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 13, 2010

# Table of Contents

# Abstract

This thesis 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 thesis finally examines formalizing SensorML using a formal specification language called Business Object Notation.

# Acknowledgments

Vieri del Bianco , Dragan Stosic and Joseph R. Kiniry.

# 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 and expensive. 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 issues. 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) initiative [1]. 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. This thesis investigates the use of these SWE SensorML and SOS standards in developing sensor systems.

A real sensor system called SenseTile[ref] 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 thesis describes a SensorML model of SenseTile, in particular the SenseTile Sensor Board. 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 [4] which has the goal of developing very robust software systems. Business Object Notatation(BON)[10] 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 thesis looks at how BON can be used to formally specify SensorML. The BON to SensorML mapping described in thesis allows SensorML sensor descriptions to then be refined to JML[2] and then to Java[3], as described in [4].

[1]http://www.opengeospatial.org/projects/groups/sensorweb

[2]http://www.eecs.ucf.edu/ leavens/JML/

[3]http://www.java.com/

# Chapter 2: **Background Research**

## 2.1 Sensor Web Enablement

As described in the introduction the SensorML and SOS specifications are developed by the Open Geospatial Consortium (OGC) as part of the Sensor Web Enablement (SWE) initiative [1]. 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:

- SensorModel 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 meaurement. For a good overview of SWE and it's use in Sensor Web Networks see [6]. 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 UML[2] models, the SensorML Conceptual Model and the SWE Common Conceptual Model, that describe SensorML. 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.

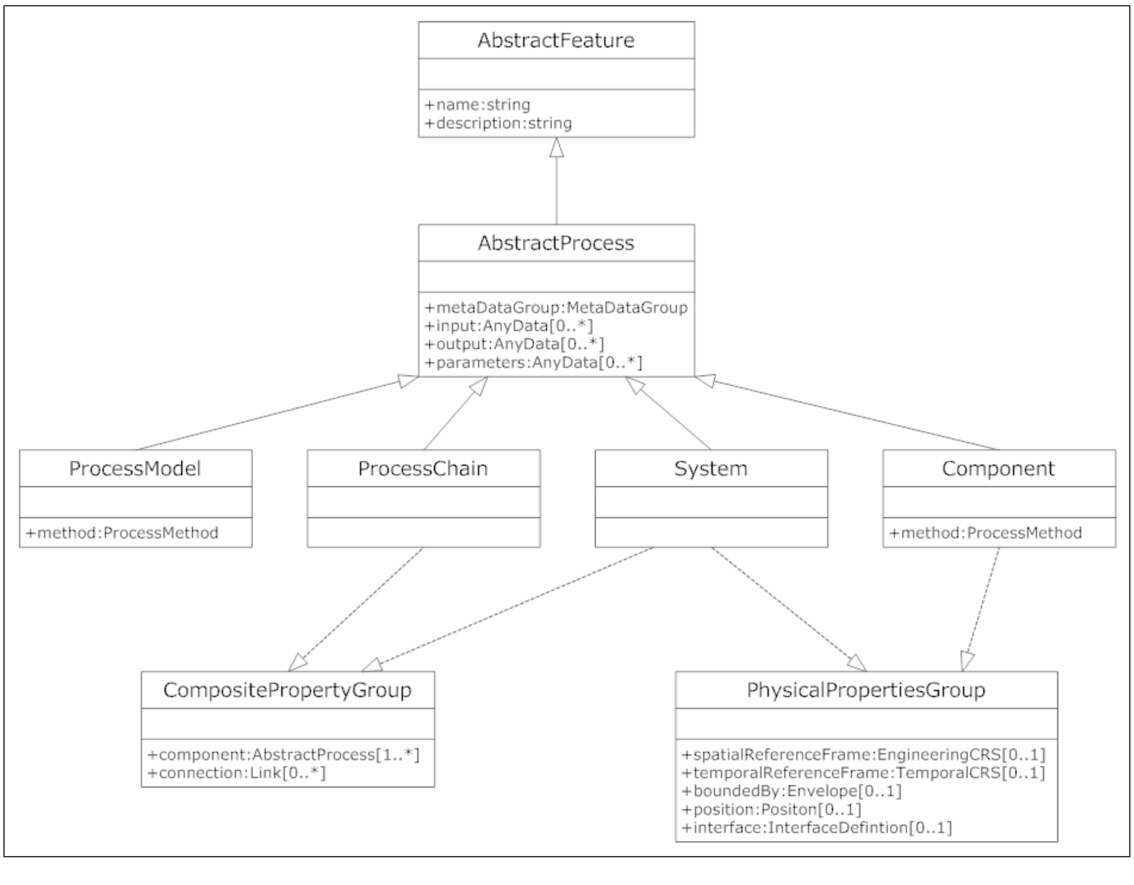

Figure 2.1: SensorML Conceptual Model

[2]http://www.uml.org/

#### Processes

The AbstractProcess models processes in SensorML and 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 MetaDataGroup. The MetaDataGroup contains properties such as capabilities of the sensor, contact information and documentation references. It is generally used to assist humans in understanding the sensor system described 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 and is modeled by the ProcessModel type. Physical processes provide physical information, such as location, which can be used in processing if required and is modeled by the Component type. Both process types contain the method property. The method property is a ProcessMethod type describes the methodology for transforming the process inputs to outputs.

#### Process Chains

There is a similar division of the process chains into the non-physical ProcessChain and physical System. Both process chains can contain a mixture of ProcessModel and Component processes if desired. 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 CompositePropertGroup referenced by the ProcessChain and System types is a sequence of type Link. The Link type contains source and destination properties that reference inputs and outputs of the processes in the process chain thus connecting the processes.

#### Linkable Properties

SensorML uses XML Linking Language (XLink) [3] to support hypertext referencing in SensorML XML documents using URNs[4] and URIs[5]. 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 2.2 model scalar primitive values. They are used to defined the type of day used by the inputs, outputs and parameters in SensorML processes. Quantity, for example,

[3]http://www.w3.org/TR/xlink
[4]http://tools.ietf.org/html/rfc2141
[5]http://tools.ietf.org/html/rfc3986

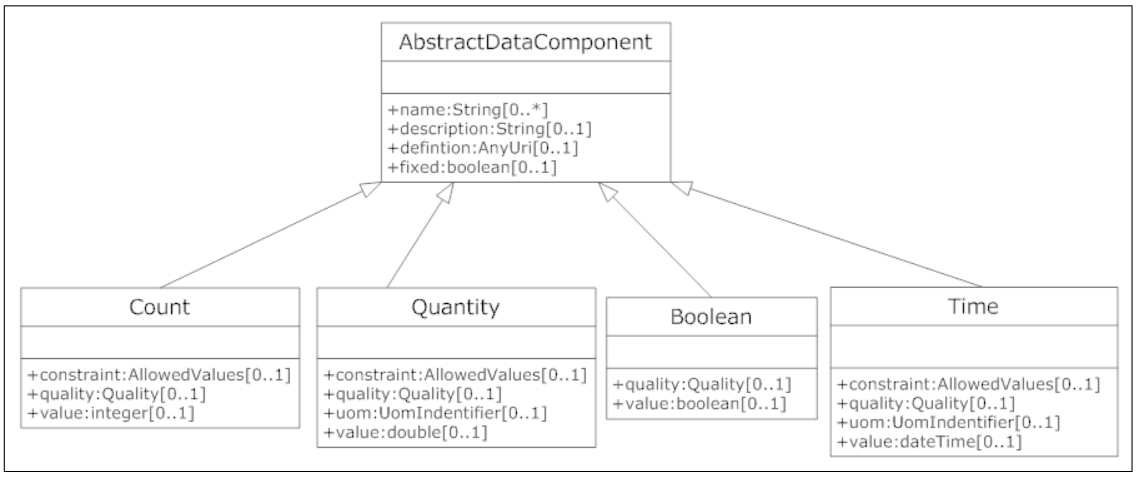

Figure 2.2: SWE Common Simple Data Types.

