Oceans of Tomorrow Sensor Interoperability for In-situ Ocean Monitoring

Jay Pearlman, IEEE
Piscataway, NJ USA

Eric Delory,
PLOCAN, Telde, Spain

Simon Jirka,
52°North GmbH, Muenster, Germany

Sergio Martinez,
Leitat, Terrassa, Spain

Joaquin del Rio,
UPC, VIlanova i la Geltru, Spain

Tom O'Reilly,
MBARI, Moss Landing, CA USA

Abstract— The Oceans of Tomorrow (OoT) projects, funded by the European Commission's FP7 program, are developing a new generation of sensors supporting physical, biogeochemical and biological oceanographic monitoring. The sensors range from acoustic to optical fluorometers to labs on a chip. The result is that the outputs are diverse in a variety of formats and communication methodologies. The interfaces with platforms such as floats, gliders and cable observatories are each different. Thus, sensor "drivers" have to be built for each kind of sensor interface, which leads to extensive efforts in developing large-scale systems and additional fielding cost. Since the price of sensor devices is expected to decrease rapidly, these adaption efforts become the key cost factor in large-scale sensor network systems. At the other end, from a system perspective, the data transmission and visualization must be individually tailored. When multi-sensors are making measurements, the challenge of interoperability is compounded.

The Oceans of Tomorrow projects are addressing interoperability in the complete data flow from sensor to user. Selected innovations at the sensor end are through implementation of the OGC PUCK protocol [PUCK web reference, O'Reilly 2006]. PUCK provides a protocol and "container" to store instrument-related information ("payload") with the instrument itself. An observing system retrieves and utilizes information through instrument's serial interface. PUCK protocol was developed by MBARI and its use is expanding [Toma 2014].

For the information flow from platform to user, an approach is the use of Sensor Observation Service (SOS), which acts as a common interface to observation data stores. For transmitting collected data from platforms to such an observation data store, the transactional and resultant handling operations of the SOS interface are used which allow the insertion/publication of measurements as well as corresponding metadata. Common SWE templates and profiles (e.g. for OGC SensorML, OGC Observations and Measurements and OGC SOS) are being used. These will be comprehensive data flow descriptions for the Oceans of Tomorrow sensors and are being created through the OoT projects in order to increase interoperability.

In addition to information flow described above, further work is necessary for a common visualization and means of sharing data that can support multiple sensor types and enable overlay of observation data. The visualization will need to be supported by data transformation capabilities such as the GEOSS DAB or ERDAPP when the information used is from an array of sensors and platforms. This presentation will consider necessary standards and best practices to extend the current implementations of sensors in an ocean observation environment. The directions and recommendations will be presented in the paper.

Keywords—in situ sensors; oceans of tomorrow; SWE; sensor web; PUCK, SEISI

I. INTRODUCTION

The 'Ocean of Tomorrow' initiative aims to foster multidisciplinary approaches and cross-fertilization between various scientific disciplines and economic sectors on key crosscutting marine and maritime challenges. Oceans of Tomorrow 2013 addressed in-situ sensor development. Nine projects were funded for sensors. They are grouped into two topics: (topic 1) Innovative biosensor designs in response to demand for real-time monitoring of marine water quality and the provision of early warning systems; and (topic 2) Cost-effective sensors to increase the amount of data for marine observation, modeling and monitoring systems. The topic 1 biosensor projects are:

BRAAVOO: Biosensors, Reporters and Algal Autonomous Vessels for Ocean Operation

ENVIGUARD: Development of a biosensor technology for environmental monitoring and disease prevention in aquaculture ensuring food safety

MARIABOX: Marine environmental in-situ Assessment and monitoring tool BOX

SEA-ON-A-CHIP: Real time monitoring of SEA contaminants by an autonomous Lab-on-a chip biosensor

SMS: Sensing toxicants in Marine waters makes Sense using biosensors

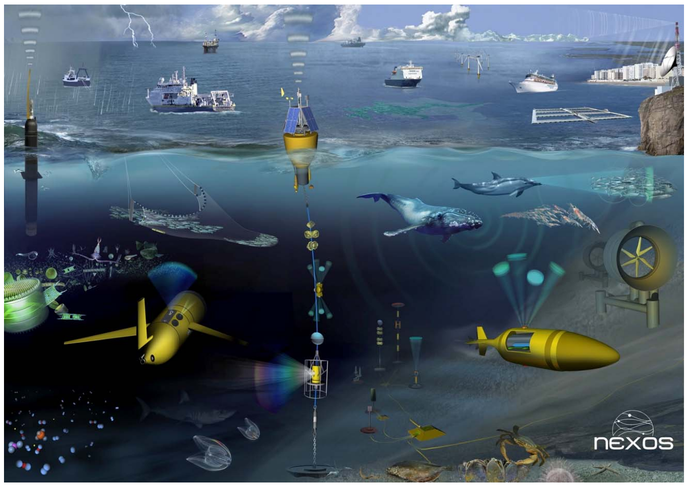

Figure 1: Ocean monitoring environment

Topic 2 projects are more comprehensive and address sensors, system level integration and field demonstrations. The four projects include:

