

Project Report

SensorML on SenseTile

Ciaran Palmer

A thesis submitted in part fulfilment of the degree of

Msc Advance Software Engineering

Supervisor: Dr. Joseph Kiniry

Moderator: -



UCD School of Computer Science and Informatics
College of Engineering Mathematical and Physical Sciences
University College Dublin

April 15, 2010

Table of Contents

Abstract	3
1 Introduction	5
2 Background Research	6
2.1 Sensor Web Enablement	6
2.2 Sensor Model Language	6
2.3 Sensor Observation Service	9
2.4 SenseTile	11
2.5 BON	12
3 Design	13
3.1 Overview	13
3.2 SensorML Model of SenseTile	13
3.3 SenseTile Web Service	18
4 Implementation	25
4.1 Overview	25
4.2 DataProducer	25
4.3 SOS	26
5 SensorML BON Specifications	27
5.1 Overview	27
5.2 SensorML BON Specification	27
5.3 ProcessChain connections problem	28
6 Results	30
7 Conclusions and Future Work	31
A Appendix: SenseTile SensorML Model	35
A.1 SenseTile SensorML System	35
A.2 SenseTile SensorML SensorBoardSystem	37
A.3 SenseTile SensorML Converter System	38

A.4	SenseTile SensorML Thermistor	40
A.5	SenseTile SensorML CelciusConvertor	42
B	Appendix: SensorML BON Specification	44

Abstract

This project evaluates the Sensor Model Language(SensorML) and Sensor Observation Service(SOS) specifications. These specifications are intended for use in the development of open sensor systems. The UCD CASL SenseTile Sensor System is used as a case study for this evaluation. A SensorML model of SenseTile is developed. A SenseTile Web Service is also developed based on the SenseTile SensorML model and the SOS specification. This Web Service provides access to SenseTile sensor metadata and sensor observations. The project finally examines formalizing SensorML types using Business Object Notation.

Acknowledgments

I would like to thank Dr. Joseph Kiniry, Dr. Vieri del Bianco and Dr. Dragan Stosic for the opportunity to do this project and for all the generous help provided to me.

I would like to thank my wife Fiona for all her support.

Chapter 1: Introduction

Sensor systems are increasingly deployed to provide sensing information for a myriad of applications. These sensor systems contain many different sensor types and are developed by communities working in different domains. The heterogeneity of these systems means that interoperability is difficult. Reliably processing sensor observations retrieved from these heterogeneous systems is also problematic. Standard ways to describe sensors and to retrieve sensor data are needed to address the problem of sensor system heterogeneity.

Standards are being developed to address the above problems. Two such standards are the Sensor Model Language (SensorML) specification[2] and the Sensor Observation Service (SOS) specification[3]. SensorML is a proposed standard for both describing sensors and describing the processing of sensor observations. The SOS specification is a proposed standard for retrieving sensor descriptions and sensor observations using Web Services.

The SensorML and SOS specifications are developed by the Open Geospatial Consortium (OGC) as part of the Sensor Web Enablement (SWE) initiative¹. The members of the OGC have the goal of improving interoperability of sensor systems by developing open standards such as the SensorML and SOS specifications.

A real sensor system called SenseTile² is used as a case study for the evaluation of SensorML and SOS specifications. SenseTile is a sensor and processing package that contains motion sensors, temperature sensors, audio sensors, pressure sensors and light level sensors among others. It is a replacement for standard ceiling tiles to provide smart building services as part of a Web Sensor Network.

This project describes a SensorML model of SenseTile, in particular the SenseTile Sensor Board and sensors. Based on this model a SenseTile Web Service prototype is developed to allow the retrieval of the SenseTile sensor descriptions and sensor observations. The Web Service is based on the SOS specification. An assessment of SensorML and SOS specifications is provided based on this work.

Currently there is no widely available formal definition of sensors or sensor data processing. Formal sensor descriptions allows sensor system development to use a formal verification based process. A verification based software development processes is described in [5] which has the goal of developing very robust software systems. Business Object Notation(BON)[4] is used in this process to formally specify the system being developed. BON is a notation and a method for analysis and design of Object Oriented systems. This project looks at how BON can be used to formally specify SensorML. The SensorML BON specification developed in this project allows SensorML sensor descriptions to then be refined to JML³ and then to Java⁴, as described in [5].

¹<http://www.opengeospatial.org/projects/groups/sensorweb>

²<http://kind.ucd.ie/products/opensource/SenseTile>

³<http://www.eecs.ucf.edu/leavens/JML/>

⁴<http://www.java.com/>

Chapter 2: Background Research

2.1 Sensor Web Enablement

The SensorML and SOS specifications are developed by the Open Geospatial Consortium (OGC) as part of the Sensor Web Enablement (SWE) initiative¹. The initiative is establishing interfaces and protocols to enable a standardized Sensor Web. This allows for easier integration of sensor applications across sensor systems. SWE provides a framework of open standards for use with Web-connected sensors and sensor systems. There are seven main specifications currently:

- Sensor Model Language(SensorML) - models and schema for sensor system description and sensor measurement processing
- Sensor Observation Service (SOS) - standard web interface for accessing sensor observations
- Observations & Measurements (O&M) - models and schema for packaging sensor observation values
- Transducer Markup Language (TML) - models and schema for multiplexed data from sensor systems
- Sensor Planning Service (SPS) - standard web interface for tasking sensor systems
- Sensor Alert Service (SAS) - standard web interface for publishing and subscribing to sensor alerts
- Web Notification Service (WNS) - standard web interface for asynchronous notification

SWE services based on the above specifications allow the discovery of sensors, access to the sensor data, as well as subscription to alerts, and tasking of sensors to control sensor measurement. For a good overview of SWE and it's use in Sensor Web Networks see [7]. The SensorML and SOS specifications, as mentioned, are used to develop the SenseTile Web Service. The O&M specification is also used in developing the SenseTile Web Service. This specification defines the encoding of sensor observations and is used by the SOS specification.

2.2 Sensor Model Language

Sensor Modeling Language (SensorML) is primarily used to describe sensor systems and the processing of observations from sensor systems. The SensorML specification[2] provides a functional model of sensors and an XML encoding to describe sensors and their observations. Some of the main uses of SensorML described in the specification are:

¹<http://www.opengeospatial.org/projects/groups/sensorweb>

- Support sensor and sensor observation discovery.
- Support processing of the sensor observations.
- Support geolocation information about sensors.
- Provide accuracy information of sensors measurements.
- Provide an executable process chain for deriving new data on demand.

XML Schema is used to specify the XML encoding of SensorML in the SensorML specification. This XML schema is based on two conceptual UML² models, the SensorML Conceptual Model and the SWE Common Conceptual Model. These models are explained in the following sections.

2.2.1 SensorML Conceptual Model

In SensorML sensors are modeled as processes that convert physical phenomena to data. Processes take inputs, process them and result in one or several outputs. For example, a process can take a measurement generated by a thermistor which might be an electrical resistance value and transform it to a Celsius value. Processes can also be connected together in process chains to allow the sensor data to be processed by several processes sequentially. These processes and process chains are modeled in SensorML as shown in the below SensorML Conceptual Model 2.1.

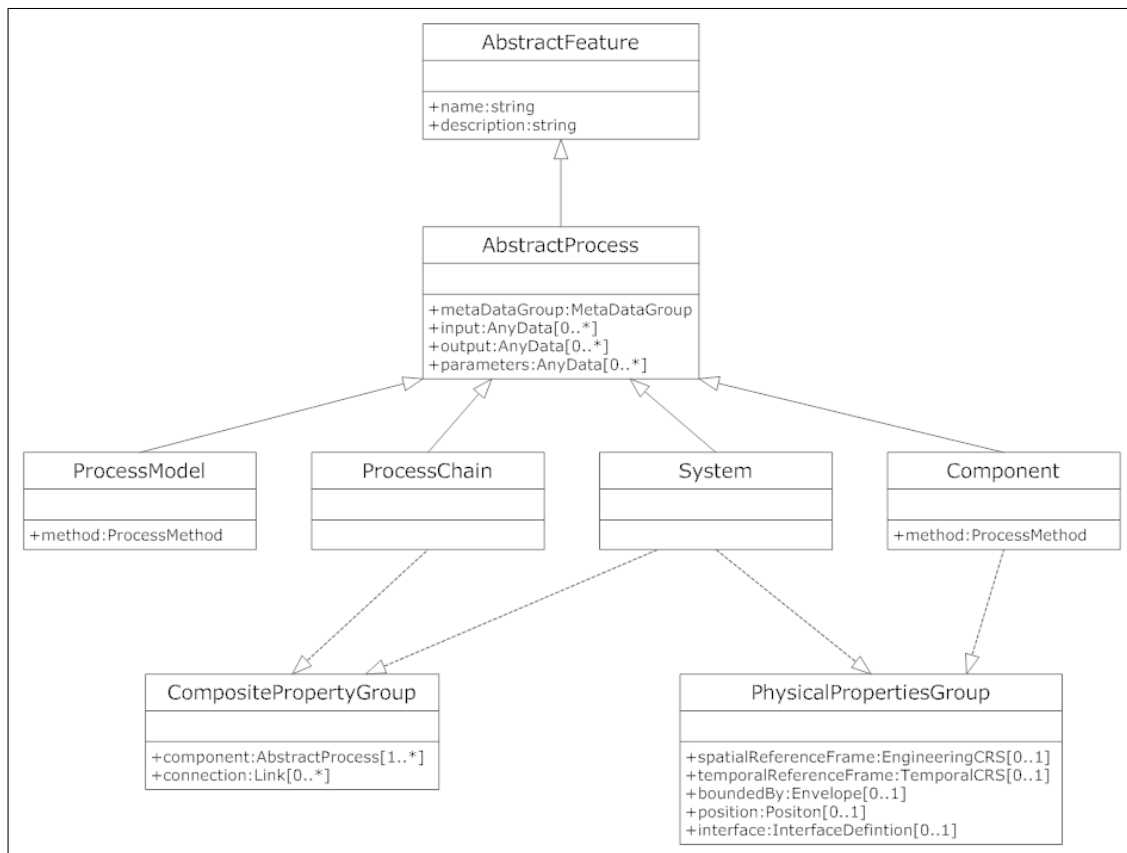


Figure 2.1: SensorML Conceptual Model

²<http://www.uml.org/>

Processes

The `AbstractProcess` models processes in SensorML and it contains input, output, parameter and `metaDataGroup` properties. The input and output properties represent sensor data measurements or connections to other processes. The parameter property is used to configure the process or provide other inputs that are not sensor measurements. For example, a parameter might be the latency time of a measurement needed for a calculation performed by the process. Finally each process provides sensor metadata as modeled by the `Meta-DataGroup`. The `MetaDataGroup` contains properties such as the capabilities of the sensor, contact information and documentation references. It is generally used to assist humans in understanding the sensor system and not used when executing a process. `AbstractProcess` derives from `AbstractFeature` which contains a name and description properties.

SensorML divides processes into two types, physical and non-physical. The non-physical process is one where location, position etc. is not relevant to the process. This type of process is modeled by the `ProcessModel` type. Physical processes provide physical information, such as location, which can be used in processing if required. This is modeled by the `Component` type. Both process types contain the method property. The method property is a `ProcessMethod` type which describes the methodology for transforming the process's inputs to outputs.

Process Chains

There is a similar division of process chains into the non-physical `ProcessChain` type and the physical `System` type. Both process chain types can chain a combination of `ProcessModel`, `Component`, `ProcessChain` and `System` types. Process chains have inputs and outputs which define the beginning and end of the processing chain. They are points of connection to other processes and process chains. The connection property contained in the `CompositeProperty-Group` referenced by the `ProcessChain` and `System` types is a sequence of type `Link`. The `Link` type contains source and destination properties that reference the inputs and outputs of processes. The sequence of `Links` define the process chain.

Linkable Properties

SensorML uses XML Linking Language (`XLink`)³ to support hypertext referencing in SensorML XML documents using URNs⁴ and URIs⁵. This allows SensorML models to be distributed over several XML documents. `XLink` attributes are also used to reference on-line unit definitions used to define a sensor measurement.

2.2.2 SWE Common Conceptual Model

The SensorML specification currently contains the Sensor Web Enablement (SWE) Common specification. This specification defines basic types and data encodings used by the SWE specifications. It describes both simple data types as well as aggregate types such as arrays and records. There are a large number of types defined in this specification, too many to show here. Below is a Conceptual Model of some of the simple data types.

The data types in figure 2.2 model scalar primitive values. They are used to defined the type of data used by the inputs, outputs and parameters properties in SensorML processes.

