White Paper

Creation of Specific SensorML Process Models

January 27th, 2006

(Draft)

Alexandre Robin
(robin@nsstc.uah.edu)

Michael E. Botts
(mike.botts@uah.edu)

Earth System Science Center - NSSTC
University of Alabama in Huntsville (UAH)
HUNTSVILLE, AL 35899

Table of Contents

Important Definitions

- **Process Model** – Atomic processing block usually used within a more complex *Process Chain*. It is associated to a *Process Method* which defines the process interface as well as how to execute the model. It also precisely defines its own inputs, outputs and parameters.

- **Process Chain** – Composite processing block consisting of interconnected sub-processes, which can in turn be *Process Models* or *Process Chains*. A process chain also includes possible data sources as well as connections that explicitly link input and output signals of sub-processes together. It also precisely defines its own inputs, outputs and parameters.

- **Process Method** – Definition of the behavior and interface of a *Process Model*. It can be stored in a library so that it can be reused by different *Process Model* instances (by using 'xlink' mechanism). It essentially describes the process interface and algorithm, and can point the user to existing implementations.

- **Detector** – Atomic part of a composite *Measurement System* defining sampling and response characteristic of a simple detection device. A detector has only one input and one output, both being scalar quantities. More complex *Sensors* such as a frame camera which are composed of multiple detectors can be described as a detector group or array using a *System* or *Sensor*. In SensorML a detector is a particular type of Process Model.

- **System** – Composite model of a group or array of components, which can include detectors, actuators, or sub-systems. A *System* relates a *Process Chain* to the real world and therefore provides additional definitions regarding relative positions of its components and communication interfaces.

- **Measurement System** – Specific type of *System* involving primarily sampling devices and *Detectors*.

- **Sensor** – Specific type of *System* representing a complete *Sensor*. This could be for example a complete airborne scanner which includes several *Detectors* (one for each band).

I. Scope

SensorML provides a powerful and modular way of describing data processing chains by allowing the definition of reusable processing components called *Process Models*. In addition to metadata useful for discovery, *Process Models* define model inputs, outputs and parameters, as well as a method for executing the model. These methods need to be defined inline or in a library before the model can be used in a *Process Chain*.

After a short overview of SensorML possibilities, this document describes the different aspects of the creation of a new *Process Model*:

- The definition of the *Process Method* which consists in an I/O interface, an algorithm and eventual implementations. (Part III)
- The instantiation of a *Process Model* in a *Process Chain* using the previously defined *Process Method*. (Part IV)
- The creation of a *Process Method* library to increase reusability. (Part V)
- The implementation of a *Process Method* using the open source Java based Data Processing Framework designed by UAH. (Part VI)
- The validation of SensorML documents and particularly *Process Chains* including custom *Process Models*. (Part VII)
- The online submission of newly created *Process Method* for review by SensorML experts and approval. (Part VIII)

For details about the SensorML language and XML encodings, please refer to the normative SensorML document ref. OGC-05-086r2.

II. Overview

A. *Introduction to SensorML Processing*

The SensorML language was first designed as an XML standard for the description of measurement devices (detectors, sensors, etc…) and more complex measurement systems (platforms, sensor groups, etc…), in order to enable automatic processing of sensor data by generic software.

Given the variety of available systems, the language has quickly evolved to a more generic way of describing *Processing Chains* and how they transform data. In this context, devices such as *Detectors*, *Actuators*, and *Filters* are simply modeled by specific processes that can be part of a more complex *Process Chain*. This allows the description of both real world *Systems* and Digital *Processing Chains* using the same syntax, thus making system descriptions and processing software even more interoperable.

Although the SensorML language can be used as a standalone technology, it is also highly coupled with other OGC Sensor Web Enablement (SWE) components such as Sensor Observation Service (SOS) and Sensor Planning Service (SPS). The same syntax for the description of data structures is used within these services and SensorML, thus improving the connectivity between SensorML *Process Chains* and data coming from or going to these services. The following diagram summarizes the different uses of SensorML within OGC Sensor Web Enablement Framework.

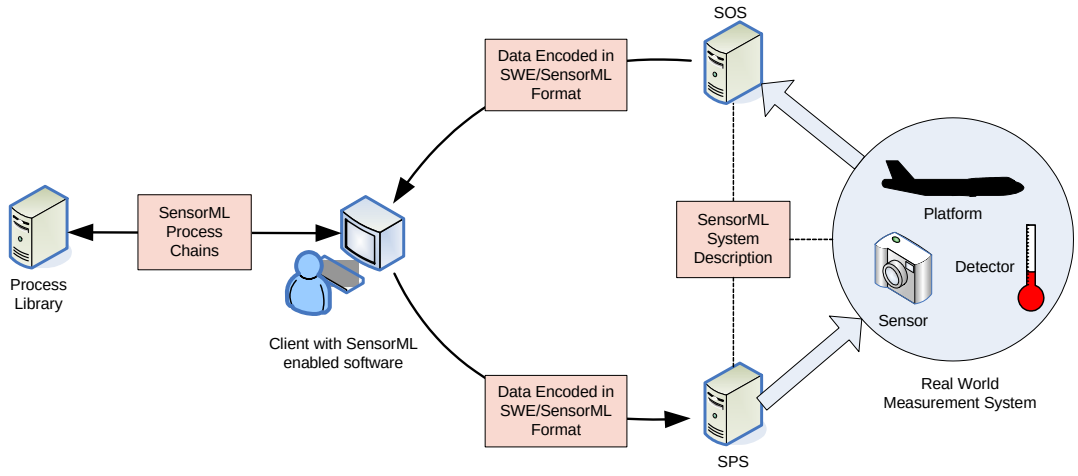


Figure 1: Use of SensorML within OGC Sensor Web Enablement framework

The VAST team at the University of Alabama in Huntsville is currently creating Open Source Java software for dealing with SensorML and other SWE documents. An initial release of the software should be available during the year 2006 but a SourceForge project is already opened for contributions.

This Java based software currently has the following components:

- A SensorML validator which uses both the official XML Schema and process specific Schematron schemas to fully validate SensorML instance documents.

- A SensorML parser that generates Java object after reading a SensorML document. This parser automatically handles xlink mechanism, and will be updated to resolve URNs when necessary.

- A SensorML/SWE Data parser useful for parsing SensorML and SWE services data structures. This handles the data description section as well as the data stream itself. (API and Implementation)

- A SensorML *Process Chain* Executor providing a framework for executing *Process Models* based on their methods and including the logic for running complete *Process Chains* using either a centralized (synchronous) or distributed (asynchronous) approach. (API and Implementation)

More information about implementing new models within this processing framework is given in section *VI*.

B. Sensors and Systems Description

SensorML can describe real world *Measurement Systems* by providing component structure, position information (location and orientation) and technical characteristics. Since only data outputted by these systems is available, a SensorML *System* is equivalent to a *Process Chain* describing how the measured quantities were transformed in order to obtain new values, and thus represents the lineage of the data. Relative positions (or absolute if known) of components then complete the definition of a *System*.

