DECLARATIVE SENSOR INTERFACE DESCRIPTORS FOR THE SENSOR WEB

Arne Broering[1,2], Stefan Below[1] and Theodor Foerster[1]

[1]IfGI, University of Muenster, Germany, http://swsl.uni-muenster.de, (arneb, stefan.below, theodor.foerster)@wwu.de
[2]ITC, University of Twente, Netherlands

KEY WORDS: Sensor, Spatial Infrastructures, Integration, Internet/Web, Interoperability, Standards

ABSTRACT:

The Sensor Web Enablement (SWE) initiative of the Open Geospatial Consortium (OGC) defines standards for Web Service interfaces and data encodings usable as building blocks to implement a Sensor Web. These standards encapsulate sensors for web-based discovery, tasking and access. In recent years, SWE has been applied in a multitude of projects, demonstrating its suitability in real world scenarios. However, there is still a fundamental challenge to be tackled. While SWE enables interoperability with the upper application layer, the connection between SWE and the underlying sensor layer and its heterogeneous protocols is not yet sufficiently described. To close this gap, a declarative model for *Sensor Interface Descriptors* (SID) based on OGC's SensorML standard is presented here. An SID for a particular sensor enables a so called SID interpreter to translate between the communication protocol of the sensor and the Sensor Web. We have developed a generic SID interpreter capable of connecting sensors to Sensor Observation Services and Sensor Planning Services based on their SID. For illustration, an SID describing a radiation detector of the German Federal Office for Radiation Protection is presented and used for integration with the SWE services. The presented approach of SIDs is the basis for realizing our vision of sensor plug & play and will make sensors on-the-fly available on the Sensor Web.

1 INTRODUCTION

The vision of the (Geo-)Sensor Web is that access to (geo-)sensors is as uniform and easy as access to resources on the World Wide Web today [Nittel et al., 2008]. The goal is to enable Web-based sharing, discovery, exchange and processing of sensor observations, as well as task planning of sensor systems [Gong et al., 2010]. The Sensor Web Enablement (SWE) initiative of the Open Geospatial Consortium (OGC) defines standards which can be utilized to build such a Sensor Web [Botts et al., 2008]. SWE standards make sensors available over the Web through standardized formats and Web Service interfaces by hiding the sensor communication details and the heterogeneous sensor protocols from the application layer.

In recent years, the SWE standards have been applied in various projects (e.g. [Chung et al., 2009, Jirka et al., 2009, Stasch et al., 2008, Schimak and Havlik, 2009] showing their practicability and suitability in real world scenarios. However, there is still an essential challenge to be tackled. There is a gap of interoperability between the SWE services and the sensors [Walter and Nash, 2009]. SWE defines service interfaces from an application-oriented perspective. The connection between a SWE service and a sensor is not yet sufficiently defined by the specifications.

Although, the Sensor Observation Service (SOS) (Section 2) provides operations for insertion of sensors and their data, the utilization of the according operations still requires reformatting of the native sensor protocol to the SWE protocol. How and where this is done is not defined by the specifications. Due to bandwidth and processing power limitations, a sensor itself is usually not able to transform and upload its data to an SOS. Most obvious is the interoperability gap at the Sensor Planning Service (SPS) (Section 2) which enables an interoperable tasking of sensors. It is not defined how an SPS transforms a retrieved sensor task to a command of the sensor protocol. Today, the connection between sensors and SWE services is usually established by manually adapting the internals of the SWE service implementation to the specific sensor interface. Such adaptations have to be built for each pair of service implementation and sensor interface which leads to extensive efforts in developing large-scale systems [Aberer et al., 2006].

Minimizing these efforts is a further step towards an on-the-fly integration, a plug & play of sensors with the Sensor Web. Such functionality would significantly support for example disaster management applications where an ad-hoc densification of a sensor network is demanded. Examples range from flooding scenarios, in which the affected river courses are not densely enough covered with water gages, to incidents in nuclear plants, which require ad-hoc deployments of radiation detectors. Assuming a Sensor Web is already in place and used by disaster relief organizations as a coherent infrastructure to access sensors, an integration of new sensors in a most efficient way becomes necessary.