COMMON SENSE project provides cost-effective sensors to quantify new emergent pollutants (Eutrophication, microplastics, Heavy metals and Underwater noise (including noise recognition)), and transversals sensors (pH, pCO2, temperature and pressure), to be used as measurement reference. Furthermore a Mini Sea Sampling system (MISS) has been developed to operate with the different sensors. The particularity is that this system includes amount of sensors such as conductivity, temperature, pressure, salinity, dissolved oxygen, pH, Turbidity, Chl(a) and PhycoErithrin.A Common Sensor Web Platform supports the data flow from sensor to user.

NeXOS is developing cost-effective, innovative and compact integrated "plug and play" multifunctional sensor systems deployable from mobile and fixed ocean platforms. Key measurement parameters acquired by the sensors are useful for a number of objectives, ranging from more precise monitoring and modeling of the marine environment to an improved assessment of fisheries. Seven new compact, cost-efficient sensors are being developed, based on optical and acoustics technologies, addressing a majority of descriptors identified by the Marine Strategy Framework Directive for Good Environmental Status. In addition to the above, crosscutting technologies such as biofouling prevention capabilities and the application of a smart sensor interface with Web enabled components and are being developed to enhance the utility and cost-effectiveness of the sensors system.

SCHeMA is providing an open and modular sensing solution for *in situ* high resolution mapping of a range of anthropogenic and natural chemical compounds that may have feedback (synergic) interaction: toxic and/or essential Hg, Cd, Pb, As and Cu trace metal species; nitrate, nitrite, and phosphate nutrients; species relevant to the carbon cycle; volatile organic compounds; potentially toxic algae species and toxins. The SCHeMA system will consist of a plug-and-play adaptive wired/wireless chemical sensor probe network serving as a front-end for gathering detailed spatial and temporal information on water quality and status based on a range of hazardous compounds.

SenseOcean will innovate and combine state of the art sensor technologies (microfabrication, lab on chip, micro and calibration free electrochemical sensors, multiparameter optodes and multispectral optical sensors) in a modular and configurable system easily usable across multiple ocean and environmental platforms. Pre-competitive prototypes will be optimized for scale up and commercialization including preparation of data flow and data management architectures. These will be tested and demonstrated on profiling floats, deep-sea observatories, autonomous underwater vehicles, and fishing vessels. Specific objectives are the development of integrated systems with sensors for pH, carbon dioxide, carbon, alkalinity, oxygen, nutrients, metals (iron and manganese) but also for colored dissolved organic matter, chlorophylls, photopigments, primary production, organic fluorophores, etc. These integrated systems will be optimized for power consumption, chemical usage and waste production. They will be resistant to biofouling to facilitate long-term deployment in the marine environment. The resulting systems will be developed to provide near-market systems; these will then be launched as commercially available products.

The types of measurements for the four projects are summarized in following Table.

Project	Common Sense	NeXOS	SCHeMA	Sense Ocean
Temp, Pressure	X		X	
Oxygen			X	X
CDOM		X		X
Nutrients/carbonates	X		X	X
Phytoplankton		X		X
Hydrocarbons		X		X
Carbon Cycle		X		X
Fisheries		X		
Underwater noise	X	X		
microplastics	X			
Heavy/trace metals	X		X	

The sensors range from acoustic to optical fluorometers to labs on a chip. The result is that the outputs are diverse in a variety of formats and communication methodologies. The interfaces with platforms such as floats, gliders and cable observatories are each different. Thus, sensor "drivers" have to be built for each kind of sensor interface, which leads to extensive efforts in developing large-scale systems and additional fielding cost. Since the price of sensor devices is expected to decrease rapidly, these adaption efforts for sensor fielding become the key cost factor in large-scale sensor network systems. At the other end, from a system perspective, there will be cases where the data transmission and visualization must be individually tailored. When multi-sensors are making measurements, the challenge of interoperability is compounded.

