Standards-based Plug & Work for Instruments in Ocean Observing Systems

Joaquin del Rio, Daniel Mihai Toma, Thomas C. O'Reilly, Arne Bröring, David R. Dana, Felix Bache, Kent L. Headley, Antoni Manuel, Duane R. Edgington

***Abstract*—Ocean observing systems may include a wide variety of sensor and instrument types, each with its own capabilities, communication protocols and data formats. Connecting disparate devices into a network typically requires specialized software drivers that translate command and data between the protocols of the individual instruments, and that of the platform on which they are installed. In addition, such platforms typically require extensive manual configuration to match the driver software and other operational details of each network port to a specific connected instrument. In this paper we describe an approach to "*plug & work*" interoperability, using standardized protocols to greatly reduce the amount of instrument-specific software and manual configuration required for connecting instruments to an observatory system. Our approach has two main components. First, we use the Sensor Interface Descriptor (SID) model, based on the Open Geospatial Consortium's (OGC) SensorML standard, to describe each instrument's protocol and data format, and to provide a generic driver/parser. Second, a new OGC standard known as PUCK protocol enables storage and retrieval of the SID file from the instrument itself. We demonstrate and evaluate our approach by applying it to three commonly used marine instruments in the OBSEA observatory test bed.**

***Index Terms*— OGC, standards, metadata interoperability cabled observatories, PUCK, SensorML, OBSEA.**

I. Introduction

OCEANOGRAPHIC instruments are traditionally developed by small companies, with little standardization of the protocols used to control and configure the instruments, and to retrieve their data. RS232 and RS485 serial are the dominant physical layer protocols (though these may be increasingly displaced by Ethernet), but in general each manufacturer defines its own syntax and command sets for its instruments.

Manuscript received AUGUST, 2011. This work was supported in part by the EU funded project ESONET (*European Seas Observatory NETwork*; grant number 036851), the project *Flexible and Efficient Integration of Sensors and Sensor Web Services* funded by the ERDF program for NRW (contract number N 114/2008), as well as the 52°North Initiative for Geospatial Open Source Software.

J.D.R., D.M.T. and A.M authors are with SARTI Research Group. Electronics Dept. Universitat Politècnica de Catalunya, UPC. Spain; e-mails: joaquin.del.rio@upc.edu, daniel.mihai.toma@upc.edu, antoni.manuel@upc.edu

T.C.O., K.L.H and D.R.E. authors are with Monterey Bay Aquarium Research Institute, MBARI. US; e-mails: oreilly@mbari.org, headley@mbari.org, duane@mbari.org

A.B. and F.B. are with 52°North - Initiative for Geospatial Open Source Software GmbH, Germany. A.B is also with ITC, University of Twente. Enschede, the Netherlands, as well as Institute for Geoinformatics IfGI, University of Münster, Germany; e-mails: broering@52north.org, bache@52north.org

D.R.D is with HOBI Labs, Inc. USA. E-mail: dana@hobilabs.com

Those instruments are often integrated into an observing system or sensor network, which provides software infrastructure for many useful functions, including instrument data acquisition, data logging, and data transfer to other locations via hard-wired or wireless telemetry links. Most observing systems utilize generic or standard protocols for these functions; thus in most cases driver software that translates between specific instrument and generic system protocols must be written for each kind of instrument (Figure 1). Once written, the driver must be properly configured when the instrument has been physically installed into a communication port on the observing system. Driver software development and installation require significant effort by skilled software engineers and technicians. Once an instrument is connected to an observing system's network infrastructure, the next challenge is to provide real-time remote access to instrument data via the Internet. Few instruments provide data in a standard format - thus observatory or shore-based software is required to transform the instrument data format to a standard form.

We address these challenges with interoperability standards at multiple levels. Standards minimize the need for software development and manual configuration steps, thereby reducing system complexity, development and operational cost. Installation and maintenance operations on remote platforms are especially expensive and challenging. Standardizing and streamlining installation and operating processes can dramatically reduce costs, as well as the risk of failures due to manual errors. Standardization also facilitates easier maintenance and replacement of observatory instruments, and traceability of the data they generate.

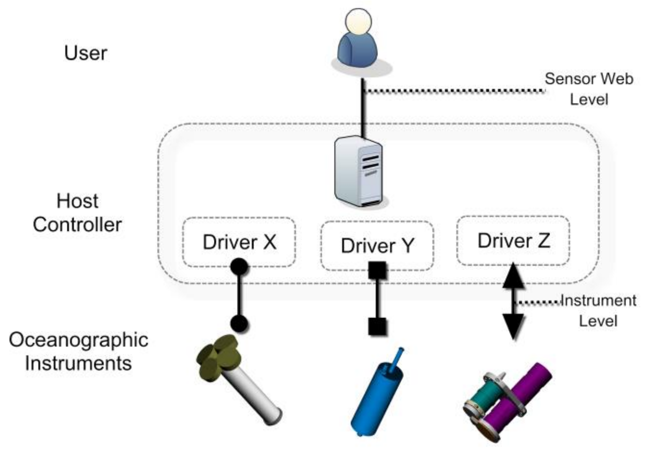


Figure 1: Accessing instrument's data requires different instrument driver implementations, due to different instrument proprietary protocols.

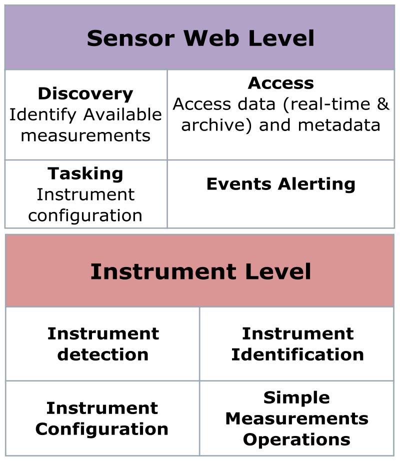


Figure 2: Simple processes susceptible to standardization.

a) At Sensor Web Level, b) At Instrument Level

In short, the ultimate goal is *plug & work*; upon connection to the observatory network, each device should be automatically detected and identified, and should provide all the information necessary for the network to collect and interpret its data, and ultimately disseminate them through standard web services.

In Figure 2 we divide the interoperability problem into two blocks: the *instrument level*, which includes the interface between a device and the network, and the *sensor web level*, providing the interface between the sensor network and the World Wide Web. We present an approach for each level, as well as for interoperability between the two levels, as shown in Figure 3. Our approach is based on open standards that are maintained by the Open Geospatial Consortium (OGC). At the instrument level, instrument and host controller use OGC PUCK protocol to handle the initial connection to the instrument and to retrieve information stored in the instrument about its own identity and configuration. The instrument side of the PUCK protocol can be programmed directly into an instrument's firmware as an extension of its native command set, or implemented through a separate PUCK adapter device. PUCK protocol is described further in sections II.B and III.A.

At the World Wide Web level, the host controller provides various Sensor Web Enablement (SWE) services that allow a data user or system operator to control individual instruments and retrieve their data. SWE is described in section II.B.

Finally, the Sensor Interface Descriptor (SID) model provides a bridge between the instrument and Web levels. Built on the OGC SensorML standard, SID is a framework to precisely describe the command protocol and data formats of a particular instrument type. A properly constructed SID file - which may be stored in the instrument and retrieved through the PUCK protocol - allows a SID Interpreter on the host controller to control the instrument and parse its data without requiring any instrument-specific driver software. We describe SID in section III.B.

These elements, PUCK, SID and SWE, can be combined to implement *plug & work* operation of a sensor network.

