“Bringing together Research and Industry for the Development of Glider Environmental Services”

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 635359

DELIVERABLE D3.3

“Interface standards for applications of deep and ultra-deep glider”

Abstract

The overall goal of BRIDGES is to develop two autonomous gliders to operate in the deep-sea environment (up to 5000m). The use of the acquired knowledge from previous glider projects such as the SEA EXPLORER product, the SPAN vehicle, and the AUTOSUB LR from NOC will be an asset and helpful for this purpose.

Current glider platforms capture the data of the payload and navigation sensors in a proprietary manner. Though the storage of the gathered data is harmonized to a certain extent (i.e. using the EGO Glider profile for NetCDF), discovery and access of these is not established in an interoperable way that allows straight-forward integration of different platforms.

This document provides an assessment of the current state of glider data storage and retrieval, identifies the existing gaps for further data exploitation and provides a proposal for a standards based approach for the discovery and access of glider data. The Sensor Web Enablement (SWE) standards family of the Open Geospatial Consortium (OGC) will be used to outline a service infrastructure that allows data analysts to easily search for and integrate glider data of different vendors.

During the remainder of the BRIDGES project this document will be further refined, taking also into account experiences gathered through prototyping activities and exchange with other relevant projects.

Document Type	Deliverable
Document Name:	BRIDGES_D3.3_Interface standards for applications of deep and ultra-deep glider_vfinal.docx
Version:	vfinal
Date:	13.05.2016
Status:	S0
Dissemination level:	PU

Authors, Reviewers				
Author(s):	Matthes Rieke, Simon Jirka			
Affiliation(s):	52°North GmbH			
Further Authors:	Ehsan Abdi, Justin Buck, Louise Darroch, Daniel Hayes, Alexandra Kokkinaki, David White, Michael Field, Laurent Beguery			
Peer Reviewers:	Daniel Hayes, Michael Field			
Review Approval:	Approved	Yes	Rejected (to be improved as indicated below)	No
Remarks / Improvements:				

Version History			
Version:	Date:	Comments, changes, Status:	Person(s) / Organisation Short Name:
0.1	01/02/2016	Initial version	Matthes Rieke, Simon Jirka
0.2	06/02/2016	Sections on Related Work and Approach	Matthes Rieke
0.3	07/02/2016	Diagrams for system architecture and O&M model	Matthes Rieke
0.4	07/02/2016	Current state of Glider data workflow (Seaglider)	Daniel Hayes, Ehsan Abdi
0.5	17/02/2016	Sections on Related Work (OGC, ISO)	Simon Jirka
0.6	22/02/2016	Sections on Related Work (Research Projects)	Justin Buck, Simon Jirka
0.7	11/03/2016	Abstract, SensorML Approach, System Architecture	Simon Jirka, Matthes Rieke
0.8	31/03/2016	First full version for review	Simon Jirka
1.0	03/05/2016	Integrating Review Feedback; SEA EXPLORER content	Michael Field, Laurent Beguery, Matthes Rieke
1.1	11/05/2016	Integrate additional Review Feedback	Daniel Hayes, Matthes Rieke
VFINAL	13/05/2016	Final version	Matthes Rieke

Version Numbering	
v0.x	draft before peer-review approval
v1.x	After the first review
v2.x	After the second review
vfinal	Deliverable ready to be submitted!

Status / Dissemination Level			
Status		Dissemination Level	
S0	Approved/Released/Ready to be submitted	**PU**	Public
S1	Reviewed	**CO**	Confidential, restricted under conditions set out in the Grant Agreement
S2	Pending for review		
S3	Draft for comments	**CI**	Classified, information as referred to in Commission Decision 2001/844/EC.
S4	Under preparation		

TABLE OF CONTENTS

FIGURES

TABLES

LISTINGS

CONTENT

1 Introduction

The overall goal of this project is to develop two autonomous gliders to operate in the deep-sea environment (up to 5000m). The use of the acquired knowledge from previous glider projects such as the SEA EXPLORER product, the SPAN vehicle, and the AUTOSUB LR from NOC will be an asset and helpful for this purpose. The diversity of autonomous ocean observing platforms and sensors is growing massively. At the same time standardised data exchange and formats from other disciplines are being adopted and adapted by the oceanographic data community (e.g. the SWE developments within the ODIP and ODIP II projects). Use of such standards will reduce the number of proprietary formats and protocols. It is hoped that in the long term standardisation will reduce the cost of data dissemination, dataset integration and long-term archiving of valuable data.

Current glider platforms capture the data of the payload and navigation sensors in a proprietary manner. Though the storage of the gathered data is harmonized to a certain extent (i.e. using the EGO Glider profile for NetCDF), discovery and access of these is not established in an interoperable way that allows straight-forward integration of different platforms.

This document provides an assessment of the current state of glider data storage and retrieval, identifies the existing gaps for further data exploitation and provides a proposal for a standards based approach for the discovery and access of glider data. The Sensor Web Enablement (SWE) standards family of the Open Geospatial Consortium (OGC) will be used to outline a service infrastructure that allows data analysts to easily search for and integrate glider data of different vendors.

In summary, this document is intended to provide recommendations on how Sensor Web technology may be used to facilitate the distribution of collected glider data and to support the integration of this data with observations and other geospatial data from other sources. Thus, the re-use of glider observations will become easier for scientists. At the same time, glider operators will receive support and guidance how to optimise glider data management and distribution.

2 Reference Documents

Abramowitz, M., Stegun, I. A., Eds., Handbook of Mathematical Functions (Applied Mathematics Series 55), Washington. DC: NBS, 1964, pp. 32-33.

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

Bröring, A., J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, R. Lemmens, "New Generation Sensor Web Enablement," MDPI Sensors, vol. 11, pp. 2652-2699, 1 March 2011 2011.

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

Carval, T. et al, EGO gliders User's manual. 2013. Online at: http://dx.doi.org/10.13155/34980

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

Devaraju, A., S. Jirka, R. Kunkel, J. Sorg, "Q-SOS - A Sensor Observation Service for Accessing Quality Descriptions of Environmental Data". ISPRS International Journal of Geo-Information, vol. 4, pp. 1346-1365, 10 August 2015 2015.

Echterhoff, J., T. Everding, "OGC Discussion Paper: OGC Sensor Event Service (SES) 0.3.0 (08-133)". Wayland, MA, USA: Open Geospatial Consortium Inc., 2008.

INSPIRE Cross Thematic Working Group on Observations & Measurements, "D2.9 Guidelines for the use of Observations & Measurements and Sensor Web Enablement-related standards in INSPIRE Annex II and III data specification development,". Ispra, Italy: European Commission Joint Research Centre, 2014.

ISO TC 211, "ISO 19115-1:2014 Geographic information -- Metadata -- Part 1: Fundamentals". Geneva, Switzerland: International Organization for Standardization, 2014.

ISO TC 211, "ISO 19156:2011 - Geographic information -- Observations and measurements - International Standard". Geneva, Switzerland: International Organization for Standardization, 2011.

Jirka, S., D. M. Toma, J. del Rio, 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.

Klaus, B., P. Horn, Robot Vision. Cambridge, MA: MIT Press, 1986.

Müller, M., B. Proß, "OGC Implementation Specification: Web Processing Service (WPS) 2.0.0 (14-065)". Wayland, MA, USA: Open Geospatial Consortium Inc., 2015.

Myer, R. L., "Parametric oscillators and nonlinear materials," in Nonlinear Optics, vol. 4, P. G. Harper and B. S. Wherret, Eds. San Francisco, CA: Academic, 1977, pp. 47-160.