The projects in Topic 2is for thus need to address the data flow from sensor to user. The projects are Sensor Web Enabled (SWE), which is an OGC standard for information flow and control between sensor and user. The SWE has component standards for its various functionalities. These include the following standards, which are considered in this work:

- ISO/OGC Observations and Measurements (O&M) 2.0 (Cox, 2011): Modelling and encoding of measurement data.
- OGC Sensor Model Language (SensorML) 2.0 (Botts et al., 2014): Modelling and encoding of sensor metadata (see above).
- OGC Sensor Observation Service (SOS) 2.0 (Bröring et al., 2012): Web service interface for pull based access to observation data and the corresponding metadata.
- OGC Sensor Planning Service (SPS) 2.0 (Simonis and Echterhoff, 2011): Controlling and configuring sensors through a Web service interface.

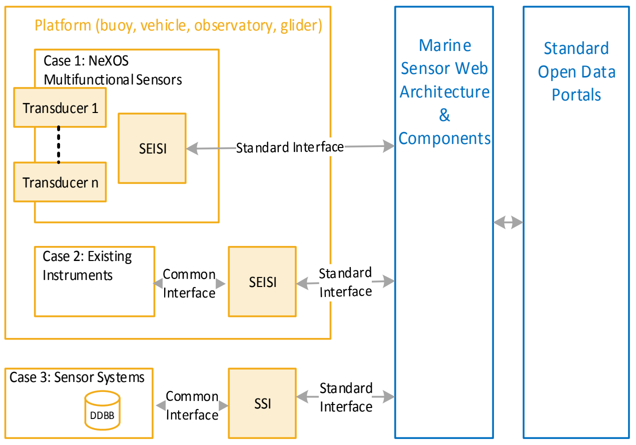
Platform (buoy, vehicle, observatory, glider)
Case 1: NeXOS Multifunctional Sensors
Transducer 1
Transducer n
SEISI
Standard Interface
Case 2: Existing Instruments
Common Interface
SEISI
Standard Interface
Case 3: Sensor Systems
DDBB
Common Interface
SSI
Standard Interface
Marine Sensor Web Architecture & Components
Standard Open Data Portals

Figure 2: Smart Sensor Interface for Sensors and Instruments (SEISI)

A typical configuration (from NeXOS) is given in Figure 2 that uses the Smart Sensor Interface for Sensors and Instruments (SEISI). Three cases are shown: case 1 is a new generation of sensor with multiple sensor arrays; case 2 is an existing sensor which uses SEISI as an interface with the sensor web; case 3 is for sensor systems with traditional interfaces. The information flow is given in Figure 3.

To address the information flow from platform to user, an approach is the use of Sensor Observation Service (SOS), which acts as a common interface to observation data stores (see Figure 3). For transmitting collected data from platforms to such an observation data store, the transactional and result handling operations of the SOS interface are used which allow the insertion/publication of measurements as well as corresponding metadata. Common SWE templates and profiles (e.g. for OGC SensorML, OGC Observations and Measurements and OGC SOS) are under development.

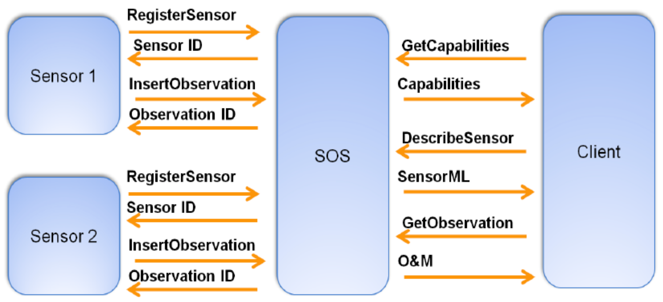

Figure 2: Information exchange in the Sensor Web enabled system

II. Sensor Descriptions – SensorML

An important foundation of the Sensor Web applications described in this paper is the standardized provision of sensor metadata. For example, the provision of comprehensive metadata is essential for enabling the discovery of sensors and their measurement data (Jirka et al., 2009), the interpretation of data (e.g. assessing precision and reliability), and the management of observatories and their components (e.g. documentation of events such as maintenance and calibration). Furthermore, sensor metadata describing the interface of an instrument helps to build automated plug-and-play mechanisms that dynamically generate drivers to access and control the described sensors.