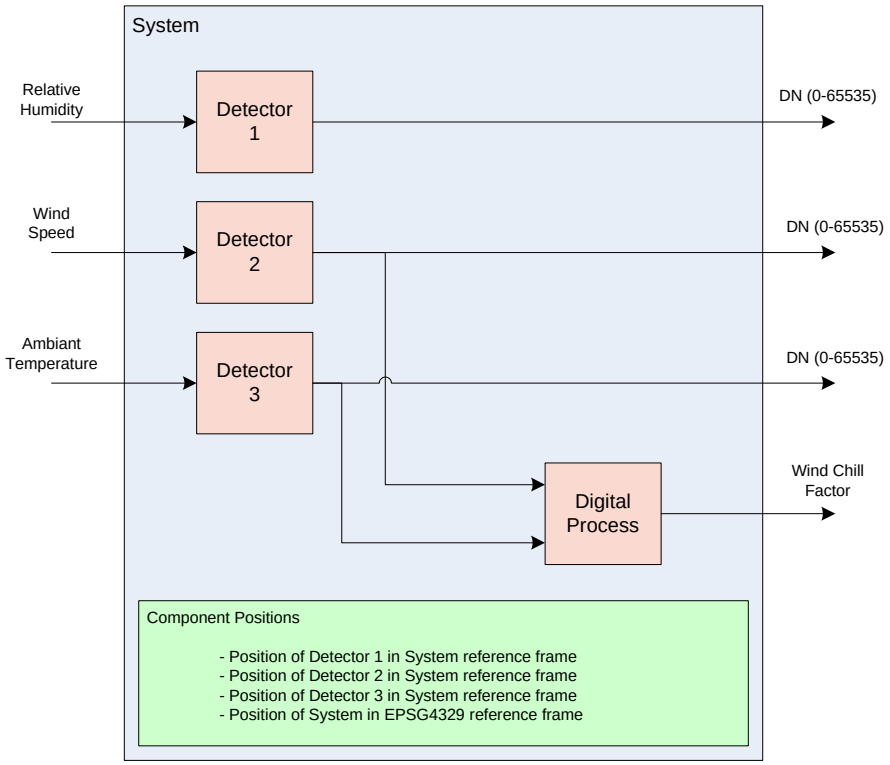
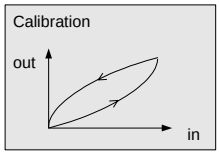


Figure 2: Block diagram of a Weather Measurement System

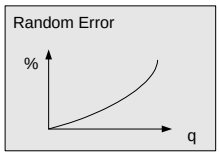
Figure 2 shows an example of block diagram for a weather station measuring relative humidity (1st input), wind speed (2nd input) and ambient temperature (3rd input) using 3 different *Detectors*, and calculating the wind chill factor with an embedded digital processor.

In this example, outputs for measured quantities are all 16 bits digital numbers (DN) without physical meaning. The SensorML *System* description then aims to provide all the necessary information to compute real quantity values from these meaningless digital numbers while still being able to archive the raw data.

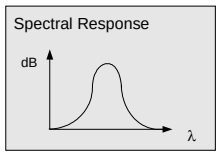
Each *Detector* in the *System* is described by a set of generic measurement parameters applicable to any detector type. These parameters include:



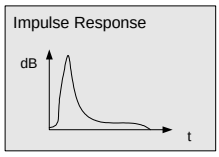
Calibration Curve
Gives the mapping of input to output values for a steady state regime. Two curves are used to describe a Hysteresis behavior.



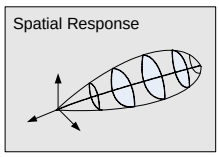
Random Error Curve
Gives the relative measurement error versus the input value itself or any other environmental quantity such as temperature.



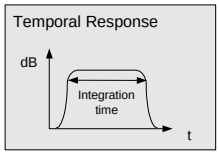
Spectral Response Curve
Specifies dynamic characteristics of the detector in the frequency domain. It gives the sensitivity of the detector versus the frequency or wavelength of the input signal.



Impulse Response Curve
Specifies dynamic characteristics of the detector in the time domain. It represents the normalized output of the detector for an impulse (Δ function) input.



Spatial Response Curve(s)
Gives the sensitivity of the detector relative to spatial coordinates (location of the source, or orientation of the incoming signal, e.g. point spread function, polarization)



Temporal Response Curve
Gives the sensitivity of the detector relative to a temporal coordinate frame (e.g. sampling time). This is a more descriptive form of the integration time.

The position information (in green on the diagram) allows derivation of the exact spatial and temporal position of the measured phenomenon. Each system component as well as the system itself is attached to a mathematical reference frame making the relative position unambiguous. This section is especially important when dealing with remote sensors for which deriving the precise location of the measured phenomenon is fundamental.

By providing a standard way of describing *Sensors* and *Systems*, SensorML allows data fusion or interpretation software to derive parameters of the initial phenomenon such as real value, position, time, intensity and frequency using only the raw data and the *System* characteristics. The *System* description also provides information for simulating detectors and complete systems behavior in response to a known input or sequence of inputs such as a specific signature or phenomenon.

C. Data Processing Chains Description

SensorML also enables the description of explicit data *Processing Chains* which can be used to process any type of archived or real-time data. The SensorML processing language was designed to maximize component reusability and support data streaming from the start.

When associated to a given *Measurement System*, these processing chains can be used by data fusion or data processing software to ingest data coming from this *System* without having to understand the *System* description completely. It is a helper provided by the *System* integrator or other party which facilitates data processing by generic SensorML enabled software (Directly interpreting *Process Chains* that are not ambiguous is indeed much simpler than writing software than can read and understand any type of *Measurement Systems*). In a sense, this allows the use of SensorML to provide "derivable observations" by describing the process by which an application can generate higher level data from the raw data without a-priori knowledge of the data and its source.

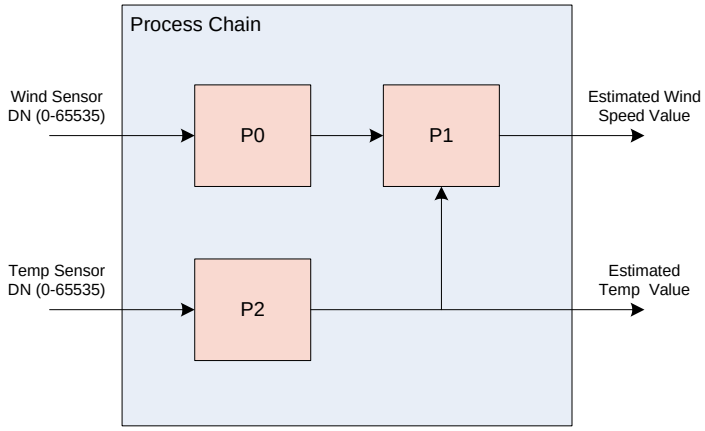


Figure 3: Block diagram of a simple “Reverse Calibration” process

Figure 3 is the diagram of a *Process Chain* used to compute the real value of measured quantities from the digital number outputted by the *Detectors*. The three sub-processes involved in the chain are set up with parameters directly associated to the ones given in the *System* description. ‘P0’ and ‘P2’ would for example use the calibration curve of the wind speed and temperature *Detectors*, respectively, to compute the estimated measured value, while ‘P1’ is used to correct the wind speed value using the error curve of the corresponding *Detector* (here, error varies with temperature). Outputs of this *Process Chain* provide meaningful values which are closer estimations of the real wind speed and ambient temperature.