This work addresses the identified interoperability gap. We develop a model for Sensor Interface Descriptors (SID) which enables the declarative description of sensor interfaces, including the definition of the communication protocol, sensor commands, processing steps and metadata association (Section 3). The model is designed as a profile and extension of OGC's Sensor Model Language standard. Based on this model, SID interpreters can be built which are able to translate between sensor protocol and Sensor Web protocols and hence close the described interoperability gap. Such interpreters for SID instances can be built independently of particular sensor technology. They establish the connection to a sensor and are able to communicate with it by using the sensor protocol definition of the SID. This work presents the implementation of a generic SID interpreter (Section 4). It transfers data, retrieved from a sensor, to a Sensor Observation Service. Also, it transforms tasks, submitted to a Sensor Planning Service, to commands which are then forwarded to a sensor. SID instances for particular sensor types can be reused in different scenarios and can be shared among user communities. The ability of an SID interpreter to connect sensors and Sensor Web services in an ad hoc manner based on the sensor's SID is a next step towards realizing sensor plug & play.

2 BACKGROUND & RELATED WORK

The main Web Services of the SWE framework are the Sensor Observation Service (SOS) and the Sensor Planning Service

(SPS). The SOS [Na and Priest, 2007] provides interoperable access to real to sensor data as well as sensor metadata. To control and task sensors the SPS [Simonis, 2007] can be used. A common application of SPS is to define simple sensor parameters such as the sampling rate but also more complex tasks such as mission planning of satellite systems. Apart from these Web Service specifications, SWE incorporates information models for observed sensor data, the Observations & Measurements (O&M) [Cox, 2007] standard, as well as for the description of sensors, the Sensor Model Language (SensorML) [Botts, 2007]. SensorML specifies a model and encoding for sensor related processes such as measuring or post processing procedures. Physical as well as logical sensors are modeled as *processes*. The functional model of a process can be described in detail, including its identification, classification, inputs, outputs, parameters, and characteristics such as a spatial or temporal description. Processes can be composed by process chains. O&M defines a model and encoding for *observations*. An observation has a result (e.g. 0,7 mSv/a) which is an estimated value of an *observed property* (e.g. radiation), a particular characteristic of a *feature of interest* (e.g. the city of Muenster). The result value is generated by a *procedure*, e.g. a sensor such as a radiation detector described in SensorML. These four central components are linked within SWE.

Bridging the interoperability gap between the Sensor Web layer, consisting of those SWE components, and the lower sensor layer can be generally addressed from two directions. On the one hand, the interoperable access on the sensor layer can be improved. On the other hand, it can be approached from the Sensor Web layer by introducing mechanisms to abstract from the variety of sensor protocols.

The first direction is addressed by several standardization approaches. Most promising is the IEEE 1451 family of standards[1] since it is backed by a large number of vendors. IEEE 1451 is a universal approach to connect sensors to diverse networks and systems. An important feature of this standards family is the definition of a Transducer Electronic Data Sheet (TEDS) which is a small memory device attached to the transducer describing for example its identification, calibration, correction data, measurement range, and manufacturer related information. Nevertheless, the expressiveness of TEDS is limited and it cannot capture all metadata of a sensor. For example, higher level processing of sensor data cannot be described in TEDS. This requirement is addressed by SensorML. Therefore, [Hu et al., 2007] convert TEDS to SensorML by creating a knowledge base which maps each TEDS property to an appropriate SensorML description. It would be promising to extend this approach and to combine it with our work to automatically generate SIDs for IEEE 1451 sensors so that an SID interpreter can connect IEEE 1451 sensors on-the-fly with SWE services.

However, in today's real world applications not only IEEE 1451 but in fact a huge variety of sensor interfaces (standardized or proprietary) are utilized. Hence, different projects are approaching the interoperability gap from the upper Sensor Web layer. AnySen [Bleier et al., 2009] is capable of reading and interpreting data from sensor nodes by abstracting the sensor protocols and reading the sensor description from an external file. The authors do not detail but claim that AnySen allows the formatting of these sensor descriptions compliant to the SensorML standard. While AnySen supports the provision of sensor data by connecting to an SOS, other SWE services, in particular tasking of sensors through an SPS, are not supported. Walter & Nash [Walter and Nash, 2009] identify the interoperability gap and analyze different system models which may lower the implementation barrier for coupling sensor systems and SWE services. The authors suggest lightweight SWE connectors which can be adapted to different raw sensor formats to convert them to SWE-based data models. They state that such SWE connectors could be implemented for a wide range of different sensor types. They come up with design approaches, but do not detail them. The Sensor Abstraction Layer (SAL) [Gigan and Atkinson, 2007] is most similar to the SID concept. SAL makes use of SensorML to describe sensor interfaces. As a library, it offers high-level functions to access sensors by hiding their specific technological details. The architecture follows a split design consisting of lightweight SAL agents running on the sensor gateways to handle the communication with the hardware and SAL clients usable by application developers to invoke specific actions on sensors managed by an agent. Missing are mechanisms for the final connection to SWE services and the integration of sensors with the Sensor Web.

