Project Report

SensorML on SenseTile

Ciaran Palmer

A thesis submitted in part fulfilment of the degree of

Msc Advance Software Engineering

Supervisor: Your Supervisor Name

Moderator: Your Moderator Name

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

March 11, 2010

Table of Contents

Abstract

SenseTileSensor Board

SensorML description.

WebServer to access sensor observations

SensorML Bon Mapping with a tool and method to develop sensorML descriptions

Acknowledgments

Chapter 1: **Introduction**

The aim of this thesis project was to model the SenseTile Sensor Board using Sensor Model Language (SensorML)[1] . A prototype of a Web Service for accessing SenseTile Sensor Board sensor observations was to be developed using this SensorML description of the SenseTile Sensor Board. An evaluation of the suitability of SensorML for use as part of the SenseTile System was also to be performed.

SenseTile is a sensor and processing package including motion sensors, RFID sensors, temperature sensor, audio sensors, pressure sensors, light level sensors video sensors among others. It is used as a replacement for standard ceiling tiles to provide smart building services as part of a Web Sensor Network. The SenseTile System is made up of two units a Sensor Board and Processor Unit. The sensors are interfaced or hosted on the Sensor Board. The Processor Unit interfaces to the Sensor Board allowing data from the Sensor Board to be processed futher as required.

SensorML is one of a number of specifications provided by the Open Geospatial Consortium (OGC) as part of the Sensor Web Enablement(SWE) initiative[ref]. The members of the OGC have the goal of developing open standards to exploit Web connected sensors of all types. SensorML is used to describe sensor systems and the processing of sensor observations. It provides standard models and XML encodings to describe the process of sensor measurement. SensorML is described further in chapter x.

The SenseTile Web Service prototype allows access to the Sensor Board sensor observations and it's SensorML description. It runs on the SenseTile Processor Unit. The Web Service is based on another OGC SWE specification, the Sensor Observation Service (SOS)[ref]. This specification defines a standard Web interface for retrieving information about sensor systems and sensor observations. Sensor Board measurements that have been post processed and converted to real values or the raw Sensor Board data can be accessed using the Web Service. SOS Lite.

There are othe SOS specification depend on another SWE specification, Observation and Measurements(O&M)[ref]. This specification provides standard models and XML encoding for sensor observations.

A SensorML to BON Mapping was also to be explored . BON[6] is a notation and a method for Object Oriented systems analysis and design. Based on this mapping it was proposed to develop a tool/method to automatically generate SensorML descriptions. Generally SensorML is hand generated with all the shortcomings of such an approach. The proposed approach would allow sensor processing to defined formally in BON and then refined to JML/Java.

Chapter 2: **Background Research**

2.1 Sensor Web Enablement

The Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) [ref] is establishing interfaces and protocols that will enable a standardised Sensor Web. It is hoped that an easier integration and development of sensor applications across sensor networks will be facilitated by this standarising activity.The OGC 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 systems and processes surrounding measurements
- Observations & Measurements (O&M) - models and schema for packaging observation values
- Transducer Markup Language (TML) - models and schema for multiplexed data from sensor systems
- Sensor Observation Service (SOS) - standard web interface for accessing observations
- Sensor Planning Service (SPS) - standard web interface for tasking sensor systems and model and requesting acquisitions
- Sensor Alert Service (SAS) - standard web interface for publishing and subscribing to sensor alerts
- Web Notification Service (WNS) - standard web interface for asynchronous notification

There is also the SWE Common that provides common data models and schemas for the above.

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 SWE framework seems to cover all needs for modeling sensors and observations. Support the use of Web Services is the main rational and fits the needs of a SenseTile Sensor Network. In this case the SOS and SensorML are core.

Scalable - can be large systems. Discovery. Process data without having to define an implementation as this can be provided as part of SensorML description.

DAML?

2.2 Sensor Modeling Language

Sensor Modelling Language (SensorML) is used to describe sensor systems and the processing of observations from sensor systems.

The purposes of SensorML are:

- Provide sensor and process information in support of resource and observation discovery
- Support the processing and analysis of the sensor observations
- Support the geolocation of observed values (measured data)
- Provide performance characteristics (e.g., accuracy, threshold, etc.)
- Provide an explicit description of the process by which an observation was obtained (i.e., its lineage)
- Provide an executable process chain for deriving new data products on demand (i.e., derivable observation)
- Archive fundamental properties and assumptions regarding sensor systems

SensorML provides a functional model of sensors and an XML encoding to describe sensors and their observations. Sensors are modelled as processes that convert real phenomena to data. Processes take inputs, process them and result in one or several outputs.