In a *Process Chain* like the one on figure 3, all sub-processes are associated with a method defined either inline or in a dictionary. This *Process Method* contains all the information needed to validate and implement this specific processing block. Additionally, each block in a chain can also contain a sub-chain, thus making the SensorML language highly modular.

Thanks to its generic syntax, SensorML can even be used to describe a *Process Chain* that is not directly related to a given *System*. This type of chains can be used to process data directly without dealing with any kind of *System* or *Sensor* description. They can for instance be used to derive high level products from basic measurements (e.g. sea surface temperature from radiance measurements).

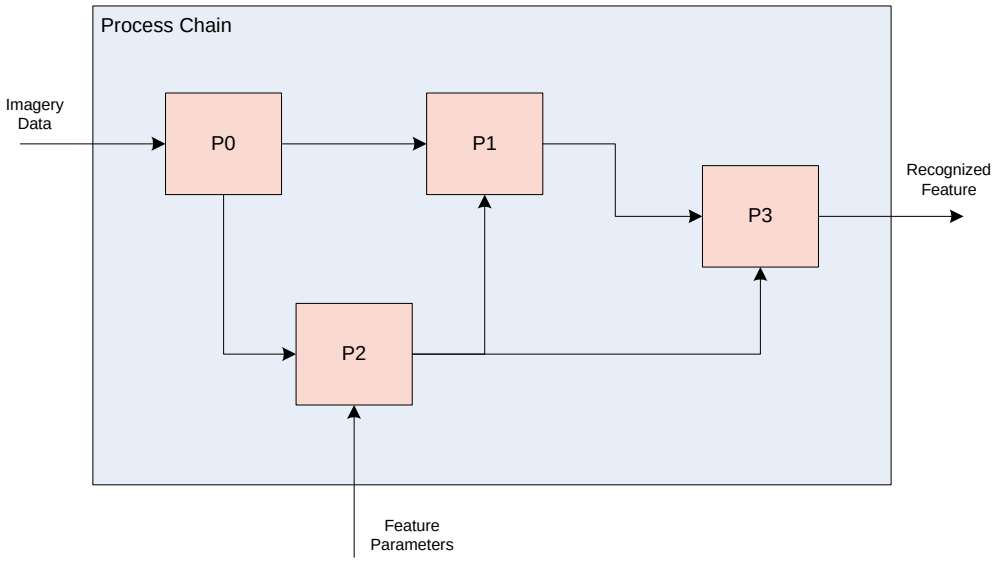


Figure 4: Block diagram of a “Feature Extraction” process

The example diagram on figure 4 shows a *Process Chain* used to extract specific features from imagery data. This type of processing chain is obviously domain specific and requires the knowledge of complex algorithms specially designed for a given application, but can typically be constructed using common reusable components that can be defined as custom *Process Models*.

That is why, additionally to generic *Process Models* that will be defined in shared dictionaries, SensorML provides mechanisms to define highly specialized *Process Methods* and store them in domain specific libraries. This capacity of extension is one of the most important strengths of the SensorML language, and will be very helpful for numerous scientific communities.

III. Defining a new Process Method

A. Design steps

The definition of a *Process Method* is absolutely necessary when reusability of a *Process Model* is needed. The *Process Method* is the element that will be stored in a library or dictionary for reuse in future *Process Chains*.

The definition of a *Process Method* can be done by following 4 steps:

- Gather all references and documents concerning the process of interest (algorithm, contacts, datasheets, etc) and populate the metadata section.
- Write a Schematron schema to define characteristics of process inputs, outputs and parameters (I/O) such as name, unit of measure, definition, reference frames, etc...
- Write a detailed and clear MathML (or Text) description of the algorithm.
- Write an implementation of the algorithm using any programming language (This document only describes the use of the UAH Java framework) or find an existing one and reference it.

These four steps are described in detailed in the following sections using the National Weather Services (NWS) wind chill temperature formula as a simple example of *Process Method*. This process calculates the wind chill temperature by using values of wind speed and ambient temperature. This is shown again on figure 5.

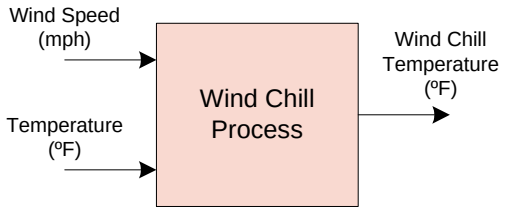


Figure 5: Wind Chill Temperature derivation process

B. Providing method metadata

The first step in the definition of a *Process Method* is to gather all references and documents related this particular method. The *Process Method* metadata section provides plenty of room to reference any relevant documents or contacts within the XML content. It is a good place to centralize information about the algorithm, the implementation, the experts involved, etc as well as any information used for discovery of the process by search engines.

The metadata section of the *Process Method* contains:

- **Description**: A general textual description
- **Identifiers**: allows one to identify the process with unique numbers or tokens for further reference.
- **Classifiers**: allows one to classify the process into specific categories such as application, domain, etc…
- **Legal and Security Constraints**: allows one to define special constraints on the use, or even the access to this method such as secrecy or copyrights.
- **Contacts**: allows one to list persons or organizations related to this method (patent owners, experts, implementers, etc…)
- **Documentation**: allows one to identify, describe and reference external documents in any format.
- **History**: provides links to previous versions and document updates.
- **Capabilities and Characteristics**: mainly used for discovery of the method by specifying characteristics such as result accuracy, complexity, estimated computation requirements, etc…

Please refer to "Appendix A: Process Method Example" as well as the SensorML language documentation for more details on XML encodings of this metadata content. The method document for the wind chill example is also available online at http://vast.uah.edu/SensorML/process/def/WindChill_Method.xml.

C. Using Schematron to define I/O constraints

Inputs, Outputs and Parameters (referred to as I/Os) need to be strictly defined in the *Process Method* description. This section is crucial since it defines how the algorithm can be interfaced to other components. To insure a proper integration of the new model into future *Process Chains*, several constraints need to be defined on all inputs, outputs and parameters:

- **Name**: a token giving the local name of an I/O component. Names defined by these constraints will need to be used by Process Method implementations and algorithm descriptions in order to reference a specific signal. Since I/O components can be nested using groups and arrays, the fully qualified name is given by a hierarchical list of local names (path).

- **Unit of Measure**: a URI pointing to a dictionary entry defining a unit of measure (meter, watt, etc…). This will be used while validating a *Process Chain* by making sure that only I/Os with compatible units are connected.

- **Definition**: a URI pointing to a dictionary entry defining a phenomenon (temperature, radiance, etc…) or other quantity (index, array size, etc…). This will be used while validating a *Process Chain* by making sure that only I/Os with compatible definitions are connected.

- **Reference Frame**: a URI pointing to a dictionary entry defining a spatial reference frame (only with position data). This will be used to enforce compatibility between spatial reference frames in a *Process Chain*.

- **Reference Time**: a URI pointing to a dictionary entry defining a temporal reference frame (only with time data). This will be used to enforce compatibility between time reference frames in a *Process Chain*.