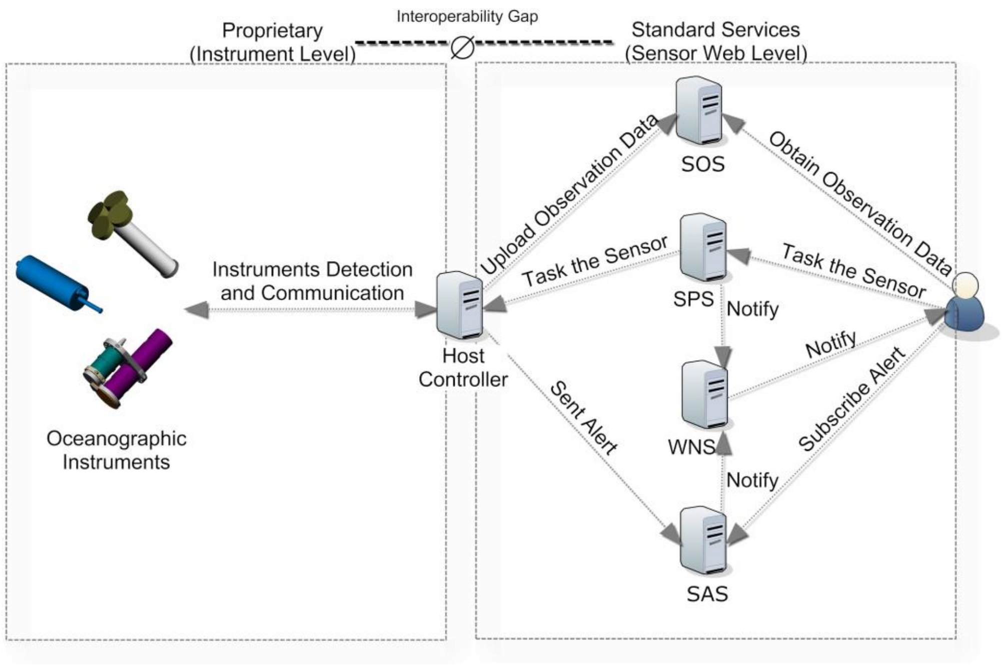
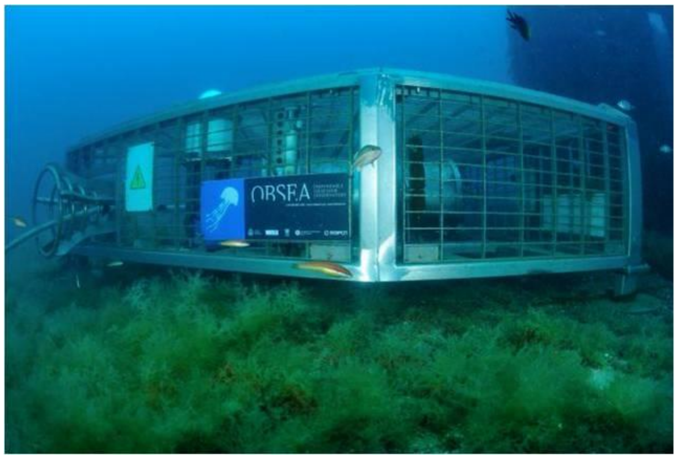


Figure 3 Bridging the interoperability gap between proprietary instrument protocols and standard web services

We tested our approach at the OBSEA Cabled Observatory [1,2]. OBSEA is a seafloor observatory 4 km offshore from Vilanova i la Geltrú (Barcelona, Spain), at a depth of 20 m in an area protected from fishing. The cable connection to shore provides power to its instruments, and continuous real-time communication through an optical Ethernet network. Figure 4-a is a picture of the deployed OBSEA junction box, instruments, and protective enclosure, and Figure 4-b shows a simplified diagram of the observatory.



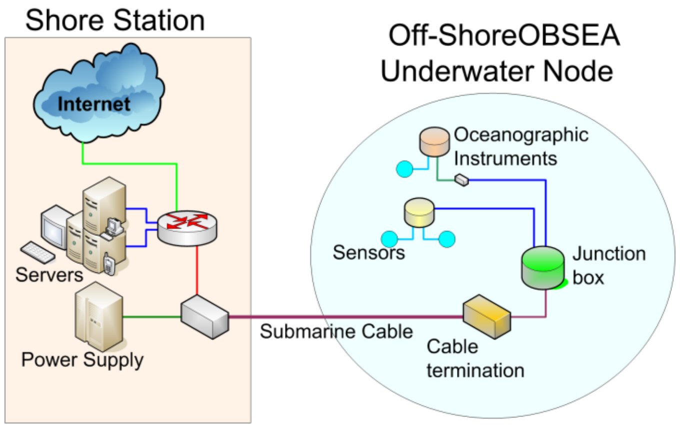


Figure 4. a-Western Mediterranean Cabled Observatory OBSEA picture.

b- OBSEA architecture

II. Background and Related Work

A. Interoperability Approaches

Makers of laboratory instrumentation have pursued various approaches to standardizing instrument interfaces, starting in the 1960s. For example, IEEE-488 (General Purpose Interface Bus) is a mature and widely used standard defining the physical layer and transport protocol for delivering commands and data between instruments and controllers. The industry standard SCPI (Standard Commands for Programmable Instrumentation) provides software-level syntax and commands for operating instruments over IEEE-488 or other transport protocols, such as Ethernet and USB.

Instrumentation interoperability has also benefitted from the rapid growth of technology and standards used by personal computers, such as PCI (Peripheral Component Interconnect), which has been extended and adapted to form the PXI bus (PCI eXtension for Instrumentation). However, these standards were all designed for a laboratory or office environment, and are not well suited to the particular demands of underwater measurements. Also, they do not address the specialized measurement types found in marine research.

Standardization efforts in the marine research community have largely focused on standard formats for data and metadata to ensure interoperability between data producers and consumers [26,27,28,29]. NMEA 0183 defines a simple ASCII serial protocol consisting of standardized "sentences" that are passed between "talkers" and "listeners" [35]. Many navigational marine instruments implement NMEA 0183, but the standard's restrictions to ASCII formats and a 4800 baud serial data bus have limited its application to "science" instruments. In terms of metadata formats, most viable standards are based on XML, and use the ISO19115 schema for describing geographic information, with some extensions to cover marine data characteristics [30].

The International Oceanographic Data and Information Exchange (IODE) of the Intergovernmental Oceanographic Commission (IOC) of UNESCO promotes XML and ISO19115 for metadata encoding, with the WMO (World Meteorological Organization) Core Metadata Profile.

B. IEEE 1451

The IEEE 1451 Smart Sensor Interface Standard provides a common communications architecture with sensors over different communication protocols at the physical level. Different protocols are addressed by different branches of the standard, for example, 1451.2 for RS232, I^2C, and SPI; 1451.4 for analog sensors, 1451.6 for CAN (controller area network), and so on. This standard has not yet been widely used, especially in marine sensors, and is adoption is hampered by a lack of software tools for implementation. However, it has some capabilities that may be useful in marine networks.

Marine instrumentation most commonly uses serial links, so IEEE 1451.2 would apply. However, the version of IEEE 1451.2 from 1997 is now obsolete due to the complexity of its 10-wire hardware interface, and there are no real industrial implementations using this standard. In order to simplify the use of IEEE 1451 for serial instruments, release of a new draft is expected in 2011. This new version of IEEE 1451.2 is fully compatible with an RS232 instrument, using the communication and measurement services described in IEEE 1451.0. This will allow manufactures to implement the protocol in their new generation of instruments.

Figure 5 shows the main components of an IEEE 1451 standard approach. IEEE 1451 uses the term TIM (Transducer Interface Module) to refer to a sensor or actuator, and NCAP (Network Capable Application Processor) to mean a controller interfacing to one or more TIMs. In this case, data coming from non-IEEE 1451 instruments are processed to inject data into an IEEE 1451.0 server [21]. This server publishes data using the HTTP 1451 standard [22]

One key feature of IEEE 1451 is the definition of a standard transducer electronic datasheet (TEDS). IEEE 1451 specifies many standard templates for describing sensors and actuators with a TEDS, and IEEE 1451.4 promotes the idea of storing TEDS information within the device itself. The system we describe includes this approach as well, with PUCK protocol used to store and retrieve the instrument description, whether as an IEEE 1451 TEDS, or in some other format.

Java Distributed Data Acquisition and Control (JDDAC), created by Agilent Technologies Inc. and Sun Microsystems Inc., is another related effort that used the IEEE 1451 TEDS.

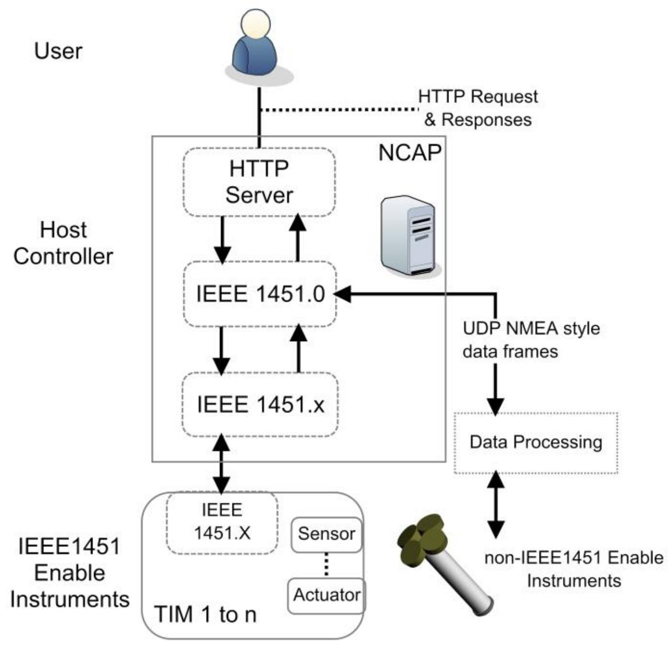


Figure 5. IEEE1451 HTTP System Architecture with additional non IEEE1451 instruments

In summary, though there have been many incremental steps toward instrument integration as it applies to marine sensor networks, there has not been a standards-based architecture that offers practical end-to-end *plug & work* capability. The architecture we describe below demonstrates one possible end-to-end solution using Open Geospatial Consortium Sensor Web Enablement (OGC SWE) standards to implement distributed and geospatially referenced sensor networks.