[1] http://ieee1451.nist.gov/

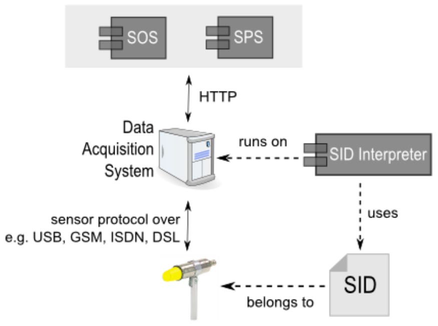

Figure 1: Usage of SID in SWE deployment

3 SENSOR INTERFACE DESCRIPTORS

This section presents the SID model. Figure 1 shows the deployment of a Sensor Web infrastructure including the usage of SIDs. A sensor communicates with a data acquisition system in its specific sensor protocol over a transmission technology such as ISDN or GSM. This sensor can also act as a sensor gateway (network sink) so that other nodes of a (possibly mobile) sensor network communicate with it. The SID interpreter runs on the data acquisition system and uses SID instances for the different sensors of the sensor network to translate between the sensor specific protocol and the SWE protocols. The interpreter is responsible to register a sensor at a SWE service and to upload sensor data to an SOS. Also, it is responsible for the opposite communication direction and forwards tasks received by an SPS to a sensor.

A strong requirement on the design of the SID model is the strict encapsulation of the SID within the SensorML document. The SID part of the SensorML document is specific for a certain sensor type, not a particular sensor instance. Hence, an encapsulation allows to reuse it in the SensorML descriptions of different sensors which are of the same type. The approach developed here, encapsulates the SID within the *interface* element of a SensorML document.

The *interface* element contains a stack of layers (Figure 2), aligned with the Open System Interconnection (OSI) reference model [ISO/IEC, 1996]. In contrast to the OSI model, SensorML does not further define how to use these layers. The SID model makes use of this layer stack and concretes its usage to describe the sensor interface.

Essential for the SID model is the ability to reflect the data flow between the components of the different layers, which is illus-

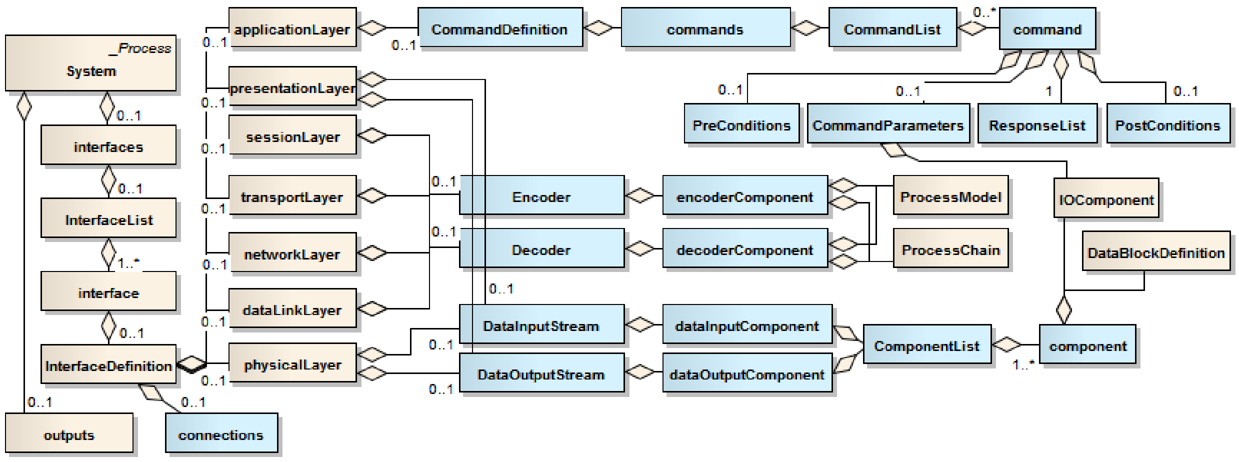

Figure 2: Overview of SID model included in SensorML. SID elements are colored in blue.