The Sensor Model Language (SensorML) 2.0 standard (Botts et al., 2014) of the Open Geospatial Consortium (OGC) is an ideal basis for modelling and encoding such sensor metadata. However, as SensorML is intentionally designed in a domain independent manner (it can be applied not only to marine sensors but for example also to satellites, drones, weather stations, and air quality sensors), the specification offers a large degree of flexibility. While this flexibility is crucial for supporting a broad range of thematic domains, it may reduce interoperability within a specific domain. Thus, a core objective of the work presented in this paper is the definition of a SensorML profile for marine applications to ensure a high level of interoperability.

Figure 4 illustrates the different elements that are covered by the proposed marine SensorML profile. A network may comprise multiple platforms such as gliders or buoys. Each of these platforms may contain one or more instruments. Finally, each instrument could consist of a number of sensors. It is important to note that not all of these levels of metadata may be necessary for all applications. Thus, it is also possible to use only selected elements of this model in a certain use case. For each of these elements, the marine SensorML profile includes a subset of required/recommended fields of metadata that shall be provided (e.g. keywords, classifiers, contact information, configuration parameters, etc.). These syntactic definitions are complemented by ongoing activities to define a set of common terms in vocabularies that ensure semantic interoperability, as well.

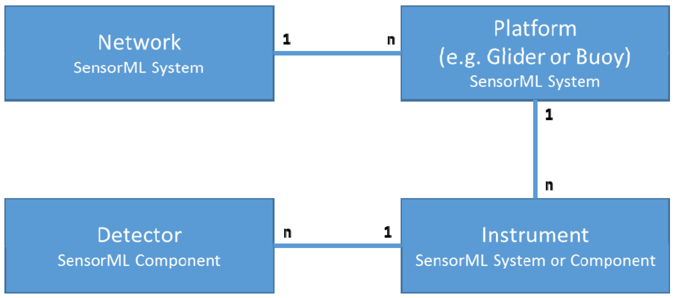

Figure 3: Overview of SensorML Model for Marine Applications

III. Instrument Protocol Description

The SensorML can also provide a standard description of the instrument's command protocol, e.g. which command should be issued to begin acquiring data, configuration commands, etc. When the instrument is attached to a 'host' computer aboard a vehicle, buoy or other platform, the host use the protocol description to operate the instrument. The host may have the SensorML installed on board, might download it through a network connection, or could retrieve it from the instrument itself with OGC PUCK protocol.

OGC PUCK describes a standard serial protocol to retrieve information about an instrument from the device itself through the instrument's RS232 or Ethernet port. A 'host' computer aboard a vehicle, buoy or other platform can interrogate its ports to detect when a PUCK-enabled instrument is attached, then retrieve and utilize the instrument description to run driver software that acquires the instrument's data and processes it. These standard features enable automated instrument integration; a human operator can merely plug the instrument into any available port without needing to manually install the driver or associate it with a particular port.

OGC PUCK specifies two accessible memory areas within the instrument. The 96-byte instrument datasheet includes a universally unique identifier (UUID) that is guaranteed to be unique among all PUCK-enabled instruments, as well as manufacturer and model codes. Thus a host can determine exactly what kind of instrument it is dealing with by reading the instrument datasheet. Optional PUCK payload memory can store additional information useful in operating the instrument, e.g. SensorML documents, and the host can retrieve this information as well. The standard protocol specifies instrument commands to read the datasheet, and to write as well as read the payload

The SensorML file provides to the platform all the necessary information about the instrument in order to communicate. This file will be used also as a configuration file to configure the way that the platform interact with the instrument and if required, how the platform is going to export the data to the user. In the Appendix, an example with some small fragments of a real SensorML file is described in order to make it more understandable. The file describes how a commercial instrument (RBR XR-420 CTD) is managed by a platform.

Figure 5 shows the full process between the platform (labeled as Smart Buoy) and an instrument (labeled in this case as an Hydrophone). What was explained before is related to the "Autoconf" and "Scheduler" actions inside the SWEBridge.