Other more complex constraints can also be defined such as coherency between component units, multiplicity of certain signals, etc… A complex I/O structure with multiple choices of components or choices of units is though not recommended as it makes the development, testing and validation of robust implementations much harder.

Note: It is foreseen that unit conversion will be taken care of at the Process Chain level (i.e. units will be automatically converted between processes when needed and possible), thus removing the need for individual Process Models to support different units. It is also advised that specific processes should be developed for spatial coordinate transforms (converting spatial coordinates from one reference system to another) rather than leaving each process to deal with the responsibility of supporting different reference frames. This should make atomic processes and the overall chain more robust since the programmer doesn't have to worry about unit or coordinate system conversion in every processing block.

Schematron has been chosen as the language to define the structure of process inputs, outputs and parameters associated with a given *Process Method*, because its rich pattern definition syntax answers the need for the description of complex constraints on an abstract model such as the SensorML *Process Model* object. Additionally, this schema language will easily allow the defined I/O structure to be used for validation of existing *Process Model* descriptions and generation of SensorML document templates within SensorML design software.

Schematron is a simple but yet powerful schema language built on top of XPath that allows the clear definition of the type of constraints listed above. A Schematron schema is mainly a list of patterns, each of which is composed of rules defining assertions expressed relative to a given XML context. The structure of a schema is shown on the following XML snippet:

```
<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">

    <sch:title>Schema Title</sch:title>

    <sch:pattern id="PATTERN_ID" name="Pattern Description">

        <sch:rule context="xpath expression">
            <sch:assert test="xpath expression "> Message if incorrect </sch:assert>
            <sch:assert ... />
            <sch:assert ... />
        </sch:rule>

        <sch:rule context="xpath expression">
            <sch:assert ... />
            <sch:assert ... />
            <sch:assert ... />
        </sch:rule>

        <sch:rule ... />
        <sch:rule ... />
        <sch:rule ... />

    </sch:pattern>

    <sch:pattern ... />
    <sch:pattern ... />
    <sch:pattern ... />

</sch:schema>
```

For a pattern to match XML content in a document, all asserts contained in this pattern active rules must be true. Active rules are rules which context was found in the document (i.e. the XPath expression given in a rule context was evaluated to true). Each assertion specifies a test expression (typically checking if an XML element is present or if it has a given value) and a message that should be displayed by the validator when the assertion is not correct. Assertions are evaluated in the context specified by the enclosing rule. The following code snippet shows an example assert statement checking that at least one element called ‘input’ should have an attribute called ‘uom’ with a text value of ‘degF’.

```
<sch:assert test="input/@uom = 'degF'">
       Unit of 'temperature' must be 'degF'
</sch:assert>
```

This document recommends the following Schematron schema template to be used for the definition of a *Process Method* interface. This schema contains only one pattern which should be divided in separate sections:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE schema [
<!ENTITY process
"//sml:ProcessModel[sml:method/@xlink:href='urn:{authority}:def:process:{processName}:{version}']">
<!ENTITY input   "sml:inputs/sml:InputList/sml:input">
<!ENTITY output  "sml:outputs/sml:OutputList/sml:output">
<!ENTITY param   "sml:parameters/sml:ParameterList/sml:parameter">  ]>

<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">

    <sch:title>Wind Chill Process Interface</sch:title>

    <!-- Namespaces definitions -->
    <sch:ns prefix="sml" uri="http://www.opengis.net/sensorML"/>
    <sch:ns prefix="swe" uri="http://www.opengis.net/swe"/>
    <sch:ns prefix="xlink" uri="http://www.w3.org/1999/xlink"/>

    <!-- Unique Pattern -->
    <sch:pattern id="IO_CHECK" name="Check I/O Characteristics">

       <!-- First rule: Simple assertions for basic I/O structure -->
       <sch:rule context="&process;">
          <sch:assert test="simple xpath expression"> Message </sch:assert>
          <sch:assert test="simple xpath expression"> Message </sch:assert>
       </sch:rule>
       <!-- End of first Rule -->

       <!-- Other rules containing more complex assertions -->
       <sch:rule context="any sub context"/>
       <sch:rule context="any sub context"/>
       <sch:rule context="any sub context"/>
       ...

    </sch:pattern>
</sch:schema>
```

- Several XML entities are first defined in the header section (DOCTYPE) in order to keep the document simpler by avoiding repetitions of XPath expressions. For instance, this means that every time the entity “&input;” is used in the document, it will be replaced by the full corresponding XPath expression “sml:inputs/sml:InputList/sml:input” before evaluation. The first entity “&process;” is used as the context of the first rule in order to match all processes containing the corresponding method URN. This is a way of specifying that this schema applies only to *Process Models* using to this particular URN.

- The first rule is given in the process context. It defines basic inputs, outputs and parameters characteristics. This includes the type of component (Quantity, Count, Time, Category, DataGroup, DataArray, etc) as well as qualification attributes (name, uom, definition, axisCode, referenceFrame, referenceTime, etc) when necessary. This section allows only simple Xpath expressions and can be easily used to automatically generate the skeleton of an instance document. (only this rule is given in the example in Appendix B)

- The following rules can be given in any context that is a descendant of the one given by the "&process;" entity. They can contain more complex assertions to be used for further validation. These patterns do not have to be used for automatic templates generation and can contain complex XPath statements.

Since the XPath syntax used to check for valid inputs, outputs and parameters can be somewhat hard to understand. Here are two simple examples showing a *Process Model* instance and the corresponding interface schema for a process computing speed by the famous ratio V = D/T. Matching parts of schema and instance documents have been high lightened.

Process Model Instance:

```
<sml:SensorML xmlns:sml="http://www.opengis.net/sensorML" xmlns:swe="http://www.opengis.net/swe"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="1.0">

   <sml:ProcessModel id="PROCESS_ID">
      <sml:inputs>
         <sml:InputList>
            <sml:input name="distance">
               <swe:Quantity definition="urn:ogc:phenomenon:distance" uom="m"/>
            </sml:input>
            <sml:input name="duration">
               <swe:Quantity definition="urn:ogc:phenomenon:duration" uom="s"/>
            </sml:input>
         </sml:InputList>
      </sml:inputs>
      <sml:outputs>
         <sml:OutputList>
            <sml:output name="speed">
               <swe:Quantity definition="urn:ogc:def:phenomenon:speed" uom="m.s-1"/>
            </sml:output>
         </sml:OutputList>
      </sml:outputs>
      <sml:method xlink:href="urn:uah:def:process:speedComputation:1.0"/>
   </sml:ProcessModel>

</sml:SensorML>
```

Corresponding schema:

```
<!DOCTYPE schema [
<!ENTITY process
"//sml:ProcessModel[sml:method/@xlink:href='urn:uah:def:process:speedComputation:1.0']">
<!ENTITY input   "sml:inputs/sml:InputList/sml:input">
<!ENTITY output  "sml:outputs/sml:OutputList/sml:output">
<!ENTITY param   "sml:parameters/sml:ParameterList/sml:parameter">  ]>

<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">
    <sch:title>Speed Computation Process Interface</sch:title>

