discussed in Section 1.1, we separate the description of the sensor from that of its mount and platform. The main importance of the associated platform(s) is in providing the relationship of the sensor and its observations to some relevant external coordinate system (for example, a geospatial reference system).

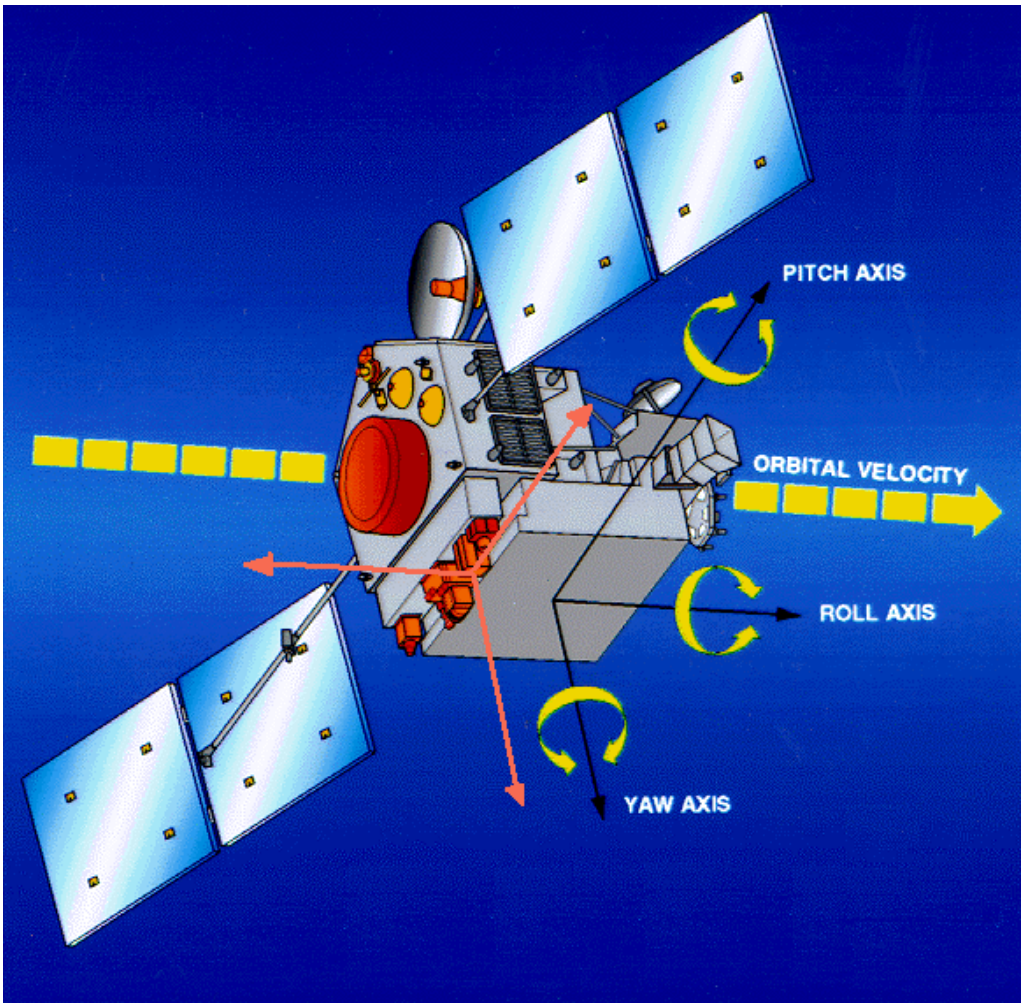

Figure 2.2 Relationship of sensor frame (pink) to the moving platform frame (black).

2.1.4. Concept of coordinate frames

All geometric and temporal characteristics of a sensor system must be related to a specified coordinate frame. Within the SensorML, all definitions for sample geometry, look angle, and collection geometry are described relative to the sensor's spatial frame. It is only through the sensor's relationship to its mount and platform(s), that the sensor and its measurements can be related to an external coordinate system, such as geographic latitude and longitude.

This is accomplished through the use of defining nested coordinate frames and their relationships to one another. For instance, an individual sample's geometry (e.g. shape and size) is defined in the localized coordinates of that sample. Its relationship to a sensor's frame may be specified through a collection geometry definition. The sensor's frame may, in turn, be related to its platform's coordinate frame through its mounting angles and position. Finally, the platform's frame is related to a geospatial coordinate frame by defining its position and orientation within that frame. The successive transformation of each of these coordinate frames into its parent frame provides the information necessary to georegister the sensor's measurements.

For a remote sensor, it is necessary to determine the intersection of a pixel's look ray and the surface of the sensor's target (e.g. the Earth's ellipsoid). Typically the look angle and the sensors target are transformed into a common spatial reference frame, such as the Earth Centered Fixed (ECF) or Earth Centered Inertial (ECI) reference system.

The frame concept will also be applied to temporal domain when applicable. One local time frame that is useful for defining the geometry and dynamics of scanners, is seconds past the start of a scan (scan start time). Also, for some sensor systems, time is recorded relative to a local clock or the start of the mission. In such cases, time frames and their transforms to "Earth time" will be defined in SensorML.

2.1.5. Measurement / observation concepts

A sensor is designed to *measure* a particular property within a given *sample* space. When these measurements are taken, they result in an *observation* that may be immediately utilized or stored. In its lowest level, this observation is typically a proxy measurement of some property other than the desired physical property, itself. For example, an observation may be the height of mercury in a thermometer or the voltage across a circuit. In order for these observations to be related to a more useful physical property, a new observation must be derived using known sensor calibration functions and perhaps other processing algorithms.

A SensorML document will describe what physical properties are measured by the sensor, as well as information concerning the processing and quality of these measurements. However, the SensorML document does not define the value of these observations, nor where the values are stored.

The definition of the Observation schema is outside of the scope of this document. However, it is important that Observations be capable of being associated with the appropriate sensor and with the appropriate measurement description is associated with

that sensor. For a definition of the Observations and Measurements schema, see the Observations and Measurements IPR [Cox, 2002].

2.1.6. Sensor Response Characteristics

The response characteristics of a sensor determine how the sensor will react to a particular stimulus (i.e. Observable) and how it will operate under given environmental conditions. Within the sensor response characteristics will be specifications for sensitivity (e.g. threshold, dynamic range, capacity, band width, etc.), accuracy and precision, and behavior under certain environmental conditions (e.g. survivable range and operational range).

While some of these parameters are relevant for a wide range of sensor types, other sensor types may require their own collection of response characteristics. For example, many of the response characteristics describing a radiation-based sensor (e.g. peak wavelength, band width, polarization angle, spectral response curve, etc.) will be different from those defining a water temperature probe. It is anticipated that while many sensors will be able to reuse some base level response characteristics, others may require appropriately defined response schemas for specifying different or additional parameters.

2.1.7. Sample and collection geometry concepts

As discussed above, a sensor measures some property within a spatially and temporally defined sample. In the case of an in-situ sensor, this sample includes some spatial volume in the immediate vicinity of the sensor. This volume may be infinitesimally small or it may be unknown or unimportant. For remote sensors, the sample involves some volume or surface area located away from the immediate vicinity of the sensor.

Regardless of whether it describes an in-situ or remote sensor, the geometric descriptions in SensorML are relative to the sensor's local coordinate frames and not a geospatial coordinate frame. As discussed before, this allows the same sensor model to "attached" to any stationary or dynamic mount and platform without a need to significantly change the SensorML description.

An individual sample's geometry, such as perhaps its size, shape, or point-spread function, is described relative to a local sample coordinate frame. This sample frame can be related to the sensor's frame by either a simple transformation or in the case of collection of samples, by a more complex transformation involving arrays or scan patterns. Possible transformations for sample collections include unstructured grids, regular arrays, scanners, and frame cameras. These will be discussed in more detail in Section 2.2.11.

2.2. SensorML Definition

2.2.1. Main components of the sensor description

Based on the concepts above, the sensor description is divided into eight main components of information, including:

- Basic **sensor information**, such as a unique identifier and names, as well as sensor type and last modification date of the sensor document
- The **properties** that can be **measured** (observableTypes)
- The **platform** on which the sensor is attached
- Definition of the sensor's **coordinate frame** and it relationship to an external frame
- The sensor **metadata**, including the manufacturer, model name, serial number, and history of the sensor
- SensorML **documentation metadata,** such as who created this description, what modifications have been made/when, what assumptions were used, what other references and informative web sites are available
- The **response** characteristics of the sensor to the external stimuli and what is **quality** of the measurements it obtains
- Information required to determine the **geometric and temporal characteristics** of the samples and sample collections related to this sensor in geodetic space and real-world time

The UMLs in the following section provide the model design of the SensorML. As shown in the UML of Figure 2.3, the root for all SensorML documents is *Sensor*. The *Sensor* has a unique id (of type xs:id) as a required attribute, and the two required properties, *shortName* (e.g. SSM/I) and *longName* (e.g. Special Sensor Microwave Imager), both of which take an xs:String. The *sensorType* property (e.g. scanner) has a value type of xs:anyURL that references a definition in a sensor taxonomy dictionary. The intent of the lastModification property, `which takes an xs:date, is to all parsers to quickly check the "version" of the document. Together these properties make up the "basic sensor information" component listed above.

Each of the other components will be discussed in detail in the sections below. Notice that the only required properties for the *Sensor* are *longName*, *shortName*, *sensorType*, *lastModification*, and *measures*. The other properties are optional so that a sensor description need only contain those properties that are relevant to the particular sensor type or sensor purpose.

As discussed in previous sections, a collection of sensors can itself be a sensor, as shown in the UML in Figure 2.4. The derived *SensorCollection* types include the *SensorArray*, which consist of a group of sensors which provide an array of like measurements and

SensorPackage which can provide a derived measurement based on the collection of dissimilar sensor measurements, as discussed in Section 2.1.2.

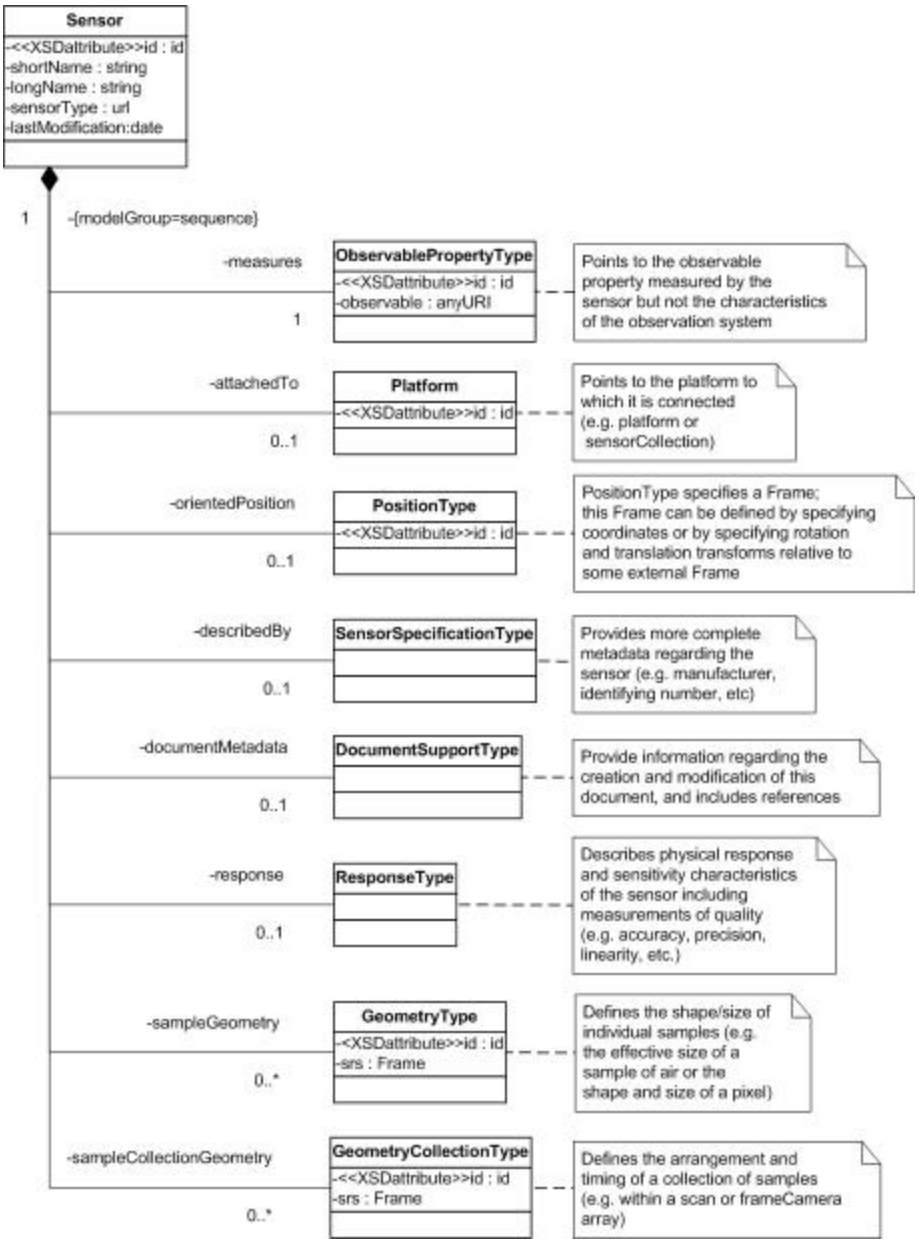

Figure 2.3 UML showing the first tier of the SensorML document structure.

Issue Name: [**Sensor Collection Division**. (meb, 2002-03-26)]

Issue Description:

Although these divisions make conceptual sense, it is unclear at present whether they

need to be defined and utilized in the *SensorCollection* cases. We may find that its better not to make a definite distinction of the two within the SensorML schema; it may complicate issues, particularly when a sensor collection may have properties of both

Resolution:

A *SensorCollection* may define its *Sensor* members immediately within the *member* property, as in:

```
<SensorCollection …>
    …
    <member>
        <Sensor …>
            <shortName>  myName  </shortName>
            <longName>  myLongName  </longName>
            <sensorType>  http://myType  </ sensorType >
            <measures>  ….  </ measures >
            …
        </Sensor>
    </member>
    …
</SensorCollection>
```

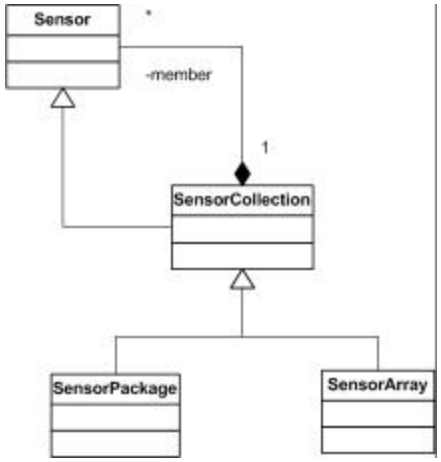

Figure 2.4 UML defining Associations and Inheritance between Sensor and SensorCollection.

Alternatively, a *SensorCollection* can use a *SensorReference* to link to either an internally defined sensor:

```
<member>
    <SensorReference ref=#sensor_1/>
    <SensorReference ref=#sensor_2/>
</member>
```

or to an externally-defined Sensor:
<member>

```
    <SensorReference remoteRef=http://some.url/sensor_1/>
    <SensorReference remoteRef =http://some.url/sensor_2/>
</member>
```

2.2.2. Observable Type

The classification of measurements obtained or derived from the sensor will be described by the *measures* property. As shown in the URL of Figure 2.5, the value for *measures* is an *ObservablePropertyType*. The *ObservablePropertyType* has two required attributes: a unique *id* (of type xs:id) and an *ObservableType* (type xs:anyURL) that points to an *ObservableType* definition in an *ObservableDictionary*. The definitions of the *ObservableType* and *ObservablesDictionary* is outside of the scope of this document, and has not been well-defined at this time. It is believed that the *ObservedProperty* and *ObservableType* values will be mined by registry engines in order to discover sensors that are capable of measuring particular physical properties.