O'Reilly, T., "OGC Implementation Specification: OGC PUCK Protocol 1.4 (09-127r2)". Wayland, MA, USA: Open Geospatial Consortium Inc., 2012.

Simonis, I., "OGC Best Practice: Sensor Alert Service (SAS) 0.9 (06-028r3)". Wayland, MA, USA: Open Geospatial Consortium Inc., 2006.

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

Stein, L., “Random patterns,” in Computers and You, J. S. Brake, Ed. New York: Wiley, 1994, pp. 55-70.

3 Glossary

AUV	Autonomous Underwater Vehicle
CSW	Catalogue Service for the Web
CTD	Conductivity Temperature Depth
DAC	Data Assembly Centre
DBMS	Database Management System
EGO	Everyone’s Gliding Observatories
ERDDAP	Environmental Research Division's Data Access Program
EXI	Efficient XML Interchange
GEOSS	Global Earth Observing System of Systems
GIS	Geographic Information System
GOOS	Global Ocean Observing System
ISO	International Standards Organisation
JSON	JavaScript Object Notation
JSON-LD	JSON for Linking Data
KVP	Key Value Pair
NetCDF	Network Common Data Format
NOAA	National Oceanic and Atmospheric Administration
O&M	Observations & Measurements
OGC	Open Geospatial Consortium
OPeNDAP	Open-source Project for a Network Data Access Protocol
OWL	Web Ontology Language
RDF	Resource Description Framework
SensorML	Sensor Model Language
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SSN	Semantic Sensor Networks
SWE	Sensor Web Enablement
UoM	Unit of Measurement

URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

4 Related Work

4.1 International Standards

4.1.1 Sensor Web Enablement

The Sensor Web Enablement (SWE) framework developed by the Open Geospatial Consortium (OGC) aims to develop and maintain standards for the interoperable integration of sensors and their observation data into Web-based (spatial) data infrastructures (Bröring et al., 2011). There exist several document types within the OGC, representing the maturity of a specification (e.g. discussion paper, best practice paper or standard). A specification can be understood as a technical definition for a web service or data model (independent of the grade of maturity) while a standard is the document that has been officially adopted by the OGC. The following subsections introduce the core elements of the SWE architecture which comprises specifications for Web service interface as well as data models and encodings.

4.1.1.1 Sensor Observation Service

The OGC Sensor Observation Service (SOS) interface standard (Bröring et al., 2012) is the most widely used Web service of the SWE suite. It has reached version 2.0 which incorporates experiences gained during practical application of the first SOS version over several years.

The SOS interface allows pull-based access to observation data as well as sensor metadata. This means that the SOS acts as a mediator between clients and a measurement archive (e.g. database) or sensor system. Through the SOS, it is possible for clients to query observation data of heterogeneous sources via a standardized interface. On the one hand the SOS standard defines a set of operations and their parameters and on the other hand it relies of the data model/encoding standards of the SWE framework to provide standardised outputs.

The core operations of the SOS interface are:

- GetCapabilities: Retrieve metadata about a SOS server (e.g. supported operations and available data sets)
- DescribeSensor: Access metadata about the sensors or processes which have generated the observation data offered by the SOS server
- GetObservation: Retrieval of observation data/measurements

An important extension of the SOS interface is a group of transactional operations (InsertSensor and InsertObservation) for publishing new sensors and observations data on a SOS server

Another important operation is the GetFeatureOfInterest operation which allows the retrieval of the geometric features to which observations are associated. It provides the required spatial context, by serving e.g. point or polygon features of the feature that is being observed (see section 4.1.1.5 for an illustration of the relationship between observations and features).

Figure 1 illustrates the four interface methods and their corresponding response formats.

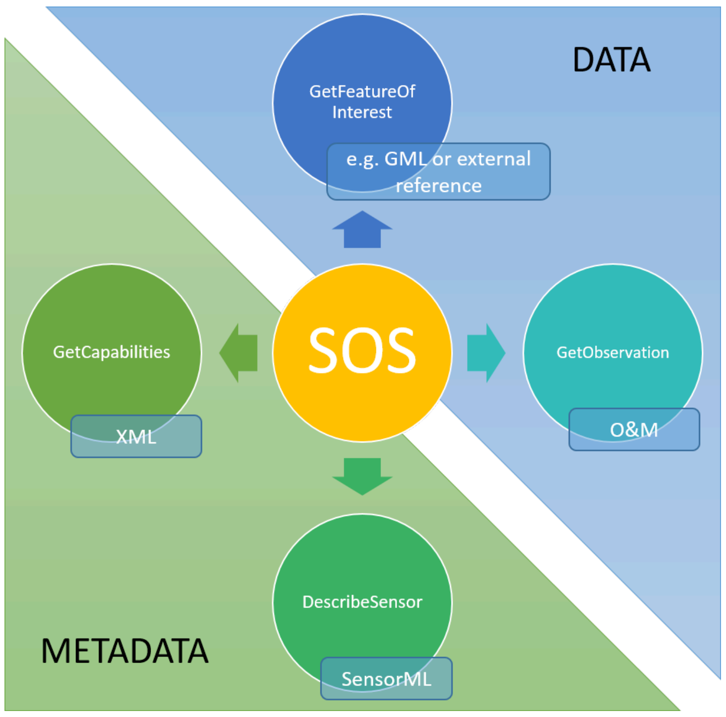


Figure 1: SOS Interface Method Overview

4.1.1.2 Sensor Planning Service

The OGC Sensor Planning Service (SPS) (Simonis and Echterhoff, 2011) interface standard offers functionality for controlling sensors and measurement processes. This means that the SPS is not suited for accessing observation data but to control the process how this data is generated.

Important operations of the SPS interface, which is also available in version 2.0, comprise:

- GetCapabilities: Retrieve metadata about a SOS server (e.g. supported operations, sensors which can be controlled, tasks that can be executed by the sensors)
- DescribeTasking: Access information on how to formulate tasking requests (i.e. required parameters and their types)
- GetFeasibility: Checking whether a task for a specific sensor is feasible or not (e.g. a sensor might be blocked at a specific point in time by another task)
- Submit: Send a tasking request to an SPS server
- GetStatus: Determining the status of the execution of a task
- DescribeResultAccess: Determine how the data collected as result of a task can be accessed (e.g. this operation might return a reference to a SOS server that provides the collected data)

4.1.1.3 Eventing

While the SOS offers pull-based data access (i.e. a request-response communication pattern) in some cases it is necessary to deliver observation data to a consumer as soon as the data is available (e.g. in alerting applications).

Such functionality requires push-based, asynchronous delivery of sensor data. Such an eventing mechanism is based on a publish/subscribe communication pattern: A consumer subscribes for a notification, the eventing component analyses incoming sensor data, and forwards the new (relevant) observations to the subscriber in (near) real-time.