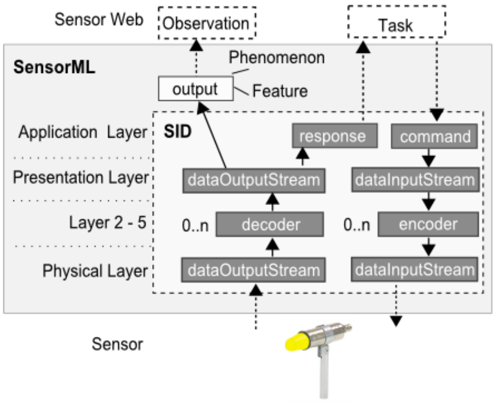

Figure 3: Data flow between sensor and Sensor Web through the SID.

trated in Figure 3. To define this data flow, we reuse the *connections* element originally associated with the SensorML *System* and associate it with the *InterfaceDefinition* (Figure 2). Describing the internal data flow in an element which is part of the SID is necessary to ensure its encapsulation.

Next, the different aspects of the SID model are described. First, we outline, how the basic addressing parameters of a physical connection to the sensor are specified (Section 3.1). After establishing the physical connection, a definition of the raw sensor protocol is needed, which is described in Section 3.2. With its definition, the sensor protocol can be interpreted and further processed (Section 3.3). Before retrieved, interpreted and processed sensor data can be forwarded to Sensor Web services, certain observation metadata has to be added, which is outlined in Section 3.4. To enable tasking of sensors, the commands accepted by the sensor interface need to be defined (Section 3.5).

3.1 Definition of Addressing Parameters

The addressing parameters (e.g. port and baud rate of a serial connection) are the basis for establishing a physical connection to the sensor. This physical connection is established through the operating system which runs the SID interpreter. The addressing parameters are stored externally in a document accompanying the SID, since the SID can be published publicly (e.g. via a SWE service) and the addressing parameters are security relevant.

```
<sml:interface name="serial_connector"
  xlink:role="urn:connection:serial">
<sml:InterfaceDefinition>
  :
```

Listing 1: Definition of addressing parameters.

As shown in Listing 1, the *name* and *role* attributes of the *interface* are used to specify identifier and type of the connection whose details are stated in an external file. The type of connection is specified by using a Unified Resource Name (URN) which points to globally defined semantics (e.g. 'urn:connection:serial' for a serial connection).

3.2 Definition of Sensor Protocol

For the declarative definition of a sensor protocol, the exact definition of the raw data streams exchanged between sensor and data acquisition system is essential. We describe the structure of this raw data within the lowest, the *physicalLayer* element. As shown in Figure 2, new elements for the data input and data output stream are attached to this element. The two elements are necessary to support duplex communication with sensors. To describe the structure of the incoming and outgoing streams the SensorML *DataBlockDefinition* type is reused.

```
Station|1275482685|33UUU 932 592|10530Q|#
Status|1275482686|2|43|72|0|#
M01|1275482698|147.0|150.0|23.0|16.3|#
:
```

Listing 2: Example data stream of MWS3 sensor.

Listing 2 shows an example of a data stream output of an MWS3 station[2]. Such stations are used in the sensor network of the German Federal Office for Radiation Protection. Besides a radiation detector, an MWS3 station carries sensors to measure atmospheric phenomena. The gateway of the MWS3 communicates either via ISDN, GSM, or DSL to a base station. Listing 3 is an excerpt of the MWS3 protocol definition. The *DataRecord* element of the *DataBlockDefinition* defines the structure and meaning of each token of a single data block within the data stream. Data blocks are separated by the '#' sign as defined in the *encoding* element. The first field within the data record specifies to which data blocks of the data stream the structure definition refers. In the example of Listing 3, the data record defines the structure for those data blocks where the first token has the value 'M01'.