C. Sensor Web Enablement (SWE)

The goal of OGC's Sensor Web Enablement is to enable Web-based sharing, discovery, exchange and processing of sensor observations, as well as task planning of sensor systems [3,4]. 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 [7].

The main Web Services of the SWE framework are the Sensor Observation Service (SOS) and the Sensor Planning Service (SPS). The SOS [31,8] provides interoperable access to real time sensor data as well as sensor metadata, while the SPS can be used to control and task sensors [32]. 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 the OGC Observations and Measurements (O&M) standard information model for sensor data [10], the Observations & Measurements (O&M) [10] standard, as well as OGC Sensor Model Language (SensorML) to describe sensor characteristics and processing steps [5].

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 with SensorML, including its identification, classification, inputs, outputs, parameters, and characteristics such as a spatial or temporal description. Processes can be composed of process chains.

O&M defines a model and encoding for observations: an observation has a result that is an estimated value of an observed property, a particular characteristic of a feature of interest. For example, 14°C is an estimated value of temperature at the OBSEA observatory. The value of the result is generated by a procedure, e.g. a sensor such as a thermometer described in SensorML. These four central components (SensorML, O&M, SPS, SOS) are linked within SWE.

OGC PUCK protocol is a new addition to SWE, establishing a protocol to retrieve descriptive information from PUCK-enabled instruments [37]. At a minimum, this includes a formatted electronic datasheet containing information needed to identify the instrument manufacturer, model, and unique instance of the model. Optionally, a PUCK-enabled instrument may also carry an additional payload with a user-defined format. The optional payload typically includes some or all of the information needed to operate the instrument and interpret its data. A host computer that implements PUCK protocol can automatically retrieve and used this information from the instrument when the device is installed, as illustrated in Figure 6. This process reduces the number of manual steps required for installing the instrument in the network. For example, a SensorML document and instrument driver code can be physically stored in the instrument's PUCK memory before deployment. When the host controller connects to the network, this information can then be retrieved and used by a host controller to configure the itself and communicate with other (SWE) services.

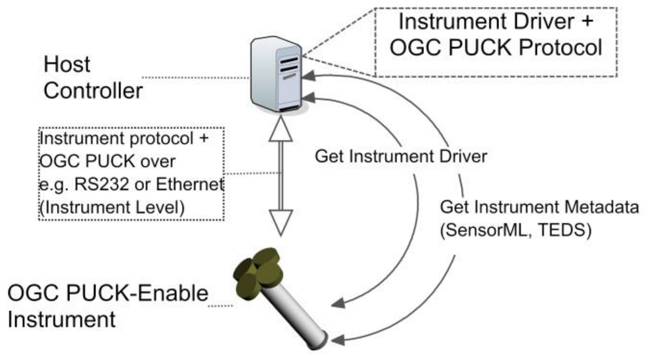

Instrument Driver + OGC PUCK Protocol
Host Controller
Instrument protocol + OGC PUCK over e.g. RS232 or Ethernet (Instrument Level)
Get Instrument Driver
Get Instrument Metadata (SensorML, TEDS)
OGC PUCK-Enable Instrument

Figure 6: Relation between OGC SWE services, SensorML, and OGC PUCK

III. A STANDARDS-BASED ARCHITECTURE FOR THE AUTOMATIC INTEGRATION OF MARINE SENSORS

In this section we describe how we use SWE standards combined with SID to implement automatic integration of marine instruments into a network.

A. Instrument Level: PUCK protocol

Figure 7 schematically shows components of a typical marine instrument. The instrument's embedded microprocessor is physically connected via I/O channels to transducers and sensors, as well as to an external physical interface (such as RS-232 or Ethernet). PUCK protocol currently supports EIA232 (aka "RS232") or Ethernet physical/electrical interface. The microprocessor executes algorithms within its embedded "firmware" which retrieve data from the sensors and may process the data further. The firmware also implements a command protocol on the external interface; an application running on an external host computer can use the protocol to configure the instrument, retrieve its sensor data, and get a description of the instrument characteristics and state.

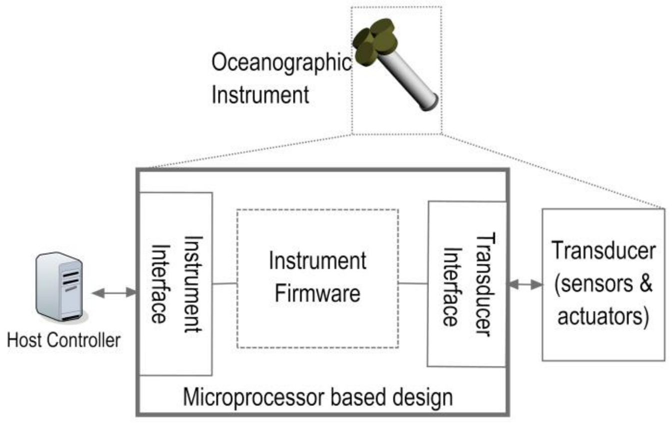


Figure 7 Simple block diagram of an Instrument

Thus the host application includes *instrument driver* software that communicates on the appropriate physical port using the specific instrument protocol. The application also must "know" the instrument's data and metadata formats in order to process the data further. Oceanographic instruments most commonly provide an external RS232 serial interface, which is compatible with the limited power, limited number of wires, and long cable lengths associated with many marine applications. In the RS232 case, the instrument driver must also be configured to use the proper baud, parity and stop bits to communicate with the device. Instrument manufacturers often provide a driver for use in a desktop environment (e.g. for Microsoft Windows), but often a driver that can be run in non-desktop environments is needed, e.g. on an ARM-based Linux instrument controller. Thus several steps are needed to install an instrument into an observing system:

- For new kinds of instruments, an instrument application and driver must be developed for the host, utilizing the specific protocol and formats to command the instrument and process its data
- The instrument must be physically installed into a host port,
- The driver must be installed on the host, and configured to use the correct port at the correct baud rate

OGC PUCK protocol (section II.B) enables automatic system configuration when the instrument is plugged into the host. Figure 8 depicts a block diagram of an RS232 PUCK-enabled instrument, and the possible interactions from the standpoint of a host controller.

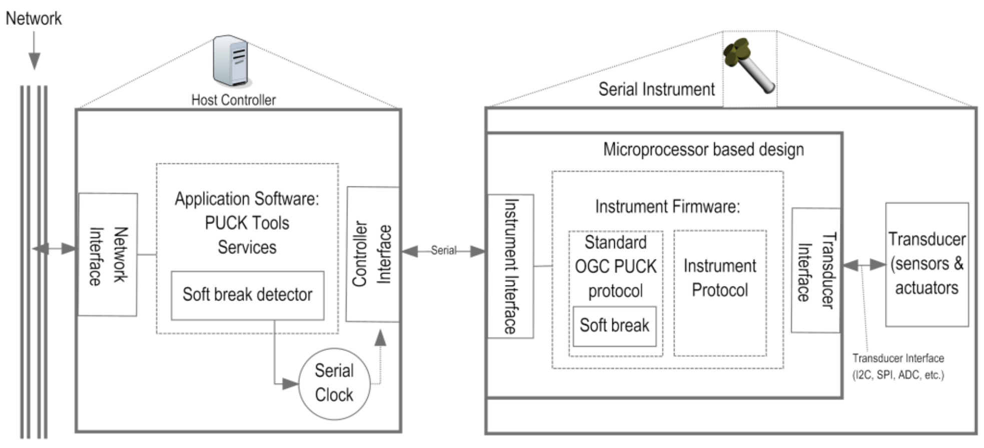


Figure 8 Serial Puck Instrument Model

A PUCK-enabled RS232 instrument can operate in “instrument mode” or “PUCK mode”. In instrument mode, the device responds to the protocol defined by its manufacturer, including "non-standard" commands. While in PUCK mode, the device responds to the standard PUCK protocol. PUCK mode is typically used when the instrument is first connected to the host or the host is rebooted. To switch a PUCK-enabled device into PUCK mode, a “PUCK soft break” command is sent to the device at several baud rates until a valid PUCK response is received by the host as shown in Figure 8. While in PUCK mode the host can retrieve the PUCK datasheet and any optional payload information, from which the host can infer the instrument model and manufacturer. The host then switches the device into "instrument mode" and begins issuing instrument-specific protocol commands to it.