Within the OGC SWE framework, there is not yet a corresponding adopted standard available. However there are several specifications available:

- OGC Sensor Alert Service (SAS) (Simonis, 2006): This specification is rather old and was published as an OGC Best Practice Paper. However it has some drawbacks because it is relatively tightly coupled to a specific communication protocol (XMPP) and it does not support complex event processing concepts for detecting relevant events.
- Sensor Event Service (SES) (Echterhoff and Everding, 2008): This specification (not released as an official standard, but available as discussion paper) can be considered as successor of the SAS: It supports the definition of complex event patterns that shall be detected. For defining these rules the SES relies on the OGC Event Pattern Markup Language (EML) Discussion Paper. However, the SES has not been advanced to an official standard.
- OGC Pub/Sub standard[1]: This standard has been adopted in 2016 and it offers a specification how to implement publish/subscribe communication for OGC Web services. Thus, this standard should be considered as basis for implementing eventing functionality. However, the OGC Pub/Sub standard goes beyond the SWE framework and it does not offer further details on how to define patterns for event subscriptions. This needs to be specified as an addition.
- OGC Web Processing Service (WPS) (Müller and Pross, 2015): In the OGC IMIS IoT Pilot (2015-2016) an event processing profile of the Web Processing known as Web Event Processing Service or WEPS, was discussed. It can be expected that this profile together with the OGC Pub/Sub standard may help to build interoperable eventing applications.

4.1.1.4 SensorML

While the previous three specifications define Web service interfaces for sensor related functionality, the OGC Sensor Model Language (SensorML) 2.0 (Botts and Robin, 2014) offers a data model and XML encoding for metadata about sensors and measurement processes (thus, the SOS uses SensorML as a response format of the DescribeSensor operation). Measurement processes can range from singular sensor stations to complex sensor platforms that form a system which also describes measurement processing steps in high detail (e.g. that and how a normalization process is applied).

Typical elements which may be included in a SensorML document comprise for example:

- Keywords characterising a sensor
- Identifiers (e.g. serial numbers, sensor names/ids)
- Classifiers (e.g. information about application domains of a sensor)
- Characteristics (e.g. size, weight of a sensor)
- Capabilities (e.g. resolution, sampling rate, etc. of a sensor)
- Valid time (information for which time period a sensor description is valid)
- Input and outputs of a sensor or measurement process

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

- Contact information (e.g. of the sensor operator, responsible scientist, manufacturer)

The SensorML standard has intentionally been defined in an application and domain independent manner. This means that only a very small set of mandatory information is provided in all SensorML documents. To increase interoperability between different communities, it is important to develop sensor profiles with additional restrictions and requirements that specify that certain information will be provided in a certain way. In this context it is important to refer to the ongoing activities for developing a marine profile for SensorML. BRIDGES is contributing to these efforts together with other projects such as those listed in **Error! Reference source not found.**.

4.1.1.5 Observation & Measurements

Complementary to the metadata model and encoding of SensorML 2.0, the Observation and Measurements (O&M) 2.0 standard offers a model (ISO TC 211, 2011) and an XML encoding (Cox, 2011) for data observed by sensors (archived/delayed-mode as well as real-time data). For O&M it is important to state that the data model has been adopted as an ISO standard while the XML encoding is an OGC standard.

Typical information required for an observation conforming to the O&M standard comprises:

- Process: The sensor, model, or algorithm which has delivered the observed value
- Observed Property: The parameter which is observed (e.g. water temperature)
- Feature of Interest: The geographic feature to which the observed value is associated
- Observation Result: The measured value itself (including unit of measurement, if applicable)
- Time Stamps: Different time stamps relevant to the observation

Figure 2 illustrates the relationship of the above described concepts.

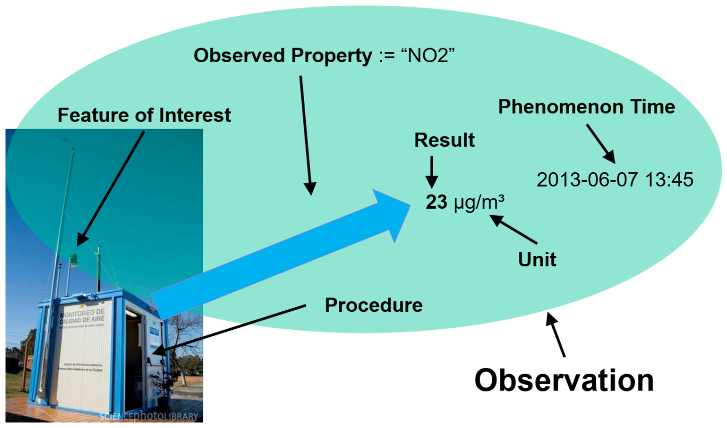


Figure 2: Observation & Measurements Example

4.1.1.6 OGC Standards and EXI

XML documents may become quite large due to the way data are encoded. In certain applications (e.g. sensor platforms that have limited communication bandwidth such as gliders on the open ocean) there is a need to reduce the data volume. To allow a more efficient transmission of XML documents, the World Wide Web Consortium (W3C) has developed the Efficient XML Interchange (EXI)[2] format which ensures a compact

[2] https://www.w3.org/TR/exi/

representation of XML documents. Within the NeXOS project this approach has already been successfully tested, so it can now be recommended for BRIDGES as well.

4.1.1.7 PUCK Protocol Standard

The OGC PUCK standard (O'Reilly, 2012) is a complementary element of the SWE framework. It is intended to ensure interoperability between sensor platforms and instruments by defining mechanisms for automatic configuration.

The idea of PUCK is that instruments shall carry and be able to provide all information needed for connecting to the instrument and accessing its data. For this purpose, PUCK provides a simple protocol to access this configuration information from an instrument via communication channels such as RS-232 and Ethernet.

The information provided by an instrument (an instrument datasheet) includes at least a unique identifier (serial number) of the instrument, an identifier of the manufacturer, and a few further metadata elements.

While the other SWE standards are usually implemented on a higher abstraction layer, PUCK is intended to be implemented in the firmware of instruments.

For BRIDGES, PUCK is relevant for facilitating the link between instruments and Gliders, providing a harmonized access to metadata and data, but not for the communication between Gliders and stations on the shore.

4.1.2 NetCDF

The Network Common Data Form (NetCDF)[3] is a data format that is designed for sharing array-oriented scientific data. Developed by the University Corporation for Atmospheric Research (UCAR), the latest version 4.1 has support of many libraries from all major established programming languages and client software is available for all common operating systems.

A NetCDF dataset is always self-describing, i.e. it provides metadata in the header that provides details on the contained data and corresponding units of measure. Platform independency is achieved by a set of software libraries, many of them available as open source software.

NetCDF is well-established in many scientific domains such as climatology, meteorology and oceanography. It is a supported input/output in many GIS applications and data analysis tools.

4.1.2.1 EGO Profile

Everyone's Gliding Observatories (EGO) is an international network of partners in the oceanology domain dedicated to the promotion of glider technology and its applications. Besides coordination, workshops and training, EGO also works on the definition of a NetCDF profile to streamline the storage of glider-gathered data sets.

The current version of the EGO gliders User's manual (Carval et al., 2013) firstly defines a workflow for data management and access that takes the roles of three organizational units (Principal Investigator, Data Assembly Centre (DAC), Global DAC) into account. The goal is

[3] http://www.unidata.ucar.edu/software/netcdf/

to ensure data quality among different glider operators and to provide transparency on the quality of data sets to users. All data sent to Ifremer and the EGO GADC following the EGO process is made freely available to everyone.

The NetCDF EGO implementation is based on the community-supported Climate and Forecast specification. This specification defines standard vocabulary and metadata conventions. EGO extends these conventions with a special focus on in-situ measurements and to support the EGO workflow described above. The profile covers data structures for navigational properties (trajectory data based on GPS and inertial sensors), global attributes (spatial and temporal coverage, producer information, etc.) and the actual measured data variables (e.g. salinity, water temperature).

The system architecture proposed in this document makes full use of the NetCDF EGO profile in order to benefit from its establishment among the glider community. Following this approach ensures that existing systems can continue to operate in the same manner and facilitates the acceptance of the architecture design.