[2]http://de.wikipedia.org/wiki/MWS3-Messwertsender

```
<swe:DataBlockDefinition>
  <swe:components>
    <swe:DataRecord>
      <swe:field name="datasetID">
        <swe:Text>M01</swe:Text>
       </swe:field>
      <swe:field name="time" />
      <swe:field name="radiation"/>
      <swe:field name="temperature"/>
      <swe:field name="precipitation"/>
       :
     </swe:DataRecord>
  </swe:components>
  <swe:encoding>
    <swe:TextBlock
       decimalSeparator="."
       tokenSeparator="|"
       blockSeparator="#" />
  </swe:encoding>
</swe:DataBlockDefinition>
```

Listing 3: Excerpt of MWS3 protocol definition.

3.3 Definition of Protocol Processing

For enabling the definition of processing steps which are necessary to translate between the sensor protocol and the SWE protocol, the *dataLinkLayer*, *networkLayer*, *transportLayer*, and *sessionLayer* are utilized. To allow data processing in both directions, from sensor domain to SWE domain and the other way round, elements for data decoding and encoding are added to each layer (Figure 2). Instances of these elements contain either a SensorML *ProcessModel* or *ProcessChain* to define the process. A process model can be used to describe a single non-physical process with its inputs, outputs, parameters and its computational method. A process chain can be used to represent a chain of multiple processes and to encapsulate them as one process.

Each layer element is optional. Which kinds of processes are described in which layer, depends on the design of a particular SID. An interpreter executes the processes defined in these layers sequentially. An example for a typical usage of the layers in an SID to process a data stream coming from a sensor and to encode it to SWE protocols can look like this: the data link layer specifies a process for character escaping, the network layer computes a checksum validation, the transport layer transforms the raw data to observations by applying an interpolation, and the session layer computes a date conversion.

Four process types, essential for sensor communication, shall be natively supported by an SID interpreter. These process types are described in the next subsections. The process type which shall be applied is referenced by its URN in the *method* property of the process model. Besides the four natively supported processes, other process methods can be incorporated by describing them inline using Content MathML[3].

3.3.1 Character Escaping Process Type In sensor communication, escape characters are used to induce an alternative interpretation of a transmitted character. As seen in the example of Listing 2, the end token of a data set within a data stream is indicated by a particular control character, the '#' sign. In the raw sensor data, this control character is masked by an escape character (e.g. '\'). Which characters are used for escaping is defined by the sensor protocol. Every SID interpreter needs to support this process type for removing and adding escape characters.

3.3.2 Checksum Validation Process Type For a reliable sensor communication, the computation and validation of checksums is essential. Each SID interpreter shall offer a process type for that purpose. The most widely used checksum method is the Cyclic Redundancy Check (CRC). However, there is no standard describing how to compute a CRC. Hence, the SID model supports the parameterized model for the definition of CRC algorithms, the Rocksoft Model [Williams, 1993]. The parameters of this model (e.g. polynomial, and name of the algorithm to be used) are passed along in the parameters element of this process type.

[3] http://www.w3.org/TR/MathML2

3.3.3 Interpolation Process Type Interpolations need to be computed to transform raw sensor data to observations, to compute calibrations, or to correct measurements. For example, an electric current returned by a detector needs to be transformed to an actual measurement value of a particular phenomenon (e.g. radiation or temperature). Parameters of this process type are multiple x,y-tuples defining a spline curve, as well as the kind of interpolation which shall be used (e.g. cubic, or linear).

3.3.4 Date Conversion Process Type If sensors tag their data with a timestamp, usually a conversion of the sensor time (e.g. seconds since Unix Epoch) to another time representation (e.g. ISO 8601) is necessary. This date conversion shall be natively supported by every SID interpreter. An instance of this process type is shown in Listing 4. In the parameters element of the process type, literals are used to define the input and output time formats (e.g. an 'T' Unix timestamp in seconds, and 'YYYY' for years). By specifying the URN "urn:process:dateConversion" it is indicated to the SID interpreter that this process method shall be applied.

```
<sml:ProcessModel>
   :
  <sml:parameter name="dateSettings">
    <swe:DataRecord>
      <swe:field name="inputFormat">
        <swe:Text>
          <swe:value>T</swe:value>
         </swe:Text>
       </swe:field>
      <swe:field name="outputFormat">
        <swe:Text>
          <swe:value>
            YYYY-MM-DD:HH:MM:SS
           </swe:value>
         </swe:Text>
       </swe:field>
     </swe:DataRecord>
   </sml:parameter>
   :
  <sml:method
     xlink:role="urn:process:dateConversion"/>
</sml:ProcessModel>
```

Listing 4: Excerpt of a date conversion process instance.