Modern cable-to-shore observatories such as OBSEA are not as power-constrained as buoy-based systems and standard Ethernet is an option for instrument interfaces. Likewise low-power Ethernet is now available even for buoy-based applications [23]. For these instruments, the OGC PUCK standard specifies PUCK over IP ("IP PUCK"). In addition to basic PUCK protocol, IP PUCK specifies the existing Zeroconf standard as a means to automatically assign the instrument's link-local IP address and hostname, and a mechanism for hosts to discover the instrument address and “PUCK port” number [14]. Thus an IP PUCK instrument can automatically acquire an IP address and name when it is physically installed into the network. A host can then use Zeroconf's service discovery protocol to discover PUCK-enabled instruments in the marine IP network and retrieve their metadata and payloads using PUCK commands issued to the instrument's specified "PUCK port". Figure 9 depicts a block diagram of an IP PUCK-enabled instrument, and the component system protocols: Zeroconf, OGC PUCK protocol, and the instrument proprietary protocol. Figure 9 also shows the possible interactions with a host device connected in the same network.

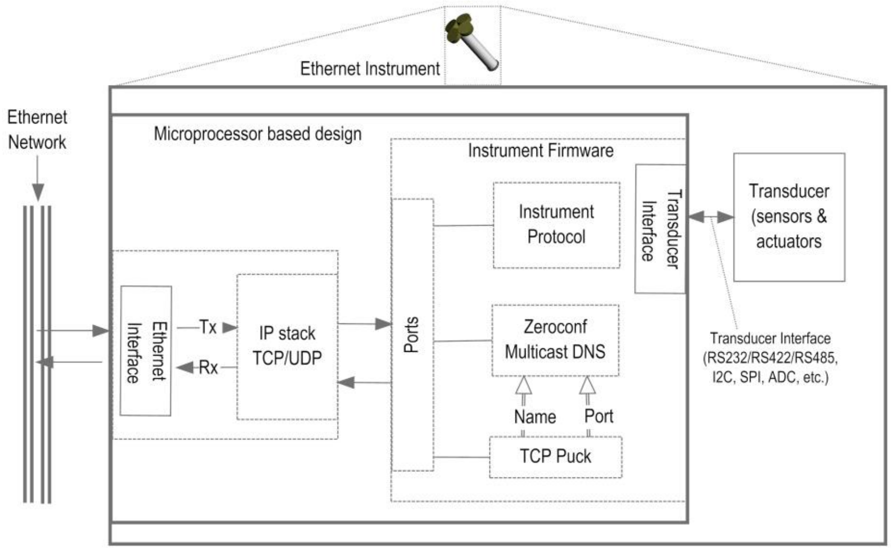


Figure 9 IP Puck Instrument Model

An implementation of IP PUCK has been developed on the Stellaris ARM® Cortex-M3™-based microcontroller, LM3S9B96 [13]. This "Smart Sensor Board ("SSB") has been designated as a standard oceanographic instrument platform by the French *Institut Français de Recherche pour l'Exploitation de la Mer* (IFREMER) [18]. The device includes serial ports for RS232 instruments, and can include embedded sensors as well. In either case a host can automatically detect the SSB in the network with ZeroConf and retrieve the instrument information using PUCK protocol. In this implementation, the unique instance name advertised by Zeroconf discovery protocol is derived from the PUCK datasheet, and consists of instrument manufacturer, model and serial number.

The PUCK standard designates "PUCK payload" as an optional feature, as the available memory in some instruments may be very limited. In such cases, an observing system can provide an external database of information that would otherwise be stored in PUCK payload. A host computer can then retrieve the relevant information from the database, using the PUCK UUID retrieved from the instrument as a database key.

B. Data Integration Level: Sensor Interface Descriptors (SID)

A system for integrating data from an arbitrary set of instruments must accommodate a wide variety of protocols. Each instrument's protocol includes the commands used to trigger its collection and transmission of data, and the formats in which data are returned. As noted in section I, there is virtually no standardization of protocols among different manufacturers or types of instruments. Because of this, significant effort is required to develop and maintain instrument driver code across different software platforms; Sensor Interface Descriptors help to mitigate this by defining a generalized and platform-independent encoding of instrument protocols.

Some instruments return data in a nominally human-readable format, typically ASCII-encoded decimal numbers separated by commas or tab characters. However there are many other formats, with varying degrees of complexity, tailored to the characteristics of each instrument. These may include fields defined by a fixed or delimited formats, and binary encoding of various integer and floating-point number types. Different data fields within a packet may be encoded differently. A single instrument may also produce multiple packet formats within the same data stream. For example, an instrument may report "housekeeping" data in separate packets, and at different rates, from its primary measurement data. The contents of a data field could even indicate the length or format of subsequent data within the same packet.

Instruments also vary in the way that data output is triggered; some stream data continuously (with or without initial prompting from the host), while others must be polled.

The proposed Sensor Interface Descriptors (SID) standard provides a framework for generalized descriptions of protocol features. While SID does not accommodate every possible variation, it does encompass a wide variety, covering the majority of protocols used in ocean observing systems today.

The architectural principle of a system infrastructure incorporating the SID concept as defined by Bröring et al. (2010) [6] is shown in Figure 10. An instrument communicates with a data acquisition system in its specific protocol over a transmission technology such as RS232 or Ethernet. The instrument can also act as a gateway (network sink) so that other nodes of a (possibly mobile) sensor network communicate with it. The SID Interpreter runs on the host controller and uses SID instances for the different instruments of the sensor network to translate between the specific instrument protocol and the protocols used on the application level, e.g. the protocols of the Sensor Web Enablement (SWE) suite of standards (Section II.B). The interpreter is responsible for registering an instrument at the application (e.g. an SWE Sensor Observation Service) and to upload the measured data. It is also responsible for the opposite communication direction, to forward tasks from an application (e.g. a SWE Sensor Planning Service) to an instrument.

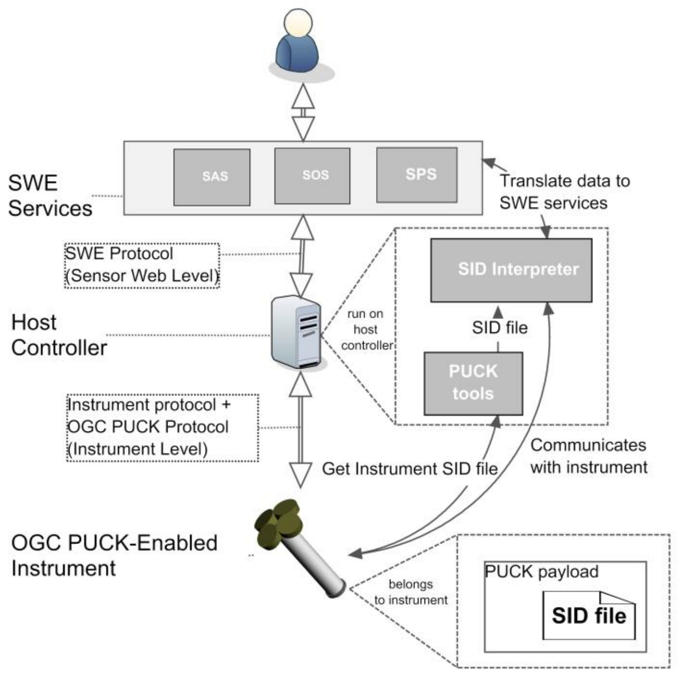

SAS
SOS
SPS
SWE Services
Translate data to SWE services
SWE Protocol (Sensor Web Level)
SID Interpreter
Host Controller
run on host controller
SID file
PUCK tools
Instrument protocol + OGC PUCK Protocol (Instrument Level)
Get Instrument SID file
Communicates with instrument
OGC PUCK-Enabled Instrument
belongs to instrument
PUCK payload
SID file

Figure 10: Connection of a sensor to SWE services through an SID Interpreter.

1) The SID Model

A strong requirement of the design of the SID model [10] is the strict encapsulation of the SID within a SensorML document. The SID part of the SensorML document is specific for a certain instrument type, not a particular instrument instance. Hence, an encapsulation allows reusing the SID in the SensorML descriptions of different instruments that are of the same type. The approach developed here, encapsulates the SID within the *InterfaceDefinition* element of a SensorML document.

The *InterfaceDefinition* element contains a stack of layers (Figure 11), aligned with the Open System Interconnection (OSI) reference model. In contrast to the OSI model, SensorML does not further define how to use these layers. The SID model uses this layer stack to describe the instrument interface.