4.1.3 W3C Semantic Sensor Networks (SSN)

The W3C Semantic Sensor Network Incubator Group has worked on the development of an ontology for describing sensors and sensor networks in the context of Sensor Web applications. Furthermore the SSN Incubator Group has provided recommendations on how this ontology may be used for enabling SWE based infrastructures. This group has completed its work and the resulting findings are documented in a final report[4]:

The resulting Sensor and Sensor Network ontology (SSN ontology) combines aspects related to both sensor descriptions (SensorML) as well as observation models (i.e. O&M). Thus, it offers a way to capture semantics of observation data and the underlying processes. This allows to comprehend how data was captured, processed as well as embedding observations into a context (e.g. by linking to a phenomena ontology).

In addition, the results of the SSN Incubator Group also provide direction on how to end existing SWE infrastructures with semantic annotations.

Although implementation of this ontology in productive systems are limited, its concepts can be applied within the BRIDGES SWE architecture, e.g. by linking the observed properties of a glider to existing vocabularies such as the British Oceanographic Data Centre (BODC) Dictionary.

4.1.4 ISO 19115 - Geographic Metadata

The ISO 19115 family of standards offers standardised models as well as encodings for metadata describing geospatial information resources. For BRIDGES, especially the following two documents are relevant:

- ISO 19115-1:2014: Geographic information -- Metadata -- Part 1: Fundamentals: Defines a model for describing geospatial information resources (ISO TC 211, 2014)
- ISO/DTS 19115-3: Geographic information -- Metadata -- Part 3: XML schema implementation of metadata fundamentals: This standard, which is currently in development provides an encoding for the model specified in ISO 19115-1

[4] https://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/

These two documents are complemented by ISO 19139 which provides an XML encoding for the metadata handled by a Catalogue Service for the Web (CSW) server implementing the ISO Metadata Application Profile. A CSW can thus be used to discover glider data served by SOS instances.

For BRIDGES, the ISO 19115 standard family is relevant to provide geospatial metadata. While there is no standard explicitly supporting the description of sensors, ISO 19115 can be used for indirectly describing the sensors through the data sets they are acquiring.

4.2 Existing Software Solutions

This sections provides a brief overview of existing software packages that relate to the current state of glider software or the proposed approach described in section 7.

4.2.1 ncSOS

Based on the THREDDS[5] (Thematic Real-time Environmental Distributed Data Services) server, the ncSOS[6] provides an OGC SOS 1.0.0 implementation with a special focus on the NetCDF data format. Using the underlying THREDDS instance, this implementation maps and outputs NetCDF files to the corresponding SOS interface methods. For example, a DescribeSensor request (see section 4.1.1.1) will retrieve the required information from the header part of the NetCDF file. It has full support for geometries defined in the CF-1.6 Discrete Sampling Geometries which are also used by the NetCDF EGO Profile.

4.2.2 ERDDAP

The Environmental Research Division's Data Access Program (ERDDAP) is a web service software, developed by NOAA, which provides harmonized access to data stored in many different data formats or even different web services. It does not only provide data format transformations but also a URL-based API, based on the OPeNDAP (Open-source Project for a Network Data Access Protocol) specification, to specifically define filters (e.g. spatial and temporal, keyword or category filters).

The basic concept of ERDDAP is the mapping of existing data sets into the two internal data set formats: *griddap* and *tabledap.*

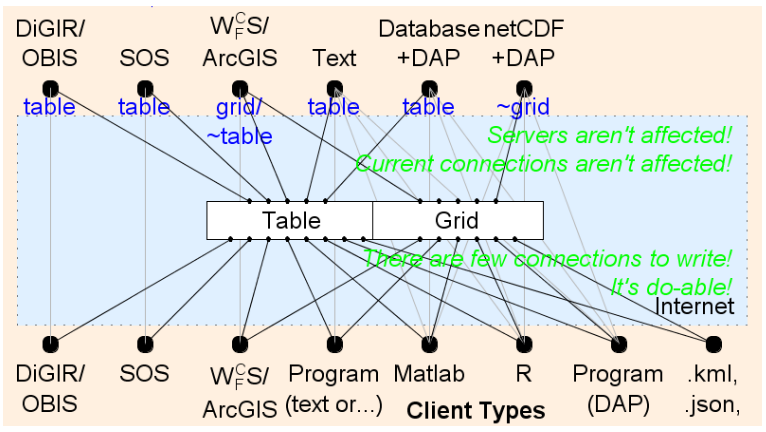


Figure 3: ERDDAP service types (Source: ERDDAP Tech Talk[7])

[5] http://www.unidata.ucar.edu/publications/factsheets/2007sheets/threddsFactSheet-1.pdf
[6] https://github.com/asascience-open/ncSOS
[7] http://coastwatch.pfel.noaa.gov/erddap/images/erddapTalk/erddapTechTalk.html

For example, classic coverage data (e.g. satellite imagery) is stored as a griddap structure, whereas in-situ time series data will be held as tabledap. This segmentation into two categories makes it easy to transform between different data formats. The user can request time series data for a given measurement station, ERDDAP would request an SOS, transform the data into the internal tabledap format and then return it to the user in the desired format (e.g. NetCDF).

4.2.3 52°North SOS

The 52°North Sensor Observation Service 4.x[8] implements the OGC SOS standard versions 1.0.0 and 2.0. The implementation comprises all extensions defined in the specification.

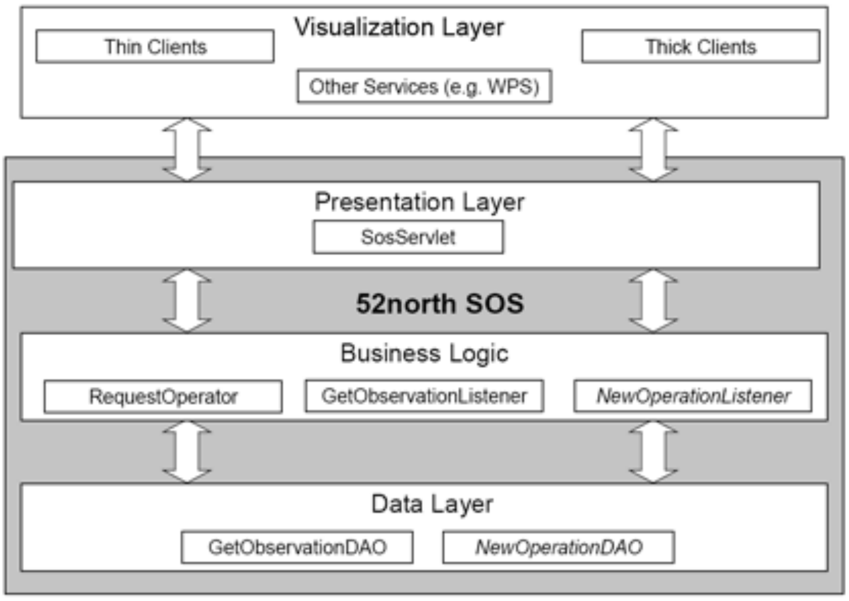


Figure 4: Layered architecture of the 52°North Sensor Observation Service

With its layered architecture, shown in Figure 4 the SOS can be flexibly connected to different data sources ranging from file-based approaches to different database systems. By default, PostGIS is used as the Database Management System (DBMS). By customizing the Business Logic layer, new functionality may be added. Thereby, new encoders and decoders can be added in a plug and play fashion. For example, prototypical support for an EXI encoding for SOS messages, O&M as well as SensorML has already been implemented in the European research project NeXOS[9].