    <!-- Namespaces definitions -->
    <sch:ns prefix="sml" uri="http://www.opengis.net/sensorML"/>
    <sch:ns prefix="swe" uri="http://www.opengis.net/swe"/>
    <sch:ns prefix="xlink" uri="http://www.w3.org/1999/xlink"/>

    <sch:pattern id="IO_CHECK" name="Check I/O Characteristics">

       <sch:rule context="&process;">

          <sch:assert test="&inputs;[@name='distance']/swe:Quantity>
             Input 'distance' should be a Quantity
          </sch:assert>
          <sch:assert test="&inputs;[@name='distance']/swe:Quantity/@uom = 'm'">
             Unit of distance should be 'm'
          </sch:assert>
          <sch:assert test="&inputs;[@name='duration']/swe:Quantity>
             Input 'duration' should be a Quantity
          </sch:assert>
          <sch:assert test="&inputs;[@name='duration']/swe:Quantity/@uom = 's'">
             Unit of duration should be 's'
          </sch:assert>
          <sch:assert test="&outputs;[@name='speed']/swe:Quantity>
             Output 'speed' should be a Quantity
          </sch:assert>
          <sch:assert test="&outputs;[@name='speed']/swe:Quantity/@uom = 'm.s-1'">
             Unit of speed should be 'm.s-1'
          </sch:assert>

       </sch:rule>

    </sch:pattern>
</sch:schema>
```

Please refer to “Appendix B: Schematron Example” as well as Schematron and XPath documentations for more details on how to encode process I/O interface using Schematron. The schema for the wind chill example is also available online at http://vast.uah.edu/SensorML/process/sch/WindChill_Interface.sch.

D. Using MathML to define the algorithm

MathML is a W3C standard using XML to define mathematical equations. It allows the definition of a rigorous executable algorithm based on these equations (MathML content) and provides mechanisms for rendering equations using mathematical notations (MathML presentation). It can also be embedded into HTML for direct presentation on a web page.

In order to describe equations, MathML (content) uses a very rigorous syntax to specify the order of operations and what they apply to. The following XML snippet shows how a simple equation like $y = x + 2$ can be encoded:

```
<apply>
   <eq/>
   <ci>y</ci>
   <apply>
      <plus/>
      <ci>x</ci>
      <cn>2</cn>
   </apply>
</apply>
```

The first child of each 'apply' element identifies the operator to be used and the next children describe the operation parameters. By nesting 'apply' elements containing different operators, it is possible to describe any mathematical equations from elementary calculus to multivariable analysis and linear algebra. This type of content MathML can be interpreted and used directly for processing since it rigorously specifies mathematical operations, but can also be transformed to MathML presentation for direct visualization using mathematical notations.

The latest formula used by the United States National Weather Services to compute wind chill temperature is:

$$T_c = 35.74 + 0.6215 \cdot T - 35.75 \cdot V^{0.16} + 0.4275 \cdot T \cdot V^{0.16},$$

with T: Ambient Temp (ºF), V: Wind Speed (mph), T_c: Wind Chill Temp (ºF).

Please refer to "Appendix C: MathML Example" for the complete MathML content code of the wind chill equation. You can also see a rendered version of the HTML/MathML document given in appendix using Mozilla Firefox 1.5 at http://vast.uah.edu/SensorML/process/math/WindChill_Algorithm.xml.

E. Linking to source and binaries of specific implementations

A *Process Method* description also provides a mean of linking to existing or newly created implementations of the method in various languages.

The XML encoding allows the linking to both source code and binary distributions for an unlimited number of programming languages such as Java, C++, LISP, etc… Code can be designed in a modular way so that it can be dynamically loaded into the corresponding processing framework at runtime. (For instance, Java provides mechanism to easily load classes at runtime).

The implementation list is intended to be used for code that is strictly implementing the specified algorithm and interface, so that it can be used for dynamically loading new processing modules in SensorML enabled software. Reference implementations which are not directly applicable but useful in order to better define the algorithm can be specified in the documentation section.

IV. Referencing a Process Method in a Chain

A. *Writing a Process Model*

Once the *Process Method* definition is written, it can be referenced in an instance document by pointing to it via the "xlink:href" attribute present on the "method" element of the *Process Model* object, as shown on the following XML fragment:

```
<sml:ProcessModel id="PROCESS_ID">
   <sml:metadata ... />
   <sml:inputs ... />
   <sml:outputs ... />
   <sml:parameters ... />
   <sml:method xlink:href="urn:{authority}:def:process:{processName}:{version}"/>
</sml:ProcessModel>
```

The *Process Model* instance basically defines the inputs, outputs and parameters used for this particular instance of the process. It needs to be valid with respect to the Schematron given as part of the *Process Method* description. Particular parameter values can be chosen and specific combinations of I/O and units can be selected if the schema allows multiple possibilities. Inputs, outputs and parameters are encoded using XML structures like the ones shown on the following XML snippet:

```
<sml:inputs>
   <sml:InputList>
      <sml:input name="inputName1">
         <swe:Quantity definition="urn:ogc:phenomenon:temperature" uom="degC"/>
      </sml:input>
      <sml:input name="inputName2">
         <swe:Quantity definition="urn:ogc:phenomenon:pressure" uom="Pa"/>
      </sml:input>
   </sml:InputList>
</sml:inputs>

<sml:outputs>
   <sml:OutputList>
      <sml:output name="outputName1">
         <swe:Quantity definition="urn:ogc:def:phenomenon:speed" uom="m.s-1"/>
      </sml:output>
      <sml:output name="outputName2">
         <swe:Quantity definition="urn:ogc:def:phenomenon:distance" uom="m"/>
      </sml:output>
   </sml:OutputList>
</sml:outputs>

<sml:parameters>
   <sml:ParameterList>
      <sml:parameter name="paramName1">
         <swe:Count definition="urn:ogc:def:data:index"/>1200</swe:Count>
      </sml:parameter>
      <sml:parameter name="paramName2">
         <swe:Time definition="urn:ogc:def:time:iso8601"/>2006-01-27T12:00:00.00</swe:Time>
      </sml:parameter>
   </sml:ParameterList>
</sml:parameters>
```

Please refer to "Appendix D: Process Model Example" for complete XML encodings of a *Process Model*. The wind chill example is also available online at http://vast.uah.edu/SensorML/instances/process/WindChill_Process.xml.

B. Writing a Process Chain

A *Process Model* is often part of a complete *Process Chain* which combines several atomic processes in order to define more complex processing rules. A Process Chain defines its own inputs, outputs and parameters and how they are connected to internal sub-processes. *Process Models* used in a chain can be included inline or referenced using the "xlink:href" attribute. Internal connections are specified using a path like pointer to link to source and destination of a given signal. These pointers are made of a hierarchy of names corresponding to values of name attributes given on "process" and I/O component elements. This is shown on the following XML fragment:

```
<sml:ProcessChain id="PROCESSING_CHAIN_ID">
      <sml:inputs>
         <sml:InputList>
            <sml:input name="inputName">
               <swe:Quantity definition="urn:ogc:phenomenon:inputPhenomenon" uom="inputUnit"/>
            </sml:input>
         </sml:InputList>
      </sml:inputs>