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 allow a more reliable processing of sensor data the types contain a number of properties to provide metadata about the value carried in the simple type. 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 Engine parses SensorML models and instantiates the required objects to performed the intended processing. The code for the SensorML processing engine is found at [6]. 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 types described in the section 2.2.2. This code is found at [7].

## 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

[6]http://code.google.com/p/sensorml-data-processing

[7]http://code.google.com/p/swe-common-data-framework

a collection called an observation offering. The offering information is requested by clients to get the identity of senors they are interested in getting observations from.

The SOS specification uses four main entities to describe the SOS's operation. The entities are Data Consumers, Data Producers, OGC Catalogs and the SOS itself.

- Data Consumers are clients of the SOS that request sensor observations and sensor metadata.
- Data Providers are entities read the 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 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 entities described in the SOS in an operational context.

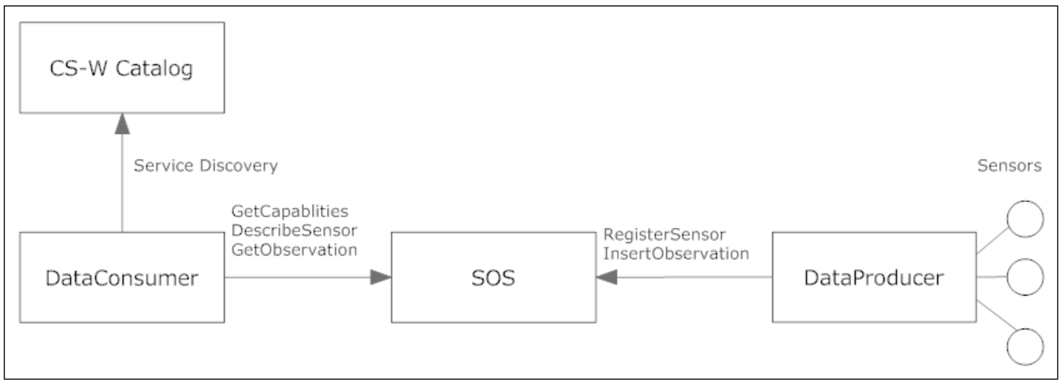

Figure 2.3: SOS Operational Context

The diagram above shows the core and transactional operations provided by an SOS. The core operations are GetCapabilities, DescribeSensor and GetObservations. These are mandatory operations used by Data Consumers to request sensor information. The core operations perform the following function:

- 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 http operations. The sensor observations contained in GetObservation and InsertObservation operations are encoded according to the OGC O&M Observations [5]. Sensor systems metadata contained in DescribeSensor and RegisterSensor is defined in SensorML.

The SenseTile Web Service developed in this thesis is based on the concepts described in the Sensor Observation Service (SOS) specification. The SOS Data Consumer and SOS Data Provider entities will 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 thesis. The Data Consumer uses a CS-W catalog to find SOS instances. The GetCapabilites operation is then used toward each of the SOS instances found to find sensor offerings that are of interest to the Data Consumer. The CS-W catalog is covered in a separate specification [9] and is not covered in any real detail in the SOS specification.

There several optional SOS operations not covered in this thesis. The SOS refers to these as enhanced operations. They are GetResult, GetFeatureOfInterest, GetFeatureOfInterestTime, DescribeFeatureOfInterest, DescribeObservationType, and DescribeResultModel.

## 2.4 SenseTile

A case study of the use of SensorML in sensor networks is based on the SenseTile System. SenseTile is a general purpose sensor system developed at UCD CASL [8]. 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 **??** shows a outline of this system. The sensor platform is called the SenseTile Unit and is composed of one or more

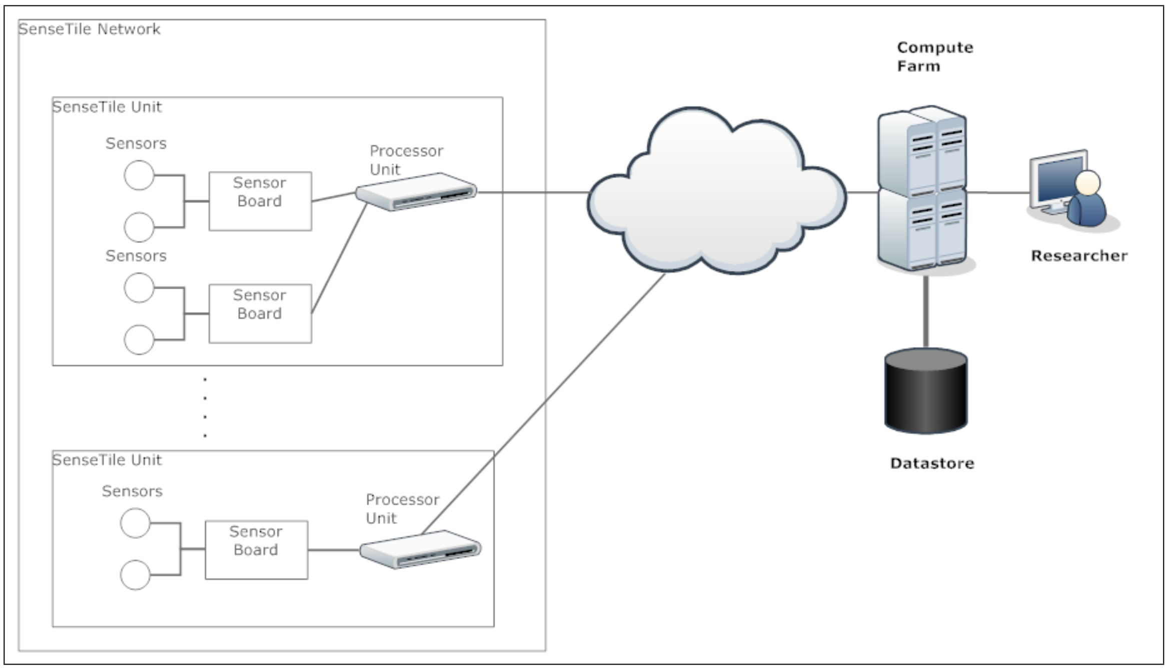

Figure 2.4: SenseTile System

[8]http://kind.ucd.ie/ products/opensource/SenseTile

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.

This SenseTile unit is used here as case study on the use of SensorML, though SensorML can be used in other the parts of the SenseTile system. As the processor unit is a relatively powerful computer, providing a Web Service on the SenseTile Unit is feasible. The part SensorML plays in this Web Service is explored in the context of the SenseTile system.

The TMP175 and TMP75 are digital temperature sensors that are optimal for thermal management and thermal protection applications. The TMP175 and TMP75 are Two-Wire and SMBus interface-compatible, and are specified over a temperature range of 40C to +125C.

### 2.4.1 UCD Sensor Board Driver

## 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 supports design by contract**??** where a function will guarantee 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 [10].

# 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 section2.1 seems to cover all that is needed for modeling sensors and observations. Central to the design is a SensorML model of the SenseTile System. The section SensorML Model of SenseTile covers the details of this model. This model acts as both metadata about the SenseTile system and provides a description of how to process the sensor data for the SenseTile Web Service. The Web Service design is based on the SOS specification and allows access to the SensorML model 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 thesis. The SenseTile Sensor Board and its sensors is modeled in the most detailed. The model of the Sensor Board is placed in a context to examine SensorML process chaining where 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 in figure 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.

The block diagram shows the physical phenomena measured by the sensors being feed as inputs to the Components in the Sensor Board System. The SensorBoard System outputs digital numbers based on the sensors measurement that have no physical meaning yet. The SenseTile 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 provide the raw data from the sensors or the data converted to meaningful values to clients.

Only the Thermistor is modeled in this thesis. The CelsiusConv Component converts the digital number form the Thermistor Component to a Celcius value. The same approach can applied to the other senors though for some components no conversion may be necessary.

A description of the SenseTile Systems and Components used in the SenseTile model is provided in the following sections.