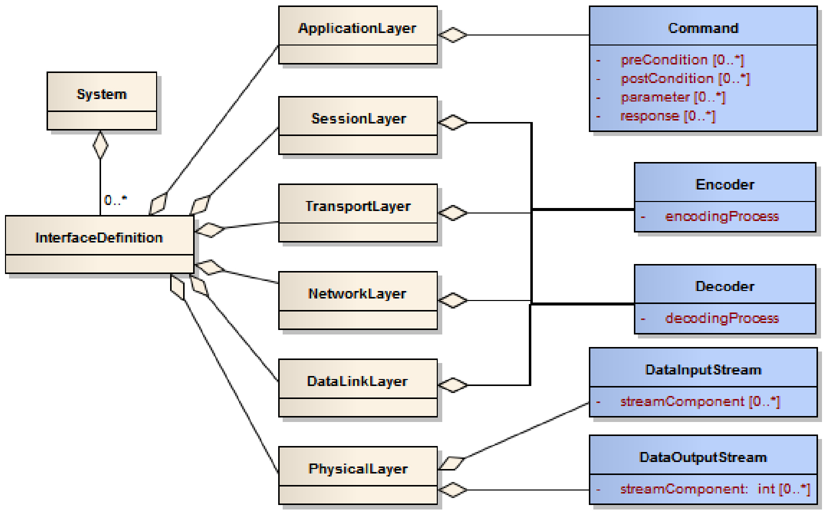


Figure 11: Excerpt of SensorML schema (beige colored types) including the SID extension (blue colored types).

A definition of the raw instrument protocol exchanged between instrument and data acquisition system is essential. The structure of these raw data is described within the lowest, the *physicalLayer* element. As shown in Figure 11, 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 instruments.

The *dataLinkLayer*, *networkLayer*, *transportLayer*, and *sessionLayer* define the processing steps that are necessary to translate between the instrument protocol and the target protocol. To allow data processing between the instrument and application, elements for data decoding and encoding are added to each layer (Figure 11). Instances of these elements contain descriptions of applied processing steps. Here, the SID model reuses existing SensorML types to define processes with its inputs, outputs, parameters and its computational method.

The encoding of an instrument data stream into target protocols typically uses the SID layers in this way: 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.

The data resulting from the preceding processing steps have to be associated with certain metadata needed by the target protocol, e.g. the SWE Observation and Measurement (O&M) protocol. The measured data need to be associated with units of measure. Further, the data need to be linked to the core SensorML measurement concepts, the observed property and the feature of interest.

While the association of the data with a unit of measure is done on the *presentationLayer*, the link to observed property and the feature of interest is established in the outputs element of the SensorML document. This *outputs* element is not part of the SID, since it is not a sub-element of the *InterfaceDefinition* (Figure 11). The information contained in the *outputs* element is intentionally kept out of the SID, since the linkage of an instrument to feature of interest and observed property is dependent on the particular use case, not the interface of the instrument type. By not including this information in the SID, it is possible to reuse the SID in different applications; i.e. SID enables interoperability because it is platform-independent..

The application layer of the OSI model describes interfaces to access the OSI stack. Consistent with this view, the *applicationLayer* is used here to define the commands accepted by the instrument. As shown in Figure 11, the *command* element contains sub-elements to describe possible instrument responses, the pre- and post-conditions for executing the command, as well as the command parameters.

2) The SID Creator

The creation of the SID file (SensorML and contained SID code) without tool support is tedious and error-prone, since plain XML has to be written by hand. For this reason, the visual SID Creator has been developed by Bröring et al. 2011 [9], which enables a semi-automatic generation of SID instances.

Implemented in Java, the SID Creator allows the description of an instrument protocol by following a wizard-style user interface. First, the user provides basic metadata about the instrument, such as identification, a human readable name and description. The metadata are used to compose SensorML tags when the SID file is generated. Since SensorML is generic and does not explicitly specify where to put this information, a public profile of SensorML that is optimized for discovery of instruments [11] is applied to encode these data.

Next, the wizard enables the definition of the instrument protocol. The user can define how the SID Interpreter physically connects to the instrument (e.g., RS232, or Ethernet). Further, the structure of ASCII based protocols can be described. For example, the separator symbols are defined; these are utilized by the protocol to parse blocks, fields within a block and decimal numbers. The SID Creator may be used to specify multiple blocks within the data stream coming from the instrument. An example of such a block within a data stream is given in Listing 1.

```
…#thermometer123|2010-09-02T13:05|22.34|C#…
```

List. 1 - A single block within a data stream.

The block is identified by the value of its first field, *thermometer123* in case of Listing 1. This block ID is also specified in the wizard to be stored in the SID file. Further, three fields are added to the block. The second field is the value of the measured data. Finally, this field is referenced as the actual sensor data output. The values of this field are uploaded to the application. The wizard generates a complete SensorML description for the instrument containing the user-defined SID element. Table 6 shows in detail the inputs to each page of the SID Creator for describing the interface of the Seabird SBE-37SM instrument.

Implementations of the SID Interpreter and SID Creator are available as open source software from 52°North (http://52north.org/sid).

C. Combining PUCK and SID

Combining SID and PUCK enables *plug & work* capability without the need create platform-specific instrument driver code. As described above, SID provides a standard description of a particular instrument’s native protocol, including the commands to configure the instrument and trigger data acquisition. If the instrument also recognizes PUCK protocol, the instrument-specific SID file could either be stored in the instrument's PUCK payload, or stored in an external database keyed to PUCK UUID. We have tested this approach, and describe our results below.

If the instrument and host computer are connected with RS232, the host computer runs a "PUCK detector" algorithm to automatically detect when a new PUCK-enabled instrument is plugged into a serial port and to respond accordingly. This detection protocol is shown in Figure 13. The PUCK detector periodically interrogates the serial port for a PUCK-enabled instrument via the PUCK "soft break" command. When the host receives a PUCK response from the serial port, the host retrieves the 96-byte PUCK datasheet and examines the UUID to determine if a new instrument has been installed. If so, the host retrieves the SID file either from the instrument's PUCK payload or from an external database keyed to PUCK UUID, and configures a new SID Interpreter with the appropriate SensorML, serial port name and serial baud rate. Finally the SID Interpreter begins to retrieve data samples from the instrument at the sampling frequency specified in the instrument's SID file.

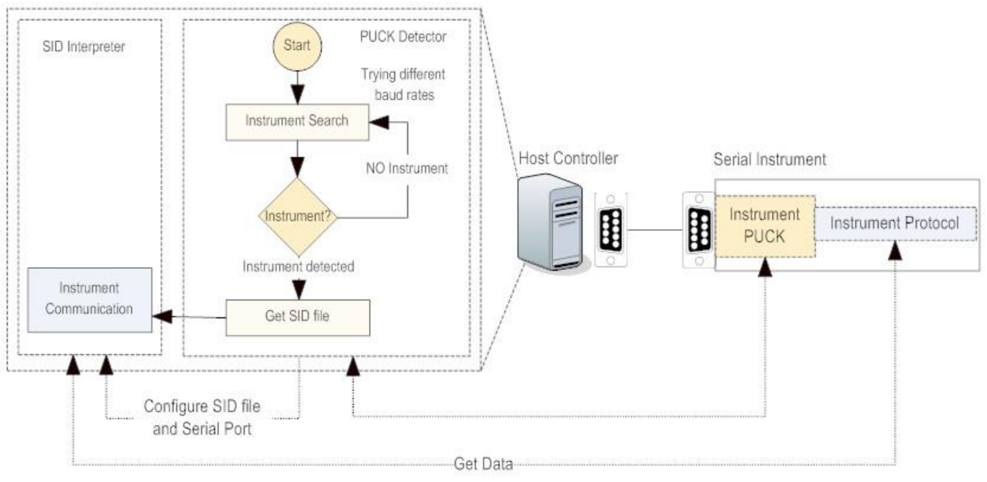


Figure 13 Automatics serial puck detection process diagram and SID enable

Figure 14 shows the detection algorithm for IP PUCK instruments. A Zeroconf mDNS browser runs on the host, and is capable of discovering all IP PUCK instruments in the local network. When the browser finds an IP PUCK instrument, the host reads the IP address and TCP "PUCK port" number of the service from the ZeroConf DNS record, retrieves the 96-byte PUCK datasheet and examines the UUID to determine if this instrument was previously installed. If not, the host retrieves the SensorML and SID, and uses them to configure a new SID Interpreter. Finally the SID Interpreter acquires data from the instrument at the frequency specified by the SID.