4.2.4 Geoinformation Enabling Toolkit

The Geoinformation Enabling Toolkit (GET-IT)[10], formerly known as RITMARE StarterKit, is a software suite developed to help research data production units to establish their own data provision servers. It utilizes the 52°North SOS implementation as a data backend and GeoNode as map data provider. The user can manage, share, visualize and download the underlying data.

[8] www.52north.org/sos
[9] http://www.nexosproject.eu/
[10] https://github.com/SP7-Ritmare/starterkit

In addition to the provision of data, GET-IT features a metadata editing tool (see [11] for detailed information) which simplifies the creation and management of metadata in order to fulfil certain criteria (e.g. INSPIRE compliancy).

4.3 Research Projects

The following research projects conduct work in the oceanographic domain and are therefore relevant for BRIDGES as they represent a large marine community of sensor metadata and observation data providers, managers, and users.

4.3.1 The Ocean of Tomorrow Projects

In the context of the Seventh Framework Programme (FP7), the European Commission has launched a series of projects which are summarised under to topic 'The Ocean of Tomorrow'. This initiative is focused on bringing researchers from different domains together in order to develop new solutions for marine and maritime challenges.

Among the different topics addressed by this programme, the projects funded under the topic "Innovative Multifunctional Sensors" (FP7-OCEAN-2013-2) are particularly relevant to the work performed within BRIDGES. These projects are described in the following paragraphs in more detail:

- COMMON SENSE
- NeXOS
- SCHeMA
- SenseOcean

An overview of all Ocean of Tomorrow projects has been published by the EC in a dedicated book[12].

4.3.1.1 COMMON SENSE

The COMMON SENSE (Cost-effective sensors, interoperable with international existing ocean observing systems, to meet EU policies requirements) project[13] aims at developing innovative and cost-effective sensors. The data collected by these sensors shall be made available in a standardised, and thus interoperable, manner. The motivation for following this approach is to facilitate the integration of measured data into larger-scale systems such as the Global Ocean Observing System (GOOS) and the Global Earth Observing System of Systems (GEOSS).

In this context, the COMMON SENSE project also envisions a Sensor Web Platform based on the OGC SWE standard. This platform will rely on SWE both for the transfer of data from sensors on the platform and also for providing data access to external users.

This approach is fully in-line with the Sensor Web architecture planned for the BRIDGES project. In order to ensure a harmonised approach it is important to organise an exchange with partners of the COMMON SENSE project (e.g. for the development of a marine SensorML profile which can be re-used across multiple projects).

[11] https://media.readthedocs.org/pdf/getit/latest/getit.pdf
[12] http://ec.europa.research/eu/bioeconomy/pdf/ocean-of-tomorrow-2014_en.pdf
[13] http://www.commonsenseproject.eu/

Furthermore, the COMMON SENSE project is considering the 52°North Sensor Web implementations (i.e. the SOS) for building the Sensor Web Platform. Also planned work of COMMON SENSE on a sensor registry (potentially based on the 52°North Sensor Instance Registry) might lead to results that are also of interest for the BRIDGES project.

4.3.1.2 NeXOS

The NeXOS (Next generation Low-Cost Multifunctional Web Enabled Ocean Sensor Systems Empowering Marine, Maritime and Fisheries) project[14] aims at the development of innovative, integrated multifunctional sensor systems (e.g. optics and passive acoustics) that can be deployed on mobile and fixed platforms. This is combined with the development of an interoperable Sensor Web architecture which facilitates the sharing of the collected data through standardised interfaces, data models and formats.

Within this project an approach is developed to facilitate the data flow from the sensor hardware into central Sensor Web servers. For this purpose developments are performed on a level closer to the hardware (e.g. intelligent data loggers, such as those implementing PUCK, that are able to use OGC SWE protocols for transmitting data) as well as on a Web server level (e.g. improved OGC Sensor Observation Service implementations to manage collected sensor data as well as corresponding viewer tools).

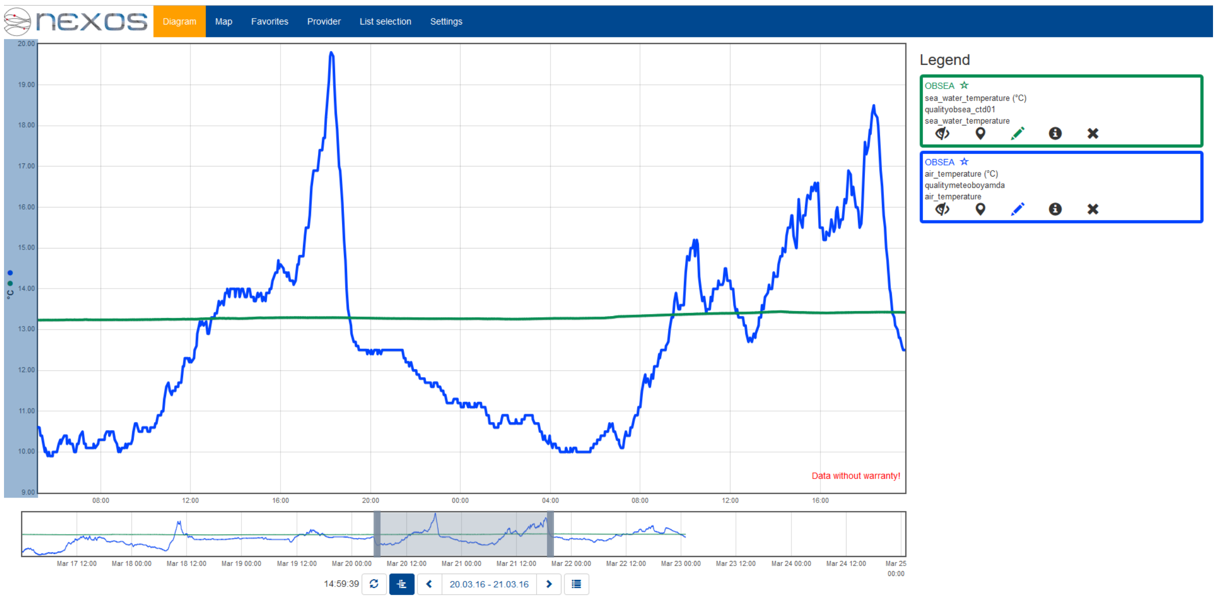


Figure 5: NeXOS Sensor Observation Service Viewer

Within BRIDGES it will be possible to build upon the open source developments resulting from NeXOS in order to build prototypes for testing the BRIDGES Sensor Web specifications.

Furthermore, NeXOS has contributed to the development of a SensorML 2.0 profile for marine sensors to which the NeXOS SensorML profile should be aligned. Thus, the SensorML 2.0 recommendations in this deliverable are also considering the NeXOS achievements.

[14] http://www.nexosproject.eu/

4.3.1.3 SCHeMA

The SCHeMA (Integrated In Situ CHemical MApping Probes) project[15] aims at the development of marine in-situ sensing technology for detecting anthropogenic and natural chemical compounds.

Within SCHeMA the sensor developments are complemented by work on a plug-and-play network to collect and distribute the collected observation data to users via web technologies[16]. For developing this network, the SCHeMA project will also consider the previously described SWE standards (including SensorML) as well as further European regulations such as INSPIRE[17].