      <sml:outputs>
         <sml:OutputList>
            <sml:output name="outputName">
               <swe:Quantity definition="urn:ogc:def:phenomenon:outputPhenomenon" uom="outputUnit"/>
            </sml:output>
         </sml:OutputList>
      </sml:outputs>

      <sml:processes>
         <sml:ProcessList>
            <sml:process name="localProcessName1" xlink:href="URI to Process XML"/>
            <sml:process name="localProcessName2">
               <sml:ProcessModel id="INLINE_PROCESS_ID">
                  <sml:description ... />
                  <sml:inputs ... />
                  <sml:outputs ... />
                  <sml:parameters ... />
                  <sml:method xlink:href="urn:{authority}:def:process:{processName}:{version}"/>
               </sml:ProcessModel>
            </process>
         </sml:ProcessList>
      </sml:processes>

      <sml:connections>
         <sml:ConnectionList>
            <sml:connection>
               <sml:Link>
                  <sml:source ref="this/inputs/inputName"/>
                  <sml:destination ref="localProcessName1/inputs/inputName3"/>
               </sml:Link>
            </sml:connection>
            <sml:connection>
               <sml:Link>
                  <sml:source ref="localProcessName1/outputs/outputName"/>
                  <sml:destination ref="this/outputs/outputName"/>
               </sml:Link>
            </sml:connection>
         </sml:ConnectionList>
      </sml:connections>
   </sml:ProcessChain>
```

Please refer to "Appendix E: Process Chain Example" for XML encodings of a *Process Chain*. An example using the wind chill example is also available online at http://vast.uah.edu/SensorML/instances/process/ProcessChain.xml.

V. Creating a Process Method Library

A. Writing the library document(s)

A library of *Process Method* definitions can be constituted of a single XML document grouping all method definitions for a given domain of interest, or of several documents, each of which containing one *Process Method* definition. An intermediate solution is also possible for a better categorization of definitions. If using a service interface, it is possible to store documents in any format (such as in a database) and create the XML on the fly when a given process is requested.

B. Pointing to a method in a library

1. Using a URL

A URL can be used to point directly to:

- A *Process Method* at the root of an XML document:
  http://myDomain.com/myProcessDocument.xml
- A *Process Method* in an XML fragment if appending an ID to the URL:
  http://myDomain.com/myLibraryDocument.xml#myProcess
- A service that can return the requested *Process Method* definition:
  http://myDomain.com/myLibraryService?processID=myProcess

2. Using a URN

A more abstract URN (Uniform Resource Name) can be used if the URL of the document cannot be exposed (or can change) and URN resolution capabilities (such as a service or DDNS server) are in place.

OGC is in the process of defining a URN scheme that can apply to various resources used across diverse OGC specifications. The recommended scheme to reference a given *Process Method* document is:

urn:{myNameSpace}:def:process:{myProcessName}:{version}

VI. Implementing a Process Method

Process Methods can of course be implemented in diverse programming languages and targeted to various platforms. However, this document only goes over the specifics of the UAH Java framework which was designed especially for SensorML processing.

… TBD …

VII. Validating SensorML documents

SensorML documents are intended to be validated in two steps:

- The first step uses an XML Schema to validate the overall structure of the documents and the grammar used (Elements names, etc…).

- Each process is then validated with the corresponding Schematron schema which checks for valid inputs, outputs and parameters. This should check for valid names, units, definitions and I/O structures.

A. Syntax and Grammar Validation

An *XML Schema* defines the main structure and grammar of SensorML objects like Systems, Sensors, Detectors, Process Models, Process Chains, etc…

This schema (now version 1.0) maintained by OGC members should be used as a reference for the first validation step of any SensorML document.

B. Validation of Process Model Characteristics

Specific *Process Models* need to be validated using a separate Schematron schema which checks for inputs/outputs/parameters characteristics. Schematron is a very simple and flexible way of defining patterns on an XML document. In the case of a *Process Model*, it will typically be used to check names, structure and units of inputs, outputs and parameters. Additional constraints can also be specified.

This step involves pre-parsing the document so that the *Process Method* definition containing the Schematron schema for each process type can be downloaded (or fetched locally if previously downloaded). These schemas can then be used to validate the document.

VIII. Submitting a Process for approval

A development project has been setup on SourceForge to enable team development of complex algorithms and to provide a central place for storing new process definitions and implementations.

Through this web interface, it is also possible to:

- Submit a new *Process Method* document along with its implementation in the UAH Java framework. It will be added to the library after validation and testing.

- Submit a new *Process Method* document along with its implementation in a different language/framework or without implementation at all. It will be added to the library and a request for implementation will be posted on SourceForge so that community developers can implement it using the UAH Java framework.

- Request the creation of a specific *Process Method* document and implementation. In this case, enough information such as references (at least) must be supplied. Community developers having an interest for it will then be able to write a Process Method and even implement it

Appendix A: Process Method Example

WindChill_Method.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:ProcessMethod xmlns:sml="http://www.opengis.net/sensorML" xmlns:swe="http://www.opengis.net/swe"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xlink="http://www.w3.org/1999/xlink"
xsi:schemaLocation="http://www.opengis.net/sensorML http://vast.uah.edu/schemas/sensorML/1.0.30/base/process.xsd">

   <!-- Description -->
   <sml:description>
      <swe:Discussion>Method defining the Wind Chill Temperature calculation process</swe:Discussion>
   </sml:description>

   <!-- Contact -->
   <sml:contact role="author">
      <sml:ResponsibleParty>
         <sml:individualName>Alexandre Robin</sml:individualName>
         <sml:organizationName>University of Alabama in Huntsville</sml:organizationName>
         <sml:positionName>Research Scientist</sml:positionName>
         <sml:contactInfo>
            <sml:phone>
               <sml:voice>(256)-961-7978</sml:voice>
            </sml:phone>
            <sml:address>
               <sml:electronicMailAddress>robin@nsstc.uah.edu</sml:electronicMailAddress>
            </sml:address>
         </sml:contactInfo>
      </sml:ResponsibleParty>
   </sml:contact>

   <!-- Documentation -->
   <sml:documentation role="formula">
      <sml:Document>
         <sml:description>
            <swe:Discussion>NWS document giving wind chill temperature formula</swe:Discussion>
         </sml:description>
         <sml:fileLocation xlink:href="http://www.wrh.noaa.gov/slc/projects/wxcalc/windChill.pdf"/>
      </sml:Document>
   </sml:documentation>

   <!-- I/O Interface -->
   <sml:interface>
      <sml:InterfaceDefinition>
         <sml:description>
            <swe:Discussion>
               1 input must be "ambiant_temperature" in degF.
               1 input must be "wind_speed" in mph.
               1 output must be "windchill_temperature" in degF.
               No parameters
            </swe:Discussion>
         </sml:description>
         <sml:schematron xlink:href="http://vast.uah.edu/SensorML/process/sch/WindChill_Interface.sch"/>
      </sml:InterfaceDefinition>
   </sml:interface>

   <!-- Algorithm -->
   <sml:algorithm>
      <sml:AlgorithmDefinition>
         <sml:description>
            <swe:Discussion>
               Apply following formula: Tc = 35.74 + 0.6215*T - 35.75*V^0.16 + 0.4275*T*V^0.16
            </swe:Discussion>
         </sml:description>
         <sml:mathML xlink:href="http://vast.uah.edu/SensorML/process/math/WindChill_Algorithm.xml"/>
      </sml:AlgorithmDefinition>
   </sml:algorithm>

   <!-- Implementations -->
   <sml:implementation>
      <sml:ImplementationCode language="java" framework="uah-dpf" version="1.0">
         <sml:source xlink:href="http://vast.uah.edu/SensorML/process/src/WindChill_Process.java"/>
         <sml:binary xlink:href="http://vast.uah.edu/SensorML/process/bin/WindChill_Process.class"/>
      </sml:ImplementationCode>
   </sml:implementation>

</sml:ProcessMethod>
```