³<http://www.w3.org/TR/xlink>

⁴<http://tools.ietf.org/html/rfc2141>

⁵<http://tools.ietf.org/html/rfc3986>

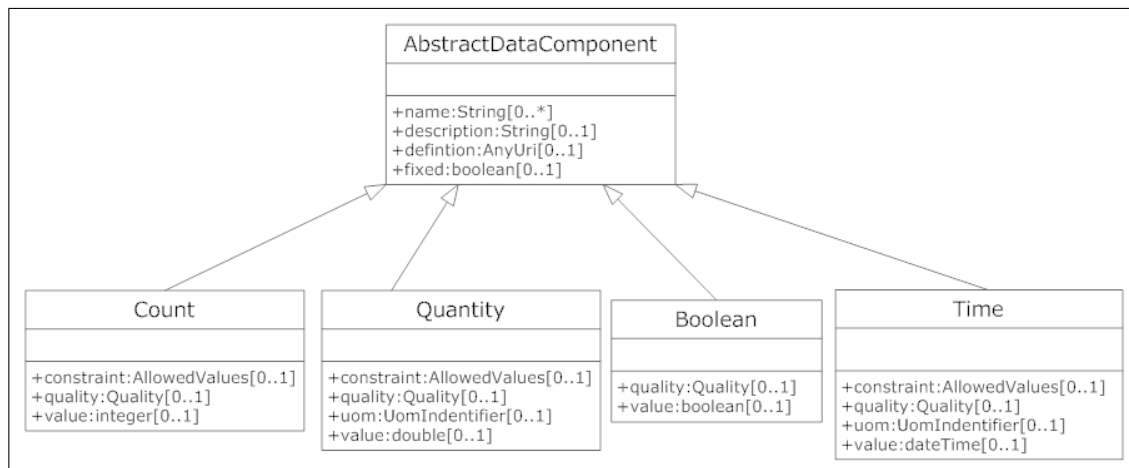


Figure 2.2: SWE Common Simple Data Types.

Quantity, for example, models a floating point number and can be used to represent Celsius temperature values. The data types in SWE Common derive from AbstractDataComponent which contains naming and description properties.

To support reliable processing of sensor data, the data types contain a number of properties to provide information on the quality it's value. The uom property provides a unit of measurement reference that indicates how the value should be interpreted. A constraint property allows a value range or a enumerated list of allowed values to be defined for the type. The quality property provides for a measure of the quality of the value. For example, the confidence level of a value being correct can be expressed as a percentage. Aggregation of these data types is also provided by several types including the DataRecord and DataArray types.

2.2.3 VAST SensorML Engine

The VAST Team at the University of Alabama in Huntsville (UAH) has developed an open source Java based SensorML processing engine. This software provides types that implement SensorML processes and process chains. The SensorML Processing Engine⁶ parses SensorML models and instantiates the required objects to performed the intended processing. This engine is used in the SenseTile Web Service and is referred to as the VastSMLEngine in rest of the thesis. The SensorML processing engine is dependent on an implementation of the SWE Common data types⁷ described in the section 2.2.2.

2.3 Sensor Observation Service

The SOS specifications defines a web service interface for the retrieval of sensor descriptions and the sensor observations from sensor systems. It organizes related sensor observations into a collection called an observation offering. The observation offering information is requested by clients to obtain the identity of sensors they are interested in getting observations from.

The SOS specification uses four main entities to describe the SOS's operation. The entities

⁶<http://code.google.com/p/sensorml-data-processing>

⁷<http://code.google.com/p/swe-common-data-framework>

are Data Consumers, Data Producers, OGC Catalogs and the SOS itself. These entities have the following function:

- Data Consumers are clients of the SOS that request sensor observations and sensor metadata.
- Data Providers are entities that read observations from the sensors and have enough processing power to generate an XML encoding of the sensor's observations. These observations are sent over a network to the SOS.
- OGC Catalogs by Data Consumers are used to locate SOS instances.
- SOS instances act as access points to a Sensor Web Network and provide the Web Service for retrieval of sensor descriptions and sensor observations.

The figure 2.3 below shows the SOS entities in an operational context.

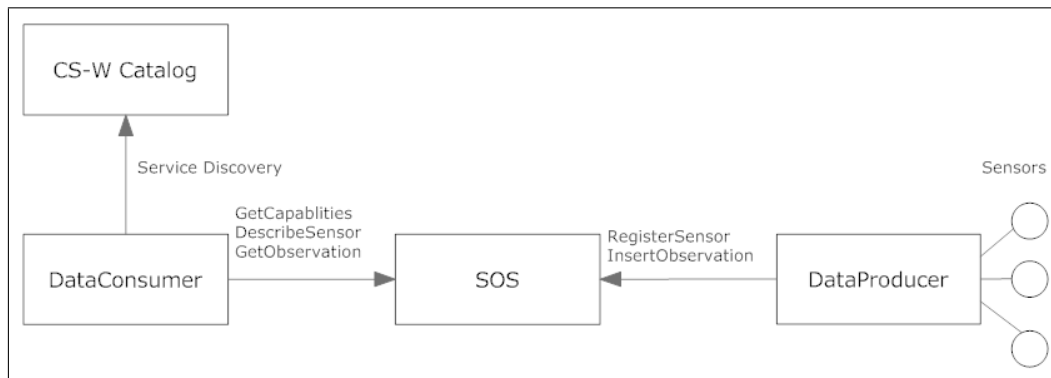


Figure 2.3: SOS Operational Context

Figure 2.3 shows the core and transactional operations provided by an SOS which are used by DataConsumers and DataProducers. The core operations are GetCapabilities, DescribeSensor and GetObservations. These are mandatory operations used by Data Consumers to request sensor information. The core operations provide the following services:

- GetCapabilities - retrieve SOS observation offering information.
- DescribeSensor - retrieve detailed information about the sensors.
- GetObservation - obtain sensor observations.

The transactional operations are RegisterSensor and InsertObservation used by the Data Producer to provided sensor information and observations. The transactional operations supported are described in the following:

- RegisterSensor - register the Data Provider with the SOS.
- InsertObservation - update SOS with a sensor observation.

These operations are encoded in XML and are intended to be carried in HTTP⁸ operations. The sensor observations contained in GetObservation and InsertObservation operations are

⁸<http://www.w3.org/Protocols/>

encoded according to the O&M Observations specification[6]. Sensor systems metadata used in DescribeSensor and RegisterSensor operations is provided with SensorML.

The SenseTile Web Service developed in this project is based on the concepts described in the Sensor Observation Service (SOS) specification. The SOS Data Consumer and SOS Data Provider entities are referred to a DataConsumer and DataProvider respectively in this thesis when describing the SenseTile Web Service.

The OGC Catalog Service shown in the SOS Operation Context diagram 2.3 is not explored in this project. The Data Consumer uses a CS-W catalog to find SOS instances. The CS-W catalog is covered in a separate specification[9] and is not covered in any real detail in the SOS specification.

There are also several optional SOS operations not covered in this project. The SOS refers to these as enhanced operations. They are GetResult, GetFeatureOfInterest, GetFeatureOfInterestTime, DescribeFeatureOfInterest, DescribeObservationType, and DescribeResult-Model.

2.4 SenseTile

SenseTile⁹ is used as case study of the evaluation of SensorML and SOS. SenseTile is a general purpose sensor system developed at UCD CASL¹⁰. The SenseTile System is used to investigate issues arising with large scale sensor networking. SenseTile is made up of a sensor platform, a datastore, and a compute farm. Figure 2.4 shows a outline of this system.

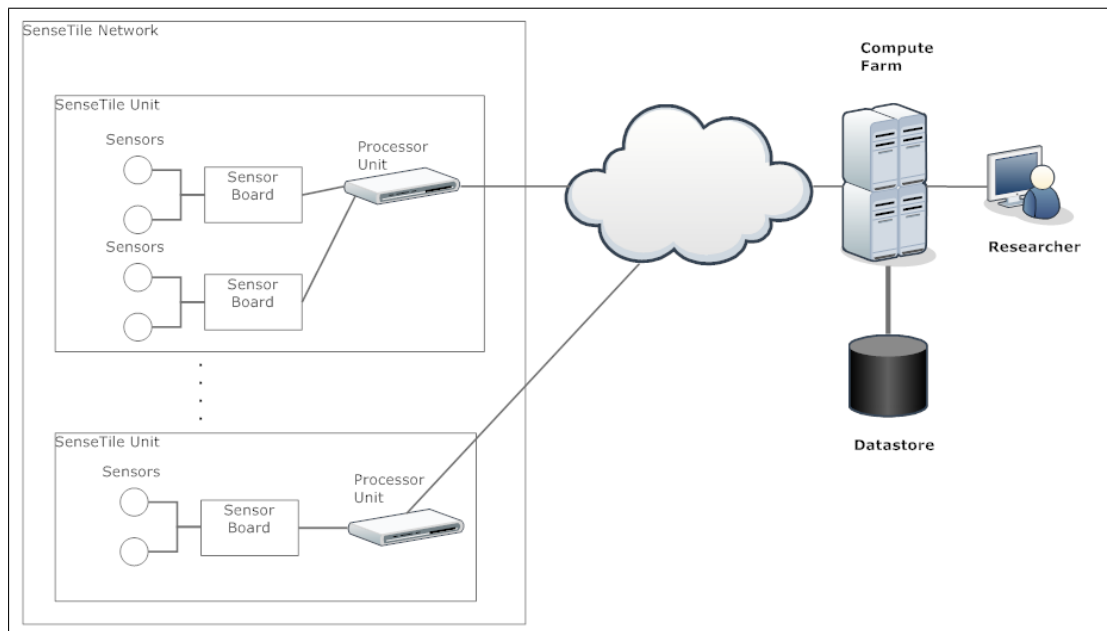


Figure 2.4: SenseTile System

⁹<http://kind.ucd.ie/products/opensource/SenseTile>

¹⁰<http://casl.ucd.ie/>

The datastore is a multi-terabyte scale datastore. The sensor data is stored here where it can be used in further processing of the data.

The compute farm is a large-scale compute farm that supports Linux, Solaris, and Mac OS and is where further processing of sensor data is performed.

The sensor platform is called the SenseTile Unit and is composed of one or more SenseTile Sensor Boards paired with a SenseTile Processor Unit. The Processor could be a PDA or small PC or any processing device that has a USB connection. The Sensor Board has a USB connection used for communication with the Processor Unit.

There are over a dozen sensors on the SenseTile Sensor Board including a thermistor and light sensor. Sensors are mounted on the Sensor Board and more sensors can be added using the USB connection. The Sensor Board and sensors are the main focus of the SensorML SenseTile model in this project and as the processor unit is a relatively powerful computer, providing a lightweight Web Service on the SenseTile Processor Unit itself is feasible.

2.4.1 UCD Sensor Board Driver

The research group¹¹ at UCD CASL have developed a custom asynchronous packet based protocol to manage the SenseTile Sensor Board and to read sensor data from the Sensor Board. A Java driver is developed also. It creates and parses the SenseTile Sensor Board streaming communication packets. The SenseTile Web Service uses this Java driver API to access the Sensor Board sensor data. This driver is referred to as the SensorBoardDriver in this thesis.

There is a Sensor Board simulator also developed by the UCD CASL group and it is used to develop the SenseTile Web Service.

2.5 BON

Business Object Notation (BON) is a notation and a method for analysis and design of Object-Oriented (OO) systems. It supports a seamless transition from the design to the source code as it based on the same Object Oriented (OO) concepts supported by most OO languages. It is reversible as changes in the source code can be easily mapped back into the design and analysis models. It also supports design by contract[10] where a function guarantees a postcondition if a specified precondition is fulfilled by the caller of the function. BON support for design by contract allows for a formal description of a system to be developed. For a detailed description of BON see [4].

¹¹<http://kind.ucd.ie/products/opensource/SenseTile>

Chapter 3: Design

3.1 Overview

This chapter describes the design of the SenseTile Web Service. Rather than define any new sensor models or XML formats for SenseTile it preferable to use the existing solutions if good enough for the task at hand. The OGC SWE framework as described in section 2.1 seems to cover all that is needed for modeling sensors and observations. Central to the design is a SensorML model of the SenseTile System.

Section 3.2 covers the details of the SenseTile SensorML model. This model acts as both metadata about the SenseTile system and provides a description of how to process the sensor data.

The Web Service design is based on the SOS specification and allows access to the SensorML description of SenseTile sensors as well as SenseTile sensor observations. Two SOS specification concepts as described in section 2.3, the Data Producer and the SOS itself, structure the design and are described in the SenseTile Web Service section.

3.2 SensorML Model of SenseTile

A SensorML model of SenseTile is developed as part of this project. The SenseTile Sensor Board and sensors are modeled in the most detail. The model of the Sensor Board is placed in a context to examine SensorML process chaining where further processing of Sensor Board data is performed in a separate SensorML process. The SensorML Systems and Components that are used to model the system are shown in a block diagram 3.1.

In the block diagram there are three SensorML Systems, the SenseTile System, the Sensor Board System and the Conversion System. The SenseTile System is used as container for the other two Systems linking them in a process chain. The Sensor Board System contains metadata about the SenseTile Sensor Board and is a container for SensorML Components that model the Sensor Board's sensors.