Similarly to the other presented Oceans of Tomorrow projects, SCHeMA is also considered as a relevant cooperation partner: Exchanging experiences on the use of SWE technology will help to take sustainable design decisions and to ensure on the other hand a harmonised approach on the application of Sensor Web technology.

4.3.1.4 SenseOCEAN

The EU H2020 SenseOCEAN project[18] aims to create a highly integrated multifunctional and cost-effective in situ marine biogeochemical sensor system. The system will be modular and deployable on multiple platforms including autonomous platforms such as gliders and profiling floats. One of the project's goals is the implementation of an interoperable data architecture that will introduce standards from the sensor through to data delivery.

Sensor data and metadata are being modelled using OGC SWE standards SensorML 2.0 and O&M. Because the SWE standards are designed in a domain independent manner, they were intentionally specified with a high degree of flexibility that enables implementation across many different domains and usage scenarios. At the same time this flexibility allows one to achieve similar goals in different ways. Thus, in order to avoid interoperability issues, partners from several projects have teamed up to develop profiles of the different OGC SWE standards that can serve as a common basis, through the Marine SWE profiles wiki. This effort includes the semantic annotation of the standards with controlled vocabularies so that they become semantically rich and interoperable

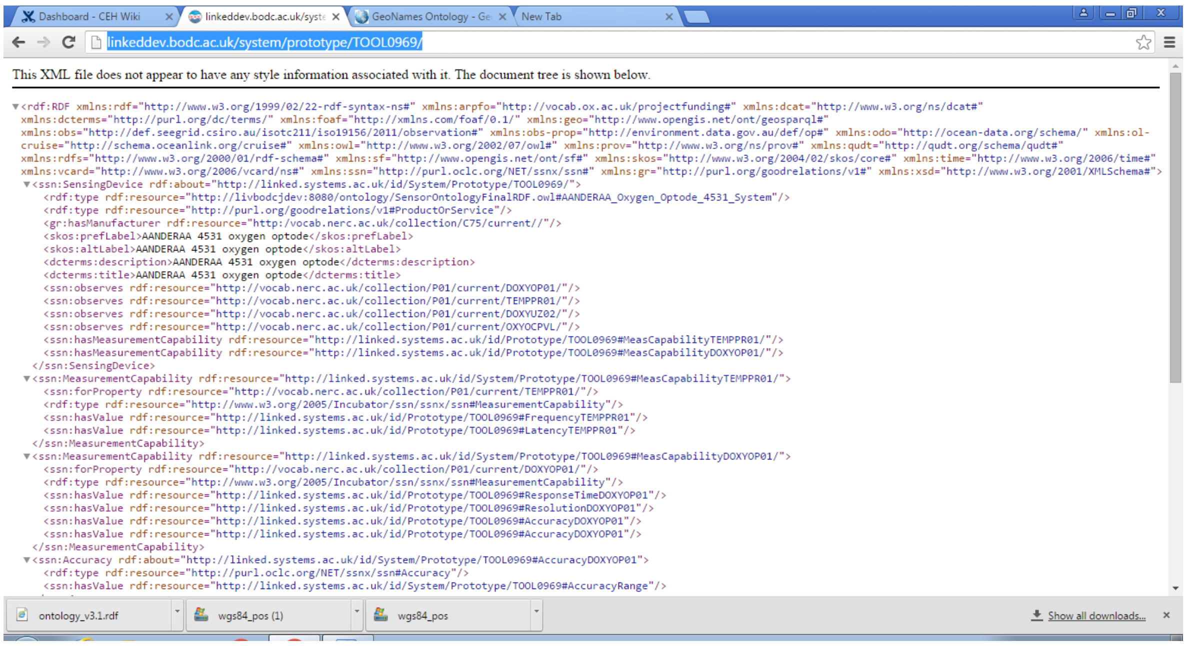
Sensor data and metadata are also being published as Linked Data by describing information with open standards like the Resource Description Framework (RDF) and using standardized ontologies like Semantic Sensor Network Ontology (SSN-O). To express data and their metadata in O&M, SSN ontology needed alignment with the Provenance Ontology (PROV-O) and om-lite (a Web Ontology Language (OWL) representation of O&M 2.0). The following ontologies were also imported to express additional features of sensor characteristics:

- the GoodRelations ontology (http://semanticweb.org/wiki/GoodRelations) to describe manufacturer metadata

[15] http://www.schema-ocean.eu/
[16] http://www.schema-ocean.eu/Docs/Confirmed/SCHeMA%20factsheet%201_vers%20end.pdf
[17] Please note: In a different project context, 52°North is currently working together with the JRC on an enhancement of the INSPIRE Technical Guidance on Download Services so that the OGC SOS standards can also be used as an INSPIRE compliant Download Service.
[18] http://www.senseocean.eu/

- the NERC vocabulary server is providing the Uniform Resource Identifiers (URIs) to parameters and units with information (https://www.bodc.ac.uk/data/codes_and_formats/vocabulary_search/)
- the PROV-O ontology to describe provenance metadata (https://www.w3.org/TR/2013/REC-prov-o-20130430/)
- Dublin Core, SKOS(Simple Knowledge Organization System)
- GeoNames Ontology, to add geospatial semantic information

A constraint on full implementation of SWE from the sensor through to the platform and data delivery is the requirement to install the sensors on legacy platforms which have limited functionality. The solution to this issue is that each sensor transmits a URI that resolves to the sensor metadata at the data centre level. Sensor metadata will be available in multiple formats including SensorML and JSON-LD. An example of the returned SensorML created with the permission of the manufacturer is shown in Figure 6. The record is for a commercially available Aanderra Oxygen Optode 4531. This is a commercially available sensor and not part of the SenseOCEAN project and the record has been created to demonstrate functionality of the data system (until SenseOCEAN metadata become public domain).