Sensor metadata is also supported by SensorML.

The processes can be connected together in chains using SensorML ProcessChains or Systems.

including measurement by a sensor system, as well as post-measurement processing.

When Modeling a sensor system in SensorML the main entities used are Systems and Components.

A SensorML System is used to group sensors that are related spatially to one another. As it is a process chain it is hierarchical and so Systems can contain Systems.

A Component is an atomic process that generally converts a physical phenomena to a digital number.

ProcessModels are another modeling that is used to describe a pure process that has no physicality. They point to an implementation or a description of one.

The data types used for inputs and out puts come another OGC sepc the SWE common. Examles are Quantity which is a decimal number, Count, Boolean, Category, and Time provide the basic primitives. Within SensorML,

Aggregation of primitive data types are also provided DataRecord or DataArray.

A sensor systems would likely to be based mainly on Systems and Detector model. Reference tutorial and sensorml specification.

The SensorML description of the SenseTile Sensor Board is described in chapter .

2.3 Sensor Observation Service

The SenseTile Web Service is based on the concepts described in the Sensor Observation Service (SOS) specification. The SOS defines a web service interface for the discovery of sensors systems and the retrieval of observations from sensor systems. Observations are encoded as OGC O&M Observations. Sensor systems metadata can be also be retrieved using the SOS interface and is generally defined in SensorML or TML. para. An SOS organizes collections of related sensor system observations into Observation Offerings. These operations are encoded in XML and used in http post and get operations. The XML message needs to be parsed to identify the operations, parameters and results. 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. para: Scalability would be achieved through aggregating a number of sensor data producers on the sensor side and by aggregating a number of SOS instances on the data center side for providing data to consumers.

2.3.1 SOS Data Consumer Operations

There are there core operations that are provided by an SOS and used by SOS Clients to request sensor information:

- GetObservation operation is used to obtain sensor observations and measurement data.
- GetCapabilities operation is used to retrieve SOS service metadata.
- DescribeSensor operation retrieves detailed information about the sensors .

2.3.2 SOS Data Provider Operations

Two other operations RegisterSensor and InsertObservation are of interest to this thesis . These are part of the SOS transactional interface. They are used with SOS data producers which are software entities connected to the sensor and have enough processing power to generate XML descriptions of the sensor data.

2.3.3 SOS Catalogs

The catalog aspect of SOS that used for discovery of the SOS instance is not covered in the Web Service Prototype described in this thesis.

2.3.4 Enhanced Operations

Also six enhanced operations, GetResult, GetFeatureOfInterest, GetFeatureOfInterestTime, DescribeFeatureOfInterest, DescribeObservationType, and DescribeResultModel are not included in the SenseTileWeb Service.

2.4 OGC Lib

The VAST Team at the University of Alabama in Huntsville (UAH) developed an open source JAVA framework call swe-common-data-framework (http://code.google.com/p/swe-common-data-framework/) to allow the quick development of SWE services such as an SOS. In this project the library is used for it's SensorML parsing and SensorML process execution functionality.

2.5 UCD Sensor Board library

UCD Research group providing a Java lib to access the data

2.6 BON

2.7 JAXB

2.8 AXIS 2.0

Chapter 3: **Design**

3.1 SensorML Model of SenseTile

A SensorML model of SenseTile was developed as part of this thesis. The model contains metadata about the SenseTile system and supports the needs of the SenseTile Web Service. Two main requirements of the SenseTile Web Service were to be supported by the SensorML model. Firstly the raw data from the senors was to be retievable to allow external system process the data as desired. Secondly there was a requirement to do post processing on the Processor Unit of the sensor data to convert it to meaningful values. A block diagram of the Sensetile SensorML model is shown in figure 3.1.