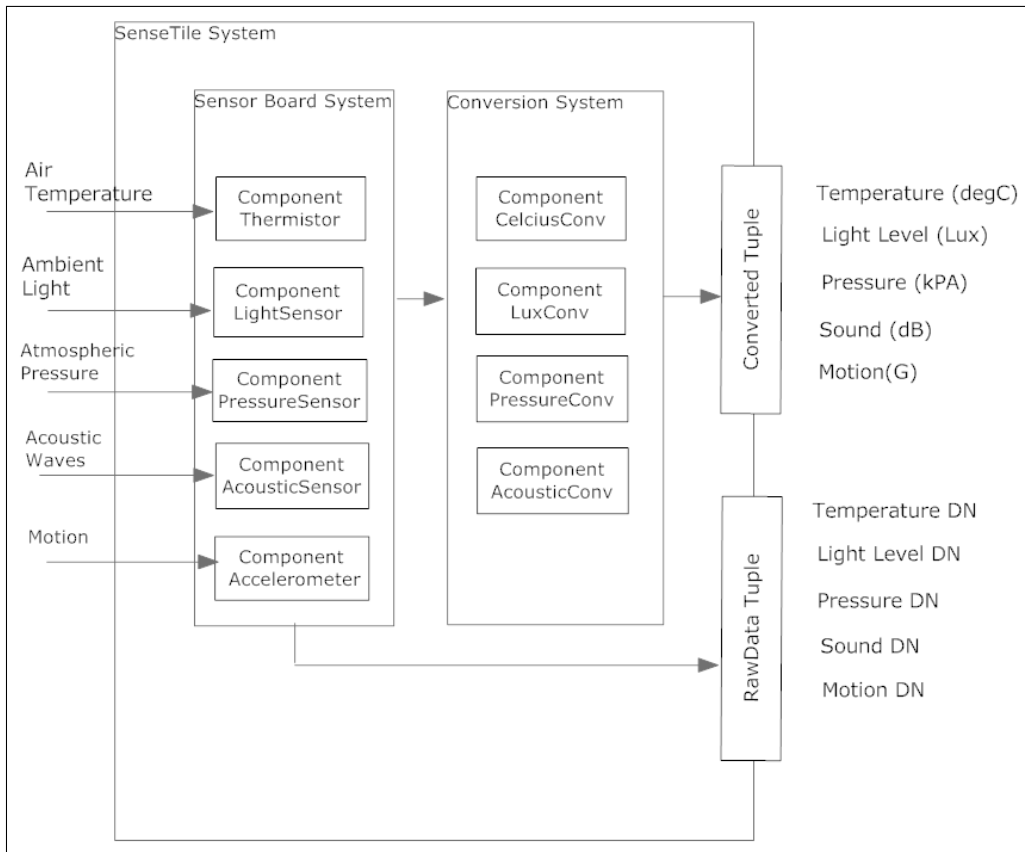


Figure 3.1: SenseTile System SensorML Block Diagram

The block diagram shows the physical phenomena measured by the sensors as inputs to the Components in the Sensor Board System. The Sensor Board System outputs digital numbers based on the sensors measurement that have no physical meaning yet. The System outputs the digital numbers directly and also connects this output to the input of the Conversion System. The digital numbers from the sensors are fed into the relevant conversion process that produce a meaningful value such as Celsius or Lumen. This allows the SenseTile Web Service to provide the raw data from the sensors and provide the data converted to meaningful values to clients.

Of the SenseTile sensors only the Thermistor sensor is modeled along with the CelsiusConv Component that converts the digital number received from the Thermistor Component to a Celsius value. However the same approach can be applied to the other sensors though for some sensor types no conversion process is needed.

A description of the important aspects of the SenseTile Systems and Components used in the SenseTile model is provided in the following sections. The complete model is provided in appendix A.

3.2.1 SenseTile System

The SenseTile System is a SensorML System as described in section 2.2 and is a container for the Sensor Board and Conversion Systems connecting their inputs and outputs. The below listing shows the XML encoding of the Systems component property.

Listing 3.1: SenseTile System Components List

```

<sml:components>
  <sml:ComponentList>
    <sml:component name="sensorBoardSystem"
                  xlink:href="SensorBoardSystem.xml">
    </sml:component>
    <sml:component name="convertorSystem"
                  xlink:href="ConverterSystem.xml">
    </sml:component>
  </sml:ComponentList>
</sml:components>

```

The components and ComponentList elements contain a list of component elements. These component elements have a name attribute to reference the component within the SenseTile System. For example, the Sensor Board System is called sensorBoardSystem as shown in the listing above. The component can be a SensorML process or process chain.

The component element also has an XLink href attribute that is used to reference the location of the SensorML definition of the component. See section 2.2 for a description of XLink. In this case the SensorML descriptions of the SensorBoard System and the Conversion System are in separate external files rather than in-line in the SensorML description. The use of XLink in SensorML is described in the SensorML section 2.2. As no path or URI is used in the hrefs, the file must be stored in the current directory.

The SenseTile System connects up the inputs and outputs of the SensorBoard and Conversion Systems using connections. The below SensorML fragment shows the SenseTile Systems connections. Here the temperature output from the SensorBoard System is connected both to the input of the Conversion System and to the output of the SenseTile System itself. Also the output of the Conversion System is connected to a different output of the SenseTile System.

Listing 3.2: SenseTile System Connection List

```

<sml:connections>
  <sml:ConnectionList>
    <sml:connection name="ambientTemperature">
      <sml:Link>
        <sml:source ref="this/inputs/ambientTemperature"/>
        <sml:destination ref="sensorBoardSystem/inputs/ambientTemperatureInput"/>
      </sml:Link>
    </sml:connection>
    <sml:connection name="temperatureDN">
      <sml:Link>
        <sml:source ref="sensorBoardSystem/outputs/temperatureDNOOutput"/>
        <sml:destination ref="this/outputs/temperatureDNOOutput"/>
      </sml:Link>
    </sml:connection>
    <sml:connection name="convertToTemperature">
      <sml:Link>
        <sml:source ref="sensorBoardSystem/outputs/temperatureDNOOutput"/>
        <sml:destination ref="convertorSystem/inputs/celciusConvInput"/>
      </sml:Link>
    </sml:connection>
    <sml:connection name="outputTemperature">
      <sml:Link>
        <sml:source ref="convertorSystem/outputs/celciusConvOutput"/>
        <sml:destination ref="this/outputs/temperatureOutput"/>
      </sml:Link>
    </sml:connection>
  </sml:ConnectionList>

```

The connection name attribute is used here to give a meaningful name to the connection. The Link type, as described in 2.2, is encoded with the Link element. The Link source and destination elements contain a ref attribute that indicates the components output and input that are connected based on the component name and the input or output names in the process referenced by component name. The special "this" string refers to the SenseTile System itself and is used to reference it's inputs and outputs.

3.2.2 SensorBoard System

This System models the Sensor Board and contains meta data about the Sensor Board. It connects up the inputs and outputs of the SenseTile sensors components. In this case it names the Thermistor Component "thermistor" and references the Thermistor SensorML file, Thermistor.xml. It connects up the Thermistor Component's inputs and outputs as shown in the below XML fragment.

Listing 3.3: Sensor Board System Components and Connections List

```

<sml:components>
  <sml:ComponentList>
    <sml:component name="thermistor" xlink:href="Thermistor.xml" />
  </sml:ComponentList>
</sml:components>

<!-- Connections -->
<sml:connections>
  <sml:ConnectionList>
    <sml:connection name="ambientTemperatureInput">
      <sml:Link>
        <sml:source ref="this/inputs/ambientTemperatureInput" />
        <sml:destination ref="thermistor/inputs/thermistorInput" />
      </sml:Link>
    </sml:connection>
    <sml:connection name="temperatureDN">
      <sml:Link>
        <sml:source ref="thermistor/outputs/thermistorOutput" />
        <sml:destination ref="this/outputs/temperatureDNOutput" />
      </sml:Link>
    </sml:connection>
  </sml:ConnectionList>
</sml:connections>

```

3.2.3 Thermistor Component

The Thermistor on the SenseTile Sensor board is the Texas Instruments TMP175 Digital Temperature Sensor and is modeled with a SensorML Component. This process can have a different location information than the Sensor Board if required. The SensorML Component allows the capabilities of the Thermistor to be described as metadata. For example the TMP175 is specified for operation over a temperature range of 40C to +125C. This is captured in the following SensorML XML fragment:

Listing 3.4: Thermistor metadata example

```

<swe:field name="TemperatureRange"
  xlink:arcrole="urn:ogc:def:property:dynamicRange">
  <swe:QuantityRange definition="urn:ogc:def:property:temperature">
    <swe:uom code="cel" />
    <swe:value>-40 125/swe:value>
  </swe:QuantityRange>
</swe:field>

```

The SensorML Thermistor temperature input is modeled using a Quantity value without any units as is a measured physical phenomena. Its output is a digital number from the sensor. This is modeled as a Count type with the range constrained to the allowed value range from the Thermistor. The following XML fragment shows how the input and outputs for the Thermistor look in SensorML:

Listing 3.5: Thermistor inputs and outputs

```

<sml:inputs>
  <sml:InputList>
    <sml:input name="thermistorInput">
      <swe:Quantity definition="urn:ogc:def:phenomenon:temperature">
    </sml:input>
  </sml:InputList>
</sml:inputs>

<sml:outputs>
  <sml:OutputList>
    <sml:output name="thermistorOutput">
      <swe:Count>
        <swe:constraint>
          <swe:AllowedValue id="outputRange">
            <swe:interval>-880 2032</swe:interval>
          </swe:AllowedValue>
        </swe:constraint>
      </swe:Count>
    </sml:output>
  </sml:OutputList>
</sml:outputs>

```

3.2.4 Converter System

The Converter System is a container for the processes that convert sensor data to a meaningful value. In this case it connects up the inputs and outputs of the CelsiusConv Component. It is very similar to the Sensor Board System in structure and so is not described here.

3.2.5 Celsius Converter Component

The CelsiusConv SensorML Component models a process that performs a conversion of the Thermistors digital number output to a real Celsius value. It is similar to the Thermistor SensorML but its inputs and outputs are have different types and are shown in listing 3.6.

Listing 3.6: CelsiusConv

```

<!-- Inputs -->
<sml:inputs>
  <sml:InputList>
    <sml:input name="celciusConvInput">
      <swe:Count />
    </sml:input>
  </sml:InputList>
</sml:inputs>

<!-- Outputs -->
<sml:outputs>
  <sml:OutputList>
    <sml:output name="celciusConvOutput">
      <swe:Quantity definition="urn:ogc:def:phenomenon:temperature">
        <swe:uom xlink:href="urn:ogc:def:unit:celsius" />
        <swe:constraint>
          <swe:AllowedValue id="temperatureRange">
            <swe:interval>-45 125</swe:interval>
          </swe:AllowedValue>
        </swe:constraint>
      </swe:Quantity>
    </sml:output>
  </sml:OutputList>
</sml:outputs>

<!-- Method -->
<method xlink:href="urn:ucd:sensetile:sensorboard:celciusConv" />

```

This process performs a simple calculation to perform the Celsius conversion. The method element in this references a URN that is used to look up an implementation class to perform the conversion algorithm. The method element can also reference a SensorML ProcessMethod. A ProcessMethod provides a description of the algorithm that is used by the process and a link to an implementation. This type is not implemented by the VastSML Engine so is not included in the SenseTile SensorML model.

3.3 SenseTile Web Service

3.3.1 Overview

The SenseTile Web Service is based on the Sensor Observation Service (SOS) as described previously. The two main entities in the SOS Specification, the Sensor Data Provider and the SOS, are used to structure the SenseTile Web Service. These entities are implemented as separate software components that together provide the service. The Sensor Data Provider component is referred to as the DataProvider in this design. The two components are run as separate processes on the SenseTile processor unit to allow a more flexible network architecture. This network architecture is described in more detail in section 3.3.2. The basic system structure is shown in figure 3.2 .

The DataProvider parses a SensorML description of the SenseTile System, reads sensor data from the Sensor Board and provides the sensor data as O&M Observations to the SOS. The SenseTile SOS implements the SOS core operations to allow a client to get sensor observations. The following sections describe the SenseTile network architecture, the supported scenarios

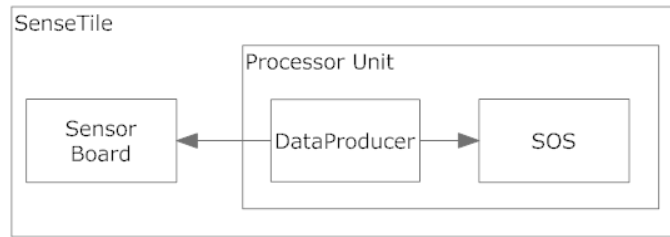


Figure 3.2: SenseTile Web Service

of the system and the design of the DataProvider and SOS components.

3.3.2 SenseTile WebService Network Architecture

The SOS Specification envisions that the SOS is run on a large external server. Based on this the SenseTile Web Service Network should be as shown in figure 3.3.

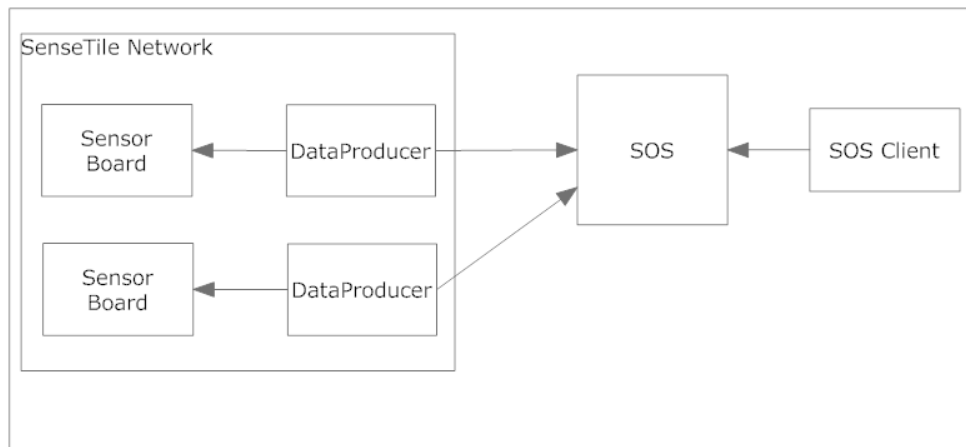


Figure 3.3: SOS Specification Architecture

The DataProducers update this external server with observations and SOS Data Consumer access this large SOS. Several SOS servers can be used in a hierarchy to allow the system to scale.