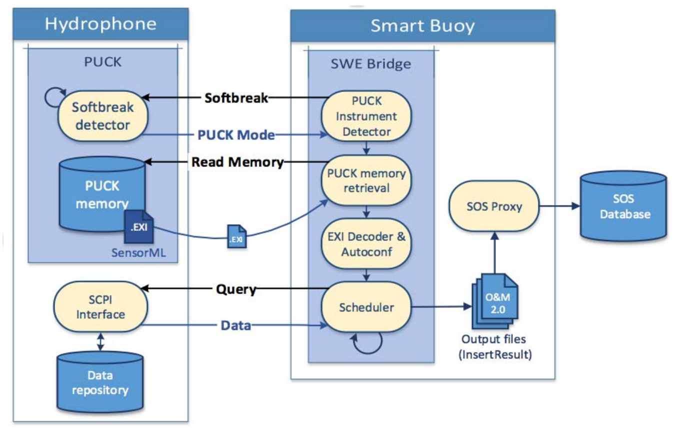

Figure 4: Data and informaiton flow processes for an integrated sensor and platform

IV. SWE Descriptions, SOS and Applications to OoT

The OGC Sensor Web Enablement (SWE) framework serves as baseline for the development of the interoperable ocean observing systems described in this paper (Jirka et al., 2014). This framework of standards offers a mechanism to extend spatial data infrastructures with capabilities to handle sensor and the observation data sets these sensors are delivering. Especially the following standards are considered in this work:

- ISO/OGC Observations and Measurements (O&M) 2.0 (Cox, 2011): Modelling and encoding of measurement data in XML. This standard offers a common data format for transferring observation data between data providers and consumers.
- OGC Sensor Model Language (SensorML) 2.0 (Botts et al., 2014): Modelling and encoding of sensor metadata (see above). While O&M is a data format for the observations performed by a

sensor, SensorML is a data format for exchanging the descriptions of the sensors and procedures that were used for generating the observation data.

- OGC Sensor Observation Service (SOS) 2.0 (Bröring et al., 2012): Web service interface for pull-based access to observation data and the corresponding metadata. Using this interface specification, data consumers have a common way to request observation data via the WWW from a data repository. The SOS interface allows users to specify in which data they are interested (e.g. spatial filters, temporal filters, filtering by the observed property or a certain observatory, etc.). Subsequently the SOS returns an O&M based response containing the requested data.
- OGC Sensor Planning Service (SPS) 2.0 (Simonis and Echterhoff, 2011): Controlling and configuring sensors through a Web service interface. This is a complementary interface standard allow to control measurement processes and sensors/platforms. Examples comprise the submission of a track that shall be monitored by an underwater vehicle or the adjustment of the sampling rate of a sensor on a buoy.

Besides these core standards, additional specifications such as the OGC SWE Common Service and Data Models are used. These standards define common principles that apply across all of the SWE standards.

V. User Access and Visualization

After sensor observations are accessible through the interoperable Sensor Web interfaces (i.e. OGC SOS), they can easily be integrated into user applications. A typical use case is the visual analysis of the collected data. One example for such a data visualisation tool is the 52°North Sensor Web Viewer "Helgoland"[1] developed within the NeXOS project. This client application can be connected to different SOS servers to explore the available observation data sets of multiple providers. Users may use a map view as well as tabular data selection menus to choose the data that shall be visualized. There are different views for displaying the data (diagram and table) as well as export functions to download the data in formats such as CSV. A central capability of this application is the combined visualisation of data from different sources covering various parameters (see Figure 6) which displays the chlorophyll concentration measured by a test deployment in Germany and salinity and sea water temperature measured by the OBSEA observatory in Spain). This way it is possible to compare different measurements of to discover correlations between different time series.