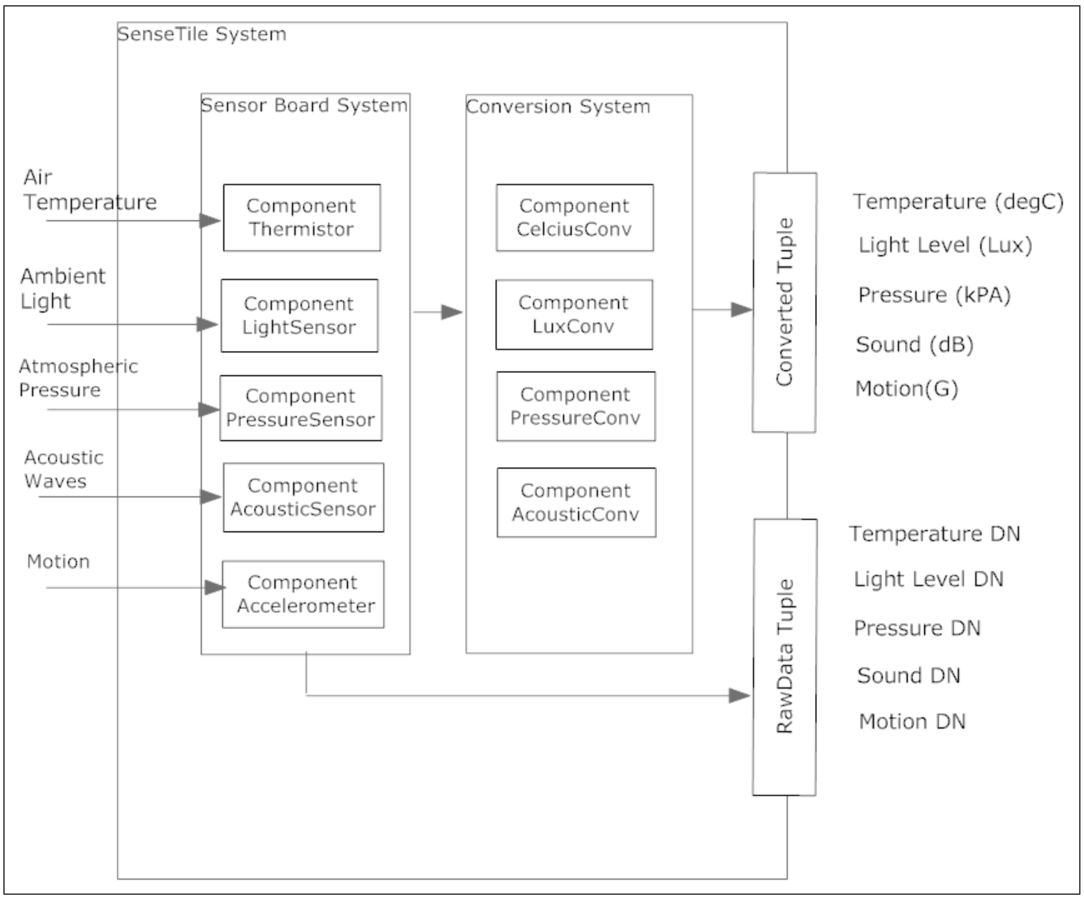

Figure 3.1: SenseTile System SensorML Block Diagram

## 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 as appropriate. The below XML fragment shows how SensorML encodes the containment of these Systems.

```
<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 tags. These component elements name the component which may be another SensorML System Component. The component tag also has an XLink href used to reference the location of the SensorML definition of the component. 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 SenseTile System description. The use of XLink in SensorML is described in the SensorML section 2.2.

The SenseTile System connects up the inputs and outputs of the SensorBoard and Conversion Systems using SensorML connections. The below SensorML fragment shows how the temperature output from the SensorBoard System is connected to the input of the Conversion System.

```
<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/temperatureDNOutput"/>
              <sml:destination ref="this/outputs/temperatureDNOutput"/>
          </sml:Link>
   </sml:connection>
   <sml:connection name="convertToTemperature">
          <sml:Link>
              <sml:source ref="sensorBoardSystem/outputs/temperatureDNOutput"/>
              <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 tag. The contained source and destination elements contain a ref that indicates the components output and input to use. Only one connection is shown here but it is contained in a elements that allow a sequence of connection elements.

### 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 the sensors.

### 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 is an indivisible process and can have a different location information than the Sensor Board if required. The SensorML Component allows the capabilities of the Thermistor to be described. For example the TMP175 is specified for operation over a temperature range of 40C to +125C. This is captured in the following SensorML:

```
<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. The range is constrained to the allowed value range from the Thermistor. The following fragment shows how the input and outputs for the Thermistor look in SensorML:

```
<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 the sensor data to a meaningful value. It this case it connects up the inputs and outputs the CelsiusConv Component.

### 3.2.5 Celsius Converter Component

The SensorML Celsius Converter SensorML Component models a process that performs a conversion of the Thermistors digital number output to a real quantity celsius value. It similar to the Thermistor SensorML description but its inputs and outputs are different. Its process method performs a simple calculation to perform the Celsius conversion.

A Sensor process method is also developed. The process method describes the conversion algorithm and points to an implementation class. Process Methods are not implemented by the Vast Lib so it not used in the running system. Instead the method tag provides a URN that is used to lookup the implementation class.

## 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 will be 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 .

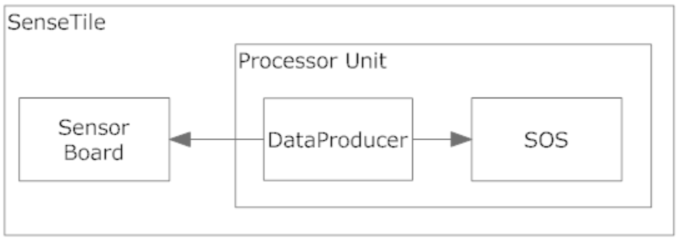

Figure 3.2: SenseTile Web Service

The DataProvider parses a SensorML description of the SenseTile System, reads sensor data form the Sensor Board and provides the sensor data as O&M Observations to the SOS. The SenseTile SOS will implement the SOS core operations to allow a client to get these observations. The following sections describe the design of the DataProvider and SOS components as well as the scenarios they support.

### 3.3.2 SenseTile WebService Network Architecture

In the SenseTile Web Service the SOS acts as a gateway to the SenseTile Sensor Network. This prototype is designed with the network architecture as shown in fig 3.3. A client accesses multiple SOS instances to get observations from the sensor network. An SOS can support several DataProducers and hence would not to be run on every SenseTile Unit.

The SOS Specification envisioned that the SOS is run on a large external server as shown in fig 3.4. The DataProducers update this external server with observations and SOS Data Consumer access this large SOS.

The SOS Specification describes the use of a Proxy SOS eeicpr fix. This architecture could be used with the SenseTile Sensor Network if accessing many SOSs is not feasible for the clients.