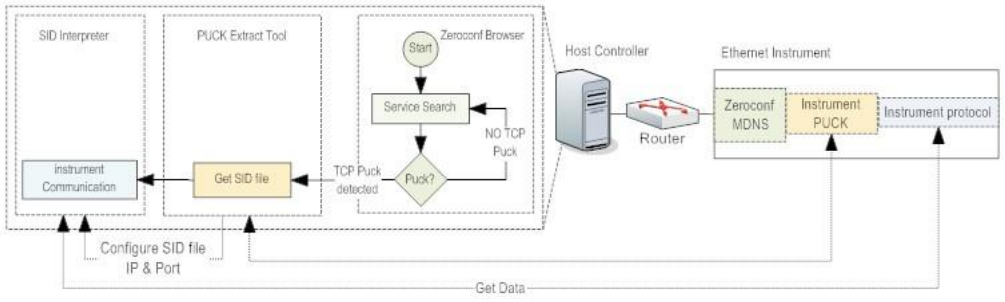


Figure 14. Automatics IP puck detection process diagram and SID enable

IV. Implementation

We implemented a prototype *plug & work* observatory based on the architecture described in the previous section. The prototype was demonstrated at the general assembly meeting of the ESONET EU funded project in December 2010 in Marseille (France) [33].

A. Prototype Architecture

Figure 15 shows the main components of the prototype architecture. The objective of the prototype is to access data generated by instruments (1,2,3), connected to a host controller (5) through standard Internet clients (6,7) without any manual configuration steps at installation or run time. Instruments 2 and 3 are PUCK-enabled serial devices; the SID describing the instrument protocol and data formats is stored within the PUCK payload (8) of its corresponding instrument. The host controller (5) is implemented by an Internet-connected Windows laptop computer, and hosts several software components:

9a) A PUCK detector is assigned to each serial port, issuing a PUCK "soft break" at different baud rates until it receives a PUCK response from an attached instrument. If the detector determines that the instrument is newly installed based on its UUID, the SensorML and SID are either retrieved from the instrument's PUCK payload or from the SID repository, and a new SID Interpreter is created (9b). The SID Interpreter executes the initialization and sampling protocol declared in the SID. Data coming from the instrument will be parsed with the data format declared by the SID and processed (if necessary). These data can be sent to higher-level data management components on the network.

Instrument	**Link layer**	**PUCK-enabled?**	**Provides PUCK payload?**	**Data format used**	**Link to SID**
Seabird SBE-16+ CTD on Smart Sensor Board	Ethernet	via Smart Sensor Board	via Smart Sensor Board	comma-delimited ASCII	http://52north.org/communities/sensorweb/examples/2011-07-28-Seabird-SBE-16P-3out.xml
Seabird SBE-37SM	RS232	yes	yes	comma-delimited ASCII	http://52north.org/communities/sensorweb/examples/2011-03-18-Seabird-SBE-37.xml
HOBI Labs HydroScat-2	RS232	no	no	comma-delimited and fixed-width ASCII	http://52north.org/communities/sensorweb/examples/2010-11-18-Hobilabs-HydroScat.xml
WET Labs ECO Triplet	RS232	yes	no	space-delimited ASCII	http://52north.org/communities/sensorweb/examples/2010-11-10-WETlabs-Triplet.xml
RBR XR-420 CTD	RS232	yes	yes	space-delimited ASCII	http://52north.org/communities/sensorweb/examples/2011-03-18-RBR-xr420.xml

Table 1: Instruments used with interoperability prototype and SID URLs

Our prototype uses the open source 52°North SOS (Sensor Observation Service) (11) and SOS client (6) to distribute the data. The data are also sent to a Data Turbine Ring Buffer [24, 25] (12), for real-time data access through the Internet using the RDV Client (7).

The instrument SID files are generated beforehand with the SID Creator. As noted earlier, SID files can be reused for instruments of the same type. Before the instrument is deployed, the SID file is stored either in the instrument's PUCK payload or in an external SID Repository (13) database that is keyed to PUCK UUID. If the SID file is stored in the instrument’s PUCK payload, the PUCK detector (9a) will retrieve the SID payload from the instrument once connected, and store it in the SID Repository (13). The SID Interpreter (9b) is then invoked for the instrument’s SID payload. If the instrument does not have a SID payload, the PUCK detector (9a) using the instrument UUID will locate the proper SID file in the repository before invoking SID Interpreter. Below we describe our observing system implemenation that used SID to connect to an OGC Sensor Observation Service.

B. Instruments

We tested our prototype with several oceanographic instruments, as shown in
. Most of the PUCK-enabled devices provide optional PUCK payload, in which we stored the appropriate SID and SensorML. The PUCK-enabled WET Labs fluorometer does not support a payload, so we implemented a simple external database or "SID Repository" in which we stored SID files, keyed to PUCK UUID.

The instruments implement a variety of data formats, and some offer multiple options. For purposes of the prototype testing, we used delimited ASCII for instruments that supported it. We defined instrument-specific SIDs and SensorML documents for the instruments in

, using the SID Creator described previously. The SID files are available online at the locations shown in

. In order to illustrate the variety of instrument response formats, the following tables (Table 2-5) show example data delivery messages for the different instruments listed in

and which have been described with SID files.

Table 2: Example response of SBE-37SM instrument

Response to "TS" command: Delimiter: comma
20.8397, 0.00006, 0.028, 0.0103, 1484.920, 01 Jan 1980, 00:00:01<CR><LF>
tokens: *temperature (deg C), conductivity (siemens/meter), pressure (decibar), salinity (psu), sound velocity (meter/sec), dd mmm yyyy,: hh:mm:ss*

Table 3: Example response of RBR instrument

Response to "F00" command: Delimiter: space
TIM 000302213620 -0.0042 20.7903 10.5324 FET
tokens: *"TIM" (fixed marker), YYMMDDhhmmss, conductivity (mS/cm), temperature (deg C), pressure (decibar), "FET" (fixed marker)*

Table 4: Example response of WET Labs Triplet

Response to "$run" command: Delimiter: space
08/04/11 14:44:28 4996 13896 17333 08/04/11 14:44:28 16777 13896 17333 08/04/11 14:44:29 16777 13896 17333
tokens: *date (mm/dd/yy), time (hh:mm:ss), chlorophyll (counts), backscatter1 (counts), backscatter2 (counts)*

Table 5: Example response of HOBI Labs HydroScat-2

Response to "C" command: Delimiter: comma.
2355,1904,1601,2472,1471,1860,1686,2950,1754,1663,1653,2472,1471,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,5,5,5,3,3,3,3,3,177,2677,

0,128,116<CR><LF>
tokens: *Time elapsed since, signal1, sigOffset1, reference1, refOffset1, signal2, sigOffset2, reference2, refOffset2,... <unused channels> ... <housekeeping data> ... pressure, 0, temperature, battery voltage*.

C. Configuration Example: Seabird SBE-37SM

As a common marine instrument type, the configuration of the Seabird SBE-37SM is described in further detail. This instrument measures water conductivity, temperature and pressure. From these parameters it also computes and outputs salinity, depth and sound propagation speed (see Table 2). The command used to collect ocean data samples with the Seabird CTD is an ASCII command type "TS" followed by a carriage return.

Listing 2 shows an excerpt of the SID created for the Seabird sensor. In the application layer of the SID, the "TS" command is stated (<swe:DataRecord gml:id="TS">). It is defined that it is automatically executed in an interval of 5 seconds to regularly retrieve the sensor's measurements (<sid:command name="getDataCommand" auto="true" interval="5">). Further, Listing 2 contains parts of the description of the Seabird's response to the TS command: the message containing measured data values. As described in section III.B.2, here the structure definition of the protocol takes place by declaring which blocks and fields are contained in the message coming from the instrument. In this case, the instrument response uses character "comma" to separate each token and a carriage return is used to identify the end of instrument responses (<swe:TextBlock tokenSeparator="," blockSeparator="&#x000D;" decimalSeparator="."/>).