[1] https://github.com/52North/helgoland

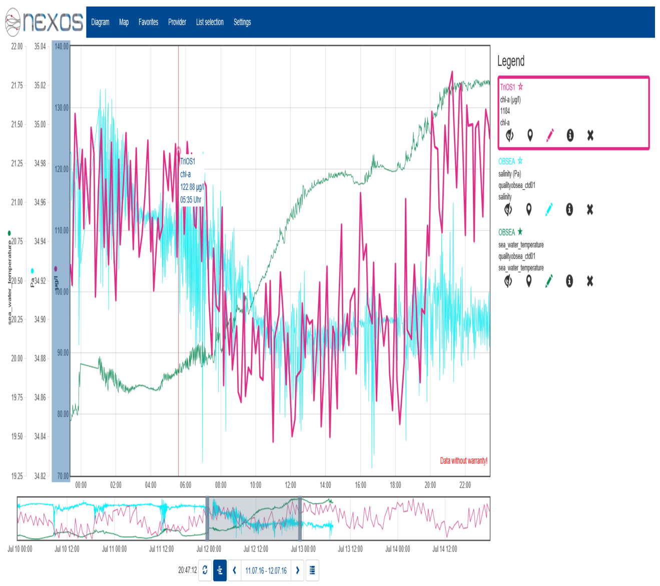

Figure 6: Sensor Web Viewer developed within the NeXOS Project

VI. Future Directions and Applications

Within the Ocean of Tomorrow projects, the discussion on dedicated marine profiles of the SWE standards has led to first results that are collected in a Wiki[2]. This comprises, for example, the collection of SWE usage examples from different projects, an ongoing analysis of commonalities and differences, as well as a list of necessary vocabulary content to achieve semantic interoperability between projects. The information is complemented by the collection of available SOS servers operated by the involved projects and a viewer implementation, which will help to discover and display the content of these servers. However, it is necessary to continue this work further, to advance the currently informally described agreements into officially adopted best practices or even standards.
Another aspect for continuing work concerns the realisation of push-based (real-time) data delivery mechanisms. While a robust data publication mechanism based on the OGC SOS interface is in place, there are several use cases, which would benefit from the delivery of data as soon as a new measurement is available. Such data delivery mechanisms can be supported by the new OGC PubSub standard[3]. Although there are successful initial implementations within the NeXOS project, further work is necessary to build robust operational components that can be flexibly applied and integrated into existing infrastructures.

[2] Currently this Wiki is not yet publically accessible. However, everyone who is interested in this work is invited to contact the authors of this paper to obtain access to this Wiki,

[3] http://www.opengeospatial.org/projects/groups/pubsubswg

VII. Appendix

The SensorML allows defining different type of information:
- *Ouputs* in order to declare the magnitudes offered by the instrument:

```
                    <sml:outputs>
                              <sml:OutputList>
                     <!-- observed physical magnitudes -
                          ->
                       <sml:output name="conductivity">
                              <sml:ObservableProperty
           definition="conductivity"/>
                     </sml:output>
```

- Parameters for general purpose information like physical interface, baud rate among others:

```
                        <sml:parameters>
                                <sml:ParameterList>
                         <!-- ============================= -->
                         <!--     Instrument Interface      -->
                         <!-- ============================= -->
                          <sml:parameter name="dataInterface">
                            <sml:DataInterface id="SB37_RS232">
                                          <sml:data/>
                                   <sml:interfaceParameters>
                                         <swe:DataRecord>
                                       <!-- Interface type -->
                                     <swe:field name="portType">
                                             <swe:Category>
                                               <swe:label>Port
                        Type</swe:label>

                  <swe:value>RS232</swe:value>
                                             </swe:Category>
                                            </swe:field>
                                      <!-- Interface number -->
                                    <swe:field name="portNumber">
                                               <swe:Count>
                                               <swe:label>Port
                       Number</swe:label>
                                                </swe:Count>
                                            </swe:field>
                                          <!-- Baud Rate -->
                                     <swe:field name="baudRate">
                                               <swe:Count>
                                               <swe:label>Baud
                        Rate</swe:label>

                   <swe:value>9600</swe:value>
                                                </swe:Count>
```

- Processes in order to define how to get an observation from the instrument:

```
                         <!-- ============================= -->
                         <!--    Get Observation Process    -->
                         <!-- ============================= -->
                                   <sml:SimpleProcess
                gml:id="getObservation">
                                       <gml:identifier
codeSpace="uid">urn:SARTI:swe:instrument:RBR_XR420:getObservation</gm
                         l:identifier>
                                  <gml:name>getObservation
                       process</gml:name>
                                        <sml:inputs>
                                         <sml:InputList>
                                    <!--  ========= Instrument
                      Command ========= -->
                                     <sml:input name="dataIn">
                                          <sml:DataInterface>
                                                <sml:data>
                                               <swe:DataStream>

               <swe:elementType name="command"/>

                        <swe:encoding>
                                                                <!--
                 Define the text encoding -->

              <swe:TextEncoding tokenSeparator=" "
               blockSeparator="&#x000A;&#x000D;"/>

                        </swe:encoding>
                                                           <!-- Define
                    the command string -->

                 <swe:values>FOO</swe:values>
```

and how the instrument responses to a "GetObservation" process:

```
                          <!-- ========= Instrument Response
                    ========= -->
                                <sml:output name="dataOut">
                                      <sml:DataInterface>
                                             <sml:data>
                                           <swe:DataStream>

                    <swe:elementType name="response">

     <swe:DataRecord>


                 <swe:field name="conductivity"></swe:field>

                  <swe:field name="temperature"></swe:field>

                     <swe:field name="depth"></swe:field>

                              </swe:DataRecord>

                             </swe:elementType>
                                                          <!-- Define
                 the response text encoding -->

                                <swe:encoding>

        <swe:TextEncoding tokenSeparator="," blockSeparator="."/>
```