3.4 Definition of Observation Metadata

The data, resulting from the preceding processing steps (Section 3.3), has to be associated with certain metadata, which is part of the O&M model (Section 2), before it can be forwarded to an SOS. The measured data needs to be associated with units of measure so that an interpretation is possible. Further, the data needs to be linked to the elementary SWE components, the observed property and the feature of interest, so that observations of the O&M model can be built and inserted into an SOS.

The association of the data with a unit of measure is done on the *presentationLayer* by means of the *DataOutputStream* element as shown in Listing 5. The field 'radiation data' represents a quantity measured in 'mSv/a'.

```
<sml:presentationLayer>
  <sid:DataOutputStream>
     :
    <swe:field name="sampling time">
      <swe:Time>
        <swe:uom code="ISO8601"/>
      </swe:Time>
    </swe:field>
    <swe:field name="radiation data">
      <swe:Quantity>
        <swe:uom code="mSv/a"/>
      </swe:Quantity>
    </swe:field>
    :
  </sid:DataOutputStream>
</sml:presentationLayer>
```

Listing 5: Definition of sensor data output on presentation layer.

In the output element of the SensorML document, the sensor data is linked to the observed property and the feature of interest as shown in Listing 6. The 'radiation output' has an *ObservableProperty* pointing to the detailed description of the observed property given by a URN, and specifies in the *metaDataProperty* element a link to the feature of interest. The output element is not part of the SID, since it is not a sub-element of the *InterfaceDefinition* (Figure 2). The contained information is intentionally kept out of the SID, since the linkage of a sensor to a feature of interest and a phenomenon is dependent on the particular use case, not the interface of the sensor type. By not including this information into the SID, a reusing of the SID in different SWE deployments is possible.

```
<sml:output name="radiation output">
  <swe:ObservableProperty
     definition="urn:phenomenon:radiation">
    <gml:metaDataProperty
       xlink:href="http://myServer.org/features/Muenster"/>
  </swe:ObservableProperty>
</sml:output>
```

Listing 6: Definition of observation metadata in output element.

3.5 Definition of Sensor Commands

The application layer of the OSI model describes interfaces to access the OSI stack. Compliant to this view, the *applicationLayer* is used here to define the commands accepted by the sensor. These command definitions can be used by an SPS so that it can provide information to the clients how to task the sensor. As shown in Figure 2, the *command* element contains sub-elements to describe possible sensor responses, the pre- and postconditions for executing the command, as well as the command parameters. Listing 7 shows an example of the parameter definition of a command which sets the sampling rate of a sensor. The command has three parameters: the first one is fixed to the value 'SR', the second one is a text representing the sensor ID, and the third one is the measuring interval, with a minimum value of 5 seconds. The SID model prescribes specific URNs to define *roles* of parameters, for example whether its required or optional. The order of the command parameters is the same as in the sensor protocol to which the command is mapped by the processes defined in the lower layers (Section 3.3).

```
<swe:Parameters>
  <swe:DataRecord>
    <swe:field name="cmd"
       xlink:role="urn:command:name">
      <swe:Text>
        <swe:value>SR</swe:value>
      </swe:Text>
    </swe:field>
    <swe:field name="sensorID"
       xlink:role="urn:command:parameter:required">
      <swe:Text/>
    </swe:field>
    <swe:field name="interval"
       xlink:role="urn:command:parameter:optional">
      <swe:Quantity>
        <swe:uom code="sec" />
        <swe:constraint>
          <swe:AllowedValues>
            <swe:min>5</swe:min>
          </swe:AllowedValues>
        </swe:constraint>
      </swe:Quantity>
    </swe:field>
    :
  </swe:DataRecord>
</swe:Parameters>
```

Listing 7: Example of a command definition to set the sampling rate.

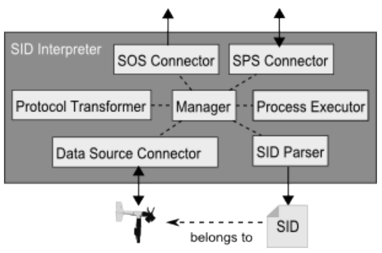

Figure 4: Design of the SID Interpreter.