The SenseTile Web Service is actually designed with the network architecture as shown in figure 3.4.

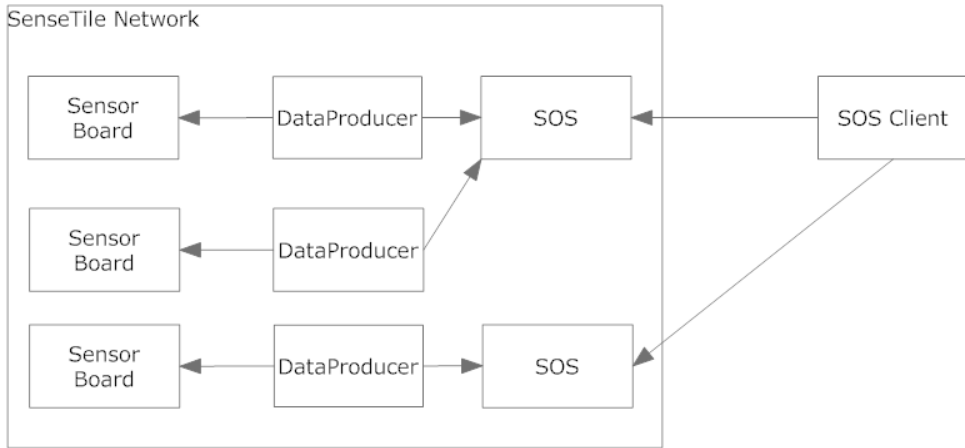


Figure 3.4: SenseTile Web Service Network Architecture

This is a different approach than outlined in the SOS specification. A client accesses multiple SOS instances to get observations from the sensor network. An SOS can support several DataProducers and hence is not needed on every SenseTile Unit. This approach is used to take advantage of the relatively powerful processing capability of the SenseTile Processor Unit and so eliminating the need for the large SOS server.

3.3.3 SenseTile Web Service Scenarios

This section contains UML sequence diagrams to show how the DataProducer and SOS interact to provide the sensor observations and sensor descriptions to the DataConsumer client. There are three main scenarios supported by the SenseTile Web Service and they are described in the following sections.

1. Generate Observations from Sensor Data

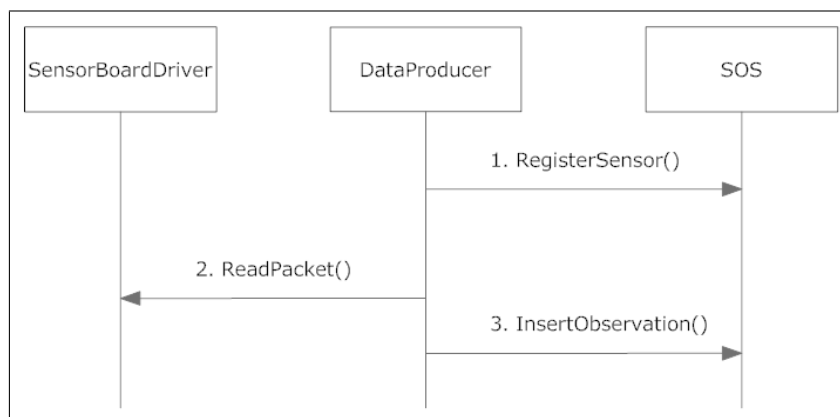


Figure 3.5: Generate Observation

1. At start up the DataProducer registers the SensorBoard System and the Converter System with the SOS in RegisterSensor operations. The SOS creates sensor offerings for each of the Systems and stores the SensorML descriptions of the Systems. The SOS returns a unique identity URN in the RegisterSensor response for use by the DataProducer for subsequent communication with the SOS.
2. The DataProducer reads Sensor Board data packets from the SensorBoardDriver. It waits one second before reading packets.
3. The DataProducer updates the SOS with sensor observations in an O&M XML format using the InsertObservation operation.

2. Get Sensor Description from SOS

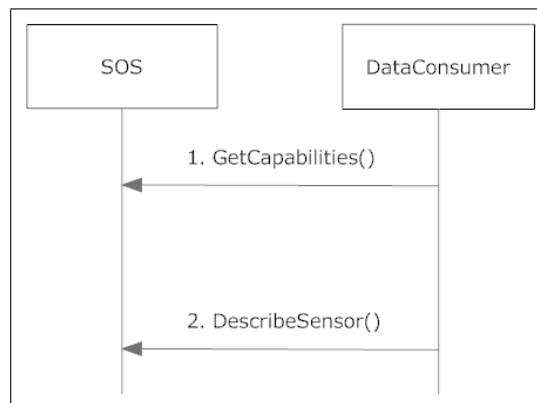


Figure 3.6: Get Sensor Descriptions

1. The DataConsumer requests information about SenseTile sensor offerings provided by the SOS using the GetCapabilities operation.
2. Using the information in the GetCapabilities response, the DataConsumer requests a description of a sensor system that is providing an offering using the DescribeSensor operation. The SOS returns the SensorML description of the sensor system to the DataConsumer.

3. Get Sensor Observation from SOS

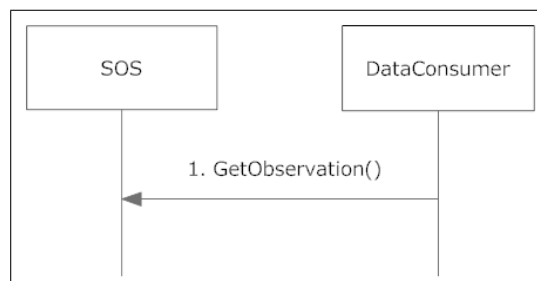


Figure 3.7: Get Observation from Sensor

1. Using the sensor identity received in a GetCapability response the DataConsumer requests a sensor observation using the GetObservation operation. The SOS returns the latest O&M format sensor observation to the DataConsumer.

3.3.4 DataProducer

The DataProducer’s role, as described previously, is to read sensor data from the Sensor Board and generate O&M observations. These observations are then sent to the SOS instance. The BON static diagram 3.8 shows the main clusters and classes of the DataProvider. BON is used to describe the class design in this thesis. An association arrow ,as in UML, is used rather than the BON double arrow due to drawing tool limitation. Refer to reference [4] for details of class design in BON.

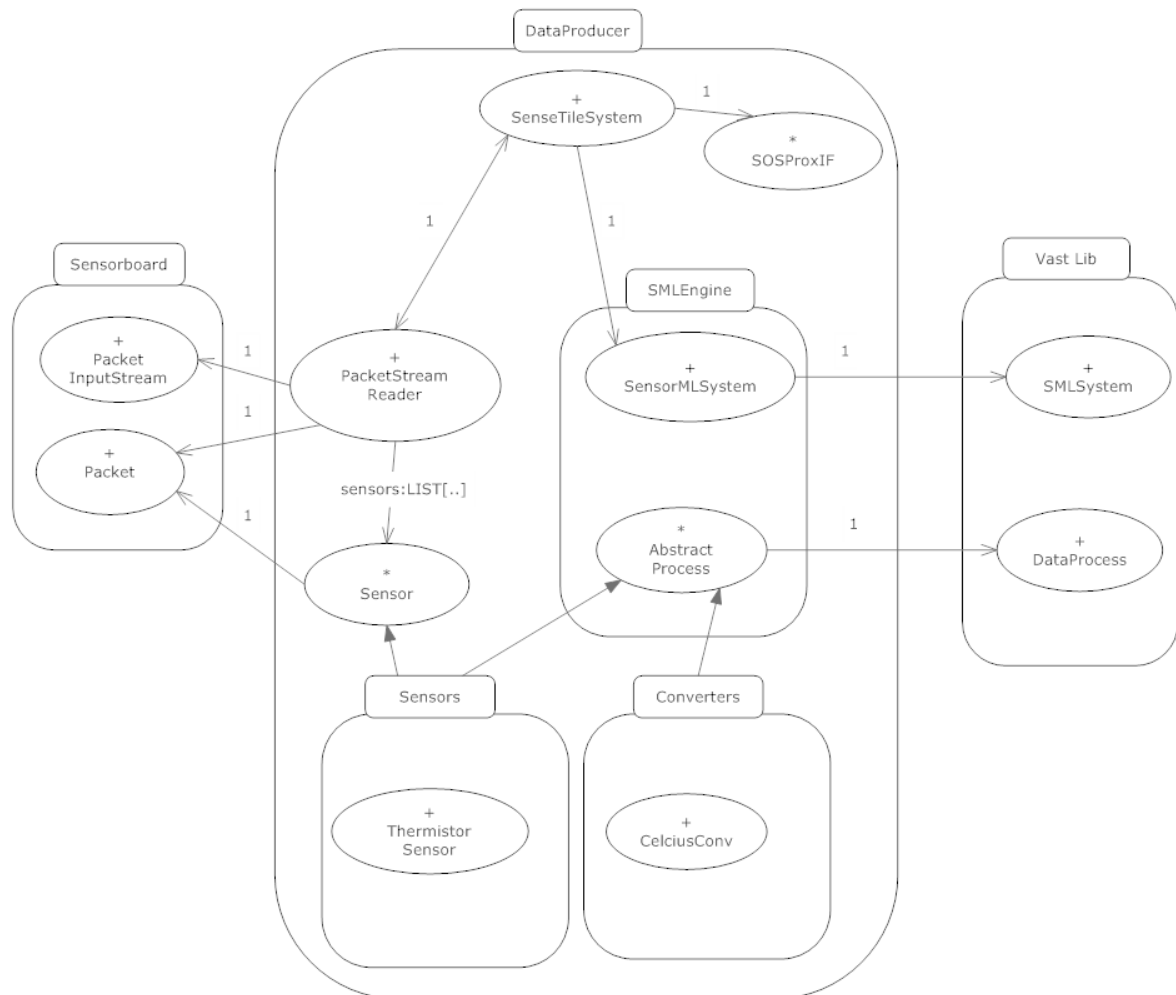


Figure 3.8: DataProducer BON Class Diagram

The processing of the sensor data to generate the observations is described by the SensorML models. The DataProducer uses the VastSMLEngine to instantiate objects that implement the required data processing chains based on the SensorML SenseTile model. The SMLEngine cluster contains classes to access the VastSMLEngine and so allow data from SensorML SenseTile model to be read.

The Sensors and Converters clusters contain classes that provide the implementation of process methods for SensorML Components in the Sensor Board and Converter SensorML Models respectively. These are referred to as sensor objects and converter objects henceforth.

The DataProducer operates by repeating the below steps continuously.

1. Read Sensor Board Data

2. Process Sensor Data
3. Create O&M Observations

1. Read Sensor Board Data

The UCD CASL SensorBoardDriver is used by DataProducer to access the data generated by the Sensor Board. The driver provides a high level interface for accessing the data from the sensor board. It reads data from the Sensor board and generates packets containing the sensor data. These packets provide functionality to access a particular sensors data such as the temperature measurement generated by the thermistor. The SensorBoardDriver classes are contained in the SensorBoard cluster in 3.8 The SensorBoard class PacketInputStream to is used access the packets from the Sensor Board.

At start up the SenseTileSystem reads the list of sensors from the Sensor Board SensorML model. These sensor objects provide an entry point to the SenseTile process chain and can read data from the SensorBoardDriver packets. The Observer pattern [11] to provide packets to SensorML sensor processes. The sensor objects are registered with the PacketStream-Reader. When the PacketStreamReader reads packets it updates all of the registered sensor object with the packet.

These sensor objects implement data reading strategies for the type of data they read. For example the Thermistor Sensor reads out temperature value, stores it and then averages the received values before sending a single value to the VastSMLEngine for processing.

2. Process Sensor Data

The PacketInputStream requests the SenseTileSystem to execute the VastSMLEngine engine to process the sensor data. The VastSMLEngine executes the process chains described in the SenseTile SensorML model.

The sensor processes output value is passed to the converter processes for further processing. The outputs of both processes are then available to sensor observation creation described next..

3. Create O&M Observations

When the VastSMLEngine Engine is finished processing an O&M Observation XML string containing the sensor data is sent to the SOS instance using the SOSProxyIF class.

3.3.5 SOS

The SOS component of the SenseTile Web Service provides a simplified SOS Web interface to access the sensor observations from the DataProducer. Below is a BON class diagram showing the important clusters and classes in the SOS design.