Appendix B: Schematron Example

WindChill_Interface.sch

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE schema [
<!ENTITY process "//sml:ProcessModel[sml:method/@xlink:href='urn:ogc:def:process:WindChill:1.0']">
<!ENTITY input   "sml:inputs/sml:InputList/sml:input">
<!ENTITY output  "sml:outputs/sml:OutputList/sml:output">
<!ENTITY param   "sml:parameters/sml:ParameterList/sml:parameter">
]>
<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">
    <sch:title>Wind Chill Process Interface</sch:title>

    <!--=========================-->
    <!-- Namespaces definitions -->
    <!--=========================-->
    <sch:ns prefix="sml" uri="http://www.opengis.net/sensorML"/>
    <sch:ns prefix="swe" uri="http://www.opengis.net/swe"/>
    <sch:ns prefix="xlink" uri="http://www.w3.org/1999/xlink"/>

    <!--=====================================-->
    <!-- Check I/O Count, Names and Contents -->
    <!--=====================================-->
    <sch:pattern id="IO_CHECK" name="Check I/O Characteristics">
        <sch:rule context="&process;">

            <!--============================-->
            <!-- Ambiant Temperature input -->
            <!--============================-->
            <sch:assert test="&input;[@name='ambiant_temperature']/swe:Quantity">
                Input Quantity named 'ambiant_temperature' must be present
            </sch:assert>
            <sch:assert test="&input;[@name='ambiant_temperature']/swe:Quantity/@uom = 'degF'">
                Unit of 'ambiant_temperature' must be 'degF'
            </sch:assert>
            <sch:assert test="&input;[@name='ambiant_temperature']/swe:Quantity/@definition = 'urn:ogc:def:phenomenon:temperature'">
                Definition of 'ambiant_temperature' must be 'urn:ogc:def:phenomenon:temperature'
            </sch:assert>

            <!--====================-->
            <!-- Wind Speed input -->
            <!--====================-->
            <sch:assert test="&input;[@name='wind_speed']/swe:Quantity">
                Input Quantity named 'wind_speed' must be present
            </sch:assert>
            <sch:assert test="&input;[@name='wind_speed']/swe:Quantity/@uom = 'mph'">
                Unit of 'wind_speed' must be 'mph'
            </sch:assert>
            <sch:assert test="&input;[@name='wind_speed']/swe:Quantity/@definition = 'urn:ogc:def:phenomenon:windSpeed'">
                Definition of 'wind_speed' must be 'urn:ogc:def:phenomenon:windSpeed'
            </sch:assert>

            <!--====================-->
            <!-- Wind Chill output -->
            <!--====================-->
            <sch:assert test="&output;[@name='windchill_temperature']/swe:Quantity">
                Output Quantity named 'windchill_temperature' must be present
            </sch:assert>
            <sch:assert test="&output;[@name='windchill_temperature']/swe:Quantity/@uom = 'degF'">
                Unit of 'windchill_temperature' must be 'degF'
            </sch:assert>
            <sch:assert test="&output;[@name='windchill_temperature']/swe:Quantity/@definition = 'urn:ogc:def:phenomenon:temperature'">
                Definition of 'windchill_temperature' must be 'urn:ogc:def:phenomenon:temperature'
            </sch:assert>

        </sch:rule>
    </sch:pattern>
</sch:schema>
```

Appendix C: MathML Example

WindChill_Algorithm.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="./ctop.xsl"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN" "http://www.w3.org/TR/MathML2/dtd/xhtml-math11-f.dtd" [
<!ENTITY mathml "http://www.w3.org/1998/Math/MathML">]>

<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html"/>
      <title>Wind Chill Temperature Formula</title>
   </head>
   <body>
      <h2 align="center">Wind Chill Temperature Formula</h2>

      <math mode="display" xmlns="&mathml;">
         <apply>
            <eq/>
            <ci>T</ci>
            <ci>ambiant_temperature</ci>
         </apply>
      </math>

      <math mode="display" xmlns="&mathml;">
         <apply>
            <eq/>
            <ci>V</ci>
            <ci>wind_speed</ci>
         </apply>
      </math>

      <math mode="display" xmlns="&mathml;">
         <apply>
            <eq/>
            <ci>Tc</ci>
            <apply>
               <plus/>
               <cn type="real">35.74</cn>
               <apply>
                  <times/>
                  <cn type="real">0.6215</cn>
                  <ci>T</ci>
               </apply>
               <apply>
                  <minus/>
                  <apply>
                     <times/>
                     <cn type="real">35.75</cn>
                     <apply>
                        <power/>
                        <ci>V</ci>
                        <cn type="real">0.16</cn>
                     </apply>
                  </apply>
               </apply>
               <apply>
                  <times/>
                  <cn type="real">0.4275</cn>
                  <ci>T</ci>
                  <apply>
                     <power/>
                     <ci>V</ci>
                     <cn type="real">0.16</cn>
                  </apply>
               </apply>
            </apply>
         </apply>
      </math>

      <math mode="display" xmlns="&mathml;">
         <apply>
            <eq/>
            <ci>windchill_temperature</ci>
            <ci>Tc</ci>
         </apply>
      </math>

   </body>
</html>
```

$$T = ambiant\_temperature$$

$$V = wind\_speed$$

$$T_c = 35.74 + 0.6215 \cdot T - 35.75 \cdot V^{0.16} + 0.4275 \cdot T \cdot V^{0.16}$$

$$windchill\_temperature = T_c$$

Appendix D: Process Model Example

WindChill_Process.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:SensorML xmlns:sml="http://www.opengis.net/sensorML" xmlns:swe="http://www.opengis.net/swe" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/sensorML
http://vast.uah.edu/schemas/sensorML/1.0.30/base/sensorML.xsd" version="1.0">
   <sml:ProcessModel id="WINDCHILL_PROCESS">

      <!--==================-->
      <!-- METADATA SECTION -->
      <!--==================-->
      <sml:description>
         <swe:Discussion>Wind chill temperature computation process</swe:Discussion>
      </sml:description>

      <!--====================-->
      <!-- INPUTS DEFINITION -->
      <!--====================-->
      <sml:inputs>
         <sml:InputList>
            <sml:input name="ambiant_temperature">
               <swe:Quantity definition="urn:ogc:def:phenomenon:temperature" uom="degF"/>
            </sml:input>
            <sml:input name="wind_speed">
               <swe:Quantity definition="urn:ogc:def:phenomenon:windSpeed" uom="mph"/>
            </sml:input>
         </sml:InputList>
      </sml:inputs>