4 INTERPRETER IMPLEMENTATION

The implementation of our SID interpreter is based on the OSGi framework[4] which is extendible by pluggable and loosely coupled components (Figure 4). A central *Manager* component controls the workflow. First, the *SID Parser* is used to read in the SID document of the sensor. Depending on the specified addressing parameters (Section 3.1), a particular *Data Source Connector* implementation (e.g. for USB connections) is chosen to connect to the sensor. Based on the protocol definition of the SID (Section 3.2), the *Protocol Transformer* communicates with the sensor in a bi-directional way. The *Process Executor* is able to execute the four native process methods (Section 3.3). Also, user-defined MathML processes can be executed by the means of the MathML Solver library[5]. The *SOS Connector* triggers the SOS operation *RegisterSensor* to add the new sensor to the Sensor Web and executes the *InsertObservation* operation to upload sensor data as observations (Section 3.4) to an SOS. The *SPS Connector* forwards the SensorML document and the contained SID to an SPS which uses the sensor command descriptions (Section 3.5) to provide detailed information how to task the sensor. Sensor tasks, submitted to the SPS, are forwarded by the SPS to the *SPS Connector*. The tasks are transformed to the sensor protocol, and passed through the *Data Source Connector* to the sensor.

5 CONCLUSIONS & FUTURE WORK

In this paper, we outline the need for mechanisms to close the interoperability gap between sensors and the Sensor Web. We bridge this gap by introducing a model based on the SensorML standard which enables the declarative description of sensor interfaces. Based on the SID model, interpreters can be built, which are independent of particular sensor technology, and are able to automatically generate communication logic to connect sensors to the Sensor Web - a next step on our long term research agenda of realizing sensor plug & play.

We presented our SID interpreter implementation which integrates sensors described by an SID with a Sensor Observation Service as well as a Sensor Planning Service. These services enable access to measured sensor data and allow tasking of a sensor in an interoperable way. We illustrated the design of the developed model

[4]http://www.osgi.org/
[5]http://sourceforge.net/projects/mathmlsolver/

by an example, the SID of an MWS3 radiation detector. We will contribute our model to the current development of the SensorML 2.0 standard. The SID model, the MWS3 example, as well as our SID interpreter implementation are published[6] as open source at $52^\circ North$.

SIDs are a basis for minimizing the efforts of integrating new sensors with the Sensor Web. Currently, we are developing tools and user interfaces to support the creation of SIDs. This will support sensor network administrators to make their sensors available on the Sensor Web. Future catalogs of SID instances will allow users to share and reuse the interface descriptions of their sensors. A further step towards realizing sensor plug & play for the Sensor Web will be the incorporation of the SID interpreter into our publish/subscribe architecture, the Sensor Bus [Broering et al., 2010; forthcoming], underlying the Sensor web and enabling an on-the-fly integration of sensors. Based on these developments, identified semantic challenges in the context of sensor plug & play [Broering et al., 2009] will be tackled. Further, the SID interpreter will be extended to not only support the connection of sensors to SWE services, but also to connect to other Sensor Web implementations (e.g. Pachube[7], or Sensorpedia[8]). Finally, the approach will be applied in real-world scenarios to demonstrate its benefits in sensor asset management. As the project develops, we anticipate that both our design and our research agenda will evolve as new issues and opportunities arise.

ACKNOWLEDGMENT

This work is financially supported by the project ”Flexible and Efficient Integration of Sensors and Sensor Web Services” funded by the ERDF program (grant N 114/2008) of the European Union, as well as the $52^\circ North$ Sensor Web community which envisions a real time integration of heterogeneous sensors into a coherent information infrastructure.

REFERENCES

Aberer, K., Hauswirth, M. and Salehi, A., 2006. A middleware for fast and flexible sensor network deployment. In: Proceedings of the 32nd international conference on Very large data bases.

Bleier, T., Bozic, B., Bumerl-Lexa, R., Da Costa, A., Costes, S., Iosifescu, I., Martin, O., Frysinger, S., Havlik, D., Hilbring, D., Jacques, P., Klopfer, M., Kunz, S., Kutschera, P., Lidstone, M., Middleton, S., Roberts, Z., Sabeur, Z., Schabauer, J., Schlobinski, S., Shu, T., Simonis, I., Stevenot, B., Usländer, T., Watson, K. and Wittamore, K., 2009. SANY - An Open Service Architecture for Sensor Networks. SANY Consortium.