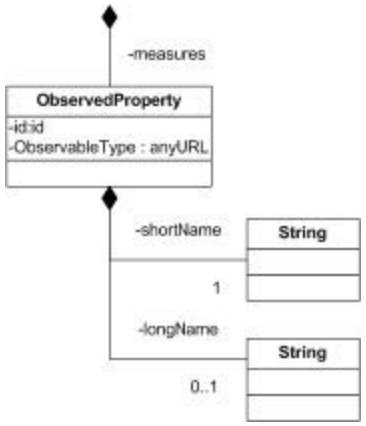

Figure 2.5 UML illustrating the abstract *ObservableType*.

In addition, the *ObservablePropertyType* has a required property, *shortName* (type xs:String) and an optional property, *longName* (type xs:String). Thus, an example of the *measures* property might be:

```
<measures>
    <ObservedProperty id="do_312"
            observableType="http:www.opengis.net/observation/type#dissolved_O2">
        <shortName> Dissolved O2 </shortName>
        <longName> Dissolved Oxygen at YSI Station 312 </longName>
    </ObservedProperty>
 </measures>
```

or

```
<measures>
    <ObservedProperty id="ssmi_f10_95V"
            observableType="http:www.opengis.net/observation/type#radiation">
        <shortName> 95 GHz V </shortName>
        <longName>
            SSM/I 95GHz vertical polarization channel on DMSP F10 satellite
        </longName>
    </ObservedProperty>
</measures>
```

2.2.3. Frames

A general concept of coordinate frames will surely prove valuable for several activities within the GIS open standards communities. Within OpenGIS, a *Frame* schema is being considered for implementation within GML. While a *Frame* schema is presented as part of the current release of SensorML, future releases will explore the possibilities of interoperability with frames defined within other schemas.

One considers that there exist well-defined "absolute frames" that can serve as common coordinate frames for relating data sets of interest. Typically, these "absolute frames" would be an inertial frame of reference, such as J2000, used in the planetary community, or Earth Centered Inertial (ECI) and Earth Centered Fixed (ECF) frames, which are defined within WGS84 specifications and used within the Earth-focused communities. An "absolute frame" could also be geodetic latitude-longitude-altitude system or a projection-based defined spatial reference system, although these frames are not typically appropriate for dynamic or remote sensors. Within SensorML, there is a definition for an abstract *_AbsoluteFrame*, as well as several concrete definitions for *EcfFrame*, *EciFrame*, *J2000Frame*, and *WGS84LatLonAltFrame*.

In addition to absolute frames, any number of local coordinate frames can exist and be defined relative to an absolute frame. Local frames can also be nested and defined relative to other local frames. All of these concepts hold true for temporal frames, as well as for spatial frames.

Frames are encoded in SensorML in several ways including:

- As a sequence of vectors.
- Via an affine transformation specified by various means, such as a matrix. The matrix of the transformation can also be encoded in several different ways, including for example:
    - Specification of the matrix elements, including 4x4 homogeneous matrices
    - Specification of a rotation matrix by specifying Euler angles.
    - Specification of a rotation matrix by specifying Pitch, Roll, Yaw.
    - Specification of a translation by specifying a vector.

As shown in the UML in Figure 2.6, a *FrameDefinedByVectors* requires an *origin* (provided by a Vector) and 2-3 *component Vectors*. A *FrameDefinedByTransform*

includes one or more *transform* definitions that are relative to some other *Frame*. The *transform* is typically defined by a *Matrix*.

A example of a *FrameDefinedByVector* is given below:

```
<Frame id = “F1”>
    <component>
        <Vector id = “v1”>
            <component relativeTo = “#e1”>1.0</component>
            <component relativeTo = “#e2”>1.5</component>
            <component relativeTo = “#e3”>-1.2</component>
        </Vector>
    </component>
    <component>
        <Vector id = “v1”>
            <component relativeTo = “#e1”>5.0</component>
            <component relativeTo = “#e2”>2.3</component>
            <component relativeTo = “#e3”>-3.2</component>
        </Vector>
    </component>
    <component>
        <Vector id = “v1”>
            <component relativeTo = “#e1”>2.7</component>
            <component relativeTo = “#e2”>2.5</component>
            <component relativeTo = “#e3”>-1.2</component>
        </Vector>
    </component>
    <origin>
        <Point gid = “v1” srsName=“..”>
            <coordinates> ….</coordinates>
        </Point >
    </origin>
</Frame>
```

The Frame can also be defined by an affine transformation. This is encoded as follows:

```
<Frame id = “F2”>
    <transform relativeTo = “#F1”>
        <Matrix id = “m1”>
            <position row=1 col=1>2.34</position>
            ….
        </Matrix>
    </transform>
</Frame>
```

Figure 2.7 illustrates potential relationships between a collection of *Frames* associated with a dynamic sensor system. Within this relationship, a sample’s frame is defined through position or transform relative to the sensor frame. The sensor frame may be related to its mount through some transform, while the mount’s frame may in turn be related to the platform frame through some mounting rotations and translations. This mount could be static or dynamic relative to the platform. Finally, the relationship of the platform’s frame relative to some geospatial “world” frame is specifies by it own transform. For a static, in-situ sensor system, these relationships could be trivial. The

sample's frame may be identical to the sensor frame, which can be easily related to a geospatial frame through orientation angles an position translation.

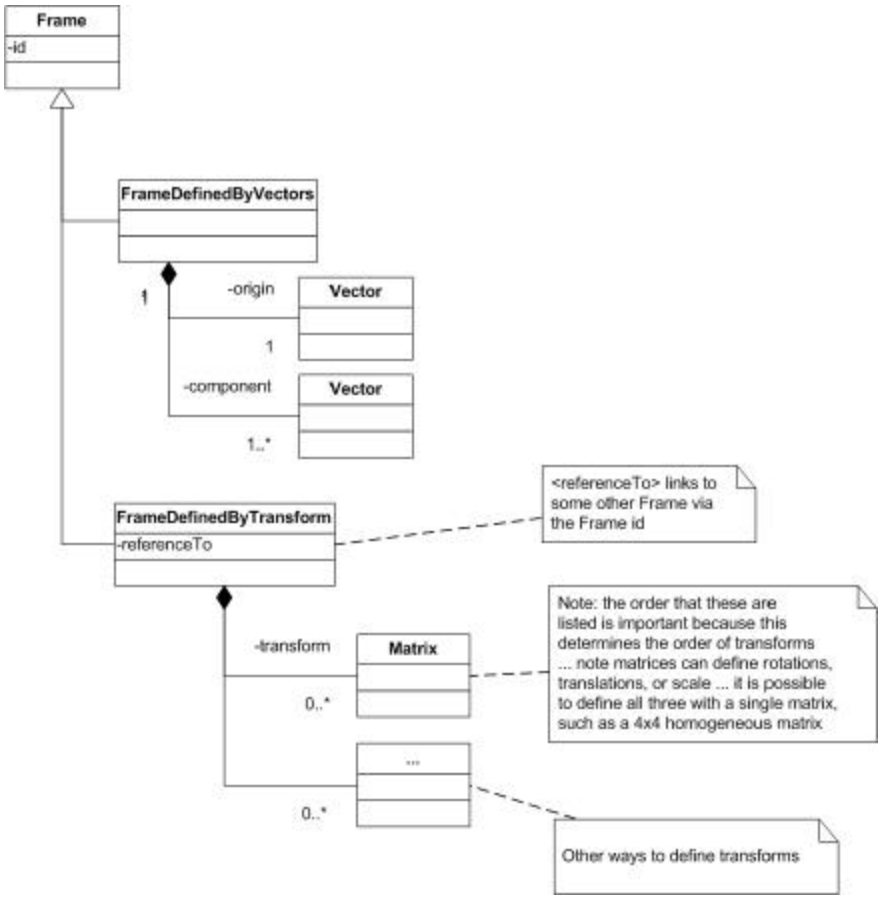

Figure 2.6 UML defining the Frame schema.

In contrast, for a dynamic remote sensor, the relationships can be much more complex. The sample frame might be related to the sensor frame through a dynamic scanning transform. Similarly, the platform frame might be dynamic relative to the geospatial frame and be defined by some space-time curve or through some transform defined by orbit propagation. For remote sensors, this information is capable of defining the origin and position of any "look rays" related to the sensor, but is not capable of defining the location of the observation without deriving the intersection of these look rays with some target body (or volume) that is defined with the same geospatial frame. This target frame might be an ellipsoid defined by WGS84 or the geometry of a building or cloud.

A *Frame* can be viewed as moving in which case the origin is viewed as lying on a curve in the vector space. Curves can be defined intrinsically using, for example, a special frame called the Frenet frame.

An affine *Frame* can be defined relative to any other frame in the vector space simply by specifying the transformation between the two frames. This transformation is always an

affine transformation. If the frames represent the position and orientation of a rigid body than this transformation is a composition of a translation and a rotation.

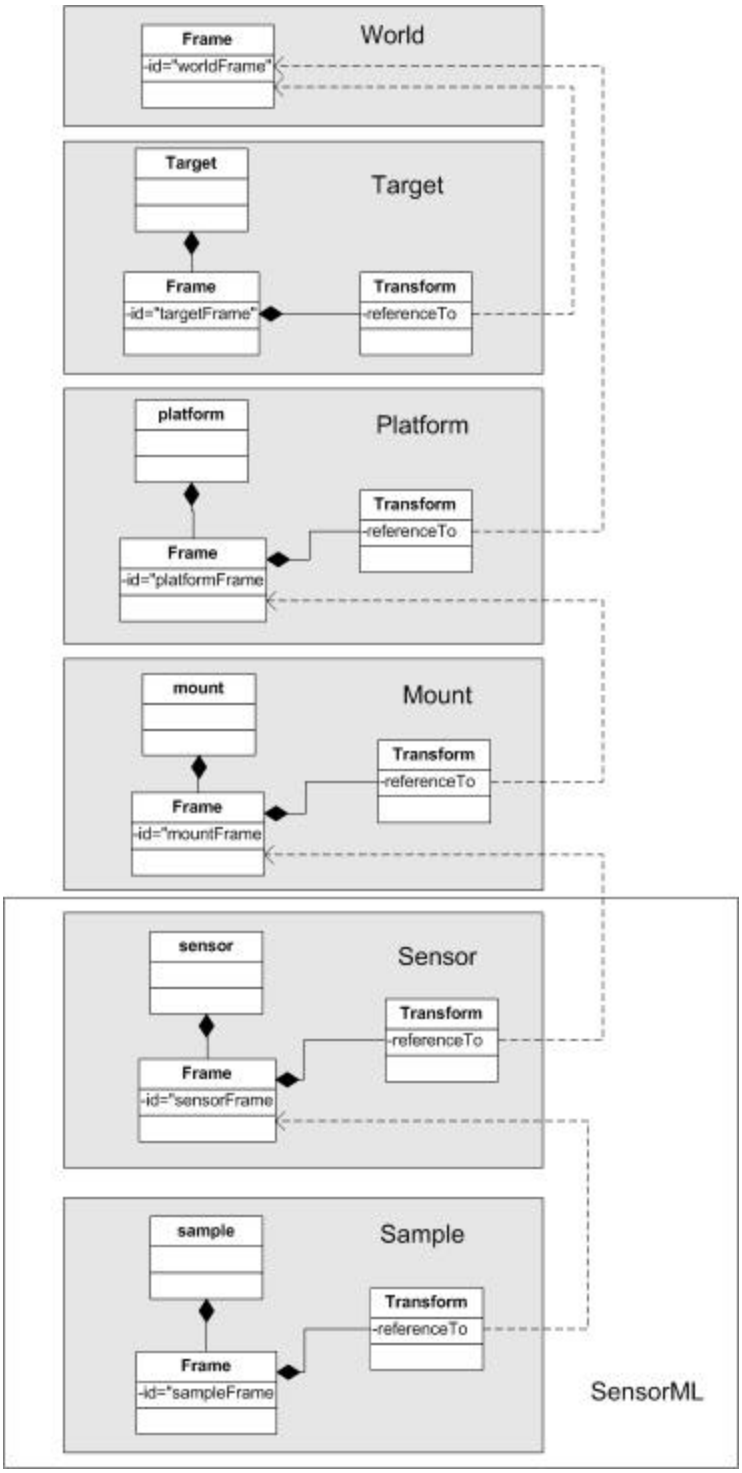

Figure 2.7 UML showing potential relationships between frames in a sensor system.

2.2.4. Platforms

The optional *attachedTo* property references a *PlatformType* as it value. The Platform reference should be either a URL pointing to a *Platform* instance or a URI pointing to a simple *Platform* definition within the SensorML instance. Typically, except in very simple cases (such as a in-situ sensor on a simple static platform), it is expected that the *Platform* definition will be external to the *Sensor* definition. As with the Sensor definition, there is a corresponding *PlatformReference* property that allows one to link to an internally or externally defined *Platform*:

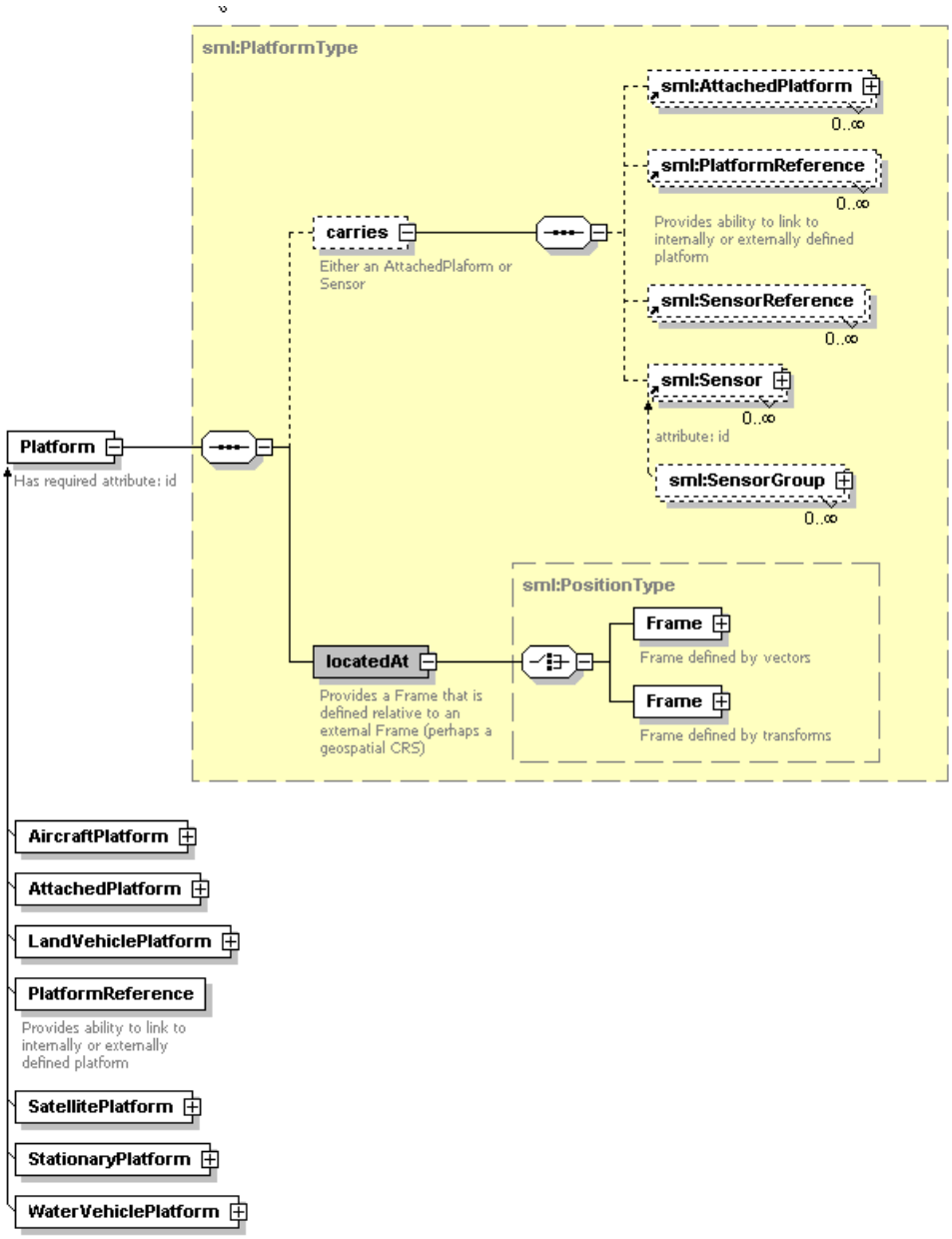

Figure 2.8. Schema for Platform.

```
<attachedTo>
    <PlatformReference remoteRef=http://some.url/myPlatform/>
</ attachedTo >
```