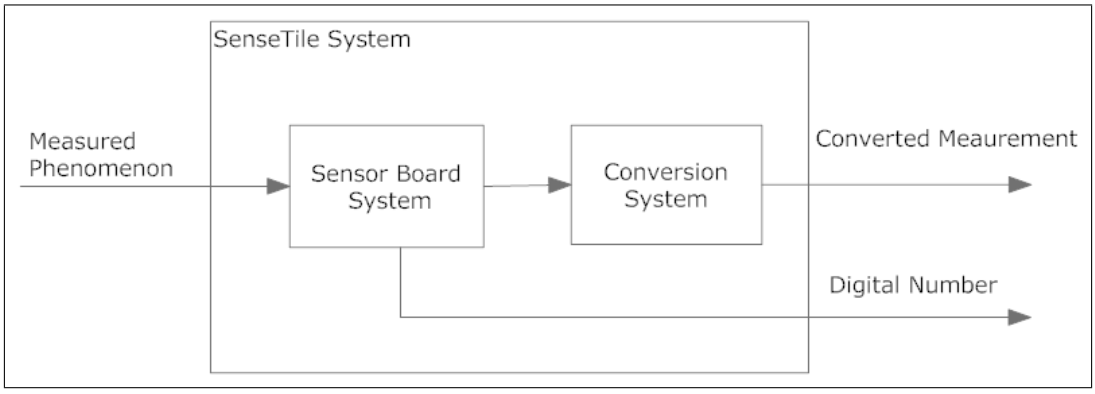

Figure 3.1: SenseTile System Block Diagram

Three SensorML Systems are used to model the SenseTile as shown in the block diagram. One SensorML System models the overall SenseTile which contains the Sensor Board and Conversion Systems. The below XML fragment shows how SensorML describes System containment using a component list and how the other SensorML Systems descriptions are referenced using xlink hrefs.

```
<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 physical phenomena measured by the sensors are the input to the SenseTile System. These values will be read in from packets received from the Sensor Board. The Digital Number output is the raw sensor data which is a digital number without a physical meaning. The Converted Measurement output contains the values from post processing the sensor data which have some physical meaning such as temperature in Celcius.

The division of the systems is based on the concept that sensor observations are based on

at least a sampling process and a conversion processref tutorial. The Sensor Board System contains SensorML Components that model the sensors on the board that perform the sampling. The SensorML Components contained in the Conversion System model processes that produce meaningful physical measurement values such as temperature and lumens from the sampled data.

The SenseTile System connects up the inputs and outputs of the SensorBoard and Convertor 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 is peformed in SensorML.

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

A more detailed block diagram of the Sensetile SensorML model with Components is shown in figure 3.2. Only the Thermistor Component and the Celcius Component are developed as part of this thesis. They are described in model detail in later sections.

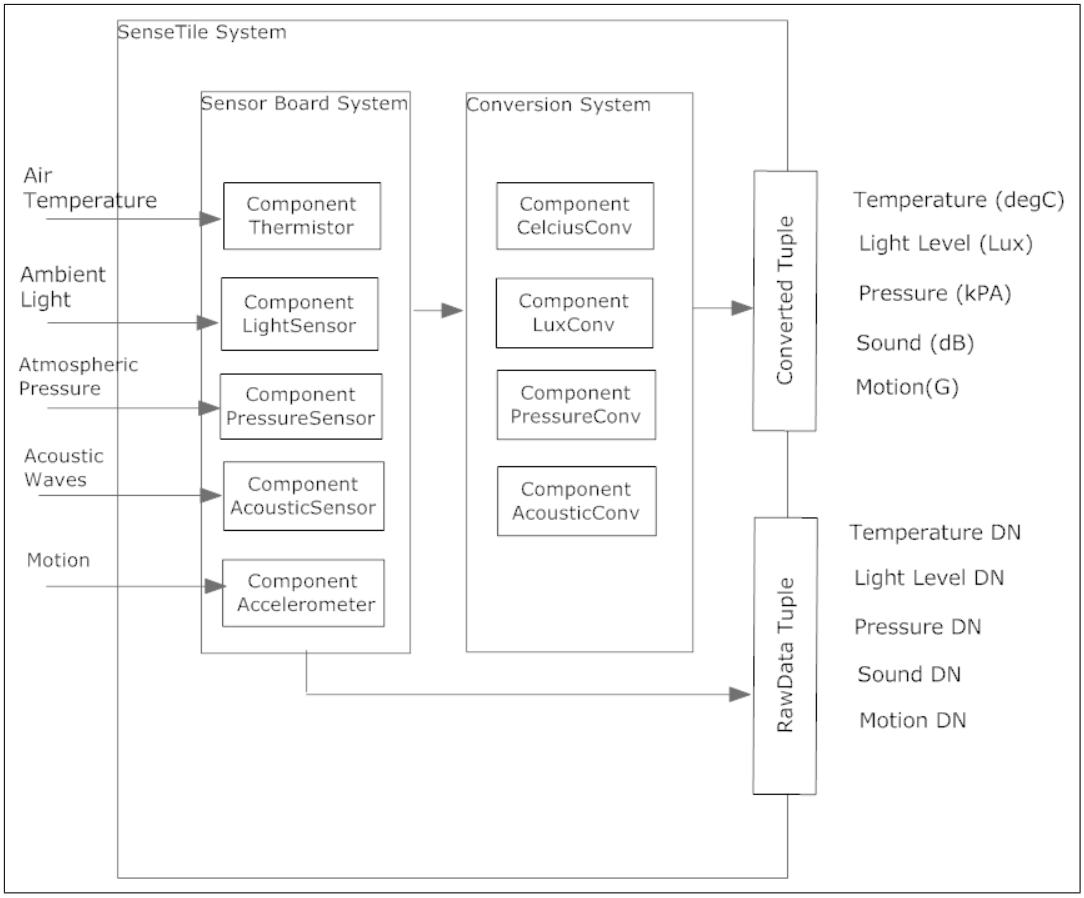

Figure 3.2: SenseTile System Components

Naming rules for inputs and outputs - allow more generic handling of implementation classes.

System and components are in several files. Inline in one file if desired.Choice to seperate files.

3.1.1 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 Thermsistor 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>
```