Listing 2 Seabird 37SMP response description and SID File excerpt representation

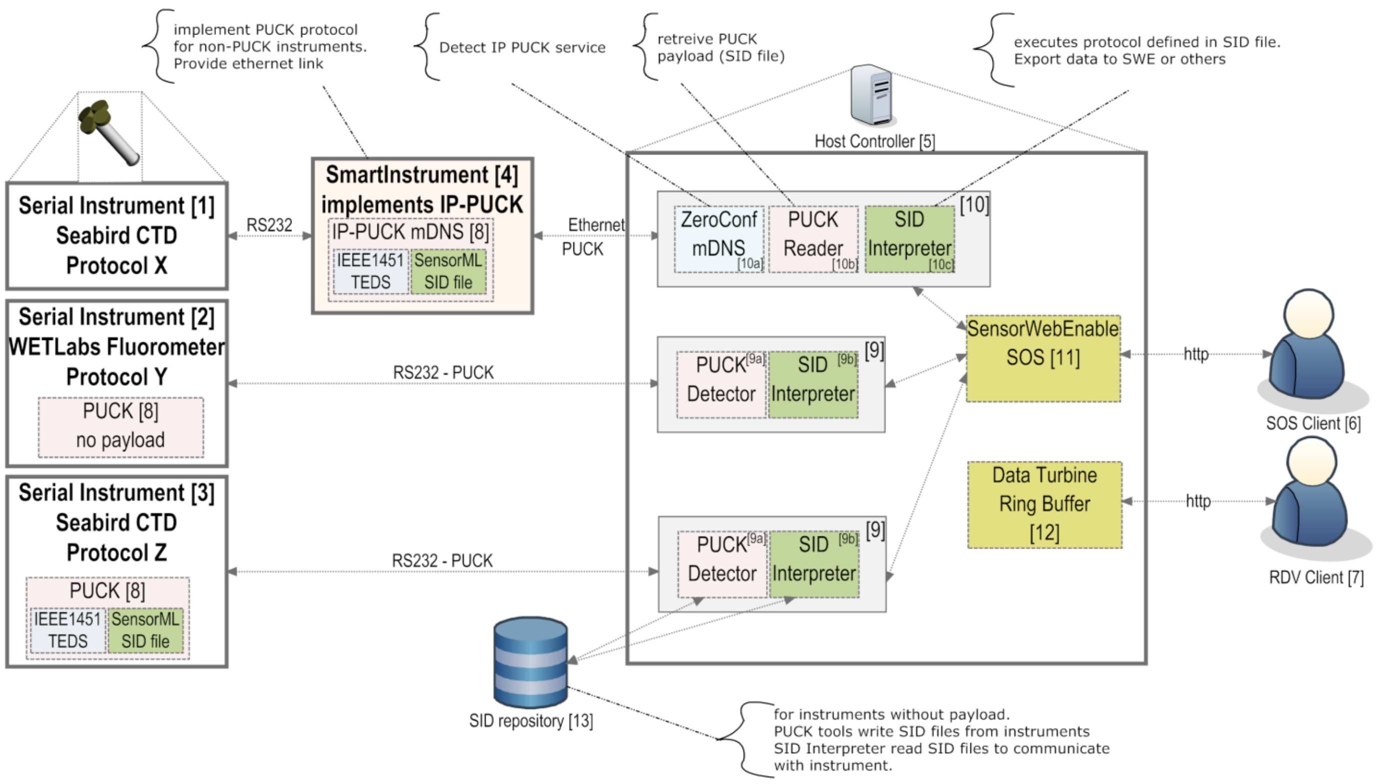
```
<sml:applicationLayer>
  <sid:CommandDefinition>
    <sid:commands>
      <sid:CommandList>
        <sid:command name="getDataCommand" auto="true" interval="5">
          <sid:Command>
            <swe:DataRecord gml:id="TS">
              <swe:field name="command" xlink:role="urn:ogc:def:command:OGC:name">
                <swe:Text>
                  <swe:value>TS</swe:value>
                </swe:Text>
              </swe:field>
..
<sml:physicalLayer>
  <sid:DataOutputStream>
    <dataOutputComponents>
      <ComponentList>
        <component>
          <swe:DataBlockDefinition>
               <swe:DataRecord>
                 <swe:field name="time" />
                 <swe:field name="conductivity" />
                 <swe:field name="pressure" />
                 <swe:field name="temperature" />
               </swe:DataRecord>
             <swe:encoding>
               <swe:TextBlock tokenSeparator="," blockSeparator="&#x000D;" decimalSeparator="."/>
            </swe:encoding>
         </swe:DataBlockDefinition>
```

The SID Creator wizard creates the SID file (Listing 2) from user-specified parameters describing the instrument and its protocol. Table 6 shows the inputs to each page of the SID Creator for describing the interface of the SBE-37SM. The pages for structure and task definition describe the structure of the data delivery command and response as shown in Table 2. The metadata definition page allows defining the fields of the data stream. The fields are further processed by the SID Interpreter and uploaded to an application (e.g., to a Sensor Observation Service). In this way, the values of those fields become associated with metadata such as an observed property and unit of measure.

Table 6: Inputs to the SID Creator wizard pages to describe the SBE-37SM protocol

Page	**SID Creator Input**	**Values for *SBE-37SM***
Structure Definition	Transmission Protocol	RS232
	Block separator	<CR><LF>
	Token separator	,
	Decimal separator	.
	Block 1 - Fields	temperature conductivity pressure salinity sound_velocity dateTime
Task Definition	**Command 1**	getDataCommand
	- Parameter Value	TS
Metadata Definition	**Output 1**	
	- Field Name	temperature
	- Observed property	http://sweet.jpl.nasa.gov/2.1/-propTemperature.owl#Temperature
	- Unit of measure	Cel
	Output 2	
	- Field Name	pressure
	- Observed property	http://sweet.jpl.nasa.gov/2.0/-hydro.owl#WaterPressure
	- Unit of measure	bar
	Output 3	
	- Field Name	salinity
	- Observed property	http://sweet.jpl.nasa.gov/2.0/-chemConcentration.owl#Salinity
	- Unit of measure	ppth

Although Seabird offers PUCK-enabled instruments, our Seabird SBE-37SM CTD does not natively implement the PUCK protocol; rather, it is connected to a serial port on the IP PUCK Smart Sensor Board (SSB) (4), which communicates over a LAN with the Host Controller (5). We stored the CTD SBE-37SM SID file in the SSB's PUCK payload. When the SSB is plugged into the network, it uses Zeroconf to automatically acquire a link-local address and name. The host controller (5) uses Zeroconf service discovery protocol to find the IP PUCK instrument in the network by means of the ZeroConf Browser (10a). When a new IP PUCK instrument is detected, the PUCK reader (10b) retrieves the instrument SID file from the SSB (4) and uses it to create new instance of an SID Interpreter (10c). The SID Interpreter subsequently generates instrument commands to trigger data acquisition, parses the instrument data and exports the parsed data to an SOS (6) and/or DataTurbine (7). A demonstration of the IP PUCK SSB is available on the World Wide Web [36].


implement PUCK protocol for non-PUCK instruments. Provide ethernet link
Detect IP PUCK service
retreive PUCK payload (SID file)
executes protocol defined in SID file. Export data to SWE or others
Host Controller [5]
Serial Instrument [1] Seabird CTD Protocol X
RS232
SmartInstrument [4] implements IP-PUCK
IP-PUCK mDNS [8]
IEEE1451 TEDS
SensorML SID file
Ethernet PUCK
ZeroConf mDNS [10a]
PUCK Reader [10b]
SID Interpreter [10c]
[10]
Serial Instrument [2] WETLabs Fluorometer Protocol Y
PUCK [8] no payload
RS232 - PUCK
PUCK Detector [9a]
SID Interpreter [9b]
[9]
SensorWebEnable SOS [11]
http
SOS Client [6]
Data Turbine Ring Buffer [12]
http
RDV Client [7]
Serial Instrument [3] Seabird CTD Protocol Z
PUCK [8]
IEEE1451 TEDS
SensorML SID file
RS232 - PUCK
SID repository [13]
for instruments without payload. PUCK tools write SID files from instruments SID Interpreter read SID files to communicate with instrument.
Figure 15 Sensor Web Enablement Architecture using PUCK protocol and SID

V. Conclusions and future work

In this paper, we identify a gap of interoperability between the variety of manufacturer specific instrument protocols and applications that need to access and use sensor data. To tackle this challenge, we have presented a software architecture for enabling end-to-end *plug & work* of sensor instruments with information infrastructures. By combining the standardized PUCK protocol and the SID concept, a promising way for automated instrument integration has been designed. We have demonstrated this new approach using instruments connected to the OBSEA test bed.

The developed architecture automates the *detection*, *identification*, *configuration* and *execution of measuring operations* at instrument level (Figure 1). In combination with OGC's SWE standards, it realizes the discovery, access, tasking and alerting on the sensor web level. Automating the interaction with the instrument and standardizing the representation of instrument protocols greatly reduces the costs and problems associated with writing and configuring instrument drivers. The PUCK protocol provides standard methods for the *detection* of a new instrument once plugged into a host controller or into a network, and *identification* via the PUCK payload. The combination of SID and PUCK enables an SID Interpreter to *automatically configure and execute measurement operations*. Coupling the SID Interpreter with SWE services enables the export of instrument data to the sensor web, closing the gap between the sensor web level and the instrument level.

With those concepts in place, we designed a framework that allows the development of tools, such as the SID Creator (section IV.B.2), which significantly decrease the administration efforts for integrating instruments with ocean observing systems.