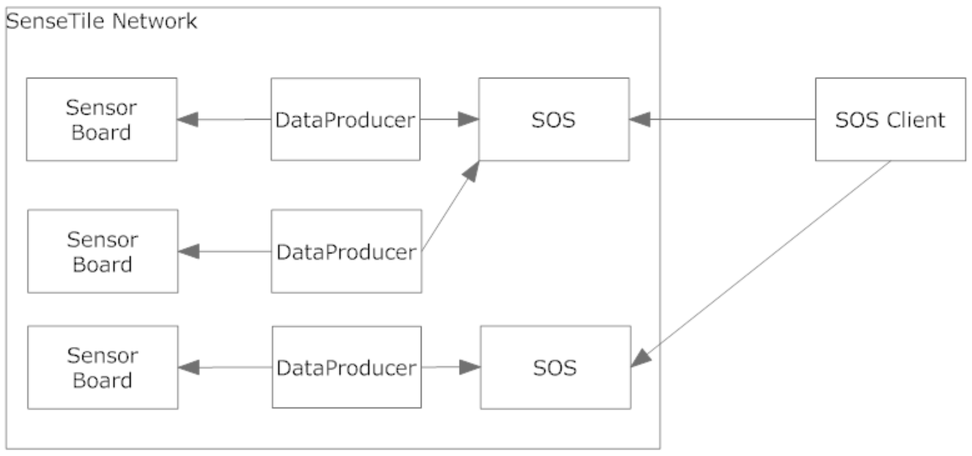

Figure 3.3: SenseTile Sensor Network Architecture

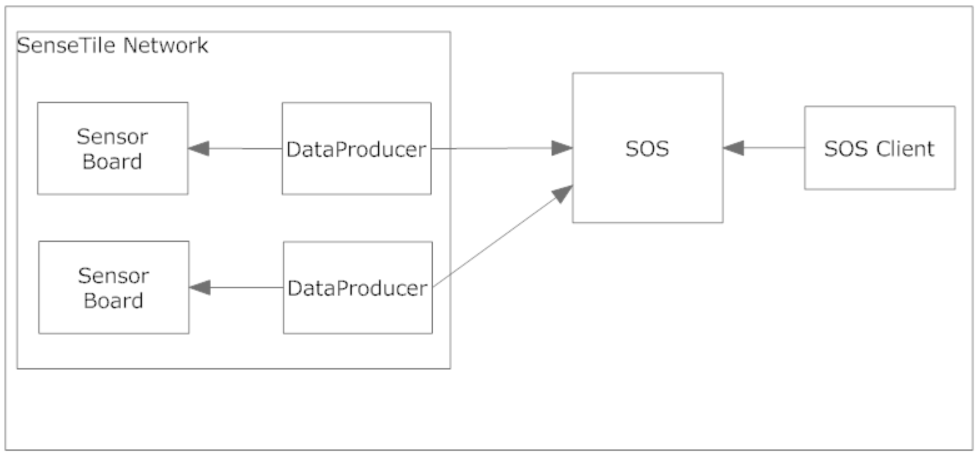

Figure 3.4: External SOS

### 3.3.3 SenseTile Web Service Scenarios

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

#### Generate Observations from Sensor Data

The DataProducer accesses data from the sensor board and updates the SOS with sensor observations.

1. At start up the DataProducer registers the SensorBoard System and the Converter System with the SOS in separate RegistorSensor operations. The SOS creates sensor offerings for each of the Systems and stores the SensorML descriptions of the Systems provided in the RegistorSensor operation. It then returns a unique id (URN) in the RegistorSensor response for use by the DataProducer for subsequent communication with the SOS.

2. The DataProducer reads Sensor Board data packets from the SensorBoardDriver.

3. The DataProducer updates the SOS with sensor observations with the InsertObservation operation.

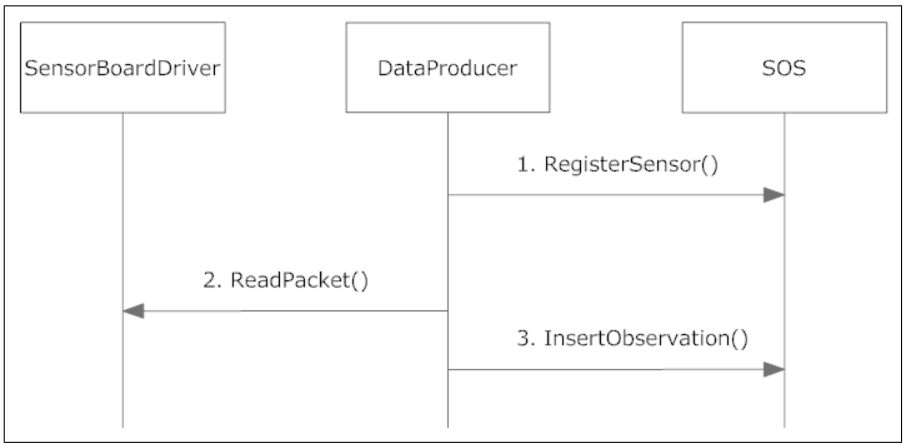

Figure 3.5: Generate Observation

### Get Sensor Description from SOS

The SOS allows DataConsumers to obtain SensorML descriptions of the SesneTile Systems.

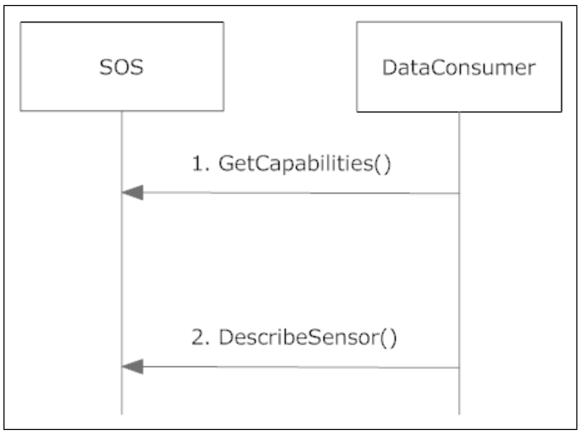

Figure 3.6: Get Sensor Descriptions

1. The DataConsumer requests information on sensor offerings provided by the SOS using the GetCapabilities operation.
2. Using the information in the GetCapabilites 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.

### Get Sensor Observation from SOS

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

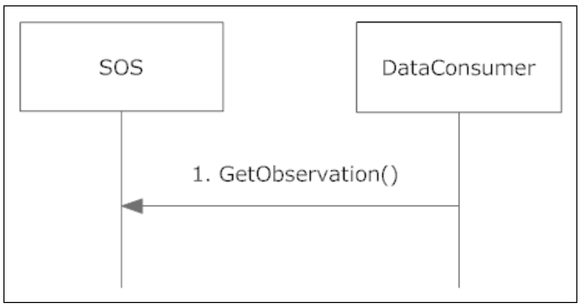

Figure 3.7: Get Observation from Sensor

### 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 processing of the sensor data to generate the observations is described by the SensorML models. The DataProducer uses the OGC Vast Library to instantiate objects that implement the required data processing chains based on the SensorML models. The BON static diagram 3.8 shows the main clusters and classes of the DataProvider.

The SMLEngine cluster contains classes that provide access to the Vast Lib that is providing the SensorML Engine. The Vast Lib implementation of the SensorML Engine is hidden from the rest of the code thus.

The Sensors and Converters clusters contain classes that provide implementation of process methods for SensorML components in the SensorBoard 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

The steps above are detailed the in following sections with reference to the BON static diagram 3.8.

#### 1. Read Sensor Board Data

The UCD SensorBoard Library is used by DataProducer to access the data generated by the Sensor Board. The library provides high level interface to the 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. This library is shown as the SensorBoard cluster in 3.8 An instance of the class PacketInputStream to is used access the packets from the Sensor Board.

The DataProducer uses the Observer pattern to provide packets to the SensorMLEngine for processing. At start up the SenseTileSystem reads the list of sensor objects from the SMLEngine that implement the Sensors interface. These objects provide an entry point to the SensorML processing chain and can read data from the UCD LIB packets. These sensor objects implement reading strategies for the type of data they read. For example the

ThermistorSensor reads out the temperature value but will average it before it sent to the SensorML Engine for processing.

### 2. Process Sensor Data

The PacketInputStream will request the SenseTileSystem to exexcute the SensorML engine to process the sensor data. The SensorML Enigine will execute the process chains described in the SenseTile SensorML model. The sensor objects output will be passed to the converter objects. The outputs of both object types are then available to generate observations.

### 3. Create O&M Observations

When the SensorML Engine is finished and there are obsevations generated these need to be sent to the SOS. An O&M Onservation XML string containing the data is sent to the SOS instance. This is timestamped at this stage. containing the output from the SensorML processes and send the observations to the SOS. -register offerings with the SOS. In this design each system is an offering. Need to generate an observation model?

The DataProducer was developed as a SensorML based software. It parses the SenseTile SensorML model to instantiate the code to execute the processes described. This required two aspects to be developed, a framework to run the processes and classes that implemented the components.