The *Platform* schema defined in SensorML is currently rather basic and perhaps incomplete. Its primary focus is to provide a means of defining a coordinate reference *Frame* for an attached *Sensor*(s) and to provide a means of relating the *Platform Frame* to some geospatial reference *Frame.*

As shown in Figure 2.8, *Platform* has a required attribute, id (type xs:id) that provides a unique identifier for that platform. It also has an optional property, carries, that takes any number of *AttachedPlatform* or *Sensor* instances. The required *locatedAt* property defines the platform's *Frame* relative to some external *Frame.* As discussed above, a *Frame* can be defined as vectors within the external *Frame* or as transforms (translations and rotations) relative to the external *Frame.*

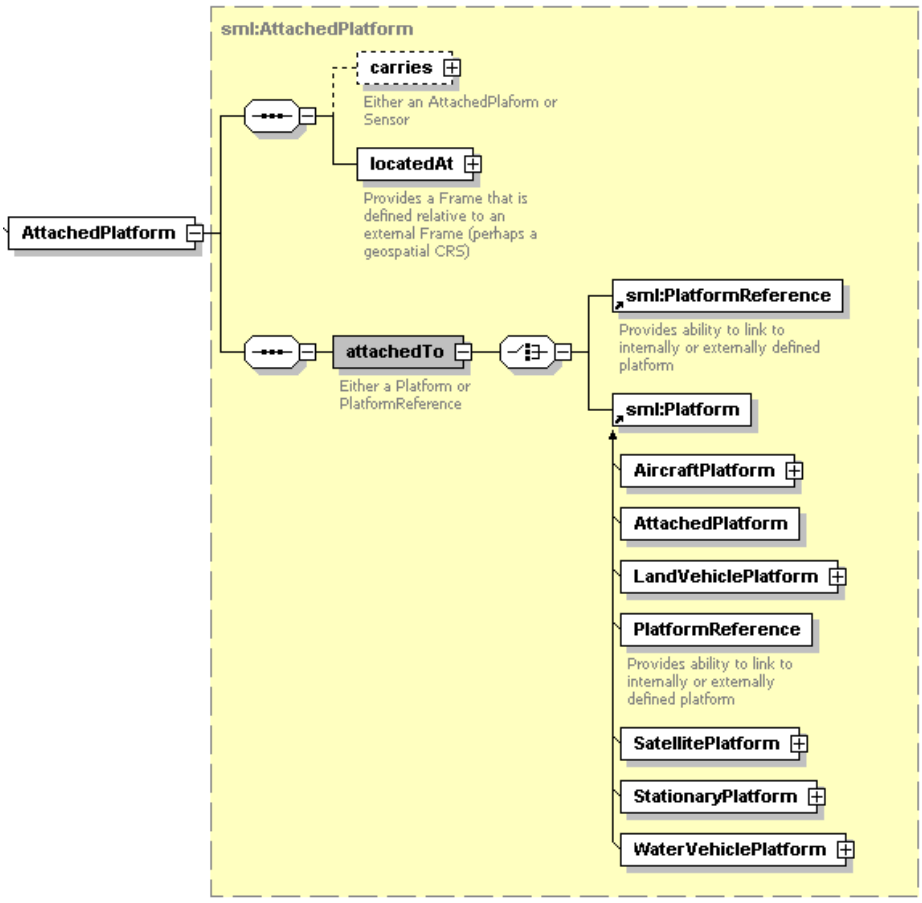

Figure 2.9. Schema for an AttachedPlatform.

There are currently several specific platforms defined through extension of *PlatformType*. Although not currently established in the schema, it is intended that a given type of

Platform will take certain *PositionTypes* as values for the *locatedAt* property. Currently, there are definitions for aircraft, satellite, land vehicle, water vehicle, and stationary platforms.

In addition, there is a special case *AttachedPlatform* that will typically derive its geospatial positioning from a parent *Platform*. A sensor mount would be one example of an *AttachedPlatform*. As shown in Figure 2.9, the AttachedPlatform has an added *attachedTo* property that takes a *Platform* as its value.

2.2.5. Oriented Position

The position and orientation of the sensor is specified within the *orientedPosition* property. As shown in Figure 2.10, the value of the *orientedPosition* is a *Frame* that defines the *Sensor's Frame* in relation to the *Frame* of a *Platform*. As was illustrated in Figure 2.7, this *Platform* could be a mount (i.e. an *AttachedPlatform*) that is itself related to a static or dynamic *Platform*.

Particularly for simple static sensors, the *orientedPosition* and associated *Frame* may completely define the sensor's location within the SensorML instance. However, for more complex dynamic sensors, the sensor's position may be relative to an externally defined *Frame* in which actual position and orientation values may need to be derived or at least interpolated using appropriate software. Although the schemas for *Platform* and *Frame* are not yet fully defined, it is anticipated that a *Platform Frame* will point to *Frame* instances that define position and orientation through time, as well as potentially be able to reference a service that can provide a position and orientation for any given time.

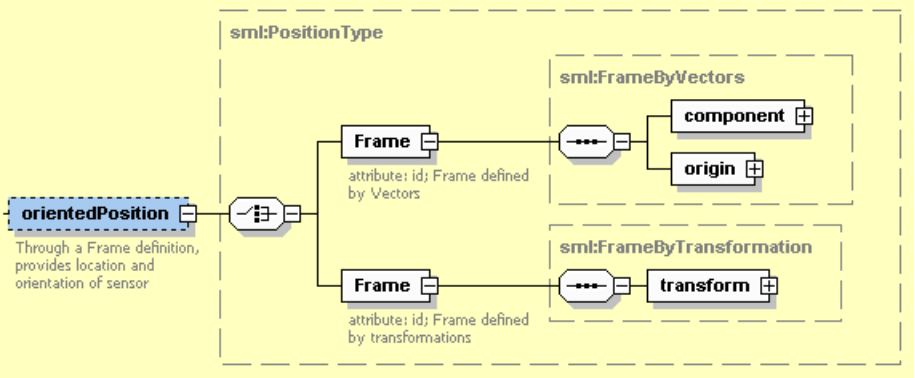

Figure 2.10. Schema for PositionType.

2.2.6. History and Actions

As described above in Section 2.1.1, a SensorML document is considered a "living" document that can accompany (or record changes to) a sensor from its design through its creation, deployment, calibration, maintenance, and ultimately its removal from service. Within SensorML, these will be recorded through a collection of *SensorAction* values. Similarly, the SensorML document itself undergoes changes, some of which result from

actions on the sensor, and others that result from a *SensorAction*, document additions, or changes to the sensor model assumptions. All of these changes are recorded within the SensorML through the use of a collection of *DocumentAction* descriptions.

The UML in Figure 2.11 defines the base *ActionType* as well as two derived definitions for *DocumentAction* and *SensorAction*. Also shown in Figure 2.11 are derived classes for *DocumentCreation*, *DocumentModification*, *SensorDeployment*, *SensorInspection*, and *SensorCalibration*. A model of the *DocumentCreation* definition is given as an example. As will be described in more detail below, *SensorAction* and *DocumentAction* will be employed in the *history* properties for *describedBy* and *documentMetadata* sections of the SensorML document.

Associated with *ActionType* are two utility schema for *Person* (Figure 2.12) and *Reference* (Figure 2.13). While these are currently defined in SensorML for completeness, they may be replaced in future releases with more wide-spread standard schema.

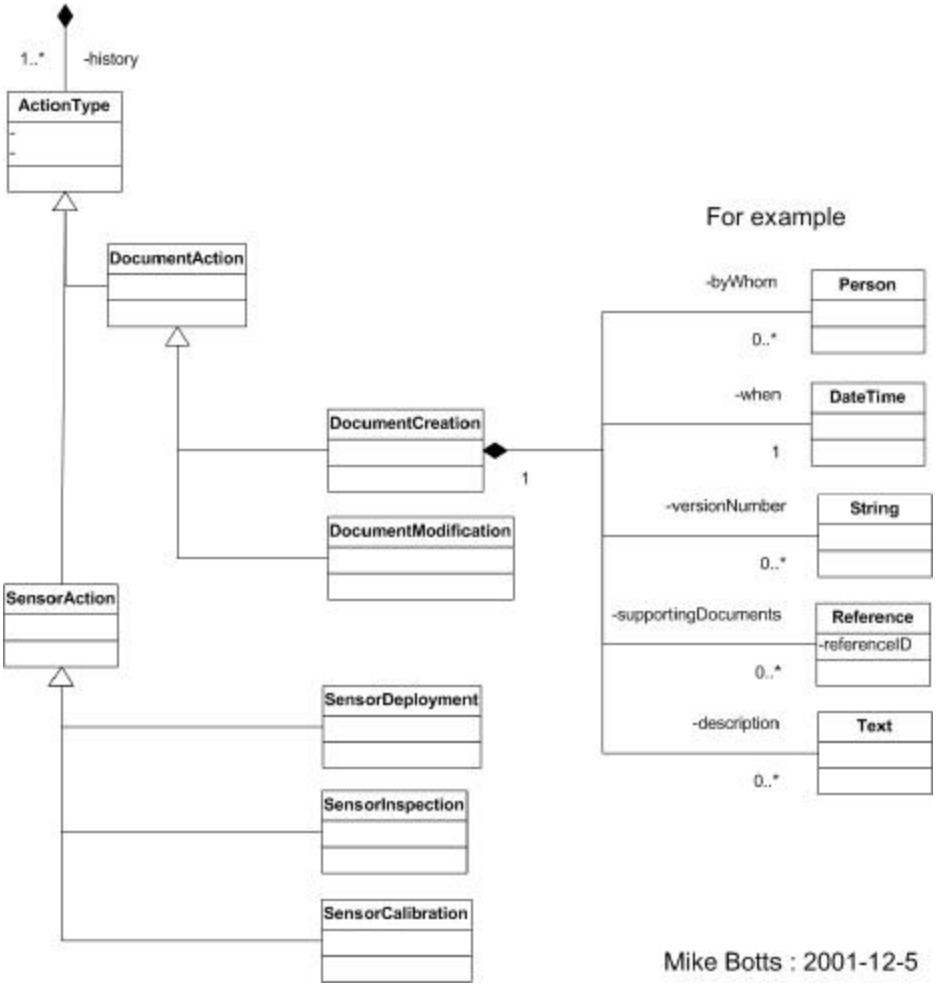

Figure 2.11 UML defining ActionType inheritance and example associations.

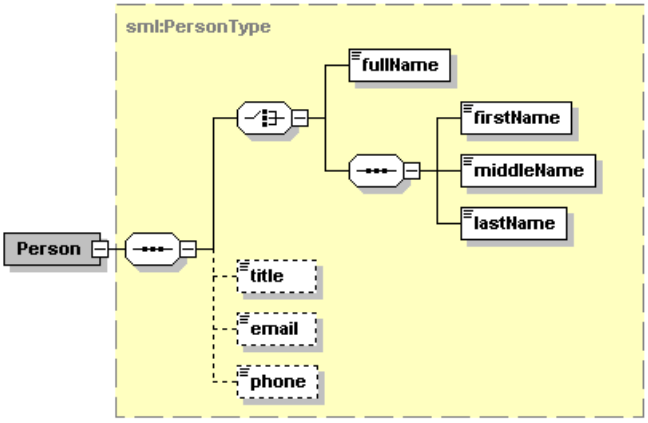

Figure 2.12. Schema for Person.

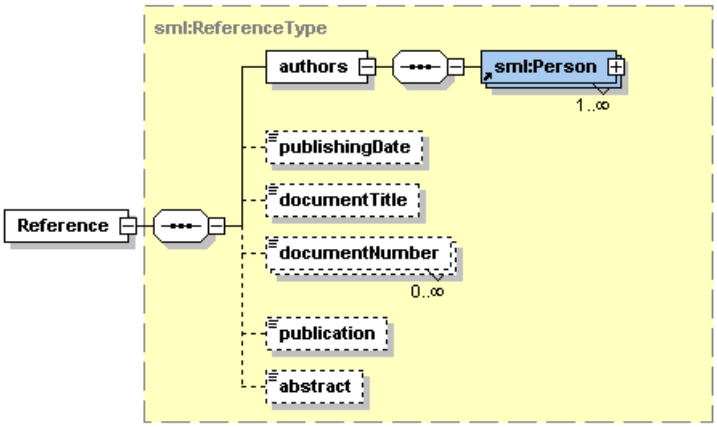

Figure 2.13. Schema for Reference.

2.2.7. Sensor Metadata

General information regarding the sensor is provided by the *describedBy* property that takes a *SensorSpecificationType* as its value. *SensorSpecification* provides metadata describing the sensor. As shown in the schema diagram of Figure 2.15, required properties include *manufacturedBy* and *model*, but of which take a xs:String as its value. Optional properties include one or more listings for *identifyingNumber* (e.g. serialNumber, modelNumber, or versionNumber) and a textual *description*.

The *history* property takes a *SensorAction* as its value, and allows one to record sensor events either as they occur or post event. Currently, *SensorActions* include *SensorCreation*, *SensorDeployment*, *SensorInspection*, and *SensorDecommission*, although it is anticipated that further *SensorAction* specifications will be defined.

SensorAction. A typical *SensorAction* includes information about by whom, when, and where the action occurred, and any supporting documents and textual descriptions regarding the action.

SensorSpecification illustrates the first application of a utility class called *Document*. *Document* includes any number of *authors* (type *Person*), reference *date* (type xs:date), *subject* (type xs:string), and *description* (type xs:string). *Document* is usable anywhere within a SensorML schema or instance.