3.1.2 Celcius Converter Component

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

A Sensor process method was also developed. The process method describes the coversion 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.

3.2 BON to SensorML mapping

Features to be elements or attributes.

Name rule suffix "Attrib" to make it an attribute.

List handling - convention for mapping.

SML prefix to handled key word clashes between sensorML and BON.

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 seperate 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 seperate processes on the SenseTile processor unit to allow a more flexible network architecture. This network architecture is described in more detail in section 3.3.3. The basic system structure is shown in figure 3.3 .

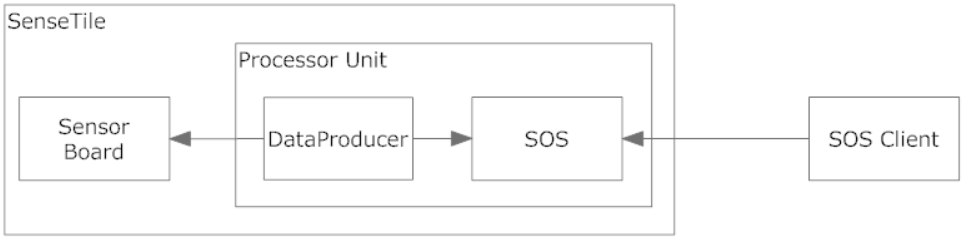

Figure 3.3: SenseTile Web Service

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 Supported Scenarios

Scenarios
Generate Observations from Sensor Data The DataProducer accesses data from the sensor board
Get Sensor Description from SOS
Get Sensor Observation from SOS

Table 3.1: Scenario Chart for SenseTile Web Service

3.3.3 SenseTile WebService Network Architecture

In the SenseTile Web Service the SOS acts as a gateway to the SenseTile Sensor Network. This prototype was designed with the network architecture as shown in fig 3.4. A client would access multiple SOS instances to get observations from the sensor network. An SOS could support several DataProducers and hence would not to be run on every SenseTile.