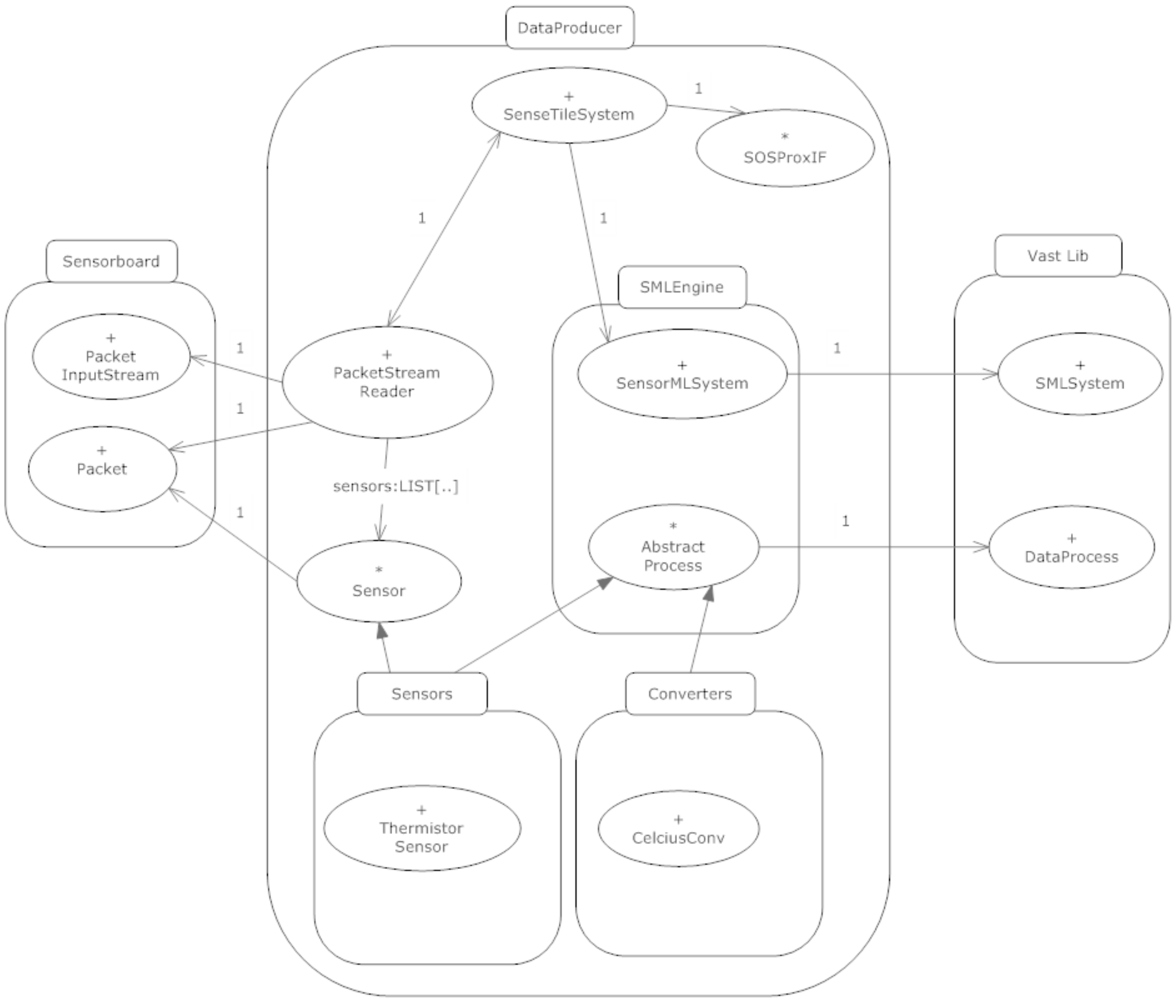

Figure 3.8: DataProducer Static Diagram

### 3.3.5 SOS

The SOS component of the SenseTile Web Service provides a simplified OGC SOS Web interface to access the observations from the DataProducer. The below is a static diagram showing the important clusters and classes in the SOS design.

The SOS is the main cluster and the Service cluster contains the classes that provide the Web Service interface.

The DataProvider 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 and provides an offering that already exists then just a new Sensor instance is added to the existing ObservationOffering.

The Sensor class contains a list of Observations that hold the observations from the DataProvider. The last 50 observations are maintained.

The ObservationService class provides the interface for the DataProducer to update with observations.

The SenseTile Service provides Web Service interface to be used by clients to access the sensor observations.

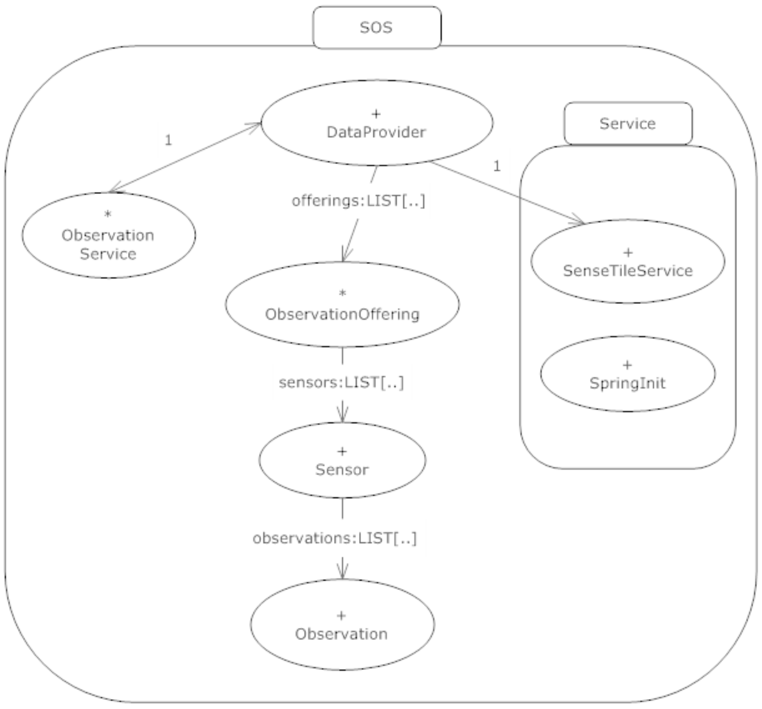

Figure 3.9: SOS Static Diagram

The SOS specification provides a very rich filtering feature that is too complex for this prototype. The operations are developed in a way to allow the sensor metadata and observations to be retrieved for a specific sensor board. The latest observation or all the stored observations can be retrieved.

# Chapter 4: **Implementation**

## 4.1 Overview

The SenseTile Web Service is written in Java. As described in the high level design 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 RMI. RMI is used due to the SOS and DataProducer processes are running in the SenseTile Network and communicate over a LAN. This communication protocol has lower overhead than a Web Service so efficient communication between the DataProducer and SOS is achieved. Though it would not be hard to change to use a Web Service as a Java Interface is used to hide the communication mechanism from the main application code. Clients, SOS DataConsumers, access the SOS using a SOAP based Web Service interface.

The SOS specification describes an implementation using HTTP carry complete operation information encoded in XML including the operations name and parameters. The SenseTile WebService takes a different approach. The operation is defined in the RMI interface and the Web Service Interface while the parameters are encoded in XML thus need parsing.

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 DOM[1] 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 to be performed.

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 website [2] that

[1] http://www.w3.org/DOM/
[2] http://www.ogcnetwork.net/SWE

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. The VastSensorMLEngine processes the model fine and was very stable during testing.

## 4.3 SOS

The SOS is the provider of the Web Service interface and is built on Apache Axis2 [3]. Axis2 is described as a "Web Services / SOAP / WSDL engine" with the Apache Axis2/Java implementation used in this thesis. The Spring framework[4] was also used. Using these libraries a light Web Service is built to run on the reasonably powerful SenseTile Processor Unit. Axis2 is run standalone here though it can be on with any Servlet engine if a more powerful Web Service is needed.