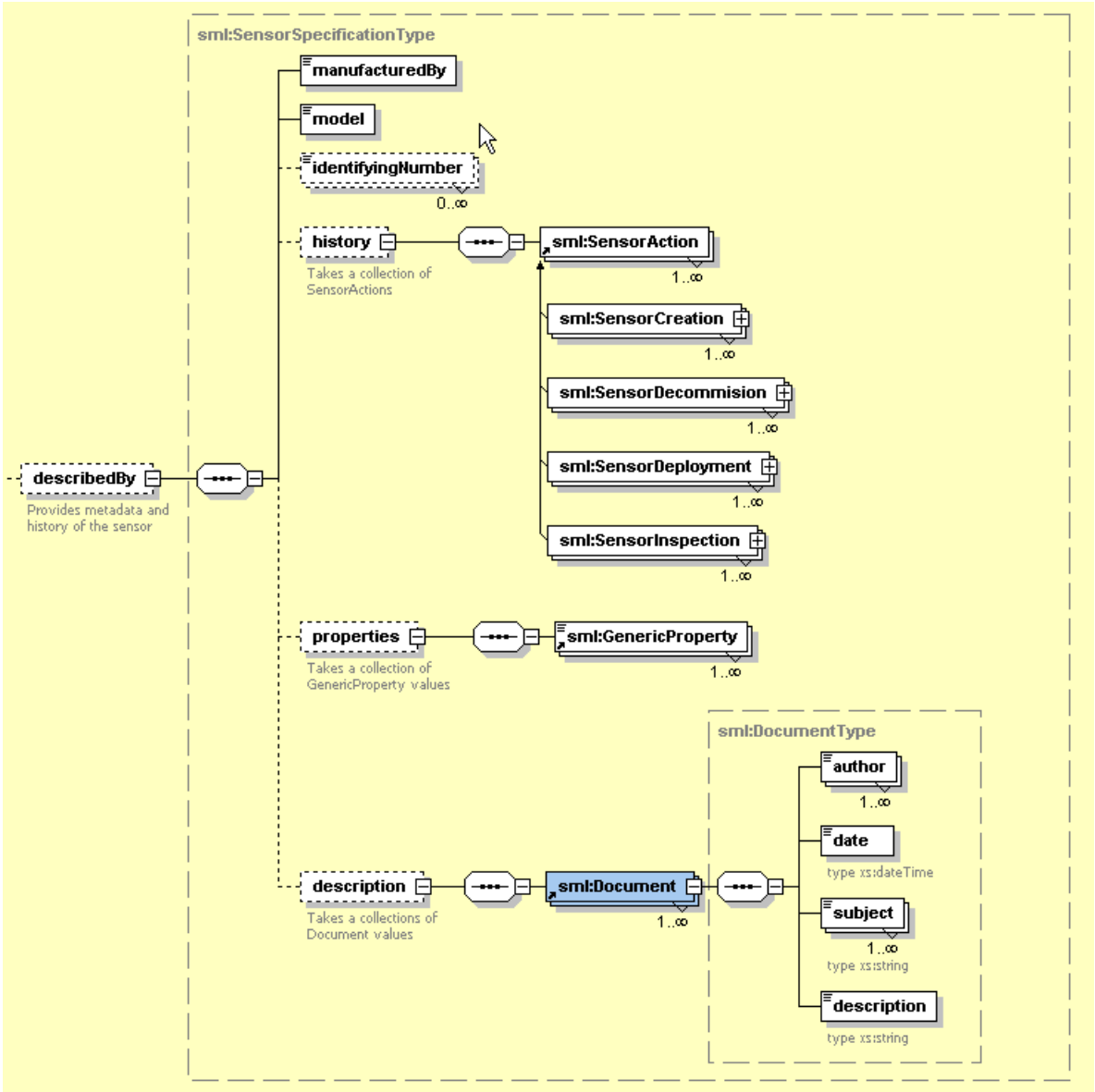

Figure 2.14. Schema for SensorSpecification.

An additional optional property within *SensorSpecification* is *properties*, which lists one or more *GenericProperty* values. *GenericProperty* allows one to include properties that might not be directed supported by SensorML, but that might be desired by some proprietary software or manufacturer data base. Examples include:

```
<genericProperty name="membrane thickness" dataType=" float"
    uom="uom:mil">1.0</genericProperty>
<genericProperty name="membrane type" dataType="string">Teflon</genericProperty>
<genericProperty name="probe solution type" dataType="chemical species"> Na2SO4
    </genericProperty>
```

Thus, an example of a *SensorSpecification* is presented below:

```
<describedBy>
    <SensorSpecification>
        <manufacturedBy> YSI </manufacturedBy>
        <model> 58 </model>
        <identifyingNumber type="serialNumber"> s454165 </identifyingNumber>
        <history>
            <SensorCreation>
                <byWhom>
                    <Person>
                        <fullName> Some Guy </fullName>
                    </Person>
                </byWhom>
                <when> 2001-12-04 </when>
                <supportingDocuments>
                    <Reference id="YSI-12">
                        <authors>
                            <Person>
                                <fullName> Some Author </fullName>
                            </Person>
                        </authors>
                        <publishingDate> 2001-01-01 </publishingDate>
                        <documentTitle> Blueprints for This Sensor </documentTitle>
                        <documentNumber> YSI-12345-678 </documentNumber>
                    </Reference>
                </supportingDocuments>
            </SensorCreation>
            <SensorDeployment>
                <byWhom>
                    <Person>
                        <fullName>SomeOther Guy</fullName>
                    </Person>
                </byWhom>
                <where> Someplace, Somewhere </where>
                <when> 2001-12-04 </when>
                <description> Attached to northern most footing on Some Bridge at a depth
                    of 5 meters </description>
            </SensorDeployment>
        </history>
        <properties>
            <genericProperty name="sensorTechnology" dataType="xs:string"> rapid
                pulse </genericProperty>
            <genericProperty name="measurementMethod" dataType="xs:string"> EPA
```

```
                accepted </genericProperty>
            <genericProperty name="membraneThickness" dataType="xs:double"
                uom="#mil"> 1.0 </genericProperty>
            <genericProperty name="membraneType" dataType="xs:string"> Teflon
                </genericProperty>
            <genericProperty name="probeSolutionType" dataType="xs:string"> Na2SO4
                </genericProperty>
        </properties>
    </SensorSpecification>
</describedBy>
```

2.2.8. Document Metadata

The *documentMetadata* property takes a *DocumentSupportType* as its value and provides metadata for the document, itself. This metadata includes (1) the documents history, such as when and by whom the document was created and modified, (2) assumptions that were used to create the sensor model, (3) general descriptions of the sensor, and (4) references for the sensor through literature bibliography or URL links.

As shown in the UML of Figure 2.15, *documentMetadata* includes the *history* property which contains one or more *DocumentAction* types. Current derived classes for

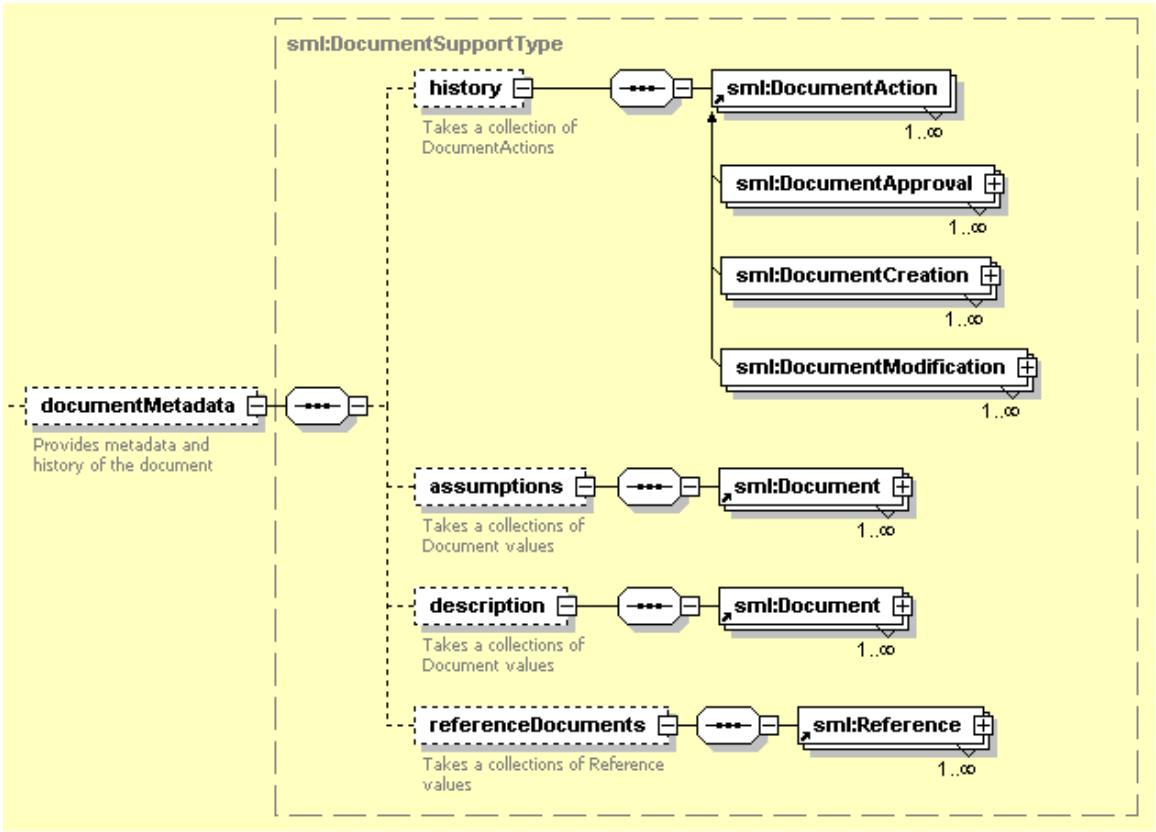

Figure 2.15. Schema diagram for DocumentSupport.

DocumentAction include *DocumentCreation* or *DocumentModification*. Both the optional *assumptions* and *description* properties take one or more *Document*(s) as their values. The *referenceDocuments* property takes a *Reference* as its value.

2.2.9. Response Characteristics

The sensor's response to stimuli, its characteristics, and behavior under certain environmental conditions are all defined within the response property. The *ResponseType* that supports these specifications is simple in design yet highly flexible. As shown in Figure 2.16, *Response* consist of any number of *_characteristic* properties. The property _characteristic is abstract and will serve as a base for many general and specific sensor characteristics. In Figure 2.16, are several such general characteristics that have been derived from _characteristic, including accuracy, capacity, dynamicRange, measurementMethod, operationalRange, resolution, survivableRange, and threshold.

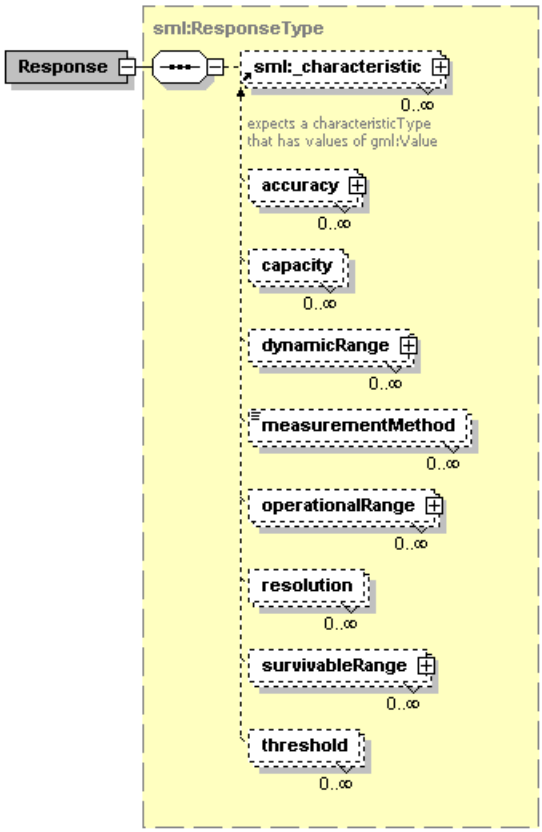

Figure 2.16. Schema diagram for basic Response.

The value for *_characteristic* is a *ValuePropertyType* which contains a value derived from *_value*. The schema definitions for *_value* are within the Observations and Measurement schema which currently reside within GML. Examples of derived definitions for _value are shown in Figure 2.17, and are defined within Cox (2002). These include types such as scalar, quantity, category, ordered category, and position, as

well as various collections of these fundamental value types. Each type derived from _value has as an attribute, a URI link to an ObservableType, such as temperature, radiation, dissolvedO2, etc. Each value type also has a specification for units of measure where appropriate.

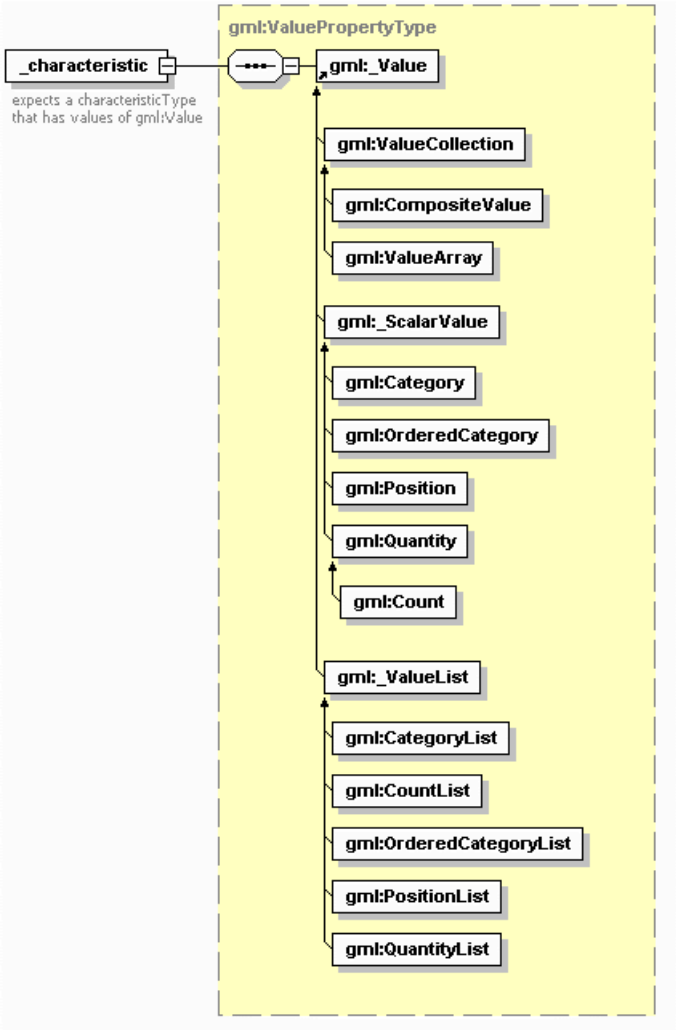

Figure 2.17. Schema Diagram showing _characteristic and ValuePropertyType realtionship. Notice that valuePropertyType is a part of the Observations schema residing within GML.

Many definitions for a characteristic can be quite simple, consisting essentially of a name and an appropriate *ScalarValueType*. This is the case for the *resolution* characteristic defined in Figure 2.18.

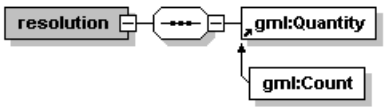

Figure 2.18. Schema diagram for resolution characteristic.

Other definitions for derived characteristics can be slightly more complex, such as the definition for *accuracy* shown in Figure 2.19. Thus, accuracy can be specified as either as "less than 5%" for example or "minus 2% and plus 5%". One can describe conditions for the accuracy measurement, such as "per deg Centigrade", "at 2 inches per hour", or "under natural daylight conditions". As an alternative, one can define the accuracy conditions as being within a range of observation values or between of range of environmental conditions.

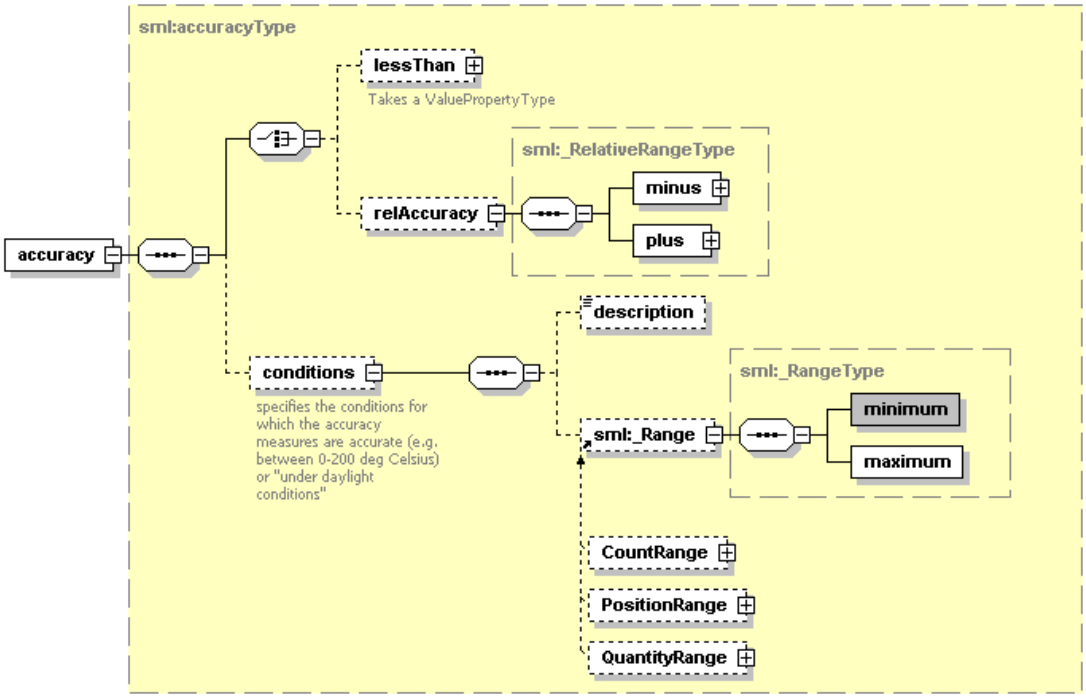

Figure 2.19. Schema diagram for accuracy characteristics.