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:arpfo="http://vocab.ox.ac.uk/projectfunding#" xmlns:dcat="http://www.w3.org/ns/dcat#"
  xmlns:dcterms="http://purl.org/dc/terms/" xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:geo="http://www.opengis.net/ont/geosparql#"
  xmlns:obs="http://def.seegrid.csiro.au/isotc211/iso19156/2011/observation#" xmlns:obs-prop="http://environment.data.gov.au/def/op#" xmlns:odo="http://ocean-data.org/schema/" xmlns:ol-
  cruise="http://schema.oceanlink.org/cruise#" xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:prov="http://www.w3.org/ns/prov#" xmlns:qudt="http://qudt.org/schema/qudt#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:sf="http://www.opengis.net/ont/sf#" xmlns:skos="http://www.w3.org/2004/02/skos/core#" xmlns:time="http://www.w3.org/2006/time#"
  xmlns:vcard="http://www.w3.org/2006/vcard/ns#" xmlns:ssn="http://purl.oclc.org/NET/ssnx/ssn#" xmlns:gr="http://purl.org/goodrelations/v1#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  ▼<ssn:SensingDevice rdf:about="http://linked.systems.ac.uk/id/System/Prototype/TOOL0969/">
     <rdf:type rdf:resource="http://livbodcjdev:8080/ontology/SensorOntologyFinalRDF.owl#AANDERAA_Oxygen_Optode_4531_System"/>
     <rdf:type rdf:resource="http://purl.org/goodrelations/v1#ProductOrService"/>
     <gr:hasManufacturer rdf:resource="http:/vocab.nerc.ac.uk/collection/C75/current//"/>
     <skos:prefLabel>AANDERAA 4531 oxygen optode</skos:prefLabel>
     <skos:altLabel>AANDERAA 4531 oxygen optode</skos:altLabel>
     <dcterms:description>AANDERAA 4531 oxygen optode</dcterms:description>
     <dcterms:title>AANDERAA 4531 oxygen optode</dcterms:title>
     <ssn:observes rdf:resource="http://vocab.nerc.ac.uk/collection/P01/current/DOXYOP01/"/>
     <ssn:observes rdf:resource="http://vocab.nerc.ac.uk/collection/P01/current/TEMPPR01/"/>
     <ssn:observes rdf:resource="http://vocab.nerc.ac.uk/collection/P01/current/DOXYUZ02/"/>
     <ssn:observes rdf:resource="http://vocab.nerc.ac.uk/collection/P01/current/OXYOCPVL/"/>
     <ssn:hasMeasurementCapability rdf:resource="http://linked.systems.ac.uk/id/Prototype/TOOL0969#MeasCapabilityTEMPPR01/"/>
     <ssn:hasMeasurementCapability rdf:resource="http://linked.systems.ac.uk/id/Prototype/TOOL0969#MeasCapabilityDOXYOP01/"/>
   </ssn:SensingDevice>
  ▼<ssn:MeasurementCapability rdf:resource="http://linked.systems.ac.uk/id/System/Prototype/TOOL0969#MeasCapabilityTEMPPR01/">
     <ssn:forProperty rdf:resource="http://vocab.nerc.ac.uk/collection/P01/current/TEMPPR01/"/>
     <rdf:type rdf:resource="http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#MeasurementCapability"/>
     <ssn:hasValue rdf:resource="http://linked.systems.ac.uk/id/Prototype/TOOL0969#FrequencyTEMPPR01"/>
     <ssn:hasValue rdf:resource="http://linked.systems.ac.uk/id/Prototype/TOOL0969#LatencyTEMPPR01"/>
   </ssn:MeasurementCapability>
  ▼<ssn:MeasurementCapability rdf:resource="http://linked.systems.ac.uk/id/System/Prototype/TOOL0969#MeasCapabilityDOXYOP01/">
     <ssn:forProperty rdf:resource="http://vocab.nerc.ac.uk/collection/P01/current/DOXYOP01/"/>
     <rdf:type rdf:resource="http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#MeasurementCapability"/>
     <ssn:hasValue rdf:resource="http://linked.systems.ac.uk/id/Prototype/TOOL0969#ResponseTimeDOXYOP01"/>
     <ssn:hasValue rdf:resource="http://linked.systems.ac.uk/id/Prototype/TOOL0969#ResolutionDOXYOP01"/>
     <ssn:hasValue rdf:resource="http://linked.systems.ac.uk/id/Prototype/TOOL0969#AccuracyDOXYOP01"/>
     <ssn:hasValue rdf:resource="http://linked.systems.ac.uk/id/Prototype/TOOL0969#AccuracyDOXYOP01"/>
   </ssn:MeasurementCapability>
  ▼<ssn:Accuracy rdf:about="http://linked.systems.ac.uk/id/System/Prototype/TOOL0969#AccuracyDOXYOP01">
     <rdf:type rdf:resource="http://purl.oclc.org/NET/ssnx/ssn#Accuracy"/>
     <ssn:hasValue rdf:resource="http://linked.systems.ac.uk/id/System/Prototype/TOOL0969#AccuracyRange"/>
```

Figure 6: An example SensorML record from the SenseOCEAN project.

Where possible the data formats for delivery of data will follow the international programmes applicable to the data e.g. profiling float data with be converted to the Argo format, submarine glider data to the Everyone's Gliding Observatories (EGO) format, mooring data to the OceanSites format. The files will link to the SensorML held in 52N and exposed to the web via ERRDAP (http://coastwatch.pfeg.noaa.gov/erddap/index.html). The implementation of ERRDAP and 52°North is shown in Figure 7. This implementation will be fully described in the SenseOCEAN deliverable due to be completed in March 2016.

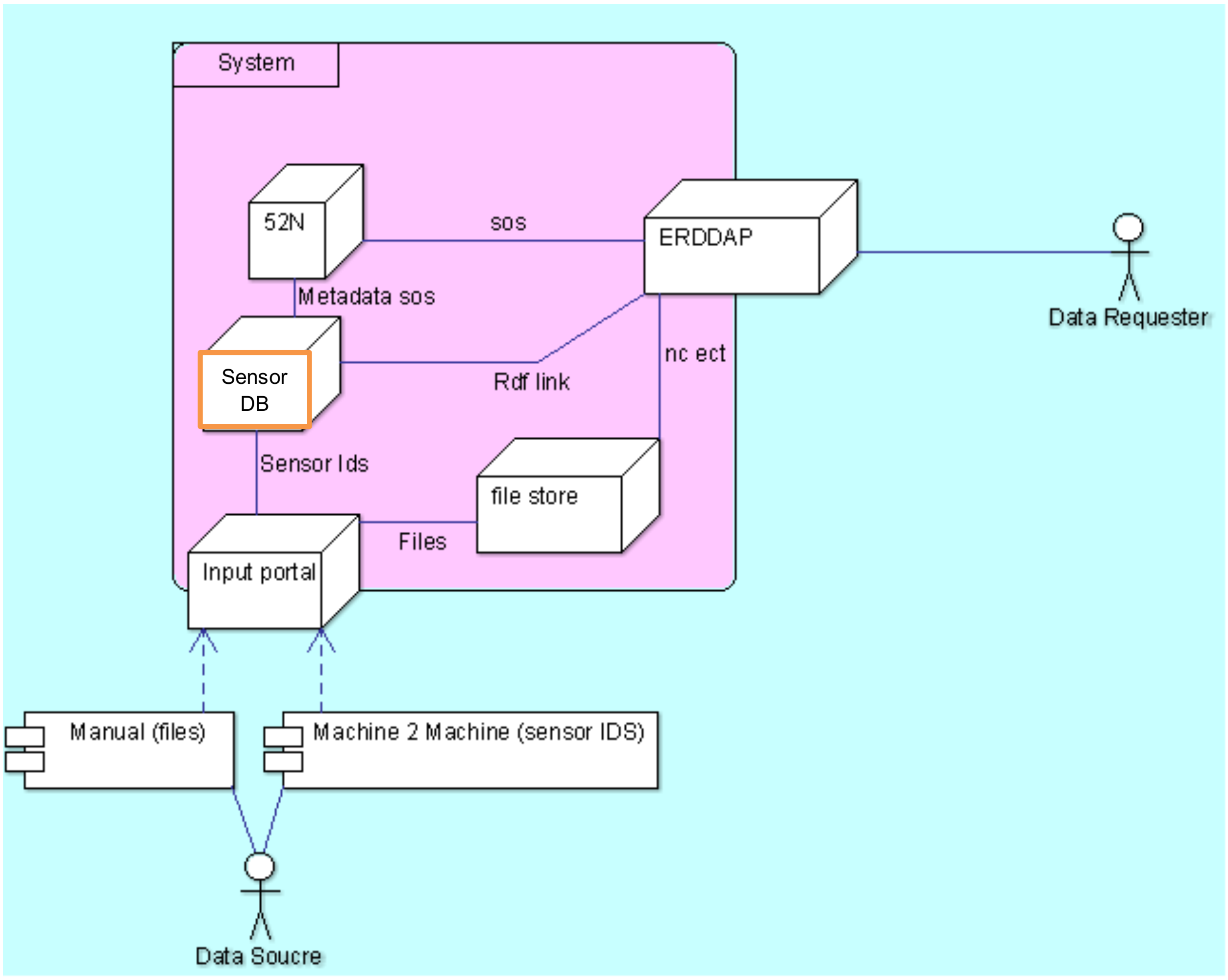


Figure 7: Schematic describing the proposed implementation or 52°North and ERRDAP with BODC as part of the SenseOCEAN project.

The goal of SenseOCEAN is to develop sensor technology from TRL of 3 to 7 i.e. prototype to pre-production commercial sensor. As such the data collected will be considered immature in the context of global programmes such as Argo whose data systems are increasingly moving towards accepting mature, well-understood data with sustainable longer term resourcing. This is to ensure that limited resources for activities such as data management are appropriately directed.

The conversion of data to the exchange formats related to each programme (gliders, profiling floats, fixed moorings, etc.), will demonstrate functionality of sensors in the context of each platform type. This leaves the potential open for inclusion of the data within the global programme specific datasets once the sensors reach commercial and operational maturity with proven quality assured data.

4.3.2 AtlantOS

AtlantOS[19] is a four year project which commenced in June 2015 and has the overarching goal:

> *"To deliver an advanced framework for the development of an integrated Atlantic Ocean Observing System that goes beyond the state-of–the-art, and leaves a legacy of sustainability after the life of the project".*

AltantOS will enhance the observation network and bring together data from ship based observation networks, autonomous observation networks and coastal observation networks. Gliders form part of both autonomous and coastal networks. The AtlantOS network is focusing on variables identified by the Ocean Observations Panel for Climate (OOPC), namely the Essential Climate Variables (ECV) and Essential Ocean Variables (EOV) (http://ioc-goos-oopc.org/obs/ecv.php).

Work package 7[20] focuses on data and one its goals is to "Integrate standardised in-situ key marine observations". Data will be delivered to their existing global repositories in their current exchange formats and the OGC SOS standard is being investigated as a method to aggregate data from data centres contributing to the AtlantOS network.

4.3.3 GROOM

A key project in establishing gliders as a tool for operational and scientific observation of the ocean was Gliders for Research, Ocean Observation and Management (GROOM)[21].