      <!--====================-->
      <!-- OUTPUTS DEFINITION -->
      <!--====================-->
      <sml:outputs>
         <sml:OutputList>
            <sml:output name="windchill_temperature">
               <swe:Quantity definition="urn:ogc:def:phenomenon:temperature" uom="degF"/>
            </sml:output>
         </sml:OutputList>
      </sml:outputs>

      <!--====================-->
      <!-- METHOD DEFINITION -->
      <!--====================-->
      <sml:method xlink:href="urn:ogc:def:process:WindChill:1.0"/>

   </sml:ProcessModel>
</sml:SensorML>
```

Appendix E: Process Chain Example

ProcessChain.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:SensorML xmlns:sml="http://www.opengis.net/sensorML" xmlns:swe="http://www.opengis.net/swe" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xlink="http://www.w3.org/1999/xlink" xsi:schemaLocation="http://www.opengis.net/sensorML
http://vast.uah.edu/schemas/sensorML/1.0.30/base/sensorML.xsd" version="1.0">
   <sml:ProcessChain id="PROCESSING_CHAIN">

      <!--====================-->
      <!-- Inputs Definition -->
      <!--====================-->
      <sml:inputs>
         <sml:InputList>
            <sml:input name="ambiantTemp"><swe:Quantity definition="urn:ogc:phenomenon:temperature" uom="degF"/></sml:input>
            <sml:input name="windSpeed"><swe:Quantity definition="urn:ogc:phenomenon:windSpeed" uom="mph"/></sml:input>
            <sml:input name="relativeHumidity"><swe:Quantity definition="urn:ogc:phenomenon:relativeHumidity"/></sml:input>
         </sml:InputList>
      </sml:inputs>

      <!--====================-->
      <!-- Outputs Definition -->
      <!--====================-->
      <sml:outputs>
         <sml:OutputList>
            <sml:output name="windChillTemp"><swe:Quantity definition="urn:ogc:def:phenomenon:temperature"/></sml:output>
            <sml:output name="dewPointTemp"><swe:Quantity definition="urn:ogc:def:phenomenon:temperature"/></sml:output>
         </sml:OutputList>
      </sml:outputs>

      <!--==================-->
      <!-- Sub-Process List -->
      <!--==================-->
      <sml:processes>
         <sml:ProcessList>
            <sml:process name="windChillProcess"
                         xlink:href="http://vast.uah.edu/SensorML/instances/process/WindChill_Process.xml#WINDCHILL_PROCESS"/>
            <sml:process name="dewPointProcess"
                         xlink:href="http://vast.uah.edu/SensorML/instances/process/DewPoint_Process.xml#DEWPOINT_PROCESS"/>
         </sml:ProcessList>
      </sml:processes>

      <!--=============-->
      <!-- Connections -->
      <!--=============-->
      <sml:connections>
         <sml:ConnectionList>
            <sml:connection><sml:Link>
                  <sml:source ref="this/inputs/ambiantTemp"/>
                  <sml:destination ref="windChillProcess/inputs/ambiant_temperature"/>
               </sml:Link></sml:connection>
            <sml:connection>
               <sml:Link><sml:source ref="this/inputs/ambiantTemp"/>
                  <sml:destination ref="dewPointProcess/inputs/ambiant_temperature"/>
               </sml:Link></sml:connection>
            <sml:connection><sml:Link>
                  <sml:source ref="this/inputs/windSpeed"/>
                  <sml:destination ref="windChillProcess/inputs/wind_speed"/>
               </sml:Link></sml:connection>
            <sml:connection><sml:Link>
                  <sml:source ref="this/inputs/relativeHumidity"/>
                  <sml:destination ref="dewPointProcess/inputs/relative_humidity"/>
               </sml:Link></sml:connection>
            <sml:connection><sml:Link>
                  <sml:source ref="windChillProcess/outputs/windchill_temperature"/>
                  <sml:destination ref="this/outputs/windChillTemp"/>
               </sml:Link></sml:connection>
            <sml:connection><sml:Link>
                  <sml:source ref="dewPointProcess/outputs/dewpoint_temperature"/>
                  <sml:destination ref="this/outputs/dewPointTemp"/>
               </sml:Link></sml:connection>
         </sml:ConnectionList>
      </sml:connections>

   </sml:ProcessChain>
</sml:SensorML>
```

Appendix F: Java Source Code

WindChill_Process.java

```
/*************************************************************
 (c) Copyright 2005, University of Alabama in Huntsville (UAH)
 ALL RIGHTS RESERVED
 This software is the property of UAH.
 It cannot be duplicated, used, or distributed without the
 express written consent of UAH.
*************************************************************/

package org.vast.sensorML.process;
import org.ogc.process.ProcessException;
import org.vast.data.*;
import org.vast.process.*;

/**
 * <p><b>Title:</b><br/>
 * WindChill_Process
 * </p>
 *
 * <p><b>Description:</b><br/>
 * Computes wind chill temperature in degrees Farenheit
 * using ambiant temperature in degrees Farenheit and
 * wind speed in miles per hour, using NWS formula:
 * Tc = 35.74 + 0.6215*T - 35.75*V^0.16 + 0.4275*T*V^0.16
 * </p>
 *
 * <p>Copyright (c) 2005</p>
 * @author Alexandre Robin
 * @date Jan 20, 2006
 * @version 1.0
 */
public class WindChill_Process extends DataProcess
{
    DataValue inputTemp;
    DataValue inputWindSpeed;
    DataValue outputTemp;

    public WindChill_Process() {}

    /**
     * Initializes the process
     * Gets handles to input/output components
     */
    public void init() throws ProcessException
    {
        try
        {
            // I/O mappings
            inputTemp = (DataValue) inputData.getComponent("ambiant_temperature");
            inputWindSpeed = (DataValue) inputData.getComponent("wind_speed");
            outputTemp = (DataValue) outputData.getComponent("windchill_temperature");
        }
        catch (ClassCastException e)
        {
            throw new ProcessException("Invalid I/O data", e);
        }
    }

    /**
     * Executes process algorithm on inputs and set output data
     */
    public void execute() throws ProcessException
    {
        double T = inputTemp.getData().getDoubleValue();
        double V = inputWindSpeed.getData().getDoubleValue();

        double V16 = Math.pow(V, 0.16);
        double Tc = 35.74 + 0.6215 * T - 35.75 * V16 + 0.4275 * T * V16;

        outputTemp.getData().setDoubleValue(Tc);
    }
}
```

References

- Sensor Model Language (SensorML) Implementation Specification, OGC-05-086r2.

- Sensor Model Language UAH - VAST webpage, http://vast.uah.edu/SensorML/

- Mathematical Markup Language (MathML) V2.0, W3C Recommendation, http://www.w3.org/TR/2003/REC-MathML2-20031021/

- MathML Official W3C webpage, http://www.w3.org/Math/

- Document Schema Definition Languages (DSDL) – Part 3: Rule-based validation – Schematron, ISO/IEC FDIS 19757-3, http://dsdl.org/

- Schematron Online Tutorial, http://www.zvon.org/xxl/SchematronTutorial/General/contents.html