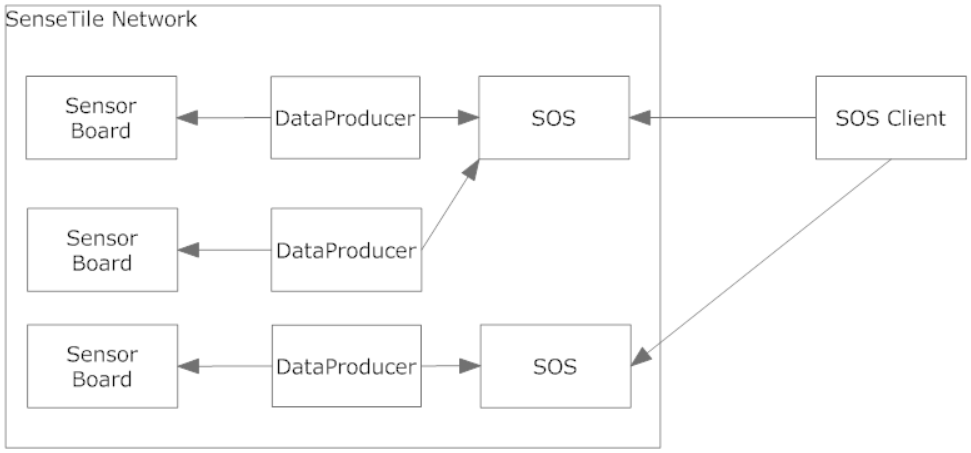

Figure 3.4: SenseTile Sensor Network Architecture

The SOS Specification envisioned that the SOS would be run on a large external server as shown in fig 3.5. The DataProviders would update this external server with observations and clients would just access this large SOS. The SOS Specification describes the use of a

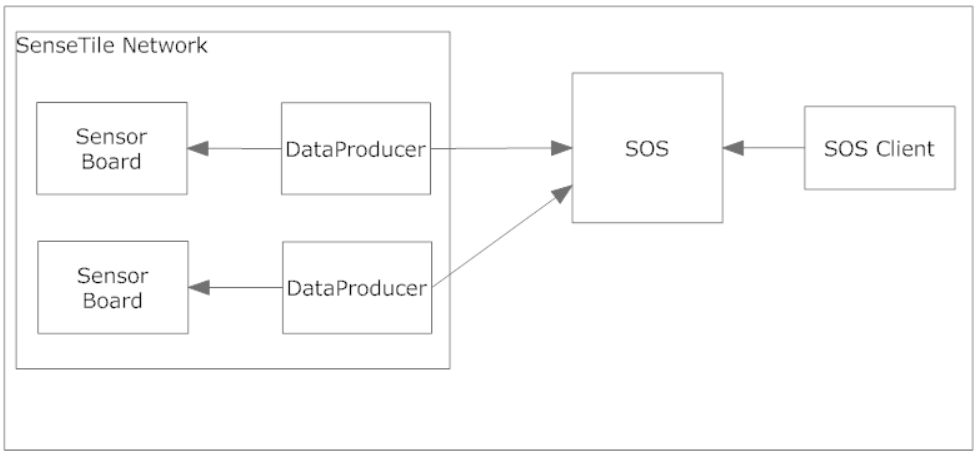

Figure 3.5: External SOS

Proxy SOS as shown in figure 3.6. This architecture could be used with the SenseTile Sensor Network if accessing many SOSs is not feasible for the clients.

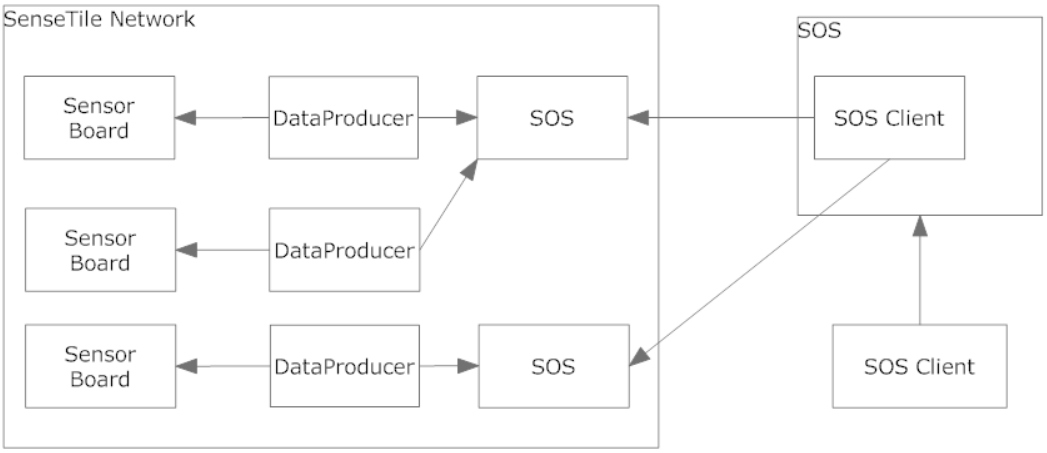

Figure 3.6: Proxy SOS

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 3.7 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 implemenation 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 continously.

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 3.7.