Developing a Web Service is straight forward with Axis2 and it supports several approaches. The supported POJOs (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. Axis2 supports the The Spring framework which is used in the SenseTile WebService for it's dependency injection functionality. A SprinitInit class is needed for Axis2 to load up the Spring framework.

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

JAXB [5] was used for SensorML XML to Java binding in the Web Service. 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. It 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 SensorML Systems.

[3]http://ws.apache.org/axis2
[4]http://www.springsource.org/
[5]http://java.sun.com/developer/technicalArticles/WebServices/jaxb

# Chapter 5: **BON to SensorML mapping**

## 5.1 Overview

In this chapter a BON(see section2.5) to SensorML( see section 2.2) mapping is described. One motivation for this mapping is take advantage of formal software development processes described in [4]. 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 it XML is verbose and a more compact description of a SensorML model as facilitated by BON is naturally preferred. Lastly SensorML is complex and it is difficult to develop correct SenorML models based only on using the SensorML XML Schema definition. How a BON specification of SensorML can help here is described later in the chapter.

## 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 BON SensorML model is shown in figure 5.1. The inheritance

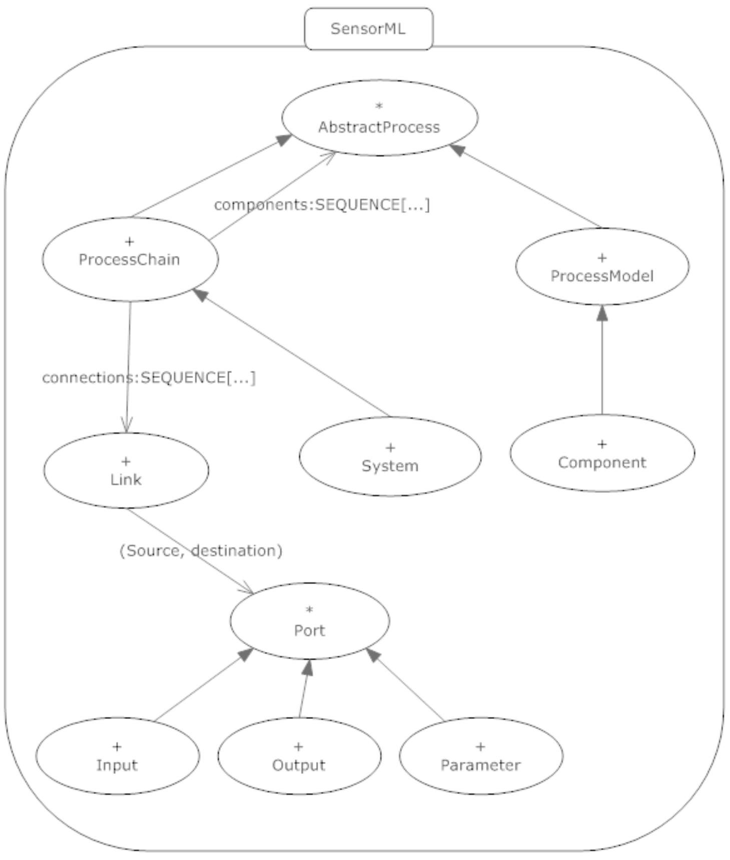

Figure 5.1: BON SensorML Model

heirarhcy is changed slightly with System inheriting from ProcessChain and Component inheriting from ProcessModel. A new abstraction called Port is added. Port contains all the properties of SensorML input, output and parameter as they all have the same properties. A PortType class that ensures legal port types of Input, Output and Parameter is also added to the SenorML 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. There are a many other types but only Count and Quantity are developed as they were the ones used in the SenseTile SensorML model.

The BON specification developed in this thesis does not address the issue of the SensorML metadata and the MetaDataGroup is not specified. 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. AssociationAttributeGroup SensorML Member The complete BON SensorML specification listing can be found in appendix B.1.

## 5.3 SensorML BON to XML Mapping

SensorML is essential XML defined by XML Schema with its main purpose being interopeability. Therefore a mapping SensorML BON specification to the SensorML XML is needed. The BON types map closely to SensorML complex types and so a relatively simple scheme is needed. Effective BON classes map to XML elements. Features in the class map either to attributes of the XML element or a contained XML element. Whether a feature is an attribute is indicated in the class as a comment using @SensorML to make this comment as a mapping instruction. Otherwise an XML element is assumed. The type names of the BON classes and features are the XML element or attribute names. Values. Case of names.

BON SEQUENCE types need a special rule for mapping to SensorML.

Connections rule.

## 5.4 ProcessChain connections problem

One of the major features of SensorML is the ablity to describe processing chains. The SensorML ProcessChain and System types have a connections property that allows process chaining as decribed in section **??**. Strings are used to describe the connection. 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 connectionconvertToTemperature

```
<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 (ensure). 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 listing below.

Listing 5.2: BON ProcessChaing

```
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 = 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
```

## 5.5 SenseTile Thermistor BON SensorML Specification

Thermistor can be generated by the following BON class.

# Chapter 6: **Results**

A SensorML Model description of a SenseTile Sensor Board and Thermistor was developed. It was run on a SensorML engine. Appendix shows the model developed.

Developed lightweight Web Service to explore the use of SensorML in Sensor Networks. running prototype tested with UCD SensorBoard Simulator code.

Tested on a low power Atom PC that is used as SenseTile Processing Unit.

BON mapping developed for main SensorML types. A BON description of SenseTile SensorML Model was developed based on the mapping.

# Chapter 7: **Conclusions and Future Work**

This thesis analyses the role of the SensorML and the SOS specifications in Sensor Systems. The UCD CASL SenseTile System is used as a case study for the use of these specifications. A SensorML model of the SenseTile system is developed to evaluate SensorML. A SenseTile Web Service prototype is used to evaluate the SOS specification which uses SensorML to provide sensor descriptions to clients.

SensorML provides a good model of sensor systems in terms of a an intuitive process concept. to describe of a sensor systems. It is hard to think of any system that could no be described by SensorML due to number of data types that can. Weather systems to Geloloaction services. SensorML address the issues of interoperabilty of Sensor Systems

For descriptions there good Metadata.

SensorML provides good support for the reliable processing of sensor data. The data types have several properties that can reference definitions to precisely describe their meaning. The accuracy of the sensor data can be modelled and provided used by systems to process the data with these variables include to get a better result.

Processing data. SensorML Frameworks. DataProducer.

It also is bases for development for Sensor Ontology work.[1] SensorML is relatively intuitive but it is difficult to know if one has developed a "good" SensorML model. There are examples and tutorials to help.

The SenseTile model uses a hierarachy to model the SenseTile. It seems repetitive to have keep defining the same inputs for each contained component. Maybe this is a flaw in the SenseTile model developed.

SensorML is defined by a heavy XML Schema. This XML Schema along with the other XML Schema for the SOS and dependent specifications are heavy indeed. Achieving a Java binding was difficult. JIBX would not generate a default XML binding. JAXB could with some help. There are many layers of abstraction in these XML Schema. This reflects the that the specifications cover a very wide range of needs of a senor system.

The idea of automatically obtaining a SensorML model and executing it depends on the SensorML Engine being used. Only one Java implementation can be pointed. In the OGC examples the Java implementations of the

Vast Lib use as a references only. Had to update library code to get it to work with the SenseTile schema even though the schema passed the validation test tool from the site. It uses DOM but XML binding is probably a better approach.

The SWE SOS specification describes the operations for accessing sensor observations and it's concepts were used to develop the SenseTile Web Service. It uses SensorML provide sensor metadata to clients and the O&M specification to encode observations. Implement all functionality would be difficult. Maybe too heavy for SenseTile System. Simple querires. Or better a push system to send the data to the client as it is generated could be a better approach here.

[1]http://www.calebgoodwin.com/SEMANTIC.html

SOS Scalable hierarcy of instances.

SWE not fully web services as W3C might define. Not SOAP. HTTP envelope flexible. There seems to be work ongoing to change this to a more industry standard view of Web Services with the use of WSDL/SOAP. I could not find this as part of current OGC SWE framework. what has been achieve

SOS on the Processor Board. Is this right? RMI only useful if SenseTile Network is a lan.

BON

Working in XML is error prone. SensorML connections type was a good example of where a graphical tool or another language, like BON, that can be parsed should be used to check the connections. Running the model through the VastSensorMLEngine without a crash was only way other than by eye to see that the connections were set up correctly.

further work. ———— SensorML sensor accuracy. Complete BON mapping and parser. Streaming the data. See SWE Architecture dpcument for a description of one approach. Other technology approaches such as . Other ways such a XQUERY? XML techs?

# 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] Open Geospatial Consortium Inc., Observations and Measurements Part 1 - Observation schema, 2007

[6] Open Geospatial Consortium Inc., OGC Sensor Web Enablement Architecture, 2008

[7] Open Geospatial Consortium Inc., Using SensorML to describe a Complete Weather Station , 2006

[8] Open Geospatial Consortium Inc.,

[9] Open Geospatial Consortium Inc., OpenGIS Catalogue Services Specification, 2007

[10] Kim Waldn and Jean-Marc Nerson , "Seamless Object-Oriented Software Architecture", 1995

[11] Bertrand Meyer., "Object-Oriented Software Construction", 1997

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

## A.1 SenseTile SensorML System

## A.2 SenseTile SensorML SensorBoardSystem

## A.3 SenseTile SensorML ConvetorSystem

## 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" xmln
  <sml:member xlink:arcrole="urn:ogc:def:process:OGC:detector">
    <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>
```

```
<!-- 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:resolut
        <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>

<!-- Manafacturer 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.xxxx/xxxxx.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

# Appendix B: **Appendix: BON to SensorML Mapping**

## B.1 SensorML BON Specifcation