An example of the response property for the YSI Wind Speed Sensor might be:

```
<response id=ysi_wss_0001>
    <dynamicRange>
        <minimum>
            <Quantity observableType=#windSpeed unitOfMeasure=#mph> 0 </Quantity>
        </minimum>
        <maximum>
            <Quantity observableType=#windSpeed unitOfMeasure=#mph> 134 </Quantity>
```

```
            </maximum>
        </dynamicRange>
        <threshold>
            <Quantity observableType=#windSpeed unitOfMeasure=#mph> 2.2 </Quantity>
        </threshold>
        <survivableRange>
            <maximum>
                <Quantity observableType=#windSpeed unitOfMeasure=#mph> 220 </Quantity>
            </maximum>
        </survivableRange>
        <operationalRange>
            <minimum>
                <Quantity observableType=#airTemperature unitOfMeasure=#celsius> -40
                </Quantity>
            </minimum>
            <maximum>
                <Quantity observableType=#airTemperature unitOfMeasure=#celsius> 40
                </Quantity>
            </maximum>
        </ operationalRange >
</response>
```

Notice in this example that for the *operationRange* and *survivableRange*, the *observableType* can be the same as the *observableType* measured by the sensor (i.e. in the *measured* property), or it could be different than the measured *observableType*.

The limited number of general characteristics shown in Figure 2.16 may be sufficient to support a large number of simple sensors. However, it is assumed that new definitions may need to be provided with SensorML instance documents or defined within application schema, as has been done for radiation sensors in Figure 2.12.

Within the *RadiationResponseType* (as defined in the application schema, radiationResponse.xsd) are several newly defined properties derived from *_characteristic*, as well as some of the general characteristics defined in the base *response* schema. Newly derived characteristics include *peakWavelength, bandWidth, spectralRange, polarizationAngle, polarizationDirection*, and *spectralResponse*, which include a *spectralCurve.*

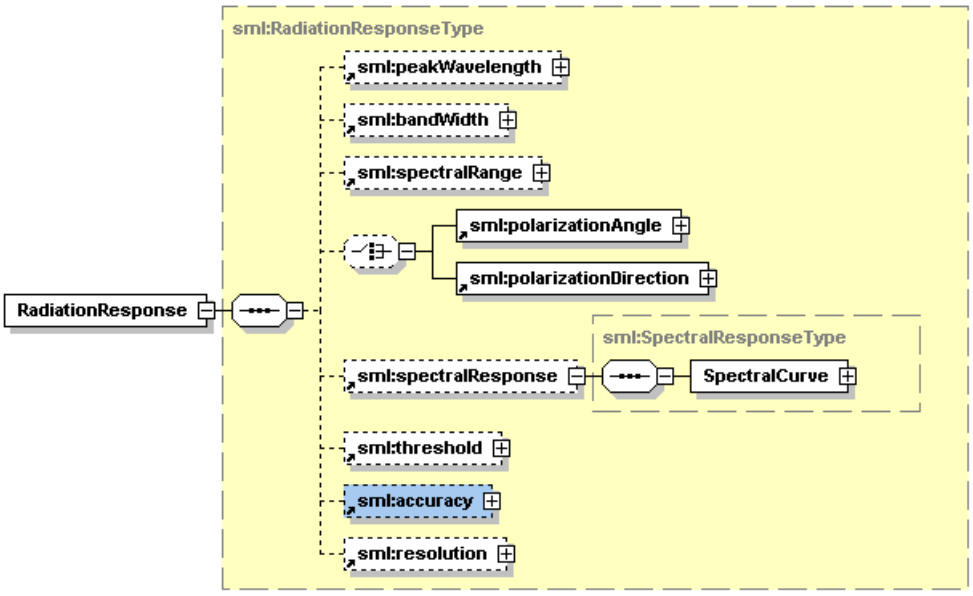

Figure 2.20. Schema diagram for specific RadiationResponse.

2.2.10. Sample Geometry

Issue Name: [Sample Geometry – Out of Scope. (meb, 2002-04-16)]

Issue Description: Geometry out of Scope for OWS 1.1.

OWS 1.2 will consider:

- Original concepts for sample geometry were presented in Botts (2001); these will be adapted to the new SensorML schema
- Frame definition plus a shape/size definition …. needs to account for distance, angle, and temporal specifications and units
- Sample geometry may be relevant to some in-situ sensors (i.e. how big of a volume do you sample?), but is most useful perhaps for remote sensors

Resolution:

2.2.11. Collection Geometry and Dynamics

Issue Name: [Collection Geometry – Out of Scope. (meb, 2002-04-16)]

Issue Description: Collection Geometry and Dynamics out of Scope for OWS 1.1.

Original concepts for collection geometry, particularly for scanner and profilers, were presented in Botts (2001); these will be adapted to the new SensorML schema during OWS 1.2

OWS 1.2 will consider following Sections:

- Collection Overview
- Sensor Arrays
- Scanners
- Frame Cameras

Also, *CollectionGeometry* may end up only being a part of *SensorCollection*, not *Sensor*

Resolution:

Annex A. XML Schemas for SensorML (normative)

A.1 SensorML.xsd.

SensorML.xsd provides the basic definitions for Sensor and SensorGroup, as well as some general utility definitions.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael E Botts (University of Alabama in Huntsville) -
->
<xs:schema targetNamespace="http://www.opengis.net/sensorML" xmlns:gml="http://www.opengis.net/gml"
xmlns:sml="http://www.opengis.net/sensorML" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xlink="http://www.w3.org/1999/xlink" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:annotation>
     <xs:appinfo>sensorML.xsd v0.4a 2002-04-18</xs:appinfo>
     <xs:documentation>Ongoing xs:schema definition for SensorML</xs:documentation>
    </xs:annotation>
    <!-- ========================================================================
        includes and imports
    ========================================================================== -->
    <xs:include schemaLocation="value.xsd"/>
    <xs:include schemaLocation="action.xsd"/>
    <xs:include schemaLocation="platform.xsd"/>
    <xs:include schemaLocation="response.xsd"/>
    <xs:include schemaLocation="frame.xsd"/>
    <!-- ========================================================================
        global elements
    ========================================================================== -->
    <xs:element name="Sensor" type="sml:SensorType">
     <xs:annotation>
         <xs:documentation>attribute: id</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="SensorGroup" type="sml:SensorGroupType" substitutionGroup="sml:Sensor"/>
    <xs:element name="SensorReference" type="sml:SensorReferenceType" substitutionGroup="sml:Sensor"/>
    <xs:element name="SensorMember" type="sml:SensorType"/>
    <!--
    <xs:element name="SensorArray" type="sml:SensorArrayType" substitutionGroup="sml:Sensor"/>
    <xs:element name="SensorPackage" type="sml:SensorPackageType" substitutionGroup="sml:Sensor"/>
    -->
    <xs:element name="_GroupGeometry" type="sml:_GroupGeometryType" abstract="true"/>
    <xs:element name="GenericProperty" type="sml:GenericPropertyType"/>
    <xs:element name="Document" type="sml:DocumentType"/>
    <xs:element name="DocumentSupport" type="sml:DocumentSupportType"/>
    <xs:element name="SensorSpecification" type="sml:SensorSpecificationType"/>
    <xs:element name="ObservedProperty" type="sml:ObservedPropertyType">
     <xs:annotation>
         <xs:documentation>Attributes: id and observableType</xs:documentation>
     </xs:annotation>
    </xs:element>
    <!-- ========================================================================
        sensor components
    ========================================================================== -->
    <xs:complexType name="SensorType">
     <xs:sequence>
         <xs:element name="shortName" type="xs:string"/>
         <xs:element name="longName" type="xs:string"/>
         <xs:element name="sensorType" type="xs:anyURI">
```

```
      <xs:annotation>
        <xs:documentation>Pointer to a sensor dictionary entry</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="lastModification" type="xs:dateTime" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Date the document was last modified</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="measures">
      <xs:annotation>
        <xs:documentation>describes what property is measured by the sensor</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="sml:ObservedProperty"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="attachedTo" type="sml:PlatformType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Points to the platform on which the sensor is attached</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="orientedPosition" type="sml:OrientedPositionType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Through a Frame definition, provides location and orientation of
          sensor</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="describedBy" type="sml:SensorSpecificationType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Provides metadata and history of the sensor</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="documentMetadata" type="sml:DocumentSupportType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Provides metadata and history of the document</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="response" type="sml:ResponseType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Defines various response and quality characteristics for the sensor taking value of
          _characteristic</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="sampleGeometry" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Defines the geometry of the measured sample</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
<!-- ========================================================================== -->
<xs:complexType name="ObservedPropertyType">
  <xs:annotation>
    <xs:documentation>observableType attributes points to entry in observables dictionary
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="shortName" type="xs:string"/>
    <xs:element name="longName" type="xs:string" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <xs:attribute name="observableType" type="xs:anyURI" use="required"/>
```

```
</xs:complexType>
<!-- ================================================================== -->
<xs:complexType name="GenericPropertyType">
 <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="dataType" type="xs:anySimpleType" use="optional"/>
      <xs:attribute name="uom" type="xs:anyURI" use="optional"/>
    </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<!-- ================================================================== -->
<xs:complexType name="SensorSpecificationType">
 <xs:sequence>
    <xs:element name="manufacturedBy" type="xs:string"/>
    <xs:element name="model" type="xs:string"/>
    <xs:element name="identifyingNumber" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="history" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Takes a collection of SensorActions</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="sml:SensorAction" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="properties" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Takes a collection of GenericProperty values</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="sml:GenericProperty" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="description" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Takes a collections of Document values</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="sml:Document" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
 </xs:sequence>
</xs:complexType>
<!-- ================================================================== -->
<xs:complexType name="DocumentType">
 <xs:sequence>
    <xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
    <xs:element name="date" type="xs:dateTime">
      <xs:annotation>
        <xs:documentation>type xs:dateTime</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="subject" type="xs:string" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>type xs:string</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="description" type="xs:string">
      <xs:annotation>
        <xs:documentation>type xs:string</xs:documentation>
```

```
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <!-- ================================================================== -->
  <xs:complexType name="DocumentSupportType">
    <xs:sequence>
      <xs:element name="history" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Takes a collection of DocumentActions</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="sml:DocumentAction" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="assumptions" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Takes a collections of Document values</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="sml:Document" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="description" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Takes a collections of Document values</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="sml:Document" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="referenceDocuments" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Takes a collections of Reference values</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="sml:Reference" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <!-- ========================================================== -->
  <xs:complexType name="SensorReferenceType">
    <xs:attribute name="sensor" type="xs:anyURI" use="required"/>
  </xs:complexType>
  <!-- ========================================================================
       sensor collections
  ========================================================================== -->
  <xs:complexType name="SensorGroupType">
    <xs:complexContent>
      <xs:extension base="sml:SensorType">
        <xs:sequence>
          <xs:element name="members" minOccurs="0">
            <xs:complexType>
              <xs:sequence>
                <xs:element ref="sml:SensorReference" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="sml:Sensor" minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
```

```
            </xs:complexType>
          </xs:element>
          <xs:element name="collectionGeometry" type="sml:_GroupGeometryType" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="_GroupGeometryType" abstract="true"/>
</xs:schema>
```

A.2 Action.xsd.

Action.xsd provide definitions for actions used in history property. These include SensorAction and DocumentAction and their derivatives.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael E Botts (University of Alabama in Huntsville) -
->
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema targetNamespace="http://www.opengis.net/sensorML" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:sml="http://www.opengis.net/sensorML" xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <annotation>
    <documentation>sensorML.xsd v0.4a 2002-04-18</documentation>
    <documentation>Defines sensor and document actions to be used in smlhistories</documentation>
  </annotation>
  <!-- ========================================================================
       global elements
  ========================================================================== -->
  <element name="Person" type="sml:PersonType"/>
  <element name="Reference" type="sml:ReferenceType"/>
  <element name="DocumentAction" type="sml:DocumentActionType"/>
  <element name="DocumentCreation" type="sml:DocumentCreationType"
      substitutionGroup="sml:DocumentAction"/>
  <element name="DocumentModification" type="sml:DocumentModificationType"
      substitutionGroup="sml:DocumentAction"/>
  <element name="DocumentApproval" type="sml:DocumentApprovalType"
      substitutionGroup="sml:DocumentAction"/>
  <element name="SensorAction" type="sml:SensorActionType"/>
  <element name="SensorCreation" type="sml:SensorCreationType" substitutionGroup="sml:SensorAction"/>
  <element name="SensorDeployment" type="sml:SensorDeploymentType"
      substitutionGroup="sml:SensorAction"/>
  <element name="SensorInspection" type="sml:SensorInspectionType"
      substitutionGroup="sml:SensorAction"/>
  <element name="SensorDecommision" type="sml:SensorDecommissionedType"
      substitutionGroup="sml:SensorAction"/>
  <!-- ========================================================================
       some utility types ... will be replaced if a better standard schema exists
  ========================================================================== -->
  <complexType name="PersonType">
    <sequence>
      <choice>
        <element name="fullName" type="string"/>
        <sequence>
          <element name="firstName" type="string"/>
          <element name="middleName" type="string"/>
          <element name="lastName" type="string"/>
        </sequence>
      </choice>
      <element name="title" type="string" minOccurs="0"/>
      <element name="email" type="string" minOccurs="0"/>
```

```
      <element name="phone" type="string" minOccurs="0"/>
    </sequence>
  </complexType>
  <complexType name="ReferenceType">
    <sequence>
      <element name="authors">
        <complexType>
          <sequence>
            <element ref="sml:Person" maxOccurs="unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element name="publishingDate" type="date" minOccurs="0"/>
      <element name="documentTitle" type="string" minOccurs="0"/>
      <element name="documentNumber" type="string" minOccurs="0" maxOccurs="unbounded"/>
      <element name="publication" type="string" minOccurs="0"/>
      <element name="abstract" type="string" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
  </complexType>
  <complexType name="ActionType" abstract="true"/>
  <!-- ================================================================
       document actions
  ================================================================= -->
  <complexType name="DocumentActionType" abstract="true">
    <complexContent>
      <extension base="sml:ActionType"/>
    </complexContent>
  </complexType>
  <!-- ================================================================ -->
  <complexType name="DocumentCreationType">
    <complexContent>
      <extension base="sml:DocumentActionType">
        <sequence>
          <element name="byWhom" minOccurs="0">
            <complexType>
              <sequence>
                <element ref="sml:Person" maxOccurs="unbounded"/>
              </sequence>
            </complexType>
          </element>
          <element name="when" type="dateTime"/>
          <element name="versionNumber" type="string" minOccurs="0"/>
          <element name="supportingDocuments" minOccurs="0">
            <complexType>
              <sequence>
                <element ref="sml:Reference" maxOccurs="unbounded"/>
              </sequence>
            </complexType>
          </element>
          <element name="description" type="string" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <!-- ================================================================ -->
  <complexType name="DocumentModificationType">
    <complexContent>
      <extension base="sml:DocumentActionType">
        <sequence>
          <element name="byWhom" minOccurs="0">
            <complexType>
              <sequence>
                <element ref="sml:Person" maxOccurs="unbounded"/>
              </sequence>
            </complexType>
```

```
            </element>
            <element name="when" type="dateTime"/>
            <element name="versionNumber" type="string"/>
            <element name="supportingDocuments" minOccurs="0" maxOccurs="unbounded">
              <complexType>
                <sequence>
                  <element ref="sml:Reference"/>
                </sequence>
              </complexType>
            </element>
            <element name="modification" type="string"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
    <!-- ========================================================================== -->
    <complexType name="DocumentApprovalType">
      <complexContent>
        <extension base="sml:DocumentActionType">
          <sequence>
            <element name="byWhom" minOccurs="0">
              <complexType>
                <sequence>
                  <element ref="sml:Person" maxOccurs="unbounded"/>
                </sequence>
              </complexType>
            </element>
            <element name="when" type="dateTime"/>
            <element name="supportingDocuments" minOccurs="0">
              <complexType>
                <sequence>
                  <element ref="sml:Reference"/>
                </sequence>
              </complexType>
            </element>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
    <!-- ====================================================================
        sensor actions
    ========================================================================== -->
    <complexType name="SensorActionType" abstract="true">
      <complexContent>
        <extension base="sml:ActionType"/>
      </complexContent>
    </complexType>
    <!-- ========================================================================== -->
    <complexType name="SensorCreationType">
      <complexContent>
        <extension base="sml:SensorActionType">
          <sequence>
            <element name="when" type="date"/>
            <element name="byWhom" minOccurs="0">
              <complexType>
                <sequence>
                  <element ref="sml:Person" maxOccurs="unbounded"/>
                </sequence>
              </complexType>
            </element>
            <element name="serialNumber" type="string" minOccurs="0"/>
            <element name="supportingDocuments" minOccurs="0">
              <complexType>
                <sequence>
                  <element ref="sml:Reference" maxOccurs="unbounded"/>
                </sequence>
```

```
            </complexType>
          </element>
          <element name="description" type="string" minOccurs="0"/>
        </sequence>
      </extension>
 </complexContent>
</complexType>
<!-- ========================================================================== -->
<complexType name="SensorDeploymentType">
 <complexContent>
    <extension base="sml:SensorActionType">
      <sequence>
        <element name="where" type="string"/>
        <element name="when" type="date"/>
        <element name="byWhom" minOccurs="0">
          <complexType>
            <sequence>
              <element ref="sml:Person" maxOccurs="unbounded"/>
            </sequence>
          </complexType>
        </element>
        <element name="supportingDocuments" minOccurs="0">
          <complexType>
            <sequence>
              <element ref="sml:Reference" maxOccurs="unbounded"/>
            </sequence>
          </complexType>
        </element>
        <element name="description" type="string" minOccurs="0"/>
      </sequence>
    </extension>
 </complexContent>
</complexType>
<!-- ========================================================================== -->
<complexType name="SensorInspectionType">
 <complexContent>
    <extension base="sml:SensorActionType">
      <sequence>
        <element name="when" type="date"/>
        <element name="byWhom" minOccurs="0">
          <complexType>
            <sequence>
              <element ref="sml:Person" maxOccurs="unbounded"/>
            </sequence>
          </complexType>
        </element>
        <element name="problem" type="string" minOccurs="0" maxOccurs="unbounded"/>
        <element name="actionTaken" type="string" minOccurs="0" maxOccurs="unbounded"/>
        <element name="supportingDocuments" minOccurs="0">
          <complexType>
            <sequence>
              <element ref="sml:Reference" maxOccurs="unbounded"/>
            </sequence>
          </complexType>
        </element>
        <element name="description" type="string" minOccurs="0"/>
      </sequence>
    </extension>
 </complexContent>
</complexType>
<!-- ========================================================================== -->
<complexType name="SensorDecommissionedType">
 <complexContent>
    <extension base="sml:SensorActionType">
      <sequence>
        <element name="when" type="date"/>
```

```
            <element name="why" type="string"/>
            <element name="byWhom" minOccurs="0">
              <complexType>
                <sequence>
                  <element ref="sml:Person" maxOccurs="unbounded"/>
                </sequence>
              </complexType>
            </element>
            <element name="supportingDocuments" minOccurs="0">
              <complexType>
                <sequence>
                  <element ref="sml:Reference"/>
                </sequence>
              </complexType>
            </element>
            <element name="description" type="string" minOccurs="0"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
</schema>
```