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 meaurement generated by the thermistor. This library is shown as the SensorBoard cluster in 3.7 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 SensoML Enigine will execute the process chains described in the SenseTile SensorML model. The sensor objects output will be passed to the coverter 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?

Questions. If have light data but no temp. As temp is only every 1000 packets?

How to know when to generate observation for the two offerings? DataProducer generates observations in Observation xml standard.

Interface to the SensorBoard Library.

Uses observer pattern to update sensors with packets from the board.

reads sensors from SensorBoard System xml

Sensors iimplement interface to facilitate this.

Vast lib connects all the processes

implementation of sensor and convertor is named using URN read from the sensorMl descption.

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

Interfaces to the Sensor Board

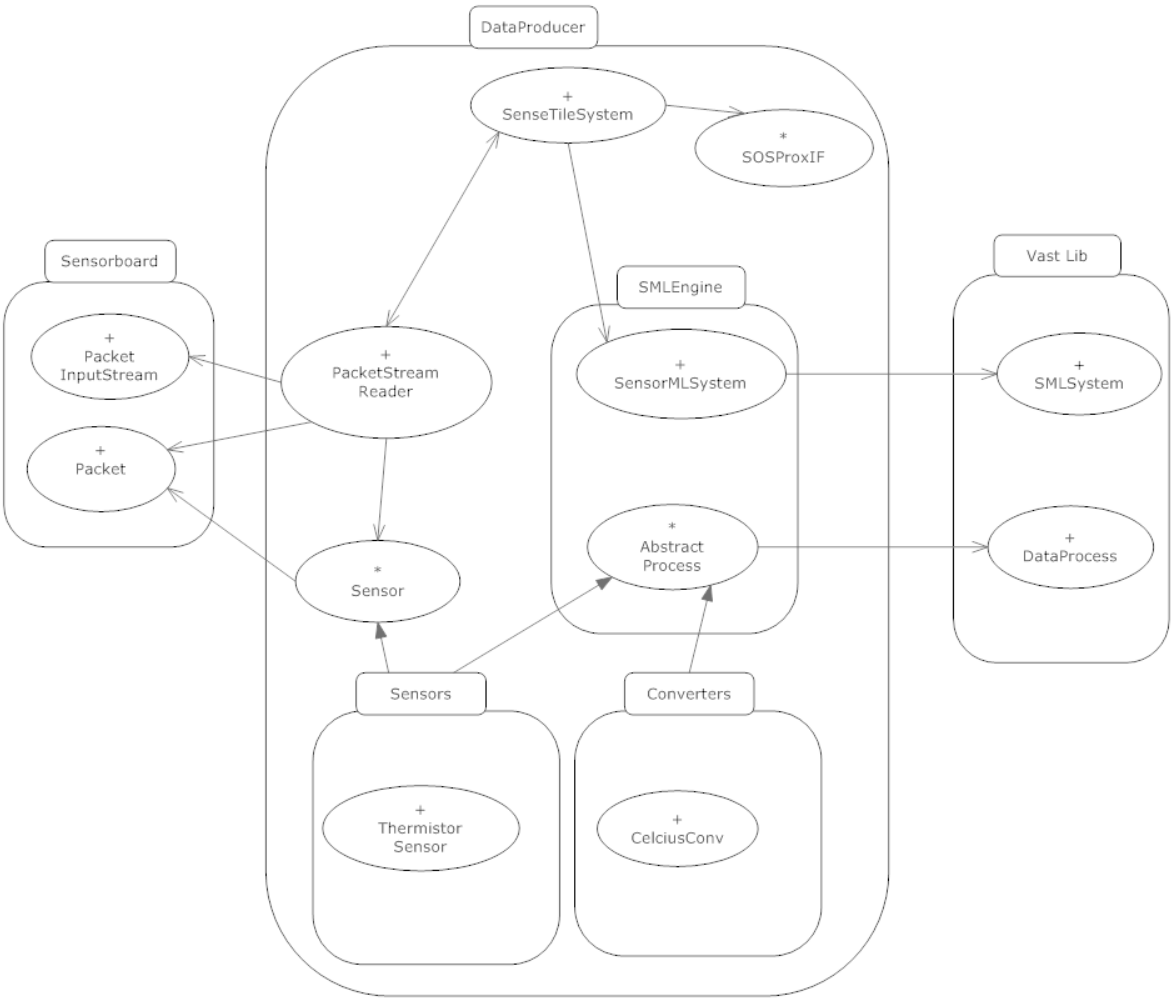

Figure 3.7: DataProducer Static Diagram

3.3.5 SOS

Keeps track of sensorboards that register

provides an RMI interface to the DataProducer to update with observations. This has lower overhead than a web service. Though it would not be hard to change to use a Web Service Interface.