```
static_diagram SENSORML_SWE
component
 cluster SENSORML
  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:          "2009/10/29.";
     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) = m
       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:          "2009/10/29.";
     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 -- attribute of member element for mapping
        ensure
            Result /= Void;
        end

    end --Member

    deferred class AbstractProcess
      indexing
      about: "base substitution group for all processes.";
      title:         "connections.";
      author:        "Ciaran Palmer.";
      copyright:     "none.";
      organisation: "School of Computer Science and Informatics, UCD.";
      date:          "2009/10/29.";
      version:       "Revision: 1.00.";

      feature

      --optional
      inputs:SEQUENCE[Input]

       --optional
      outputs:SEQUENCE[Output]

       --optional
      parameters:Parameters


      metaDataGroup:MetaDataGroup

    end --AbstractProcess

   abstract class Port
      indexing
      about: "Port for process";
      title:         "Port";
      author:        "Ciaran Palmer.";
      copyright:     "none.";
      organisation: "School of Computer Science and Informatics, UCD.";
      date:          "2009/10/29.";
      version:       "Revision: 1.00.";

      feature

      make0
        ->nameIn:STRING
        ->dataIn:ComponentData
        require
          nameIn/=Void;
          dataIn/=Void;
        ensure
        delta{name, data};
        name = nameIn;
        data = dataIn;
      end

    make1
     ->nameIn:STRING
     ->dataIn:ComponentData
      require
        nameIn/=Void;
        dataIn/=Void;
      ensure
        delta{name, data};
        name = nameIn;
        data = dataIn;
      end
   name:STRING
      ensure
          Result /= Void;
      end

   data:ComponentData

   observableProperty:ObservableProperty

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

end --Input

effective class Input
   indexing
   about: "Input Port.";
   title:         "Input.";
   author:        "Ciaran Palmer.";
   copyright:     "none.";
   organisation: "School of Computer Science and Informatics, UCD.";
   date:          "2009/10/29.";
   version:       "Revision: 1.00.";

inherit
    Port
end --Input

effective class Output
   indexing
   about: "Output Port.";
   title:         "Output.";
   author:        "Ciaran Palmer.";
   copyright:     "none.";
   organisation: "School of Computer Science and Informatics, UCD.";
   date:          "2009/10/29.";
   version:       "Revision: 1.00.";

inherit
    Port
end --Output

effective class Parameter
   indexing
   about: "Parameter Port.";
   title:         "Parameter.";
   author:        "Ciaran Palmer.";
```

```
    copyright:      "none.";
    organisation: "School of Computer Science and Informatics, UCD.";
    date:           "2009/10/29.";
    version:        "Revision: 1.00.";

inherit
     Port
end --Parameter


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:           "2009/10/29.";
    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

--System package
deferred class Component --
    indexing
    about: "Component.";
    title:          "Component.";
    author:         "Ciaran Palmer.";
    copyright:      "none.";
    organisation: "School of Computer Science and Informatics, UCD.";
    date:           "2009/10/29.";
    version:        "Revision: 1.00.";

    inherit AbstractProcess
```

```
   feature

   spatialReferenceFrame:STRING

         --spatialReferenceFrame
         --temporalReferenceFrame
         --choice
         -- location
         -- position
        --timePosition
        -- interfaces

    method:STRING
end --Component


deferred class System
   indexing
   about:         "System is a composite component containing sub-components.";
   title:         "System.";
   author:        "Ciaran Palmer.";
   copyright:     "none.";
   organisation: "School of Computer Science and Informatics, UCD.";
   date:          "2009/10/29.";
   version:       "Revision: 1.00.";

   inherit AbstractProcess
   feature

   components:SEQUENCE[AbstractProcess]

   connections:SEQUENCE[Link]

   AddLink
     ->linkIn:Link
     require
       linkIn /= Void;

       (exists i:INTEGER such_that linkIn.source.name = Current.inputs[i].name) or\
       (exists i:INTEGER and j:INTEGER such_that linkIn.source.name = components[j].outputs[i].nam

       (exists i:INTEGER such_that linkIn.destination.name = Current.outputs[i].name) or\
       (exists i:INTEGER and j:INTEGER such_that linkIn.destination.name = components[j].inputs[i].
     ensure
        delta{connections}
       (exists i:INTEGER such_that linkIn = connections[i])

end --System

deferred class ProcessModel
   indexing
   about: "ProcessModel.";
   title:         "ProcessModel.";
   author:        "Ciaran Palmer.";
   copyright:     "none.";
   organisation: "School of Computer Science and Informatics, UCD.";
   date:          "2009/10/29.";
   version:       "Revision: 1.00.";

   inherit AbstractProcess
   feature
```

```
    method:STRING
      ensure
        Result /= Void;
      end
end --ProcessModel


effective class MetaDataGroup
   indexing
   about: "Class containing all metadata information";
   title:          "MetaDataGroup.";
   author:         "Ciaran Palmer.";
   copyright:      "none.";
   organisation: "School of Computer Science and Informatics, UCD.";
   date:           "2009/10/29.";
   version:        "Revision: 1.00.";

   feature

   --make
  --    ->identificationIn:SEQUENCE[Identification]
  --    ->classificationIn:SEQUENCE[Classification]
  --    ->validTimeIn:ValidTime
  --    ->securityConstraintIn:SecurityConstraint
  --    ->legalConstraintIn:SEQUENCE[LegalConstraint]
  --    ->characteristicsIn:SEQUENCE[Characteristics]
  --    ->capabilitiesIn:SEQUENCE[Capabilities]
  --    ->contactIn:SEQUENCE[Contact]
  --    ->documentationIn:SEQUENCE[Documentation]
  --    ->historyIn:SEQUENCE[History]

   --  require
   --     identificationIn/=Void;
   --     classificationIn/=Void;
   --     validTimeIn/=Void;
   --     securityConstraintIn/=Void;
   --     legalConstraintIn/=Void;
   --     characteristicsIn/=Void;
   --     capabilitiesIn/=Void;
   --     contactIn/=Void;
   --     documentationIn/=Void;
   --     historyIn/=Void;
   --  ensure
    --    delta{identification,classification,validTime,securityConstraint,
   --               legalConstraint,characteristics,capabilities,contact,
   --               documentation,historyIn};