Botts, M., 2007. OGC Implementation Specification 07-000: OpenGIS Sensor Model Language (SensorML). Open Geospatial Consortium.

Botts, M., Percivall, G., Reed, C. and Davidson, J., 2008. OGC Sensor Web Enablement: Overview and High Level Architecture. Lecture Notes In Computer Science 4540, pp. 175–190.

Broering, A., Foerster, T., Jirka, S. and Priess, C., 2010; forthcoming. Sensor Bus: An Intermediary Layer for Linking Geosensor Networks and the Sensor Web. In: COM.Geo 2010: 1st International Conference on Computing for Geospatial Research and Applications, Washington DC, USA.

Broering, A., Janowicz, K., Stasch, C. and Kuhn, W., 2009. Semantic Challenges for Sensor Plug and Play. In: J. Carswell, S. Fotheringham and G. McArdle (eds), Web & Wireless Geographical Information Systems (W2GIS 2009), 7 & 8 December 2009, Maynooth, Ireland, LNCS, Springer, pp. 72–86.

Chung, L.-K., Baranski, B., Fang, Y.-M., Chang, Y.-H., Chou, T.-Y. and Lee, B. J., 2009. A SOA based debris flow monitoring system - Architecture and proof-of-concept implementation. In: The 17th International Conference on Geoinformatics 2009, Fairfax, USA.

Cox, S., 2007. OGC Implementation Specification 07-022r1: Observations and Measurements - Part 1 - Observation schema. Open Geospatial Consortium.

Gigan, G. and Atkinson, I., 2007. Sensor Abstraction Layer: A Unique Software Interface to Effectively Manage Sensor Networks. In: 3rd International Conference on Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007, pp. 479–484.

Gong, J., Wu, H., Gao, W., Yue, P. and Zhu, X., 2010. Geospatial Service Web. In: D. Li, J. Shan and J. Gong (eds), Geospatial Technology for Earth Observation, Springer, pp. 355–379.

Hu, P., Robinson, R. and Indulska, J., 2007. Sensor Standards: Overview and Experiences. In: Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing ISSNIP'07, Melbourne, Australia.

ISO/IEC, 1996. ISO/IEC 7498-1: Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. ISO.

Jirka, S., Broering, A. and Stasch, C., 2009. Applying OGC Sensor Web Enablement to Risk Monitoring and Disaster Management. In: GSDI 11 World Conference, Rotterdam, Netherlands.

Na, A. and Priest, M., 2007. OGC Implementation Specification 06-009r6: OpenGIS Sensor Observation Service (SOS). Open Geospatial Consortium.

Nittel, S., Labrinidis, A. and Stefanidis, A., 2008. Introduction to advances in geosensor networks. Lecture Notes in Computer Science 4540, pp. 1–6.

Schimak, G. and Havlik, D., 2009. Sensors Anywhere - Sensor Web Enablement in Risk Management Applications. ERCIM News 2009 - The Sensor Web (76), pp. 40 – 41.

Simonis, I., 2007. OGC Implementation Specification 07-014r3: OpenGIS Sensor Planning Service. Open Geospatial Consortium.

Stasch, C., Walkowski, A. C. and Jirka, S., 2008. A Geosensor Network Architecture for Disaster Management based on Open Standards. In: M. Ehlers, K. Behncke, F. W. Gerstengabe, F. Hillen, L. Koppers, L. Stroink and J. Wächter (eds), Digital Earth Summit on Geoinformatics 2008: Tools for Climate Change Research., pp. 54–59.

Walter, K. and Nash, E., 2009. Coupling Wireless Sensor Networks and the Sensor Observation Service – Bridging the Interoperability Gap. In: 12th AGILE International Conference on Geographic Information Science 2009, Hannover, Germany.

Williams, R., 1993. A Painless Guide to CRC Error Detection Algorithms. Technical report, Rocksoft Ltd., Hazelwood Park, Australia.

[6] http://52north.org/sid

[7] http://www.pachube.com/

[8] http://www.sensorpedia.org/