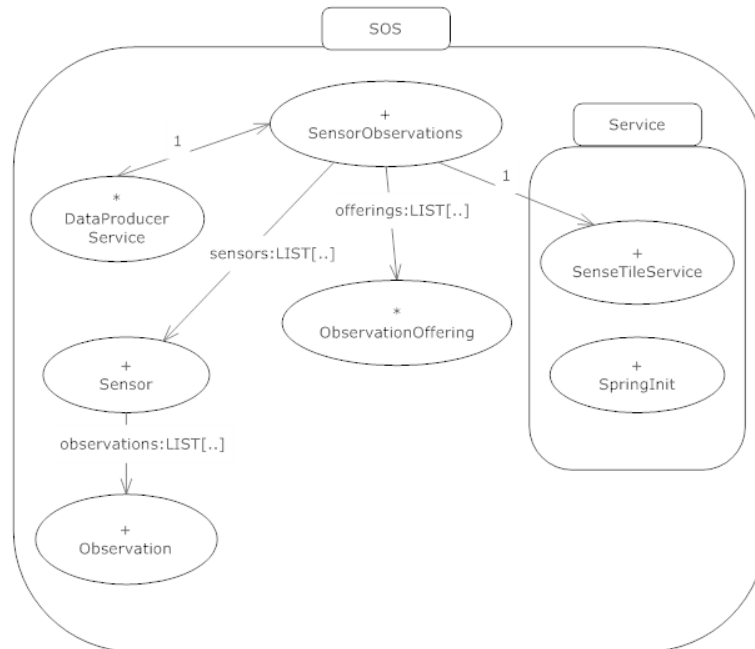


Figure 3.9: SOS BON Class Diagram

The SensorObservations class maintains a list of observation offerings provided by the SOS using the ObservationOffering class. When a new sensor type is registered a new ObservationOffering is created. If a DataProvider registers an offering that already exists then just a new Sensor instance is created and its identity is added to the appropriate ObservationOffering.

The DataProducerService class provides the interface for the DataProducer to register sensors with the SOS and update the SOS with sensor observations. These sensor observations are stored in the Sensor class in a list of Observations. The SenseTileService class provides the Web Service interface used by clients to access the sensor observations.

The SOS specification provides a very rich filtering feature in the GetObservation operation. However SenseTile Web Service only implements the functionality to get a sensor observation using sensor identity.

Chapter 4: Implementation

4.1 Overview

The SenseTile Web Service is written in Java. As described in the design chapter there are two components, the DataProducer and the SOS, that are developed to provide this service. The two components are run in separate processes. The communication between the DataProducer and the SOS uses Java RMI¹. RMI is used as the SOS and DataProducer processes are running on the same processor or communicate over a LAN. This communication has lower overhead than using HTTP so efficient communication between the DataProducer and SOS is achieved. It is not be hard to change to use a a different transport layer as a Java Interface is used to hide the communication mechanism from the main application code. Clients, SOS DataConsumers, access the SOS using a WSDL interface over SOAP².

The SOS specification describes an implementation using HTTP to carry complete SOS operations encoded in XML including the operations name and parameters. The SenseTile WebService takes a different approach. The operations are defined as RMI methods for DataProducer communication and WSDL operations for Data Consumer(SOS client) communication. These operations carry the XML encode SOS operations. This allowed a quicker development of the SenseTile Web Service but is not fully compliant with the current SOS specification.

The following sections describe the DataProducer and SOS implementation.

4.2 DataProducer

The DataProducer, described in section 3.3.4, accesses the sensor board data and inserts observations into the SOS. Two libraries are used to achieve this: the SensorBoardDriver and the VastSensorMLEngine.

The SensorBoardDriver is described previously in section 2.4.1. There is also a SensorBoard Simulator library developed by UCS CASL researchers. It produces sensor board packets that can be feed into the DataProvider. This was used during the development of the DataProvider to facilitate testing.

The VastSensorMLEngine, described in section 2.2.3, is used in conjunction with the SenseTile SensorML Model (section 3.2). The SensorML is parsed by the VastSensorMLEngine into a DOM³ tree. To perform the processing it propagates an execute method to the instantiated processes. A ProcessMap.xml file maps SensorML ProcessModel and Component method URNs (see section 2.2) to Java classes that provide the implementation of the processing that is performed.

¹<http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp>

²<http://www.w3.org/TR/soap/>

³<http://www.w3.org/DOM/>

There is no documentation that I can find for the VastSensorMLEngine. There are some reference implementations of SWE services pointed to from the OGC SWE website⁴ that provide some information how to use it. The SenseTile SensorML model was not parsed initially by the VastSensorMLEngine. As it opensource the code is available and after a few small changes the model parsed.

4.3 SOS

The SOS is the provider of the Web Service interface and is built on Apache Axis2⁵. Axis2 is described as a "Web Services / SOAP / WSDL engine" with the Apache Axis2/Java implementation used in this project. The Spring framework⁶ is also used. Using these technologies a lightweight Web Service is built to run on the relatively powerful SenseTile Processor Unit. Axis2 is run standalone here though it can run on any Servlet engine if a more powerful Web Service is needed.

Developing a Web Service is straight forward with Axis2. It supports several approaches to developing Web applications. The POJO (Plain Old Java Objects) approach was used for the SenseTile Web Service allowing the code to be reused on a different framework if AXIS2 did not work well. The Spring framework is used for it's dependency injection functionality. A SprintInit class is needed for Axis2 to load up the Spring framework.

Axis2 provides a tool that automatically generates WSDL⁷ from the running Web Service. Another tool allows client side code to be generated from the WSDL allowing easier development of the Web Service client.

JAXB⁸ was used for XML to Java binding in the SOS. JAXB is able to generate a default binding from the complex SWE schema. Some name collisions were corrected with a JAXB customization file found on the SensorML newsgroups. However the default binding was not fully successful for the System type. The Systems type input, output and parameter properties were missing and the inheritance hierarchy did not provide them. A small change was needed to add these properties and then everything worked fine with marshaling and unmarshaling the SensorML System type.

⁴<http://www.ogcnetwork.net/SWE>

⁵<http://ws.apache.org/axis2>

⁶<http://www.springsource.org/>

⁷<http://www.w3.org/TR/wsd1>

⁸<http://java.sun.com/developer/technicalArticles/WebServices/jaxb>

Chapter 5: SensorML BON Specifications

5.1 Overview

In this chapter a SensorML BON specification is described. One motivation for this is take advantage of formal software development processes described in [5]. This is a verification based process that refines BON to JML and then to Java. Robust software can thus be developed. Another motivation is that XML is verbose and a more compact description of a SensorML model as facilitated by BON is preferred. Lastly SensorML is complex and it is difficult to develop correct SensorML models based only on using the SensorML XML Schema definition. A BON specification of SensorML can help in the development of correct SensorML. One example of this is described in section 5.3.

5.2 SensorML BON Specification

The BON specification of SensorML described is very close conceptually to the SensorML model shown in section 2.2. The SensorML BON Class Diagram is shown in figure 5.1. The structure of the BON model and the names of classes and features should allow a reasonably straightforward mapping to SensorML XML elements and attributes.

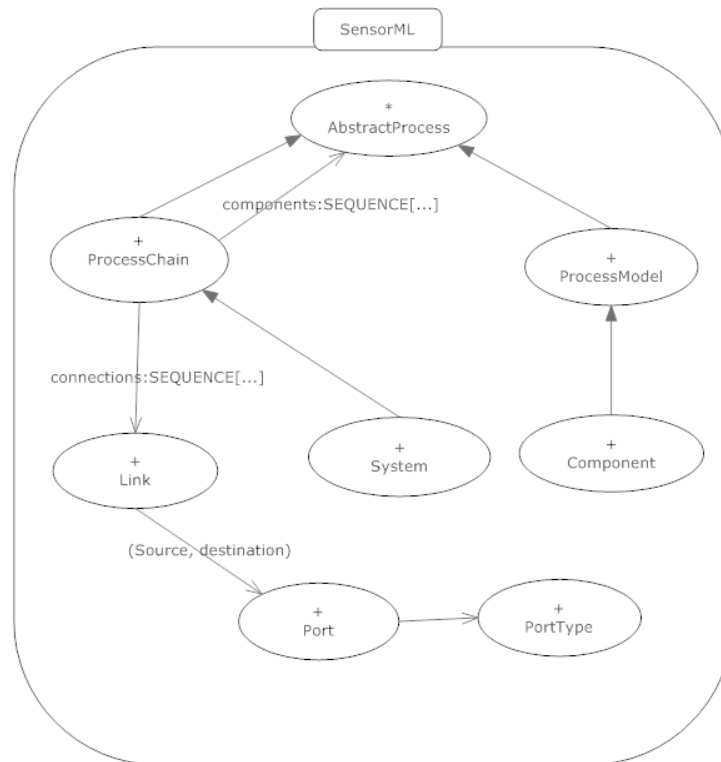


Figure 5.1: SensorML BON Class Diagram

The inheritance hierarchy is changed with System inheriting from ProcessChain and Component inheriting from ProcessModel. A new abstraction called Port is added. Port contains all the properties to describe SensorML process's input, output and parameter. A PortType class that ensures legal port types is added to the SensorML BON model.

A BON specification SWE Common data types Count and Quantity are also described along with needed dependent classes. The abstract class DataComponent is the base type for these. Only Count and Quantity are developed as they are the ones used in the SenseTile SensorML model.

The BON specification developed in this project does not address the issue of the SensorML metadata and the MetaDataGroup type is not specified in BON. This information is for descriptive purposes only and is not used for processing sensor data. BON adds no value here. A different way to add this information to SensorML sensor descriptions is suggested to be used.

The SensorML BON specifications developed in this project are be found in appendix B.

5.3 ProcessChain connections problem

One of the major features of SensorML is the ability to describe processing chains. The SensorML ProcessChain and System types have a connections property that allows process chaining as described in section 2.2. Strings are used to define connections. An example of this is shown in listing 5.1 shows one of the connections from the SenseTile System as described in section 3.2

Listing 5.1: SenseTile System connection

```
<sml:connection name="convertToTemperature">
  <sml:Link>
    <sml:source ref="sensorBoardSystem/outputs/temperatureDNOutput"/>
    <sml:destination ref="convertorSystem/inputs/celciusConvInput"/>
  </sml:Link>
</sml:connection>
```

The source and destination reference strings need to be parsed to find out the inputs and outputs to connect. This is very error prone and the ability to check that connections are correct to a least some degree is desirable. A BON specification can enforce some level of correctness in setting up the connections using preconditions. The source reference must be either an input of the process chain itself or an output of one of the contained components. In the BON ProcessChain class a feature AddLink checks that Links are legal before adding the Link to the connections feature. This is shown in the BON ProcessChain listing below.

Listing 5.2: BON ProcessChain

```

effective class ProcessChain
  indexing
  about:      "Process formed by chaining sub-processes.";
  title:      "ProcessChain.";
  author:     "Ciaran Palmer.";
  copyright:  "none.";
  organisation: "School of Computer Science and Informatics, UCD.";
  date:       "2010/04/02.";
  version:    "Revision: 1.00.";

  inherit AbstractProcess
  feature

  make
  ->componentsIn:AbstractProcess
  require
    componentsIn/=Void;
  ensure
    delta{components};
    components = componentsIn;
  end

  — Collection of subprocesses that can be chained using connections
  components:SEQUENCE[AbstractProcess]

  — Links between processes
  connections:SEQUENCE[Link]

  — Add a link with check that Link is legal
  AddLink
  ->linkIn:Link
  require
    linkIn /= Void;

    — check Link source is legal
    (exists i:INTEGER such_that linkIn.source = Current.inputs[i]) or\
    \ (exists i:INTEGER and j:INTEGER such_that linkIn.source =\
    \ components[j].outputs[i]);

    — check Link destination is legal
    (exists i:INTEGER such_that linkIn.destination = Current.outputs[i]) or\
    \ (exists i:INTEGER and j:INTEGER such_that linkIn.destination =\
    \ components[j].inputs[i]);

  ensure
    delta{connections};
    (exists i:INTEGER such_that connections[i] linkIn);

  invariant
    components /= Void;

end —ProcessChain

```

Chapter 6: Results

A SensorML Model of the UCD CASL SenseTile sensor system was developed to evaluate SensorML. The SenseTile Sensor Board and Thermistor sensor were the focus of the model. Appendix A contains the SensorML SenseTile model developed.

A lightweight Web Service was developed to explore the use of SensorML and SOS specifications in sensor systems. The Web Service is implemented in Java and is built on AXIS2.0, the Spring framework, the VASTSensorML Engine and the UCD CASL SensorBoardDriver. It was tested against SenseTile SensorBoard Simulator classes using a basic test client.

A SensorML BON specification was developed for the core SensorML types and the SWE Common Data types used in the SenseTile SensorML model. This specification is included in appendix B.

Chapter 7: Conclusions and Future Work

7.0.1 Conclusions

This project analyses the role of the SensorML and SOS specifications in sensor systems. The UCD CASL SenseTile System is used as a case study for the use of these specifications with a SensorML model of the SenseTile system developed to evaluate SensorML. A SenseTile Web Service prototype is developed to evaluate the SOS specification which uses SensorML to provide sensor descriptions to clients. The SensorML and SOS specifications are proposed open standards for sensor description and sensor data access respectively. This is part of the OGC SWE initiative goal to improve interoperability of sensor systems.