In future, we will particularly focus on solving limitations of the current SID Interpreter implementation, which does not yet cover the full SID model, and we will also extend the SID specification itself to support a broader variety of instrument protocols. Examples of not yet supported protocols are fixed-width (e.g. HOBI Labs HydroScat) formats, delimited variable-length data records, or instruments that generate multiple data record formats. In these cases, the structure of each data record is usually indicated by a "record type" field in a "record header". A new implementation of the SID Interpreter is planned which tackles those limitations and is also lightweight enough to be deployed on limited embedded environments, e.g., on a low-powered buoy controller.

Another concept we plan to evolve is the repository for SID files. While in the presented architecture a basic local repository was sufficient, in future, global and publicly accessible repositories for sensor interface descriptions shall be established. Such repositories can act as platforms for instrument manufacturers and users to exchange their knowledge on instrument protocols. As a web service, a SID repository could be incorporated in the suite of SWE specifications, comparable to but extending the existing Sensor Instance Registry [34].

VI. REFERENCES

[1] Aguzzi, Jacopo; Mànuel, Antoni; Condal, Fernando; Guillén, Jorge; Nogueras, Marc; del Rio, Joaquin; Costa, Corrado; Menesatti, Paolo; Puig, Pere; Sardà, Francesc; Toma, Daniel; Palanques, Albert. 2011. "The New Seafloor Observatory (OBSEA) for Remote and Long-Term Coastal Ecosystem Monitoring." Sensors 11, no. 6: 5850-5872.

[2] Mànuel, A.; Nogueras, M.; Del Rio, J. "OBSEA: an expandable seafloor observatory" Sea technology, ISSN: 0093-3651

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

[4] Botts, M., Percivall, G., Reed, C., & Davidson, J. (2008). OGC Sensor Web Enablement: Overview And High Level Architecture. *GeoSensor Networks*, Lecture Notes in Computer Science (pp. 175-190). Heidelberg: Springer.

[5] Botts, M., & Robin, A. (2007). *Sensor Model Language* (Implementation specification No. 07-000). Wayland, MA, USA: OGC. Retrieved from http://www.opengeospatial.org/standards/sensorML

[6] Bröring, A., S. Below & T. Foerster (2010): Declarative Sensor Interface Descriptors for the Sensor Web. In: Brovelli, M., Dragicevic, S., Li, S. & Veenendaal, B. (Eds.), WebMGS 2010: 1st International Workshop on Pervasive Web Mapping, Geoprocessing and Services. 26.-27. August 2010. Como, Italy. ISPRS, 2010, Volume 38.

[7] Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S. & Lemmens, R. (2011). New Generation Sensor Web Enablement. *Sensors* 2011, 11(3), 2652-2699.

[8] Bröring, A., Stasch, C. & Echterhoff, J. (2011, candidate standard). Sensor Observation Service 2.0 (Implementation specification No. 10-037). Wayland, MA, USA: OGC. Retrieved from http://www.opengeospatial.org/standards/requests/73

[9] Bröring, A., F. Bache, T. Bartoschek & C.P.J.M. van Elzakker (2011): The SID creator: A Visual Approach for Integrating Sensors with the Sensor Web. In: S. Geertmann, W. Reinhardt & F. Toppen (Eds.), Advancing Geoinformation Science for a Changing World, The 14th AGILE International Conference on Geographic Information Science. 18.-21. April 2011. Utrecht, Netherlands. Lecture Notes in Geoinformation and Cartography, Springer, Volume 1, pp 143-162.

[10] Cox, S. (2007). *Observations and Measurements - Part 1 - Observation schema* (Implementation specification No. 07-022r1). Wayland, MA, USA: OGC. Retrieved from http://www.opengeospatial.org/standards/om

[11] Jirka, S., A. Bröring & C. Stasch (2009): Discovery Mechanisms for the Sensor Web. *Sensors* 2009, 9(4), 2661-2681.

[12] Tidwell, J. (2006): *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly.

[13] Daniel Mihai Toma, Tom O'Reilly, Joaquin del Rio, Kent Headley, Antoni Manuel, Arne Bröring, Duane Edgington, "Smart Sensors for Interoperable Smart Ocean Environment", IEEE OCEANS11, 6-9 June, Santander, Spain.

[14] Daniel Steinberg, Stuart Cheshire, Zero Configuration Networking: The Definitive Guide, O'Reilly Media, Inc., 2005

[15] IEEE P1451.1, Draft Standard for a Smart Transducer Interface for Sensors and Actuators - Network Capable Application Processor (NCAP) Information Model. Institute of Electrical and Electronics Engineers, Inc., New York, to be submitted, 1996.

[16] IEEE P1451.2, Draft Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats. Institute of Electrical and Electronics Engineers, Inc., New York, August 1996

[17] Kent L. Headley, Thomas C. O'Reilly, et al, "Managing Sensor Network Configuration and Metadata in Ocean Observatories Using Instrument Pucks" Third International Workshop on Scientific Use of Submarine Cables and Related Technologies, 25 27 June 2003.

[18] Joaquín del Río, Yves Auffret, Daniel Mihai Toma, Shahram Shariat, Xavier André, Stéphane Barbot, Eric Menut, Yannick Lenault, Antoni Manuel, Oussama Kassem Zein, Joel Champeau, Dominique Kerjean. "Smart Sensor interface for sea bottom observatories". MARTECH'09.. INTERNATIONAL WORKSHOP ON MARINE TECHNOLOGY. November 2009.

[19] L. Kang, "IEEE 1451: A standard in support of smart transducer networking," presented at Instrumentation and Measurement Technology Conference, 2000. IMTC 2000. Proceedings of the 17th IEEE, 2000.

[20] E. Y. Song and L. Kang, "Understanding IEEE 1451-Networked smart transducer interface standard – What is a smart transducer?," Instrumentation & Measurement Magazine, IEEE, vol. 11, pp. 11-17, 2008.

[21] Joaquín del Río, Marc Martínez, Daniel Mihai Toma, Antoni Mànuel, Helena G. Ramos.”IEEE 1451 HTTP Server Implementation for Marine Data”. Marine Technology Workshop MARTECH 2011. September 2011, Cadiz, Spain

[22] IEEE Std 1451.0™-2007. IEEE 1451.0, Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats.

[23] On line. July 25, 2011. http://www.gwm-engineering.fi/RBR%20Data%20Buoy.pdf

[24] Robert Herlien, Tom O'Reilly, Kent Headley, Duane R. Edgington, Sameer Tilak, Tony Fountain, Peter Shin “An Ocean Observatory Sensor Network Application”, IEEE Sensors, 01/11/2010, Hawaii

[25] Sameer Tilak; Paul Hubbard; Matt Miller; Tony Fountain. “ The Ring Buffer Network Bus (RBNB) DataTurbine Streaming Data Middleware for Environmental Observing Systems”, e-Science, 10/12/2007, Bangalore, India

[26] Searle, Ben, "Development of a Marine XML", Sixteenth Session of the IOC Committee on International Oceanographic Data and Information Exchange (IODE), 2000, http://hdl.handle.net/1834/1627

[27] Ocean Data Standards. Online: 25 July 2011. http://www.oceandatastandards.org/

[28] Australian Ocean Data Center. Online 25 July 201. http://www.aodc.gov.au/index.php?id=37

[29] European Comission Inspire. Online 25 July 2011. http://inspire.jrc.ec.europa.eu/ Annex III section 14&15

[30] ISO 19115 Metadata Standard for Geographic Information, Online 25 July 2011. http://www.iso.org

[31] Na, A., Priest, M. (2007). OpenGIS Sensor Observation Service (SOS). Version 1.0 (Implementation specification No. 06-009r6). Wayland, MA, USA: OGC.

[32] Simonis, I. (2007). OpenGIS Sensor Planning Service (SPS) (Implementation specification No. 07-014r3). Wayland, MA, USA: OGC.

[33] Best Practices ESONET Workshop.[On line] 25 July 2011: http://www.esonet-noe.org/News-and-events/Esonet-Workshops-and-meetings

[34] Jirka, S., A. Bröring & C. Stasch (2009). Discovery Mechanisms for the Sensor Web. *Sensors* 2009, 9(4), 2661-2681.

[35] National Marine Electronics Association: NMEA 0183.[On line] http://www.nmea.org/content/nmea_standards/nmea_083_v_400.asp

[36] Automated installation and operation of sensors in an IP network.[On line] http://www.youtube.com/watch?v=W9qjX_rREWY

[37] O'Reilly, T. (2011). Candidate PUCK Protocol Standard (OGC Candidate Encoding Standard No. 09-127r1). Wayland, MA, USA: OGC. Retrieved from http://www.opengeospatial.org/standards/requests/78