A.3 Response.xsd

Defines base schema for a sensor's response characteristics, such as sensitivty, accuracy, thresholds, operating conditions, etc.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael E Botts (University of Alabama in Huntsville) -
->
<xs:schema targetNamespace="http://www.opengis.net/sensorML" xmlns:sml="http://www.opengis.net/sensorML"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:annotation>
     <xs:documentation>sensorML.xsd v0.4a 2002-04-18</xs:documentation>
     <xs:documentation>Defines basic ResponseType definition and commonly used sensor
         characteristics</xs:documentation>
    </xs:annotation>
    <!-- ========================================================================
        includes and xs:imports
    ========================================================================== -->
    <xs:include schemaLocation="value.xsd"/>
    <!--
    <xs:import namespace="http://www.opengis.net/gml" schemaLocation="gml_skeleton.xsd"/>
    -->
    <!-- ========================================================================
        global  elements
    ========================================================================== -->
    <xs:element name="Response" type="sml:ResponseType"/>
    <xs:element name="_Quality" type="sml:_QualityType" abstract="true"/>
    <xs:element name="_Range" type="sml:_RangeType" abstract="true"/>
    <xs:element name="_RelativeRange" type="sml:_RelativeRangeType" abstract="true"/>
    <xs:element name="_characteristic" type="sml:ValuePropertyType" abstract="true">
     <xs:annotation>
         <xs:documentation>expects a characteristicType that has values of sml:Value</xs:documentation>
     </xs:annotation>
    </xs:element>
    <!-- ========================================================================
        common characteristics measures
    ========================================================================== -->
    <xs:element name="accuracy" type="sml:accuracyType" substitutionGroup="sml:_characteristic"/>
```

```
<xs:element name="resolution" substitutionGroup="sml:_characteristic">
 <xs:complexType>
    <xs:complexContent>
      <xs:extension base="sml:QuantityType">
        <xs:sequence>
          <xs:element ref="sml:Quantity"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
 </xs:complexType>
</xs:element>
<xs:element name="threshold" type="sml:QuantityType" substitutionGroup="sml:_characteristic"/>
<xs:element name="capacity" type="sml:QuantityType" substitutionGroup="sml:_characteristic"/>
<xs:element name="survivableRange" type="sml:survivableRangeType"
    substitutionGroup="sml:_characteristic"/>
<xs:element name="operationalRange" type="sml:_RangeType" substitutionGroup="sml:_characteristic"/>
<xs:element name="dynamicRange" type="sml:_RangeType" substitutionGroup="sml:_characteristic"/>
<xs:element name="measurementMethod" type="xs:string" substitutionGroup="sml:_characteristic"/>
<!-- ================================================================================
   base Response types
================================================================================ -->
<xs:complexType name="ResponseType">
 <xs:sequence>
    <xs:element ref="sml:_characteristic" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="id" type="xs:ID" use="optional"/>
</xs:complexType>
<xs:complexType name="_RangeType" abstract="true">
 <xs:annotation>
    <xs:documentation>absolute range</xs:documentation>
 </xs:annotation>
 <xs:sequence>
    <xs:element name="minimum" type="sml:ScalarValueType"/>
    <xs:element name="maximum" type="sml:ScalarValueType"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="_RelativeRangeType" abstract="true">
 <xs:annotation>
    <xs:documentation>relative range</xs:documentation>
 </xs:annotation>
 <xs:sequence>
    <xs:element name="minus" type="sml:ValuePropertyType"/>
    <xs:element name="plus" type="sml:ValuePropertyType"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="_QualityType" abstract="true"/>
<!-- ================================================================================
   RangeType and elements
================================================================================ -->
<xs:element name="QuantityRange" type="sml:QuantityRangeType" substitutionGroup="sml:_Range"/>
<xs:element name="PositionRange" type="sml:PositionRangeType" substitutionGroup="sml:_Range"/>
<xs:element name="CountRange" type="sml:CountRangeType" substitutionGroup="sml:_Range"/>
<xs:complexType name="QuantityRangeType">
 <xs:complexContent>
    <xs:restriction base="sml:_RangeType">
      <xs:sequence>
        <xs:element name="minimum" type="sml:QuantityType"/>
        <xs:element name="maximum" type="sml:QuantityType"/>
      </xs:sequence>
    </xs:restriction>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="PositionRangeType">
 <xs:complexContent>
    <xs:restriction base="sml:_RangeType">
      <xs:sequence>
```

```
            <xs:element name="minimum" type="sml:PositionType"/>
            <xs:element name="maximum" type="sml:PositionType"/>
          </xs:sequence>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="CountRangeType">
      <xs:complexContent>
        <xs:restriction base="sml:_RangeType">
          <xs:sequence>
            <xs:element name="minimum" type="sml:CountType"/>
            <xs:element name="maximum" type="sml:CountType"/>
          </xs:sequence>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="accuracyType">
      <xs:sequence>
        <xs:choice>
          <xs:element name="lessThan" type="sml:ValuePropertyType" minOccurs="0">
            <xs:annotation>
              <xs:documentation>Takes a ValuePropertyType</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="relAccuracy" type="sml:_RelativeRangeType" minOccurs="0"/>
        </xs:choice>
        <xs:element name="conditions" minOccurs="0">
          <xs:annotation>
            <xs:documentation>specifies the conditions for which the accuracy measures are accurate (e.g.
              between 0-200 deg Celsius) or "under daylight conditions"</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="description" type="xs:string" minOccurs="0"/>
              <xs:element ref="sml:_Range" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="survivableRangeType">
      <xs:complexContent>
        <xs:restriction base="sml:_RangeType">
          <xs:sequence>
            <xs:element name="minimum" type="sml:ScalarValueType" minOccurs="0"/>
            <xs:element name="maximum" type="sml:ScalarValueType" minOccurs="0"/>
          </xs:sequence>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

A.4 RadiationResponse

An example of a specific ResponseType, RadiationResponse provides schema for defining the response of a radiometer.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael E Botts (University of Alabama in Huntsville) -
->
```

```
<xs:schema targetNamespace="http://www.opengis.net/sensorML"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:sml="http://www.opengis.net/sensorML"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:annotation>
     <xs:documentation>sensorML.xsd v0.4a 2002-04-18</xs:documentation>
     <xs:documentation>Defines specific responseType for radiation sensors</xs:documentation>
    </xs:annotation>
    <!-- ================================================================================
       includes and xs:imports
    ================================================================================ -->
    <xs:import namespace="http://www.opengis.net/sensorML" schemaLocation="../../base/value.xsd"/>
    <xs:import namespace="http://www.opengis.net/sensorML" schemaLocation="../../base/response.xsd"/>
    <!-- ================================================================================
       elements
    ================================================================================ -->
    <xs:element name="RadiationResponse" type="sml:RadiationResponseType"
        substitutionGroup="sml:Response"/>
    <xs:element name="peakWavelength" type="sml:QuantityType" substitutionGroup="sml:_characteristic">
     <xs:annotation>
        <xs:documentation>QuantityType</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="bandWidth" type="sml:QuantityType" substitutionGroup="sml:_characteristic">
     <xs:annotation>
        <xs:documentation>QuantityType</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="spectralResponse" type="sml:SpectralResponseType"
        substitutionGroup="sml:_characteristic"/>
    <xs:element name="polarizationDirection" type="sml:CategoryType" substitutionGroup="sml:_characteristic">
     <xs:annotation>
        <xs:documentation>CategoryType</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="polarizationAngle" type="sml:QuantityType" substitutionGroup="sml:_characteristic">
     <xs:annotation>
        <xs:documentation>QuantityType</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="spectralRange" type="sml:QuantityRangeType" substitutionGroup="sml:_characteristic"/>
    <!-- ================================================================================
       complex types
    ================================================================================ -->
    <xs:complexType name="RadiationResponseType">
     <xs:complexContent>
        <xs:restriction base="sml:ResponseType">
          <xs:sequence>
             <xs:element ref="sml:peakWavelength" minOccurs="0"/>
             <xs:element ref="sml:bandWidth" minOccurs="0"/>
             <xs:element ref="sml:spectralRange" minOccurs="0"/>
             <xs:choice minOccurs="0">
                <xs:element ref="sml:polarizationAngle"/>
                <xs:element ref="sml:polarizationDirection"/>
             </xs:choice>
             <xs:element ref="sml:spectralResponse" minOccurs="0"/>
             <xs:element ref="sml:threshold" minOccurs="0"/>
             <xs:element ref="sml:accuracy" minOccurs="0"/>
             <xs:element ref="sml:resolution" minOccurs="0"/>
          </xs:sequence>
        </xs:restriction>
     </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="SpectralResponseType">
     <xs:sequence>
        <xs:element name="SpectralCurve">
          <xs:complexType>
```