SensorML

SensorML provides a simple functional model of sensors in terms of an intuitive process concept. A process transforms sensor inputs to outputs as modeled by the SensorML AbstractProcess type. SensorML provides a process chaining capability which can be used to split sensor data processing into smaller more manageable steps. There is also a rich set of data types to describe sensor data. Due to the simple model and the rich set of data types is hard to imagine any sensor or sensor system that could not be described by SensorML. It is not intended to describe details of the sensor hardware. It is also not intended that SensorML provides the framework for encoding the actual sensor observations. This is provided by the SWE O&M specification.

SensorML also provides a good level of metadata types to assist with a human understanding of the capabilities of the sensor system. All SensorML process can provide metadata such as capabilities, properties, contacts information and documentation sources.

Good support for the reliable processing of sensor data is provided by SensorML. The data types have properties that can qualify the data. For example the data type can reference external definition libraries to precisely describe their meaning and give an indication of the quality of the data. The accuracy of the sensor can be modeled using accuracy curves which can be provided to processes using parameters to get a more accurate output from the process.

SensorML can model a simple sensor or complex sensor systems. The SensorML specification defines a simple Detector model that limits the process to one input and one output. The Detector process outputs a real value such as a Celsius temperature value. The SenseTile model uses a different approach as the unconverted data can also be retrieved from the SenseTile Web Service. There are good examples¹ and tutorials [8] available to help develop SensorML models.

The SenseTile model developed in this project is relatively simple and did not use the full richness of SensorML. It does not, for example, use the SensorML support for accurate and reliable processing data. It did examine a simple process chain which highlighted some of the difficulties of process chaining described in chapter 2.5.

SensorML is defined by a complex set of XML Schema. Achieving a Java to XML binding

¹http://www.ogcnetwork.net/SWE_Projects

is difficult based on these schema. JIBX² did not generate a default XML binding. JAXB did generate a default binding with some help but updates are still needed on the generated code to access some elements. There are many layers of abstraction in these XML Schema obviously making the binding generation difficult. This reflects the fact that the specification tries to cover all possible needs of a sensor system and is thus complex.

The VastSensorMLEngine seems to be the reference implementation of a SensorML processing framework. There was little or no documentation that I could find on this and examples of it's use needed some research. It would be good if a working example could be provided with the framework and also some basic documentation. The VastSensorMLEngine code needed to be slightly modified to parse the SenseTile model files though they passed a validation check³ provided by the Vast group. After these updates allowed the parsing to complete the code was indeed very stable and never crashed except when there were mistakes in the SenseTile SensorML XML files.

To use SensorML effectively a graphical tool or some other method rather using XML directly is needed. Working in SensorML XML is very error prone. Running the SenseTile model through the VastSensorMLEngine without a crash was the general procedure to check the correctness of the SensorML descriptions.

SensorML BON Specification

The BON SensorML developed in this project just covers the core of the SensorML types. However it has shown how some of the problems of SensorML can be addressed with a formal language such as BON. A way to check that connections in SensorML process chains are correct is described using BON assertions. The type safety and assertions provided by a BON SensorML specification allows a better way to develop SensorML based sensor systems than just using the SensorML XML encoding.

A more complete BON SensorML description is needed than provided in this project and there are still many issues that need to be addressed. For example how to represent XLinks in the SensorML BON specification or is there a need to. For a complete SensorML description the SensorML metadata needs also to be included in an SensorML description. How to do this is not covered in this project.

SOS

The SenseTile Web Service developed in this project is based on the SOS specification. The SOS specification is complex. Only a small amount of the SOS functionality is implemented in the SenseTile Web Service but this allows sensor descriptions and sensor observations to be retrieved. The SenseTile Web Service is very much a prototype and would need more work to make it a real Web Service.

Implementing all SOS functionality is a large task. If there is no requirement for interoperability or complex querying of sensor data provided by the SOS operations then it is better to go with some other simpler approach. One approach could be to use industry standard Web Services and use SensorML for sensor descriptions.

The approach used in the SenseTile Web Service of having both the SOS and the DataProducer on the SenseTile Processor unit may be too complex. It could be better to move the SOS to a separate server as described in the SOS specification. This would simplify the system

²<http://jibx.sourceforge.net/>

³<http://vast.uah.edu/SensorMLforms/validate.jsp>

design and management. The use of RMI may not be a good choice for the communication between the DataProducer and SOS as it limit this communication to LANs.

7.0.2 Future Work

It would be interesting to investigate how accurate the sensor data can be made using the SensorML support for reliable data processing. Calibration and accuracy curves could be developed based on the SenseTile sensor hardware specifications. A simple Detector model could be developed to see how this works in practice.

A complete SensorML BON specification is still to be developed. A SensorML BON to SensorML XML encoding tool would be interesting to develop. This tool could also incorporate a way to add the sensor metadata to generated SensorML XML.

Comparing a RESTful[12] approach to retrieving sensor observations rather than the SOS approach could be investigated. It may prove to be a simpler way to retrieve the sensor observations.

Bibliography

- [1] Chu, Kobialka, Durnota, and Buyya, Open Sensor Web Architecture: Core Services
- [2] Open Geospatial Consortium Inc., OpenGIS Sensor Model Language (SensorML) Implementation Specification, 2007
- [3] Open Geospatial Consortium Inc., Sensor Observation Service, 2007
- [4] Kim Waldn and Jean-Marc Nerson, Seamless Object-Oriented Software Architecture, 1995
- [5] Kiniry and Zimmerman, A Verification-centric Software Development Process for Java
- [6] Open Geospatial Consortium Inc., Observations and Measurements Part 1 - Observation schema, 2007
- [7] Open Geospatial Consortium Inc., OGC Sensor Web Enablement Architecture, 2008
- [8] Open Geospatial Consortium Inc., Using SensorML to describe a Complete Weather Station, 2006
- [9] Open Geospatial Consortium Inc., OpenGIS Catalog Services Specification, 2007
- [10] Bertrand Meyer, Object-Oriented Software Construction, 1997
- [11] Gama, Helm, Johnson, and Vlissides, Design Patterns, 1995
- [12] Fielding, Roy Thomas (2000), Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, Irvine

Appendix A: **Appendix: SenseTile SensorML Model**

A.1 SenseTile SensorML System

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:SensorML rng:version="1.0.1"
xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:rng="http://relaxng.org/ns/structure/1.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xng="http://xng.org/1.0">
  <sml:member>
    <sml:System gml:id="SENSETILE_SENSOR_SYSTEM">
      <gml:description>A SenseTile Sensor System made up
        of a sensor board and conversion components.</gml:description>
      <gml:name>SenseTileSensorSystem</gml:name>

      <!-- Keywords -->

      <sml:keywords>
        <sml:KeywordList>
          <sml:keyword>sensor sensetile</sml:keyword>
        </sml:KeywordList>
      </sml:keywords>

      <!-- Identification -->

      <sml:identification>
        <sml:IdentifierList>
          <sml:identifier name="longName">
            <sml:Term definition="urn:ogc:def:identifier:longname">
              <sml:value>UCD SenseTile Sensor System</sml:value>
            </sml:Term>
          </sml:identifier>
          <sml:identifier name="shortname">
            <sml:Term definition="urn:ogc:def:identifier:OGC:shortname">
              <sml:value>SenseTileSensorSystem</sml:value>
            </sml:Term>
          </sml:identifier>
        </sml:IdentifierList>
      </sml:identification>

      <!-- Inputs -->

      <sml:inputs>
        <sml:InputList name="inputlist">
          <sml:input name="ambientTemperature">
            <swe:Count/>
          </sml:input>
        </sml:InputList>
      </sml:inputs>
    </sml:System>
  </sml:member>
</sml:SensorML>
```

```

<!-- Outputs -->

<sml:outputs>
  <sml:OutputList>
    <sml:output name="temperatureDNOOutput"> <!-- Raw Sensor Data -->
      <swe:Count />
    </sml:output>
    <sml:output name="temperatureOutput">
      <swe:Count definition="urn:ogc:def:phenomenon:temperature">
        <swe:uom xlink:href="urn:ogc:def:unit:celsius" />
      </swe:Count>
    </sml:output>
  </sml:OutputList>
</sml:outputs>

<!-- Components -->

<sml:components>
  <sml:ComponentList>
    <sml:component name="sensorBoardSystem"
      xlink:href="SensorBoardSystem.xml"></sml:component>
    <sml:component name="convertorSystem"
      xlink:href="ConverterSystem.xml"></sml:component>
  </sml:ComponentList>
</sml:components>

<!-- Connections -->

<sml:connections>
  <sml:ConnectionList>
    <sml:connection name="ambientTemperature">
      <sml:Link>
        <sml:source ref="this/inputs/ambientTemperature" />
        <sml:destination
          ref="sensorBoardSystem/inputs/ambientTemperatureInput" />
      </sml:Link>
    </sml:connection>
    <sml:connection name="temperatureDN">
      <sml:Link>
        <sml:source ref="sensorBoardSystem/outputs/temperatureDNOOutput" />
        <sml:destination ref="this/outputs/temperatureDNOOutput" />
      </sml:Link>
    </sml:connection>
    <sml:connection name="convertToTemperature">
      <sml:Link>
        <sml:source ref="sensorBoardSystem/outputs/temperatureDNOOutput" />
        <sml:destination ref="convertorSystem/inputs/celsiusConvInput" />
      </sml:Link>
    </sml:connection>
    <sml:connection name="outputTemperature">
      <sml:Link>
        <sml:source ref="convertorSystem/outputs/celsiusConvOutput" />
        <sml:destination ref="this/outputs/temperatureOutput" />
      </sml:Link>
    </sml:connection>
  </sml:ConnectionList>
</sml:connections>
</sml:System>
</sml:member>
</sml:SensorML>

```

A.2 SenseTile SensorML SensorBoardSystem

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:SensorML rng:version="1.0.1"
xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:rng="http://relaxng.org/ns/structure/1.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xng="http://xng.org/1.0">
  <sml:member>
    <sml:System gml:id="sensorBoardSystem">
      <gml:description>A SenseTile Sensor Board.</gml:description>
      <gml:name>SenseTileSensorBoard</gml:name>

      <!-- Keywords -->

      <sml:keywords>
        <sml:KeywordList>
          <sml:keyword>sensors</sml:keyword>
        </sml:KeywordList>
      </sml:keywords>

      <!-- Identification -->

      <sml:identification>
        <sml:IdentifierList>
          <sml:identifier name="longName">
            <sml:Term definition="urn:ogc:def:identifier:longname">
              <sml:value>UCD SenseTile Sensor Board</sml:value>
            </sml:Term>
          </sml:identifier>
          <sml:identifier name="shortname">
            <sml:Term definition="urn:ogc:def:identifier:OGC:shortname">
              <sml:value>Sensor Board</sml:value>
            </sml:Term>
          </sml:identifier>
          <sml:identifier name="Model_Number">
            <sml:Term definition="">
              <sml:value>1.0</sml:value>
            </sml:Term>
          </sml:identifier>
        </sml:IdentifierList>
      </sml:identification>

      <!-- Inputs -->

      <sml:inputs>
        <sml:InputList name="inputlist">
          <sml:input name="ambientTemperatureInput">
            <swe:Count/>
          </sml:input>
        </sml:InputList>
      </sml:inputs>

      <!-- Outputs -->

      <sml:outputs>
        <sml:OutputList>
          <sml:output name="temperatureDNOOutput">
            <swe:Count/>
          </sml:output>
        </sml:OutputList>
      </sml:outputs>
    </sml:System>
  </sml:member>
</sml:SensorML>
```

```

        </sml:output>
    </sml:OutputList>
</sml:outputs>

<!-- Components -->

<sml:components>
    <sml:ComponentList>
        <sml:component name="thermistor" xlink:href="Thermistor.xml"/>
    </sml:ComponentList>
</sml:components>

<!-- Connections -->
<sml:connections>
    <sml:ConnectionList>
        <sml:connection name="ambientTemperatureInput">
            <sml:Link>
                <sml:source ref="this/inputs/ambientTemperatureInput"/>
                <sml:destination ref="thermistor/inputs/thermistorInput"/>
            </sml:Link>
        </sml:connection>
        <sml:connection name="temperatureDN">
            <sml:Link>
                <sml:source ref="thermistor/outputs/thermistorOutput"/>
                <sml:destination ref="this/outputs/temperatureDNOutput"/>
            </sml:Link>
        </sml:connection>
    </sml:ConnectionList>
</sml:connections>
</sml:System>
</sml:member>
</sml:SensorML>

```

A.3 SenseTile SensorML Converter System

```

<?xml version="1.0" encoding="UTF-8"?>
<sml:SensorML rng:version="1.0.1"
xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:rng="http://relaxng.org/ns/structure/1.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xng="http://xng.org/1.0">
    <sml:member>
        <sml:System gml:id="convertorSystem">
            <gml:description>System that contains process for
                postprocess sensetile sensorboard observations</gml:description>
            <gml:name>SenseTileConvertorSystem</gml:name>

            <!-- Keywords -->

            <sml:keywords>
                <sml:KeywordList>
                    <sml:keyword>sensors</sml:keyword>
                </sml:KeywordList>
            </sml:keywords>