   --     identification=identificationIn;
   --     classification=classificationIn;
   --     validTime=validTimeIn;
   --     securityConstraint=securityConstraintIn;
   --     legalConstraint=legalConstraintIn;
   --     characteristics=characteristicsIn;
   --     capabilities=capabilitiesIn;
   --     contact=contactIn;
   --     documentation=documentationIn;
   --     history=historyIn;
   --  end

   keywords:SEQUENCE[Keywords]


   identification:SEQUENCE[Identification]
```

```
    classification:SEQUENCE[Classification]


    validTime:ValidTime


    securityConstraint:SecurityConstraint


    legalConstraint:SEQUENCE[LegalConstraint]


    characteristics:SEQUENCE[Characteristics]


    capabilities:SEQUENCE[Capabilities]


    contact:SEQUENCE[Contact]


    documentation:SEQUENCE[Documentation]


    history:SEQUENCE[History]


end --MetaDataGroup


end - SensorML Cluster

cluster SWECOMMON
component


abstract 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:           "2009/10/29.";
    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:          "2009/10/29.";
    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:          "2009/10/29.";
    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


    --quality not mapped

    --axisID not mapped
    --referenceFrame not mapped
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:          "2009/10/29.";
    version:       "Revision: 1.00.";

    feature

    make
```

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

    --AssociationAttributeGroup not mapped

end --Uom


   effective class AllowedValuesQuantity
    indexing
    about: "AllowedValues.";
    title:          "AllowedValues.";
    author:         "Ciaran Palmer.";
    copyright:      "none.";
    organisation: "School of Computer Science and Informatics, UCD.";
    date:           "2009/10/29.";
    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(

      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. Set is better.

    allowedValue:BOOLEAN
       ->valueIn:DOUBLE
       require
         valueIn/=Void;
       ensure
          Result <-> ((min==Void or valueIn>=min) or (max==Void or valueIn<=max)\
          or ((interal==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:         "2009/10/29.";
    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(
      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 (interal==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

  end
```

## B.2 BON to SensorML mapping tables

| BON features | SensorML XML tags |
|---|---|
| deferred class AbstractProcess | not mapped |
| inputs:SEQUENCE[Input] | `<sml:inputs><sml:inputlist>` `SEQUENCE[Input]` `<\sml:InputList><\sml:inputs>` |
| outputs:SEQUENCE[Output] | `<sml:outputs><sml:outputlist>` `SEQUENCE[Input]` `<\sml:outputList><\sml:outputs>` |
| parameters:SEQUENCE[Parameter] | `<sml:parameters><sml:parametertlist>` `SEQUENCE[Input]` `<\sml:outputList><\sml:outputs>` |
| metaDataGroup:MetaDataGroup | `<\sml:metaDataGroup><\sml:metaDataGroup>` |

Table B.1: BON to SensorML AbstractProcess Mapping table

| BON features | SensorML XML tags |
|---|---|
| deferred class System | not mapped |
| inputs:SEQUENCE[Input] | `<sml:inputs><sml:inputlist>` `SEQUENCE[Input]` `<\sml:InputList><\sml:inputs>` |
| outputs:SEQUENCE[Output] | `<sml:outputs><sml:outputlist>` `SEQUENCE[Input]` `<\sml:outputList><\sml:outputs>` |
| parameters:SEQUENCE[Parameter] | `<sml:parameters><sml:parametertlist>` `SEQUENCE[Input]` `<\sml:outputList><\sml:outputs>` |
| metaDataGroup:MetaDataGroup | `<\sml:metaDataGroup><\sml:metaDataGroup>` |

Table B.2: BON to SensorML AbstractProcess Mapping table

| BON features | SensorML XML tags |
|---|---|
| deferred class C | not mapped |
| inputs:SEQUENCE[Input] | <sml:inputs><sml:inputlist> SEQUENCE[Input] <\sml:InputList><\sml:inputs> |
| outputs:SEQUENCE[Output] | <sml:outputs><sml:outputlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| parameters:SEQUENCE[Parameter] | <sml:parameters><sml:parametertlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| metaDataGroup:MetaDataGroup | <\sml:metaDataGroup><\sml:metaDataGroup> |

Table B.3: BON to SensorML AbstractProcess Mapping table

| BON features | SensorML XML tags |
|---|---|
| deferred class Link | not mapped |
| inputs:SEQUENCE[Input] | <sml:inputs><sml:inputlist> SEQUENCE[Input] <\sml:InputList><\sml:inputs> |
| outputs:SEQUENCE[Output] | <sml:outputs><sml:outputlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| parameters:SEQUENCE[Parameter] | <sml:parameters><sml:parametertlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| metaDataGroup:MetaDataGroup | <\sml:metaDataGroup><\sml:metaDataGroup> |

Table B.4: BON to SensorML AbstractProcess Mapping table

| BON features | SensorML XML tags |
|---|---|
| deferred class ProcessChain | not mapped |
| inputs:SEQUENCE[Input] | <sml:inputs><sml:inputlist> SEQUENCE[Input] <\sml:InputList><\sml:inputs> |
| outputs:SEQUENCE[Output] | <sml:outputs><sml:outputlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| parameters:SEQUENCE[Parameter] | <sml:parameters><sml:parametertlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| metaDataGroup:MetaDataGroup | <\sml:metaDataGroup><\sml:metaDataGroup> |

Table B.5: BON to SensorML AbstractProcess Mapping table

| BON features | SensorML XML tags |
|---|---|
| deferred class ProcessModel | not mapped |
| inputs:SEQUENCE[Input] | <sml:inputs><sml:inputlist> SEQUENCE[Input] <\sml:InputList><\sml:inputs> |
| outputs:SEQUENCE[Output] | <sml:outputs><sml:outputlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| parameters:SEQUENCE[Parameter] | <sml:parameters><sml:parametertlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| metaDataGroup:MetaDataGroup | <\sml:metaDataGroup><\sml:metaDataGroup> |

Table B.6: BON to SensorML AbstractProcess Mapping table

| BON features | SensorML XML tags |
|---|---|
| deferred class Port | not mapped |
| inputs:SEQUENCE[Input] | <sml:inputs><sml:inputlist> SEQUENCE[Input] <\sml:InputList><\sml:inputs> |
| outputs:SEQUENCE[Output] | <sml:outputs><sml:outputlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| parameters:SEQUENCE[Parameter] | <sml:parameters><sml:parametertlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| metaDataGroup:MetaDataGroup | <\sml:metaDataGroup><\sml:metaDataGroup> |

Table B.7: BON to SensorML AbstractProcess Mapping table

| BON features | SensorML XML tags |
|---|---|
| deferred class Quantity | not mapped |
| inputs:SEQUENCE[Input] | <sml:inputs><sml:inputlist> SEQUENCE[Input] <\sml:InputList><\sml:inputs> |
| outputs:SEQUENCE[Output] | <sml:outputs><sml:outputlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| parameters:SEQUENCE[Parameter] | <sml:parameters><sml:parametertlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| metaDataGroup:MetaDataGroup | <\sml:metaDataGroup><\sml:metaDataGroup> |

Table B.8: BON to SensorML AbstractProcess Mapping table

| BON features | SensorML XML tags |
|---|---|
| deferred class Count | not mapped |
| inputs:SEQUENCE[Input] | <sml:inputs><sml:inputlist> SEQUENCE[Input] <\sml:InputList><\sml:inputs> |
| outputs:SEQUENCE[Output] | <sml:outputs><sml:outputlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| parameters:SEQUENCE[Parameter] | <sml:parameters><sml:parametertlist> SEQUENCE[Input] <\sml:outputList><\sml:outputs> |
| metaDataGroup:MetaDataGroup | <\sml:metaDataGroup><\sml:metaDataGroup> |

Table B.9: BON to SensorML AbstractProcess Mapping table