In addition, we can use the SensorML file to describe the actions that the platform will execute with the data. We call "SWEBridge" to the application that will be running on the platform, and will manage the information coming from the sensors and will make it available to the user using and SOS: for example: once the platform retrieve data from the sensor, the platform will encapsulate the data into an InsertResult transactional operation to be pushed into an SOS server. These operations are defined using "AggregateProcess" into the SensorML file, and afterwards creating links between "sources" and "destinations", for example:
Here we can a see a code snip of the SensorML file where the AggreateProcess is declared. The SWEBridge (code running into the platform) include inputs (data coming from the instrument) and outputs (where the SWEBridge will send the data).

```
                  <sml:component name="ConfigureBridge">
                 <sml:AggregateProcess gml:id="SWEBridge_process_01">
                          <!-- ============================= -->
                          <!--       General Information     -->
                          <!-- ============================= -->
                                     <gml:description>
                               A process that configures the acquisition process
               executed by the SWEBridge application inside
                                the host platform and the generation of standard
                   O&amp;M output transactional files.
                                     </gml:description>
                          <!-- aggregate process identifier -->
                                       <gml:identifier
codeSpace="uid">urn:SARTI:swe:process:SWEBridge_process_01</gml:ident
                               ifier>
                         <gml:name>SWEBridge process</gml:name>

                          <!-- ============================= -->
                              <!--    SWE Bridge Inputs    -->
                          <!-- ============================= -->
                                      <sml:inputs>
                                       <sml:InputList>
                            <!-- each input is the instrument's response
                        to a specific command -->
                                <sml:input name="dataIn"></sml:input>
                                       </sml:InputList>
                                     </sml:inputs>

                          <!-- ============================= -->
                              <!--    SWE Bridge Outputs   -->
                          <!-- ============================= -->
                                     <sml:outputs>
                                      <sml:OutputList>
                              <!-- each output is the generation of a
    formatted file or a transaction for high-level components such as
                           SOS server, Zabbix...-->
                                  <sml:output name="dataOut">
```

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

Also different “Components“ are defined. Each component will be a process to be executed by the SWEBridge:
For a GetObservatorion:

```
                <sml:components>
                    <sml:ComponentList>
                <!-- ================================== -->
                  <!--  Each component is a process to be
        exectued, i.e. getObservation CTD is a command
               that will be sent to the instrument, afterwards the
                  intrument's output for this
                                 command -->
                <!-- ================================== -->
                <sml:component name="getObservation_CTD">
                              <sml:SimpleProcess
          gml:id="getObservation_01">
                       <!-- ============================= --
                         >
                       <!--       General Information     --
                         >
                       <!-- ============================= --
                         >
                              <gml:description >
                              Returns the last observation of
                       the system
                              </gml:description>
                       <!-- selector process identifier -->
                              <gml:name>CTD getObservation
                       process</gml:name>
                         <!-- Define the simpleProcess as a
               getObsevation command process -->
                              <sml:typeOf
        xlink:title="urn:RBR:XR420_CTD:getObservation"

       xlink:href="http://cdsarti.org/RBR-XR-420-CTDs"/>
                              <sml:configuration>
                              <sml:AbstractSettings>
                                 <swe:extension>
                                 <!-- set the sampling
                  rate to 10 s -->
                                    <sml:setValue
       ref="parameters/samplingRate">10.0</sml:setValue>
                                 <!-- set the timeout to 2
                      s -->
                                    <sml:setValue
         ref="parameters/timeOut">2.0</sml:setValue>
                                 </swe:extension>
                              </sml:AbstractSettings>
                              </sml:configuration>
                           </sml:SimpleProcess>
                           </sml:component>
```

For selecting which magnitudes will be processed:

```
          <sml:component name="fieldSelector_configured">
                              <sml:SimpleProcess
          gml:id="fieldSelector_02"

      definition="http://cdsarti.org/SWEBridge_selector_01">
                              <gml:description >
                              Selects a number of the input
                       fields for output.
                              </gml:description>
                       <!-- selector process identifier -->
                              <gml:identifier
codeSpace="uid">urn:SARTI:swe:process:fieldSelector_01</gml:identifie
                       r>
                              <gml:name>SWEBridge field selector
                       process</gml:name>
                              <sml:typeOf
      xlink:href="http://cdsarti.org/SWEBridge_selector"/>
                              <sml:configuration>
                              <sml:AbstractSettings>
                                 <swe:extension>
                                    <sml:setValue
      ref="outputs/dataOut/temperature">enable</sml:setValue>
                                    <sml:setValue
     ref="outputs/dataOut/conductivity">disable</sml:setValue>
                                    <sml:setValue
       ref="outputs/dataOut/depth">enable</sml:setValue>
                                 </swe:extension>
                              </sml:AbstractSettings>
                              </sml:configuration>
                           </sml:SimpleProcess>
                           </sml:component>
```