```
            <xs:sequence>
              <xs:element name="frequency">
                <xs:annotation>
                    <xs:documentation>QuantityList</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:sequence>
                     <xs:element ref="sml:QuantityList"/>
                    </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="power">
                <xs:annotation>
                    <xs:documentation>QuantityList</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:sequence>
                     <xs:element ref="sml:QuantityList"/>
                    </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
</xs:schema>
```

A.5 Platform

Provide intial schema for defining platforms as they relate to sensors.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael E Botts (University of Alabama in Huntsville) -
->
<xs:schema targetNamespace="http://www.opengis.net/sensorML"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:sml="http://www.opengis.net/sensorML"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:annotation>
     <xs:appinfo>platform.xsd v0.4a 2002-04-18</xs:appinfo>
     <xs:documentation>basic definitions for platforms used in conjunction with SensorML</xs:documentation>
    </xs:annotation>
    <!-- ========================================================================
        includes and imports
    ========================================================================== -->
    <xs:include schemaLocation="value.xsd"/>
    <xs:include schemaLocation="sensorML.xsd"/>
    <xs:include schemaLocation="frame.xsd"/>
    <!-- ========================================================================
        global elements
    ========================================================================== -->
    <xs:element name="Platform" type="sml:PlatformType">
     <xs:annotation>
         <xs:documentation>Has required attribute: id</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="PlatformReference" type="sml:PlatformReferenceType"
         substitutionGroup="sml:Platform">
     <xs:annotation>
         <xs:documentation>Provides ability to link to internally or externally defined platform</xs:documentation>
     </xs:annotation>
```

```
</xs:element>
<xs:element name="StationaryPlatform" type="sml:StationaryPlatformType"
    substitutionGroup="sml:Platform"/>
<xs:element name="AttachedPlatform" type="sml:AttachedPlatform" substitutionGroup="sml:Platform"/>
<xs:element name="SatellitePlatform" type="sml:SatellitePlatformType" substitutionGroup="sml:Platform"/>
<xs:element name="AircraftPlatform" type="sml:AircraftPlatformType" substitutionGroup="sml:Platform"/>
<xs:element name="LandVehiclePlatform" type="sml:LandVehiclePlatformType"
    substitutionGroup="sml:Platform"/>
<xs:element name="WaterVehiclePlatform" type="sml:WaterVehiclePlatformType"
    substitutionGroup="sml:Platform"/>
<!-- ================================================================
   Fundamental types
================================================================== -->
<xs:complexType name="PlatformType" abstract="true">
 <xs:annotation>
    <xs:documentation>Has required attribute: id</xs:documentation>
 </xs:annotation>
 <xs:sequence>
    <xs:element name="carries" minOccurs="0">
      <xs:annotation>
        <xs:documentation>Either an AttachedPlaform or Sensor</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="sml:AttachedPlatform" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="sml:PlatformReference" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="sml:SensorReference" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="sml:Sensor" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="locatedAt" type="sml:OrientedPositionType">
      <xs:annotation>
        <xs:documentation>Provides a Frame that is defined relative to an external Frame (perhaps a
          geospatial CRS)</xs:documentation>
      </xs:annotation>
    </xs:element>
 </xs:sequence>
 <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
<xs:complexType name="PlatformReferenceType">
 <xs:attribute name="platform" type="xs:anyURI" use="required"/>
</xs:complexType>
<!--================================================================================
     Platform Types
 ================================================================================-->
<xs:complexType name="StationaryPlatformType">
 <xs:complexContent>
    <xs:extension base="sml:PlatformType"/>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="SatellitePlatformType">
 <xs:complexContent>
    <xs:extension base="sml:PlatformType"/>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="AircraftPlatformType">
 <xs:complexContent>
    <xs:extension base="sml:PlatformType"/>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="LandVehiclePlatformType">
 <xs:complexContent>
    <xs:extension base="sml:PlatformType"/>
 </xs:complexContent>
</xs:complexType>
```

```
    <xs:complexType name="WaterVehiclePlatformType">
     <xs:complexContent>
        <xs:extension base="sml:PlatformType"/>
     </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="AttachedPlatform">
     <xs:complexContent>
        <xs:extension base="sml:PlatformType">
          <xs:sequence>
            <xs:element name="attachedTo">
              <xs:annotation>
                <xs:documentation>Either a Platform or PlatformReference</xs:documentation>
              </xs:annotation>
              <xs:complexType>
                <xs:choice>
                    <xs:element ref="sml:PlatformReference"/>
                    <xs:element ref="sml:Platform"/>
                </xs:choice>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:extension>
     </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

A.6 Frame

Provides definition for coordinate frame, including definition of frame by vectors and definition of frame by transforms.

********** **frame.xsd** **********

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael E Botts (University of Alabama in Huntsville) -
->
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<xs:schema targetNamespace="http://www.opengis.net/sensorML" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:sml="http://www.opengis.net/sensorML" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:annotation>
     <xs:documentation>Frame.xsd v0.4a 2002-04-18</xs:documentation>
     <xs:documentation>Defines Frame to be used fro positions and transforms in SensorML</xs:documentation>
    </xs:annotation>
    <!-- ========================================================================
        global elements
    ========================================================================= -->
    <xs:element name="Vector" type="sml:VectorType"/>
    <xs:element name="Matrix" type="sml:MatrixType"/>
    <!-- ============================================================
    Type Definitions
    ============================================================ -->
    <xs:complexType name="OrientedPositionType">
     <xs:choice>
        <xs:element name="Frame" type="sml:FrameByVectors"/>
        <xs:element name="Frame" type="sml:FrameByTransformation"/>
        <xs:element name="Frame" type="sml:_AbsoluteFrame"/>
     </xs:choice>
    </xs:complexType>
    <!-- ========================================================== -->
    <xs:complexType name="FrameByVectors">
```

```
    <xs:sequence>
      <xs:element name="component">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="sml:Vector"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="origin">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="sml:Vector" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="ID"/>
  </xs:complexType>
  <!-- ========================================================================== -->
  <xs:complexType name="FrameByTransformation">
    <xs:sequence>
      <xs:element name="transform">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="sml:Matrix"/>
          </xs:sequence>
          <xs:attribute name="relativeTo" type="anyURI"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="ID"/>
  </xs:complexType>
  <!-- ========================================================================== -->
  <xs:complexType name="VectorType">
    <xs:sequence>
      <xs:element name="component" type="xs:double"/>
    </xs:sequence>
    <xs:attribute name="id" type="ID"/>
  </xs:complexType>
  <!-- ========================================================================== -->
  <xs:complexType name="MatrixType1">
    <xs:sequence>
      <xs:element name="position" type="xs:double"/>
    </xs:sequence>
    <xs:attribute name="id" type="ID"/>
  </xs:complexType>
  <!-- ========================================================================== -->
  <xs:complexType name="MatrixType">
    <xs:choice>
      <xs:element name="Matrix" type="sml:MatrixType1"/>
      <xs:element name="Matrix" type="sml:RotationMatrix3X3PRY"/>
      <xs:element name="Matrix" type="sml:RotationMatrix3X3EulerAngles"/>
    </xs:choice>
    <xs:attribute name="id" type="ID"/>
  </xs:complexType>
  <!-- ========================================================================== -->
  <xs:complexType name="RotationMatrix3X3PRY">
    <xs:annotation>
      <xs:documentation>Matrix specified by pitch,roll,yaw angles</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="pitch" type="sml:RotationQuantity"/>
      <xs:element name="roll" type="sml:RotationQuantity"/>
      <xs:element name="yaw" type="sml:RotationQuantity"/>
    </xs:sequence>
    <xs:attribute name="id" type="ID"/>
```

```
</xs:complexType>
<!--   ========================================================================== -->
<xs:complexType name="RotationMatrix3X3EulerAngles">
 <xs:annotation>
     <xs:documentation>Matrix specified by Euler angles</xs:documentation>
 </xs:annotation>
 <xs:sequence>
     <xs:element name="theta" type="sml:RotationQuantity"/>
     <xs:element name="phi" type="sml:RotationQuantity"/>
     <xs:element name="psi" type="sml:RotationQuantity"/>
 </xs:sequence>
 <xs:attribute name="id" type="ID"/>
</xs:complexType>
<!--   ========================================================================== -->
<xs:complexType name="RotationMatrix2X2">
 <xs:annotation>
     <xs:documentation>Matrix specified by positive rotation = CCW</xs:documentation>
 </xs:annotation>
 <xs:sequence>
     <xs:element name="theta" type="sml:RotationQuantity"/>
 </xs:sequence>
 <xs:attribute name="id" type="ID"/>
</xs:complexType>
<xs:complexType name="RotationQuantity">
 <xs:simpleContent>
     <xs:extension base="double">
       <xs:attribute name="uom" type="anyURI" use="required"/>
     </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<!--=========================================================================
  Absolute Frames
 ================================================-->
<xs:complexType name="_AbsoluteFrame" abstract="true">
 <xs:sequence>
     <xs:element name="basis" type="anyURI">
       <xs:annotation>
         <xs:documentation>Points to a dictionary entry for Absolute Frames</xs:documentation>
       </xs:annotation>
     </xs:element>
     <xs:element name="axes">
       <xs:complexType>
         <xs:sequence>
           <xs:element name="x">
             <xs:complexType>
                 <xs:attribute name="id" type="ID" use="required"/>
             </xs:complexType>
           </xs:element>
           <xs:element name="y">
             <xs:complexType>
                 <xs:attribute name="id" type="ID" use="required"/>
             </xs:complexType>
           </xs:element>
           <xs:element name="z">
             <xs:complexType>
                 <xs:attribute name="id" type="ID" use="required"/>
             </xs:complexType>
           </xs:element>
         </xs:sequence>
       </xs:complexType>
     </xs:element>
 </xs:sequence>
 <xs:attribute name="id" type="ID" use="required"/>
</xs:complexType>
<xs:complexType name="EcfFrame" id="ecf">
 <xs:complexContent>
```

```
      <xs:restriction base="sml:_AbsoluteFrame">
        <xs:sequence>
          <xs:element name="basis" type="anyURI">
            <xs:annotation>
              <xs:documentation>Points to a dictionary entry for Absolute Frames</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="axes">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="x">
                 <xs:complexType>
                      <xs:attribute name="id" type="ID" use="required" fixed="ecf_x"/>
                 </xs:complexType>
                </xs:element>
                <xs:element name="y">
                 <xs:complexType>
                      <xs:attribute name="id" type="ID" use="required" fixed="ecf_y"/>
                 </xs:complexType>
                </xs:element>
                <xs:element name="z">
                 <xs:complexType>
                      <xs:attribute name="id" type="ID" use="required" fixed="ecf_z"/>
                 </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="id" fixed="ecf"/>
      </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="EciFrame" id="eci">
  <xs:complexContent>
      <xs:restriction base="sml:_AbsoluteFrame">
        <xs:sequence>
          <xs:element name="basis" type="anyURI">
            <xs:annotation>
              <xs:documentation>Points to a dictionary entry for Absolute Frames</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="axes">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="x">
                 <xs:complexType>
                      <xs:attribute name="id" type="ID" use="required" fixed="eci_x"/>
                 </xs:complexType>
                </xs:element>
                <xs:element name="y">
                 <xs:complexType>
                      <xs:attribute name="id" type="ID" use="required" fixed="eci_y"/>
                 </xs:complexType>
                </xs:element>
                <xs:element name="z">
                 <xs:complexType>
                      <xs:attribute name="id" type="ID" use="required" fixed="eci_z"/>
                 </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="id" fixed="eci"/>
      </xs:restriction>
```

```
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="J2000Frame" id="J2000">
 <xs:complexContent>
    <xs:restriction base="sml:_AbsoluteFrame">
      <xs:sequence>
        <xs:element name="basis" type="anyURI">
          <xs:annotation>
            <xs:documentation>Points to a dictionary entry for Absolute Frames</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="axes">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="x">
               <xs:complexType>
                    <xs:attribute name="id" type="ID" use="required" fixed="J2000_x"/>
               </xs:complexType>
              </xs:element>
              <xs:element name="y">
               <xs:complexType>
                    <xs:attribute name="id" type="ID" use="required" fixed="J2000_y"/>
               </xs:complexType>
              </xs:element>
              <xs:element name="z">
               <xs:complexType>
                    <xs:attribute name="id" type="ID" use="required" fixed="J2000_z"/>
               </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id" fixed="J2000"/>
    </xs:restriction>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="WGS84_LatLonAltFrame" id="wgs84">
 <xs:complexContent>
    <xs:restriction base="sml:_AbsoluteFrame">
      <xs:sequence>
        <xs:element name="basis" type="anyURI">
          <xs:annotation>
            <xs:documentation>Points to a dictionary entry for Absolute Frames</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="axes">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="x">
               <xs:complexType>
                    <xs:attribute name="id" type="ID" use="required" fixed="lat"/>
               </xs:complexType>
              </xs:element>
              <xs:element name="y">
               <xs:complexType>
                    <xs:attribute name="id" type="ID" use="required" fixed="lon"/>
               </xs:complexType>
              </xs:element>
              <xs:element name="z">
               <xs:complexType>
                    <xs:attribute name="id" type="ID" use="required" fixed="alt"/>
               </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
```

```
            </xs:element>
          </xs:sequence>
          <xs:attribute name="id" fixed="wgs84"/>
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

Annex B. Excerpts from original SensorML design regarding scanning geometry (Informative)

In the original design of SensorML (which preceded involvement with OGC), a primary focus of the design was toward defining the geometric and dynamic properties of remote sensors [Botts, 2001]. The focus was primarily on a wide variety of scanners described below.

Issue Name: [**WARNING – non schema material** . (meb, 2002-02-18)]

Issue Description:

The information provided in this Annex, should in no way be construed as part of the current definition of SensorML. It should instead serve as an initial conceptual design for supporting a *CollectionGeometryType* for scanners within future releases of the SensorML schema.

Resolution:

--- Beginning of excerpts from Botts (2001). ----

--- Note that section numbers relate to the section numbers in the original document ---

3.2. Sensor types.

Scanning Directions. Within the SensorML, a sensor can sample within three directions, referred to within the SensorML specification as sweep, elevation, and profile directions. For visualization purposes, these directions can be viewed as spherical coordinates within the sensor's right-handed coordinate frame, with the sensor's radiation detection occurring at the origin. As shown in Figure A1, the sweep angle always specifies rotation about the polar axis, Z, elevation measures the angle from the sensor's equatorial plane, and profile can be considered as a radial dimension along the line-of-sight. In the proposed SensorML, Z is always expected to designate the polar axis. Sweep and elevation angles can be specified as either positive or negative values, but they must follow the right-handed coordinate rule.

Using this general model, a conical scanner can be considered as a sensor in which the polar axis, Z, is roughly parallel to the main direction of the target, while the polar axis of a line scanner is roughly perpendicular to the main target direction (Figures A2 and A3). Thus, a line scanner and conic scanner are mathematically equivalent, and are distinguished only by the general direction of the polar axis relative to the target. Similarly, an imager is equivalent to a line scanner or conic scanner in which the entire array of pixels is sampled instantaneously.

Although the SensorML specification described below includes a ***sensorType*** element, this value will primarily be used for clarification, for cross-checking mounting angle

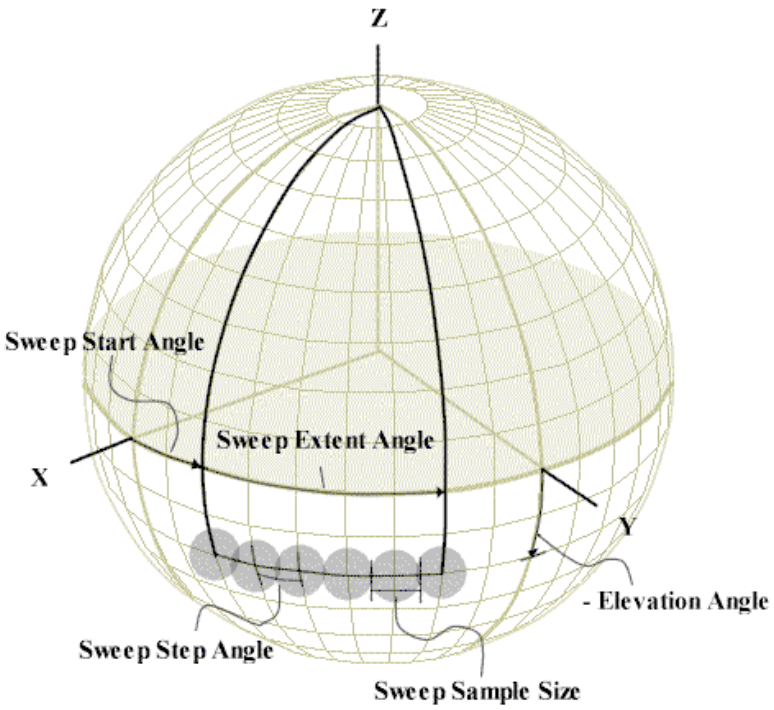

Figure A1. Sensor model for the proposed Sensor Model Language (SensorML), showing a sensor that sweeps at a constant elevation angle. Depending on the orientation of the sensor to the target body and on the timing of the scan, this sensor could either be a 1D conic scanner, line scanner, or imager.

assumptions, and possible future use with new sensor types. Most of the required knowledge for distinguishing sensor types will come from mounting and timing information.

Line-of-sight sensor. The line-of-sight sensor is the simplest, consisting of essentially a single look direction. It will thus have a constant sweep and elevation angle, sampling single or multiple points along this line-of-sight. Examples of line-of-sight sensors include:

Profile:
most altimeters (e.g. TOPEX/POSEIDEN altimeter)
most ground-based atmospheric sounders
most non-sweeping LIDAR

Conic-scanner. Conic scanners include those sensors that scan by sweeping around a rotation axis producing a cone pattern relative to the target body. Conic scanners can be one to three dimensions, possibly including sweep, elevation, and/or profile components (Figure A2). Examples of conic scanners include SSM/I (1D), which sweeps around a

single axis at a constant elevation angle, and ground-based Doppler Radar (3D), which sweeps around a vertical axis in a conic pattern at varying elevation angles, and measures atmospheric properties at multiple distances along the instantaneous line-of-sight. These and other examples include:

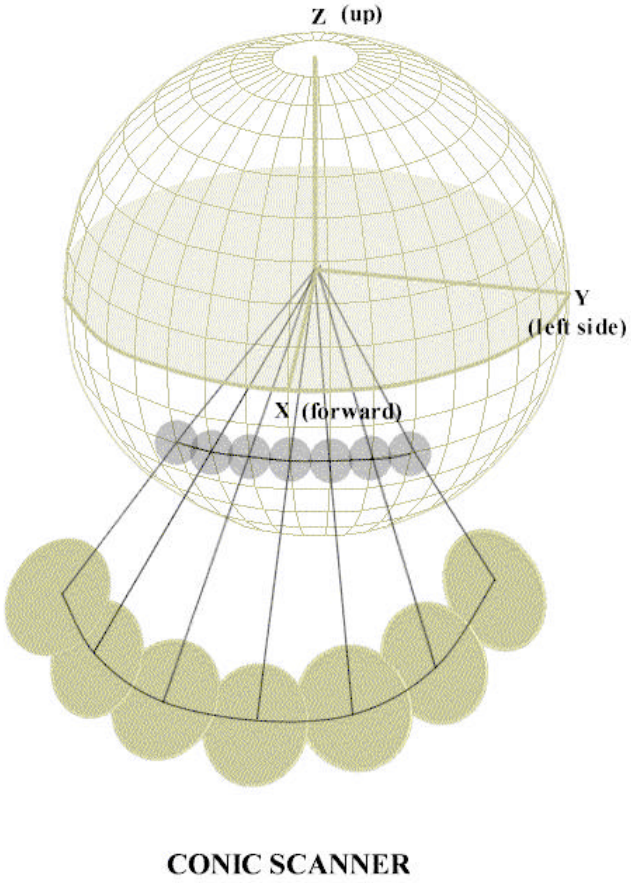

Figure A2. Schematic of a conic scanner showing the polar axis, Z, roughly perpendicular to the target surface, thereby creating a conic pattern of pixels.

Sweep:
ATSR SSM/I SMMR TMI

Sweep + elevation + profile:
ground-based Doppler Radar

Line-Scanner. Line-scanners sample by sweeping around a rotation axis such that they sample a linear array relative to the target (Figure A3). Sweeping at either a constant or varying elevation angle, scan dimensions can be one to three dimensions, and can possibly include sweep, elevation, or profile components. As discussed above, a line scanner is mathematically equivalent to a conic scanner rotated approximately 90

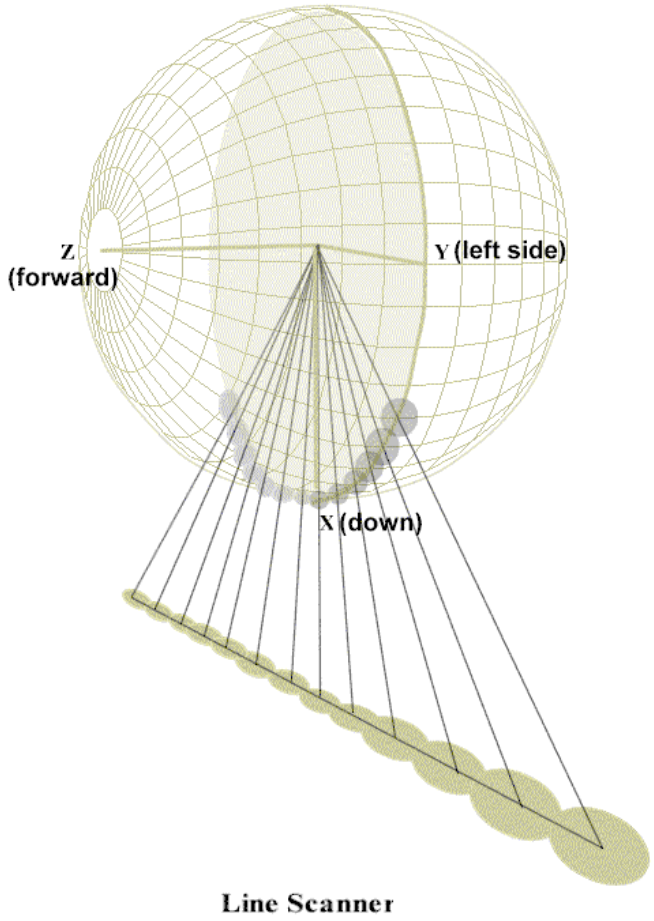

Figure A3. Schematic of a line scanner with the polar axis, Z, roughly tangential to the target surface and roughly in the velocity direction.

degrees. This is perhaps the most common sensor system used for Earth observation and includes:

Sweep:

AVHRR	AVNIR	AMPR	CZCS
LANDSAT	MODIS	MSU	OCTS
SEAWIFS	SSM/T	SSM/T2	AMSU
VIRS			

Sweep + elevation:
- GOES Imager

Sweep + profile:
- GOES Sounder
- TRMM PR

Imager. The imager class of sensors is really a special case of line or conic-scanners in which the entire array of pixels is sampled at an instant in time. The imager type sensor will be handled within the SensorML definition by simply setting step times for sweep and elevation to be zero (or by not defining these parameters). Examples of imager type sensors include:

Sweep:
any "push-broom" sensors (e.g. SPOT)

Sweep + elevation:
OTD
LIS
POLDER
Standard photographic and CCD video camera

Virtual Sensors. As discussed previously, some sensors might be describing a sensor model based not on the actual sensor design, but instead based on a design that could in essence have produced the preprocessed and distributed data set. Such sensors would NOT be of the type virtual, but would instead be of the SENSOR_TYPE equal to one of the four types defined above. Examples of sensors that might best be handled this way include:

Multi-sensor arrays:
MISR NSCAT

Active sensors:
SAR scatterometers

--------- break in excerpt ----------

3.3.7. **SCANNING** Node

The SCANNING node provides the primary information required to determine look ray directions (in sensor coordinate space) for any given pixel and to determine which pixel is being sampled at any given time. The specification is based on defining angular, spatial, and temporal measurements within the *sweep*, *elevation*, and *profile* coordinates in the spheroid model shown in Figure 1. The nesting of these elements within the SensorML specifies the order in which these coordinates vary, and allows virtually any sensor to be described using a single sensor model.

The *sweep* coordinate is an angle measurement about the Z axis, starting at the X axis and with positive direction being from the X axis to the Y axis ("right-handed rule"). The *elevation* coordinate is an angle measurement about the X axis, starting at the Y axis and with positive direction being from the Y axis to the Z axis ("right-handed rule"). The

profile coordinate is a distance measurement from the center of the sphere and outward along a radial. It is always positive.

While it is implied in the above statement, it should be explicitly noted that these angles and distances are relative to the sensor coordinate frame and not the platform coordinate frame. The geolocation will use the mounting specification described in the MOUNTING node to transform these direction to platform space, while the platform's position and orientation (not specified within the SensorML), would be used to further transform these directions into the same coordinate frame as the target body.

The allowable elements and attributes for the SCANNING node are given in Table 5. The descriptive parameters for the SWEEP and ELEVATION nodes are identical. Both are based on specifying angular and temporal measurements involved in the scanning procedure.

Element	**Attributes**
SCANNING	
SWEEP	direction, axis
fixedAngle	angleUnits, basis, value
startAngle	angleUnits, basis, value
extentAngle	angleUnits, basis, value
stepAngle	angleUnits, expression, value
startTime	timeUnits, basis, value
extentTime	timeUnits, basis, value
stepTime	timeUnits
repeatTime	timeUnits
numberOfSamples	
ELEVATION	direction, axis
fixedAngle	angleUnits, basis, value
startAngle	angleUnits, basis, value
extentAngle	angleUnits, basis, value
stepAngle	angleUnits, expression, value
startTime	timeUnits, basis, value
extentTime	timeUnits, basis, value
stepTime	timeUnits
repeatTime	timeUnits

numberOfSamples	
PROFILE	direction
fixedDistance	distanceUnits, basis, value
startAngle	distanceUnits, basis, value
extentAngle	distanceUnits, basis, value
stepAngle	distanceUnits, expression, value
numberOfSamples	

Table A**5. Elements and attributes for SCANNING node.**

The elements ***SWEEP***, ***ELEVATION***, and ***PROFILE*** all contain the attribute, *direction*, which specifies the progression of scan in that coordinate dimension. For angle measurements in ***SWEEP*** and ***ELEVATION***, the direction is positive if scanning progresses according to the right-hand coordinate rule. For ***SWEEP***, positive direction would signify a counter-clockwise rotation looking down the Z axis (or in the direction from the X to the Y axis). For ***ELEVATION***, positive direction would signify a counter-clockwise rotation looking down the X axis (or in the direction from the Y to the Z axis). For ***PROFILE***, positive direction would be from sphere center outward.

The *axis* attribute in ***SWEEP*** and ***ELEVATION*** is currently redundant since the axes of ***SWEEP*** and ***ELEVATION*** are fixed by definition as Z and X, respectively. However, this attribute will be retained in the current specification in order to provide absolute information to the software and in case there should be a need in the future to allow more flexibility with regard to the axis specification.

The parameters, ***startAngle*** and ***startTime***, indicate the starting position (within the sensor coordinate frame) and starting delta time (relative to the scan start time recorded in the data) for the first pixel sample of the sweep or elevation (Figure 1). If no values are given, they are assumed to be zero. Use of the parameter ***fixedAngle*** implies a constant angle as might be the case with a line-of-sight sensor or a sensor that scans in only one direction, yet has an offset in another. It is not necessary to specify a fixed angle of zero, since it will be implied by the absence of that coordinate in the specification. For example, a sensor that only scans in the sweep direction (along an elevation of zero), will only contain the ***SWEEP*** parameters.

To specify the extent over which sampling occurs, one may use either the ***extentAngle*** or the ***stepAngle*** element. Either of these will be combined with the ***numberOfSamples*** element to derive the other. The ***numberOfSamples*** value determines how many samples are taken during a given sweep event. The *basis* attribute specifies whether these angles are measured from the pixel center or the pixel outer edge, while the *angleUnits* attribute specifies the units of measurement for the angles. The intent in future releases is that the

stepAngle can optionally be specified as an mathematical expression or as a collection of values, but these specifications have not yet been defined.

The ***extentTime*** specifies the amount of time required to traverse the ***extentAngle***, while ***stepTime*** does the same for the ***stepAngle***. Like the ***extentAngle*** and ***stepAngle***, it is only necessary to specify one of the parameters, since the other will be derived using the ***numberOfSamples*** value. If ***extentTime*** or ***stepTime*** are not specified or are set to zero, then the entire sweep is assumed to be sampled instantaneously, as would be the case in the imager or push broom sensor type. The ***repeatTime*** specifies the amount of time that passes before the next sweep scan begins; in essence, this determines how often a new sweep event will occur.

The parameters within the ***PROFILE*** node define sampling characteristics along the line of sight direction. Unlike the *sweep* and *elevation* components, which are generally measured as angles, profile dimensions are measured as distances relative to the sensor origin. Thus, the element and attribute names are changed to reflect this distinction. There are currently no time specifications for the ***PROFILE*** element since generally samples along the profile are often treated as instantaneous. However, time specifications may be added in future releases if sampling precision is needed for specifying return times, etc.

As with sweep and elevation components, the ***stepDistance*** or ***extentDistance*** may be used along with ***numberOfSamples*** to determine the location of the sample along the line-of-sight. Likewise the *basis* attribute is used to specify if the sampling is at bin center or the outer edge. It should be noted that while different frequency bands of some sensors, such as MSU, measure values at different levels of the atmosphere, it will generally be better to support this capability using the ***heightAboveEllipsoid*** element, described below, rather than treating these sensors as profilers.

The hierarchical nesting of the three coordinate elements provides essential information for describing the order, direction, and timing of sampling. Thus, similar to do-loops or for-loops within programming languages, the SensorML is capable of describing which components vary fastest: sweep, elevation, or profile. As in programming loops, scanning steps within the inner blocks will occur first, followed by scan steps in the outer blocks.

For example, for a nesting such as