```

```

<!-- Identification -->

<sml:identification>
  <sml:IdentifierList>
    <sml:identifier name="longName">
      <sml:Term definition="urn:ogc:def:identifier:longname">
        <sml:value>UCD SenseTile Sensor Board</sml:value>
      </sml:Term>
    </sml:identifier>
    <sml:identifier name="shortname">
      <sml:Term definition="urn:ogc:def:identifier:OGC:shortname">
        <sml:value>Sensor Board</sml:value>
      </sml:Term>
    </sml:identifier>
    <sml:identifier name="Model_Number">
      <sml:Term definition="">
        <sml:value>1.0</sml:value>
      </sml:Term>
    </sml:identifier>
  </sml:IdentifierList>
</sml:identification>

<!-- Inputs -->

<sml:inputs>
  <sml:InputList name="inputlist">
    <sml:input name="celciusConvInput">
      <swe:Count />
    </sml:input>
  </sml:InputList>
</sml:inputs>

<!-- Outputs -->

<sml:outputs>
  <sml:OutputList>
    <sml:output name="celciusConvOutput">
      <swe:Quantity definition="urn:ogc:def:phenomenon:temperature">
        <swe:uom xlink:href="urn:ogc:def:unit:celsius" />
      </swe:Quantity>
    </sml:output>
  </sml:OutputList>
</sml:outputs>

<!-- Components -->

<sml:components>
  <sml:ComponentList>
    <sml:component name="celciusConv" xlink:href="CelciusConverter.xml" />
  </sml:ComponentList>
</sml:components>

<!-- Connections -->

<sml:connections>
  <sml:ConnectionList>

    <sml:connection name="convertToTemperature">
      <sml:Link>
        <sml:source ref="this/inputs/celciusConvInput" />
        <sml:destination ref="celciusConv/inputs/celciusConvInput" />
      </sml:Link>
    </sml:connection>
  </sml:ConnectionList>

```

```

        </sml:connection>
        <sml:connection name="outputRealTemperature">
          <sml:Link>
            <sml:source ref="celciusConv/outputs/celciusConvOutput"/>
            <sml:destination ref="this/outputs/celciusConvOutput"/>
          </sml:Link>
        </sml:connection>
      </sml:ConnectionList>
    </sml:connections>
  </sml:System>
</sml:member>
</sml:SensorML>

```

A.4 SenseTile SensorML Thermistor

```

<?xml version="1.0" encoding="UTF-8"?>
<sml:SensorML rng:version="1.0.1"
xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:rng="http://relaxng.org/ns/structure/1.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xng="http://xng.org/1.0">
  <sml:member>
    <sml:Component gml:id="thermistor">
      <gml:description>A Digital Temperature Sensor.</gml:description>
      <gml:name>SenseTileThermistor</gml:name>

      <!-- Keywords -->

      <sml:keywords>
        <sml:KeywordList>
          <sml:keyword>sensor thermistor</sml:keyword>
        </sml:KeywordList>
      </sml:keywords>

      <!-- Identification -->

      <sml:identification>
        <sml:IdentifierList>
          <sml:identifier name="longName">
            <sml:Term definition="urn:ogc:def:identifier:longname">
              <sml:value>UCD SenseTile Sensor Board Thermistor
                </sml:value>
            </sml:Term>
          </sml:identifier>
          <sml:identifier name="shortname">
            <sml:Term definition="urn:ogc:def:identifier:OGC:shortname">
              <sml:value>Sensor Board Thermistor</sml:value>
            </sml:Term>
          </sml:identifier>
          <sml:identifier name="Model_Number">
            <sml:Term definition="">
              <sml:value>TMP175</sml:value>
            </sml:Term>
          </sml:identifier>
        </sml:IdentifierList>
      </sml:identification>
    </sml:Component>
  </sml:member>
</sml:SensorML>

```

```

</sml:identification>

<!-- Capabilities -->

<capabilities name="Measurement_Properties">
  - <swe:DataRecord definition="urn:ogc:def:property:measurementProperties">
    <gml:description>The Senstile Sensor Board Temperature Sensor performs
      temperature measurement in a variety of communication,
      computer, consumer, environmental, industrial, and
      instrumentation applications.</gml:description>

    - <swe:field name="Temperature_Resolution"
      xlink:arcrole="urn:ogc:def:property:resolution">
      <swe:Quantity definition="urn:ogc:def:property:temperature">
        <swe:uom code="cel" />
        <swe:value>0.0625</swe:value>
      </swe:Quantity>
    </swe:field>
    <swe:field name="Temperature_Range"
      xlink:arcrole="urn:ogc:def:property:dynamicRange">
      <swe:QuantityRange definition="urn:ogc:def:property:temperature">
        <swe:uom code="cel" />
        <swe:value>-45 125</swe:value>
      </swe:QuantityRange>
    </swe:field>
    <swe:field name="Absolute_Accuracy"
      xlink:arcrole="urn:ogc:def:property:accuracy">
      <swe:QuantityRange definition="urn:ogc:def:property:absoluteAccuracy">
        <swe:uom code="%" />
        <swe:value>-2.0 2.0</swe:value>
      </swe:QuantityRange>
    </swe:field>
  </swe:DataRecord>
</capabilities>

<!-- Manufacturer contact -->

<contact xlink:href="./TexasInstruments"
  xlink:arcrole="urn:ogc:def:role:manufacturer" />

<!-- Documentation -->

<documentation xlink:arcrole="urn:ogc:role:specificationSheet">
  <Document>
    <gml:description>Specification sheet for the 175 thermistor
      </gml:description>
    <format>pdf</format>
    <onlineResource xlink:href="http://www.sensetile/sensor/Thermistor.pdf" />
  </Document>
</documentation>

<!-- Inputs -->

<sml:inputs>
  <sml:InputList>
    <sml:input name="thermistorInput">
      <swe:Count />
    </sml:input>
  </sml:InputList>
</sml:inputs>

<!-- Outputs -->

```

```

    <sml:outputs>
      <sml:OutputList>
        <sml:output name="thermistorOutput">
          <swe:Count definition="urn:ogc:def:phenomenon:temperature">
            <swe:uom xlink:href="urn:ogc:def:unit:celsius"/>
            <swe:constraint>
              <swe:AllowedValue id="temperatureRange">
                <swe:interval>-45 125</swe:interval>
              </swe:AllowedValue>
            </swe:constraint>
          </swe:Count>
        </sml:output>
      </sml:OutputList>
    </sml:outputs>

    <!-- Method -->
    <method xlink:href="urn:ucd:sensetile:sensorboard:thermistor" />

  </sml:Component>
</sml:member>
</sml:SensorML>

```

A.5 SenseTile SensorML CelciusConvertor

```

<?xml version="1.0" encoding="UTF-8"?>
<sml:SensorML rng:version="1.0.1"
xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:rng="http://relaxng.org/ns/structure/1.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xng="http://xng.org/1.0">
  <sml:member>
    <sml:Component gml:id="celciusConv">
      <gml:description>Converts digital number from thermistor to Celcius
        </gml:description>
      <gml:name>SenseTileCelciusConv</gml:name>

      <!-- Keywords -->

      <sml:keywords>
        <sml:KeywordList>
          <sml:keyword>converter</sml:keyword>
        </sml:KeywordList>
      </sml:keywords>

      <!-- Identification -->

      <sml:identification>
        <sml:IdentifierList>
          <sml:identifier name="longName">
            <sml:Term definition="urn:ogc:def:identifier:longname">
              <sml:value>UCD SenseTile Sensor Board Celcius Converter
                </sml:value>
            </sml:Term>
          </sml:identifier>

```

```

        <sml:identifier name="shortname">
            <sml:Term definition="urn:ogc:def:identifier:OGC:shortname">
                <sml:value>Sensor Board Celcius Converter</sml:value>
            </sml:Term>
        </sml:identifier>
    </sml:IdentifierList>
</sml:identification>

<!-- Inputs -->
<sml:inputs>
    <sml:InputList>
        <sml:input name="celciusConvInput">
            <swe:Count />
        </sml:input>
    </sml:InputList>
</sml:inputs>

<!-- Outputs -->
<sml:outputs>
    <sml:OutputList>
        <sml:output name="celciusConvOutput">
            <swe:Quantity definition="urn:ogc:def:phenomenon:temperature">
                <swe:uom xlink:href="urn:ogc:def:unit:celsius" />
                <swe:constraint>
                    <swe:AllowedValue id="temperatureRange">
                        <swe:interval>-45 125</swe:interval>
                    </swe:AllowedValue>
                </swe:constraint>
            </swe:Quantity>
        </sml:output>
    </sml:OutputList>
</sml:outputs>

<!-- Method -->
<method xlink:href="urn:ucd:sensetile:sensorboard:celciusConv" />

</sml:Component>
</sml:member>
</sml:SensorML>

```

Appendix B: **Appendix: SensorML BON Specification**

```
static_diagram SENSORML_SWE
component

cluster SENSORML --sml namespace prefix
component

effective class SensorML
  indexing
  about:          "SensorML document root.";
  title:          "connections.";
  author:         "Ciaran Palmer.";
  copyright:      "none.";
  organisation:   "School of Computer Science and Informatics, UCD.";
  date:           "20100402.";
  version:        "Revision: 1.00.";

  feature

  make
    ->member_array:SEQUENCE[Member]
    require
      member_array/=Void;
    ensure
      for_all i:INTEGER such_that 0 <= i and i < member_array.length\
        \it_holds member.item(i) = member_array.item(i);
    end

  member:SEQUENCE[Member]
    ensure
      Result /= Void;
    end

end --SensorML

effective class Member
  indexing
  about:          "SensorML member.";
  title:          "connections.";
  author:         "Ciaran Palmer.";
  copyright:      "none.";
  organisation:   "School of Computer Science and Informatics, UCD.";
  date:           "20100402.";
  version:        "Revision: 1.00.";

  feature

  make
    ->processIn:AbstractProcess
    ->arcroleIn:STRING
    require
      processIn/=Void;
      arcroleIn/=Void;
    ensure
```

```

    delta{process, arcrole};
    process = processIn;
    arcrole = arcroleIn;
end

process:AbstractProcess
  ensure
    Result /= Void;
  end

arcrole:STRING — not mapped
  ensure
    Result /= Void;
  end

documentList:STRING — DocumentList not mapped
  ensure
    Result = "DocumentList";
  end

contactList:STRING — ContactList not mapped
  ensure
    Result = "ContactList";
  end
end —Member

deferred class AbstractProcess
  indexing
    about:      "Base substitution group for all processes.";
    title:      "AbstractProcess.";
    author:     "Ciaran Palmer.";
    copyright:  "none.";
    organisation: "School of Computer Science and Informatics, UCD.";
    date:       "20100402.";
    version:    "Revision: 1.00.";

  feature

  make
    ->inputsIn:SEQUENCE[Input]
    ->outputsIn:SEQUENCE[Output]
    require
      inputsIn/=Void;
      outputsIn/=Void;
    ensure
      delta{inputs, outputs};
      for_all i:INTEGER such_that 0 <= i and i < inputsIn.length\
        \it_holds inputs.item(i) = inputsIn.item(i);
      for_all i:INTEGER such_that 0 <= i and i < outputsIn.length\
        \it_holds outputs.item(i) = outputsIn.item(i);
    end

  — optional
  inputs:SEQUENCE[Input] — process input ports

  — optional
  outputs:SEQUENCE[Output] — process output ports

  — optional
  parameters:Parameters — process parameters ports

```

```

metaDataGroup:STRING — MetaDataGroup type not mapped
  ensure
    Result = "metaDataGroup";
  end

setInputs
->inputsIn:SEQUENCE[Input]
  require
    inputsIn/=Void;
  ensure
    delta{inputs};
    for_all i:INTEGER such_that 0 <= i and i < inputsIn.length\
      \it_holds inputs.item(i) = inputsIn.item(i);
  end

setOutputs
->outputsIn:SEQUENCE[Output]
  require
    outputsIn/=Void;
  ensure
    delta{outputs};
    for_all i:INTEGER such_that 0 <= i and i < outputsIn.length\
      \it_holds outputs.item(i) = outputsIn.item(i);
  end

setParameters
->parametersIn:SEQUENCE[Parameter]
  require
    parametersIn/=Void;
  ensure
    delta{parameters};
    for_all i:INTEGER such_that 0 <= i and i < parametersIn.length\
      \it_holds parameters.item(i) = parametersIn.item(i);
  end
end — AbstractProcess

effective class Port
  indexing
  about:      "Port for processes";
  title:      "Port";
  author:     "Ciaran Palmer.";
  copyright:  "none.";
  organisation: "School of Computer Science and Informatics, UCD.";
  date:       "20100402.";
  version:    "Revision: 1.00.";

  feature

  make
    ->nameIn:STRING
    ->processNameIn:STRING
    ->dataIn:ComponentData
    ->portTypeIn:INTEGER
  require
    nameIn/=Void;
    dataIn/=Void;
    portTypeIn/=Void;
    PORT.TYPE.legal_portType(portTypeIn);
  ensure
    delta{name, data, processNameIn, portTypeIn};
    name = nameIn;
    data = dataIn;
    portType = portTypeIn;