GetObservaton - marshal Observation to xml string and send as response.

StreamObservation - registered clients get new observation sent over UDP connection?

Provides a Web Service interface based on Sensor Observation Service. Not fully as is a very complex interface.

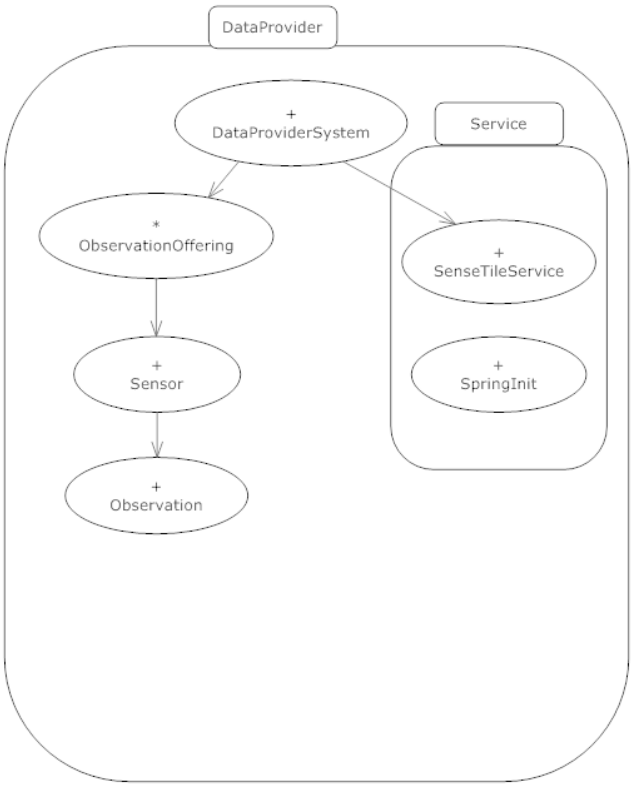

Figure 3.8: SOS Static Diagram

3.4 SensorML generation

Java binding wth JIBX generate sensorml.

Simple xml file with Sensor metadata.

Naming rules allow connection of inputs and outputs in sensorML processes.

component id +Input

component id+Output

Chapter 4: **Detailed Design and Implementation**

4.1 DataProducer

DataProducer VastLib UCD SenseTile Driver Lib

4.2 SOS

is based on AXIS 2.0/Spring

JAXB for SensorML XML to java binding.

RMI used between DataProvider and SOS.

4.3 SensorML Generation

Chapter 5: **Results**

---

running prototype tested with UCD SensorBoard Simulator code

Tested against real SenseTile and sensor data retrieved by a testclient.

Generation of SensorML description of a Senstile Sensor

Chapter 6: **Conclusions and Future Work**

---

what has been achieved

the weaknesses of your approach SOS on the Processor Board. Is this right.

difficult schemas to program against. JIBX did not generate a default binding. Jaxb did with some help but did not generate the process type correctly. Why?

Schemas cover all needs for sensor description and data.

A lot of schemas needed.

SWE not fully web services as W3C might define. Not SOAP. HTTP envelope flexible.

Vast Lib use as a references only. Other ways such a XQUERY? XML techs? As there are a large number of types etc. A quick way for a prototype. Had to update library code to get it to work with the SenseTile Schemas. Schemas passed the validation test tool from the site.

Portabiltiy, Vast Types POJOs.

Streaming the data.

Chapter 7: **References**

Bibliography

[1] Open Geospatial Consortium Inc., OpenGIS Sensor Model Language (SensorML) Implementation Specification, 2007

[2] Open Geospatial Consortium Inc., OpenGIS Implementation Specification, 2007

[3] Open Geospatial Consortium Inc., OpenGIS Implementation Specification, 2007

[4] Open Geospatial Consortium Inc., , 2007

[5] Open Geospatial Consortium Inc., , 2007

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

Chapter 8: **Appendices**

```
<?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>
```