```
<SCANNING>
    <ELEVATION direction="positive" axis="X">
        …
        <SWEEP direction="positive" axis="Z">
            …
        </SWEEP>
    </ELEVATION>
</SCANNING>
```

the scan pattern would be one of a complete sweep in the positive direction followed by an elevation step in the vertical direction. For a nesting such as

```
<SCANNING>
    < SWEEP direction=”positive” axis=”X”>
        …
        < ELEVATION direction=”positive” axis=”Z”>
            …
        </ ELEVATION >
    </ SWEEP >
</SCANNING>
```

then a complete vertical scan would occur first and then a step in the horizontal sweep direction. Finally, for the pattern

```
<SCANNING>
    <ELEVATION direction=”positive” axis=”X”>
        …
        <SWEEP direction=”positive” axis=”Z”>
            …
        </SWEEP>
        <SWEEP direction=”negative” axis=”Z”>
            …
        </SWEEP>
    </ELEVATION>
</SCANNING>
```

there would be a sweep positive, then a sweep negative, followed by a vertical step.

The following example for the SSM/I conical sensor specifies that the elevation angle is first set to a fixed angle of –45 degrees, and then a sweeping pattern from +51 to –51 degrees repeats constantly, sampling 64 pixels in low resolution mode and 128 in high resolution. These pixels are sampled over a time of 1.9 seconds and the complete scan repeats every 3.8 seconds in low resolution mode and every 1.9 seconds in high resolution mode. If the elevation angle had not been fixed, as in the case of Doppler radar, then each complete sweep would have been followed by an elevation step.

```
<SCANNING>
  <ELEVATION direction="positive" axis="X">
    <fixedAngle value="-45.0" angleUnits="degrees" basis="pixelCenter"/>
  <SWEEP direction="negative" axis="Z">
    <startAngle value="51.0" angleUnits="degrees"  basis="pixelCenter" />
    <extentAngle value="102.0" angleUnits="degrees" basis="pixelCenter " />
    <startTime value="0.0" timeUnits="seconds" basis="pixelCenter " />
    <extentTime value="1.9" timeUnits="seconds"  basis="pixelCenter"/>
    <FOR variable="mode" idref="LOW-RES">
      <repeatTime value="3.8" timeUnits="seconds" />
      <numberOfSamples value="64" />
    </FOR>
    <FOR variable="mode" idref="HIGH-RES">
      <repeatTime value="1.9" timeUnits="seconds" />
      <numberOfSamples value="128" />
    </FOR>
  </SWEEP>
```

```
  </ELEVATION>
  <description topic="scanning assumptions">Scanning parameters are
        based on a conical scanning arrangement with a constant
        elevation angle of 135.0 degrees from vertical, with scanning
        from left to right when looking in the forward direction. The
        sampling takes place over 102.4 degrees centered about nadir.
</description>
 </SCANNING>
```

3.3.8. **SAMPLES** Node

The ***SAMPLES*** node provides information regarding the characteristics of the individual pixel rather than the relationship between pixels. The currently available elements and attributes are given in Table 6. Three of the elements specify the size of the pixel in the three coordinate dimensions. As with the ***SCANNING SWEEP*** and ***ELEVATION*** parameters, the ***pixelSweepSize*** and ***pixelElevationSize*** specify these dimensions in angle units and not as pixel footprint size on the target body. The ***pixelProfileSize*** is specified in spatial dimensions. Often, these size parameters are equal to the appropriate stepAngle and stepDistance specification in the SCANNING node. However, in many sensors, such as SSM/I the pixel size can be larger or smaller than the step size, resulting in overlaps or gaps between pixels.

Element	**Attributes**
SAMPLES	
pixelSweepSize	angleUnits, expression, value
pixelElevationSize	angleUnits, expression, value
pixelProfileSize	distanceUnits, expression, value
pointSpreadFunction	expression, value
pixelHeightAboveEllipsoid	heightMeasurement, distanceUnits

Table 6. Elements and attributes for SAMPLES node.

Although the element definition is not complete, the pointSpreadFunction element will provide a description of the sensitivity distribution within a pixel. This is usually a Gaussian distribution function and will probably be defined using appropriate parameters.

The ***pixelHeightAboveEllispoid*** element can be used to support recommendations for geolocating the data at locations other than the planet's surface. For example, most channels of sounders such as MSU and AMSU might best be geolocated at various average altitudes within the atmosphere rather than the Earth's surface, in order to avoid issues of parallax when measuring atmospheric properties. Other sensors might more appropriately utilize the tropopause or perhaps a depth below the ellipsoid for geolocation. This parameter provides a suggestion for the appropriate height for geolocation, and can be specified in term of distance or atmospheric pressure.

References

Botts, M. [ed.] (2001). *An XML-based Sensor Model Language (SensorML) for Earth Observing Dynamic Sensors, Draft Version 01_001* (July 10, 2001), http://vast.uah.edu/SensorML/SensorML_01_001.doc.

Cox, S. [ed.] (2002). *Observations and Measurements IPR*, OpenGIS Consortium, OGC 02-028.