```

```

    processName = processNameIn;
end

ref:STRING — attribute ref of enclosing reference string
ensure
    Result /= Void;
    — return a STRING concat of /processName/portType/name
end

data:ComponentData

observableProperty:STRING —ObservableProperty type not mapped

AssociationAttributeGroup

invariant
    ((data /= Void) or (observableProperty /= Void)) and (name /= Void);

end —Port

effective class PORT_TYPE
    indexing
        about:      "Allowed Port Types.";
        title:      "PORT_TYPE.";
        author:     "Ciaran Palmer.";
        copyright:  "none.";
        organisation: "School of Computer Science and Informatics, UCD.";
        date:       "20100402.";
        version:    "Revision: 1.00.";

feature
    porttypes: SEQUENCE[STRING]
    ensure
        Result = old portType;
        Result /= Void;
    end
    legal_PortType: BOOLEAN
        -> portType: STRNG
    require
        e /= Void;
    ensure
        Result /= Void;
        (portType = porttypes.item(0) or portType = porttypes.item(1)\
        \or portType = porttypes.item(2)) <-> Result;
    end

INPUT: STRING
ensure
    Result = "Input";
    Result /= Void;
end

OUTPUT: STRING
ensure
    Result = "Output";
    Result /= Void;
end

PARAMETER: STRING
ensure
    Result = "Parameter";
    Result /= Void;
end

```

```

    end

feature {NONE}
    make
    invariant
        legal_PortType (INPUT);
        legal_PortType (OUTPUT);
        legal_PortType (PARAMETER);
end —PORT.TYPE

effective class Link
    indexing
    about:      "Link object used to make connections between processes.";
    title:      "Link.";
    author:     "Ciaran Palmer.";
    copyright:  "none.";
    organisation: "School of Computer Science and Informatics, UCD.";
    date:      "20100402.";
    version:   "Revision: 1.00.";

    feature

    make
        ->sourceIn:Port
        ->destinationIn:Port
        require
            sourceIn/=Void;
            destinationIn/=Void;

        ensure
            delta { source , destination };
            source = sourceIn;
            destination = destinationIn;
        end

    source:Port
        ensure
            Result /= Void;
        end

    destination:Port
        ensure
            Result /= Void;
        end
    invariant
        (source/=Void) and (destination /= Void)

end —Link

deferred class Component —
    indexing
    about:      "Component.";
    title:      "Component.";
    author:     "Ciaran Palmer.";
    copyright:  "none.";
    organisation: "School of Computer Science and Informatics, UCD.";
    date:      "20100402.";
    version:   "Revision: 1.00.";

    inherit ProcessModel
    feature

```

```

    physicalPropertiesGroup:STRING —PhysicalPropertiesGroup type not mapped to BON
    ensure
      Result = "PhysicalPropertiesGroup";
    end
end —Component

```

```

effective class System

```

```

  indexing
  about:      "System is a composite component with physical proprieties
              containing sub-components.";
  title:      "System.";
  author:     "Ciaran Palmer.";
  copyright:  "none.";
  organisation: "School of Computer Science and Informatics, UCD.";
  date:       "20100402.";
  version:    "Revision: 1.00.";

```

```

  inherit ProcessChain
  feature

```

```

    physicalPropertiesGroup:STRING —PhysicalPropertiesGroup type not mapped to BON
    ensure
      Result = "PhysicalPropertiesGroup";
    end
end —System

```

```

deferred class ProcessModel

```

```

  indexing
  about: "ProcessModel.";
  title: "ProcessModel.";
  author: "Ciaran Palmer.";
  copyright: "none.";
  organisation: "School of Computer Science and Informatics, UCD.";
  date: "20100402.";
  version: "Revision: 1.00.";

```

```

  inherit AbstractProcess
  feature

```

```

  method:STRING —ProcessMethod type not mapped
  ensure
    Result /= Void;
  end

```

```

end —ProcessModel

```

```

effective class ProcessChain

```

```

  indexing
  about:      "Process formed by chaining sub-processes.";
  title:      "ProcessChain.";
  author:     "Ciaran Palmer.";
  copyright:  "none.";
  organisation: "School of Computer Science and Informatics, UCD.";
  date:       "2010/04/01.";
  version:    "Revision: 1.00.";

```

```

  inherit AbstractProcess
  feature

```

```

  make

```

```

->componentsIn:AbstractProcess
require
  componentsIn/=Void;
ensure
  delta{components};
  components = componentsIn;
end

— Collection of subprocesses that can be chained using connections
components:SEQUENCE[AbstractProcess]

— links between processes
connections:SEQUENCE[Link]

— Add a link with check that Link is legal
AddLink
->linkIn:Link
require
  linkIn /= Void;
  — check Link source is legal
  (exists i:INTEGER such_that linkIn.source = Current.inputs[i]) or\
  \ (exists i:INTEGER and j:INTEGER such_that linkIn.source =\
  \ components[j].outputs[i]);
  — check Link destination is legal
  (exists i:INTEGER such_that linkIn.destination.name =\
  \ Current.outputs[i].name) or\
  \ (exists i:INTEGER and j:INTEGER such_that linkIn.destination =\
  \ components[j].inputs[i]);
ensure
  delta{connections};
  (exists i:INTEGER such_that connections[i] linkIn);

invariant
  components /= Void;

end —ProcessChain

end — SensorML Cluster

cluster SWECOMMON — swe namespace prefix
component

deferred class DataComponent
indexing
  about:      "Base class for SWE COMMON data types";
  title:      "DataComponent";
  author:     "Ciaran Palmer.";
  copyright:  "none.";
  organisation: "School of Computer Science and Informatics, UCD.";
  date:       "20100402.";
  version:    "Revision: 1.00.";

feature

—optional
name:STRING      —Name for data component
—optional
swe_description:STRING — Description of data component.
— Really description SWECOMMON-BON keyword clash
—optional

```

```

definition:STRING —Identifies the phenomenon or other context
                    —of the value. Takes a definition reference as its value
—optional
fixed:BOOLEAN    — Data cannot be changed if true

set_name
  ->nameIn:STRING
  require
    nameIn /= Void;
  ensure
    delta{name};
    name = nameIn;
  end

set_description
  ->descriptionIn:STRING
  require
    nameIn /= Void;
  ensure
    delta{descriptionSWE};
    descriptionSWE = descriptionIn;
  end

set_definition
  ->definitionIn:STRING
  require
    definitionIn /= Void;
  ensure
    delta{definition};
    definition = definitionIn;
  end

set_fixed
  ->fixedIn:BOOLEAN
  require
    fixedIn /= Void;
  ensure
    delta{fixed};
    fixed = fixedIn;
  end
end —DataComponent

effective class Count
  indexing
    about:          "Integer number.";
    title:          "Count.";
    author:         "Ciaran Palmer.";
    copyright:      "none.";
    organisation:   "School of Computer Science and Informatics, UCD.";
    date:           "20100402.";
    version:        "Revision: 1.00.";

  inherit
    AbstractDataComponent

  feature

—optional
value:INTEGER

—optional :
swe_constraint:AllowedValuesCount — constraint SWECOMMON-BON keyword clash

```

```

set_value
->valueIn:INTEGER
require
  valueIn/=Void;
  (swe_constraint /= Void) or (swe_constraint.allowedValue(valueIn));
ensure
  delta{value};
  value = valueIn;
end

--quality not mapped
--axisID not mapped
--referenceFrame not mapped
end --Count

```

effective class Quantity

```

indexing
about: "Decimal number with optional unit and constraints.";
title: "Quantity.";
author: "Ciaran Palmer.";
copyright: "none.";
organisation: "School of Computer Science and Informatics, UCD.";
date: "20100402.";
version: "Revision: 1.00.";

```

inherit

AbstractDataComponent

feature

```

--optional
value:DOUBLE -- value of type.

```

```

--optional
uom:Uom -- Unit of measure

```

```

--optional
swe_constraint:AllowedValuesQuantity -- constraints for Quantity value

```

```

set_value
->valueIn:DOUBLE
require
  valueIn/=Void;
  (swe_constraint /= Void) or (swe_constraint.allowedValue(valueIn));
ensure
  delta{value};
  value = valueIn;
end

```

end --Quantity

effective class Uom

```

indexing
about: "Uom type that indicates unit-of-measure,.";
title: "UomPropertyType.";
author: "Ciaran Palmer.";
copyright: "none.";
organisation: "School of Computer Science and Informatics, UCD.";
date: "20100402.";
version: "Revision: 1.00.";

```

feature

```

make

    code:STRING —Property type that indicates unit-of-measure (UCUM)
    unitDefinitionHRef:STRING —reference known unit definitions

end —Uom

effective class AllowedValuesQuantity
indexing
about: "AllowedValues.";
title: "AllowedValues.";
author: "Ciaran Palmer.";
copyright: "none.";
organisation: "School of Computer Science and Informatics, UCD.";
date: "20100402.";
version: "Revision: 1.00.";

feature

make
->minIn:DOUBLE
require
    minIn/=Void;
    max = Void;
    interval = Void;
    valueList = Void;
ensure
    delta{min};
    min = minIn;
end

make0
->maxIn:DOUBLE
require
    maxIn/=Void;
    min = Void;
    interval = Void;
    valueList = Void;
ensure
    delta{max};
    max = maxIn;
end

make1
->intervalIn:SEQUENCE[DOUBLE]
require
    intervalIn/=Void;
    intervalIn.length = 2;
    intervalIn.item(0) < intervalIn.item(1)
    min = Void;
    max = Void;
    valueList = Void;
ensure
    delta{intervalIn};
    intervalIn.item(0) = intervalIn.item(0);
    intervalIn.item(1) = intervalIn.item(1);
end

make2
->valueListIn:SET[DOUBLE]
require
    valueListIn/=Void;

```

```

    min = Void;
    max = Void;
    interval = Void;
    ensure
      delta{valueListIn};
      for_all i:INTEGER such_that 0 <= i and i < connection_array.length\
        \it_holds valueList.item(i) = valueListIn.item(i);

    end

id:String

min:DOUBLE — optional Specifies minimum allowed value for
             — an open interval (no max)

max:DOUBLE — optional Specifies maximum allowed value for
             — an open interval (no min)

interval:SEQUENCE[DOUBLE] — optional Range of allowed values
                           — (closed interval)

valueList:SET [DOUBLE] — optional List of allowed values for this component.

allowedValue:BOOLEAN
->valueIn:DOUBLE
require
  valueIn/=Void;
ensure
  Result <=> ((min=Void or valueIn>=min) or (max=Void or valueIn<=max))\
  or ((interval=Void) or (interval.item(0)<=valueIn<=interval.item(0)))\
  or ((valueList=Void) or (valueIn member_of valueList));

invariant
  (min/=Void) or (max/=Void) or (interval/= Void) or (valueList/=Void);

end — AllowedValuesQuantity

effective class AllowedValuesCount
  indexing
  about: " AllowedValuesCount .";
  title: " AllowedValues .";
  author: " Ciaran Palmer .";
  copyright: " none .";
  organisation: " School of Computer Science and Informatics , UCD .";
  date: " 20100402 .";
  version: " Revision: 1.00 .";

  feature

  make
    ->minIn:INTEGER
    require
      minIn/=Void;
      max = Void;
      interval = Void;
      valueList = Void;
    ensure
      delta{min};
      min = minIn;
    end

  make0

```

```

->maxIn:INTEGER
require
  maxIn/=Void;
  min = Void;
  interval = Void;
  valueList = Void;
ensure
  delta{max};
  max = maxIn;
end

```

```

make1
->intervalIn:SEQUENCE [INTEGER]
require
  intervalIn/=Void;
  intervalIn.length = 2;
  intervalIn.item(0) < intervalIn.item(1);
  min = Void;
  max = Void;
  valueList = Void;
ensure
  delta{intervalIn};
  intervalIn.item(0) = intervalIn.item(0);
  intervalIn.item(1) = intervalIn.item(1);
end

```

```

make2
->valueListIn:SET [INTEGER]
require
  valueListIn/=Void;
  min = Void;
  max = Void;
  interval = Void;
ensure
  delta{valueListIn};
  for_all i:INTEGER such_that 0 <= i and i < connection_array.length\
  \it_holds valueList.item(i) = valueListIn.item(i);
end

```

id:String

min:INTEGER — *optional Specifies minimum allowed value for*
— an open interval (no max)

max:INTEGER — *optional Specifies maximum allowed value for*
— an open interval (no min)

interval:SEQUENCE [INTEGER] — *optional Range of allowed values*
— (closed interval)

valueList:SET [INEGER] — *optional List of allowed values*
— for this component. Set is better.

```

allowedValue:BOOLEAN
->valueIn:INTEGER
require
  valueIn/=Void;
ensure
  Result <-> ((min=Void or valueIn>=min) or (max=Void or valueIn<=max)\
  or (interval=Void or interval.item(0)<=valueIn<=interval.item(0))\
  or (valueList=Void or (valueIn member_of valueList)));

```

```

    invariant
      (min/=Void) or (max/=Void) or (interval/= Void) or (valueList/=Void);

end — AllowedValuesCount

end — SWECOMMON cluster

cluster GML — gml namespace prefix
component
  deferred class GMLFeature
    indexing
      about: "SenseTile System.";
      title: "SenseTileSystem.";
      author: "Ciaran Palmer.";
      copyright: "none.";
      organisation: "School of Computer Science and Informatics, UCD.";
      date: "20100402.";
      version: "Revision: 1.00.";

    feature

      gml-description:STRING
      name:STRING
      boundedBy:boundedBy

    end — GMLFeature

  end — GML cluster

end

```