A component who declares that the SWEBridge will be able to send data to an SOS through an InsertResult operation

```
                        <sml:component name="insertResult">
                        <sml:SimpleProcess gml:id="insertResult">
                          <!-- ============================= --
                            >
                          <!--       General Information     --
                            >
                          <!-- ============================= --
                            >
                                <gml:description>
                                  A process that generates an
          insertResult transactional file from the input data
                                <gml:description>
                          <!-- insertResult process identifier
                          -->
                                <gml:identifier
codeSpace="uid">urn:SARTI:swe:process:insertResult_process_01</gml:id
                    entifier>
                          <gml:name>SWEBridge InsertResult
                  process</gml:name>
                          <!-- Define the simpleProcess as a
           getObsevation command process -->
                                <sml:typeOf
         xlink:href="urn:SOS:insertResult"/>
```

Finally, a connectionList defines the link between inputs (sources) and outputs (destinations):

```
      <!-- Defines the system input as the instrument command-->
                  <!-- Connects the instrument output witht the
             selector component input-->
                             <sml:connection>
                                <sml:Link>
                                  <sml:source
         ref="components/getObservation_CTD/outputs/dataOut"/>
                             <sml:destination
          ref="components/fieldSelector_CTD/inputs/dataIn"/>
                                <sml:Link>
                             </sml:connection>
                  <!-- Connects the selector output witht the
             insertResult component input -->
                             <sml:connection>
                                <sml:Link>
                                  <sml:source
         ref="components/fieldSelector_CTD/outputs/dataOut"/>
                             <sml:destination
            ref="components/insertResult/inputs/dataIn"/>
                                </sml:Link>
                             </sml:connection>
```

Full SensorML file can be downloaded from http://www.upc.edu/cdsarti/OBSEA/SWE/files/CTD_RBR_with_SWE_Bridge_configuration.xml

Acknowledgment

The authors acknowledge the strong support of the Oceans of Tomorrow data working-group for contributions to the OoT data interoperability advances. Projects in the Oceans of Tomorrow coordination have made continuing contributions. For majority of the authors, their work has been supported by the European Commission’s FP7 research and innovation program under grant number 614102.

References

A. Bröring, C. Stasch, and J. Echterhoff, "OGC Implementation Specification: Sensor Observation Service (SOS) 2.0 (12-006)", Wayland, MA, USA: Open Geospatial Consortium Inc., 2012.

M. Botts and A. Robin, "OGC Implementation Specification: Sensor Model Language (SensorML) 2.0.0 (12-000)", Wayland, MA, USA: Open Geospatial Consortium Inc., 2014.

S. Cox, "OGC Implementation Specification: Observations and Measurements (O&M) - XML Implementation 2.0 (10-025r1)", Wayland, MA, USA: Open Geospatial Consortium Inc., 2011.

S. Jirka, A. Bröring, and C. Stasch, "Discovery Mechanisms for the Sensor Web," Sensors, vol. 9, pp. 2661-2681, 16 April 2009.

S. Jirka, D. M. Toma, J. del Rio, and E. Delory, "A Sensor Web architecture for sharing oceanographic sensor data," in Sensor Systems for a Changing Ocean (SSCO) 2014 at the Sea Tech Week 2014, Brest, France, 2014.

Puck: http://www.opengis.net/spec/PUCK/v1.4/

T.O'Reilly, K.Headley et al, "MBARI technology for self-configuring interoperable ocean observatories", Proceedings of the Marine Technology Society / Institute of Electrical and Electronics Engineers Oceans Conference, Boston, Massachusetts, 2006

I. Simonis and J. Echterhoff, "OGC Implementation Specification: Sensor Planning Service (SPS) 2.0.0 (09-000)", Wayland, MA, USA: Open Geospatial Consortium Inc., 2011.

Toma, Daniel Mihai, Thomas C. O'Reilly, Arne Bröring, David R. Dana, Felix Bache, Kent L. Headley, Antoni Mànuel-Làzaro, and Duane R. Edgington. "Standards-based plug & work for instruments in ocean observing systems."IEEE Journal of Oceanic Engineering 39, no. 3 (2014): 430-443.