The objective of the GROOM project was to design a new European Research Infrastructure that uses underwater gliders for collecting oceanographic data. The new infrastructure was designed to be beneficial to a large number of marine activities and societal applications related to climate change, marine ecosystems, resources, or security that rely on academic oceanographic research and/or operational oceanography systems.

The GROOM project was a significant step towards an operational underwater glider network and included the establishment of the Global Data Assembly Centre at Ifremer, France and the definition of the EGO format for data exchange.

The final report fully documents progress made to an operational underwater glider network and is available at http://www.groom-fp7.eu/lib/exe/fetch.php?media=public:deliverables:groom_d1.10_upmc.pdf.

GROOM was a finite project and work to continue the development toward underwater gliders becoming part of the Global Ocean Observing System (GOOS) continues under the EGO banner. Despite EGO not being formally funded, projects such as AtlantOS will enhance the flow of data by continuing the development of the oceanic glider network.

[19] https://www.atlantos-h2020.eu/
[20] https://www.atlantos-h2020.eu/wp7
[21] http://www.groom-fp7.eu/

5 Status Quo and Current Workflow

This section focuses on the current state of glider data acquisition setups. Gliders from different vendors all implement a specific workflow. Identifying the differences and commonalities provides the basis for developing an interoperable approach.

The following sections provide details on the data acquisition workflow of Seaglider and Slocum AUVs. In the final version of this document descriptions of the SEA EXPLORER AUV will be added.

5.1 Internal data storage patterns, data transmission and recovery

In this section we discuss how currently-available gliders store data internally, with emphasis on sensor data (scientific and engineering types) but also fixed data like calibration information or other mission metadata. In addition, some details are provided as to how the transmission of these data to shore is achieved. Each glider uses a different approach, and the DEEP and ULTRA-DEEP EXPLORER gliders can take advantage of those early attempts to provide the most robust product from a data management standpoint during the design phase.

5.1.1 Seaglider

The Seaglider has one CPU, a Persistor TT8, which runs a special version of Microsoft DOS (and extended version of picoDOS). This means that there are limitations to the names and total number of files, besides the normal storage capacity limitation. On this disk are several types of files: the glider executable file, four specific command/control files, a compass calibration file, a battery usage tracking file, possibly bathymetry map files, various working files, and of most interest here: instrument data files. The data files are described here, the command files in the next section, while the rest are left to Appendix A.

At the conclusion of each dive (defined by a particular cycle of events that occur as configured by the glider operational "run" software), a set of files is created: log, engineering, and capture. Ultimately, if transmitted and processed successfully, these will become human-readable files on the base station on shore with extensions .log, .eng, and .cap. While on the glider they are stored in files with names such as:

- `SGNNNNLZ.A`
- `SGNNNNDZ.A`
- `SGNNNNKZ.A`

where NNNN is the dive number, "L" means log, "D" means data or engineering, and "K" means capture. The letter "Z" shows the file is zipped, and "A" means the file is archived.

When data are ready to be transmitted from the Seaglider, each file is first broken into pieces of 8 kb or less, then renamed so that intermediate files are created. For instance in `'sg0055dz.x'` might become "`sg0055dz.x00`" and "`sg0055dz.x01`". These files are then transmitted to the base station and a copy of them is renamed and saved on the Seaglider. If transmission was not successful, data is further broken down to smaller pieces and the same procedure is repeated. Checksums are run on received files and an error is generated if a failure occurs, alerting the pilot that that particular segment of data must be requested again from the glider (via a control file).

The Seaglider data transmission using the Iridium RUDICS system is very efficient. For 1000 m dives, which could take 7-8 hours, all log and eng files are sent and control files exchanged in a matter of minutes for typical sampling (CTD every 5 sec in upper 600 m, 10 sec below; Oxygen every 30 sec above 600 m only, and Optical triplet every 60 sec in upper 300 m, which is about 100 KB). Therefore, data recovery upon glider collection is rarely critical: only a few corrupted dives are often needed, as well as capture files for a complete mission log.

5.1.2 Slocum

The Slocum glider has two Persistors for "flight" and "science". Both contain a version of picodos, and a main program, "app", runs on each Persistor. The flight Persistor talks to the science Persistor via a software-controlled connection known as the "clothesline", to extract science data to be sent over Iridium or FreeWave (radio) to the server. The main mission is whichever of the available missions that the pilot chooses to run, and variables such as waypoints, sampling regimes and dive profiles are controlled by shorter mission segment files.

Both Persistors' Flash drives have a similar structure with the sub directories:

- App – the main Gliderdos app within which the mission is run
- Bin – the executables of the main program
- Config – data files referenced by the mission and the app, some of which are edited by the pilot
- Missions – Files ending in .mi for different missions that can be run by the pilot
- Mafiles – mission segments ending in .ma that are regularly edited by the pilot during a mission
- Logs – Files created by the glider when logging is invoked
- Sentlogs – files that have been successfully transmitted by the glider
- State – long term variable storage

The science Persistor does not have missions or mafiles, but otherwise the structure is the same.

Both Persistors contain a file in the config directory to determine which variables are transmitted and what subsampling rate is to be used. This is sbdlist for flight and tbdlist for science. During a mission, each dive creates a series of files in the LOGS directory on each Persistor. The naming protocol is a dive number followed by a segment number. If the mission is stopped by the pilot, the dive number increments and the segment number is reset to zero. Usually, the ASCII header is only sent in segment 0000 to keep the transmission times down. The file types for the two Persistors are defined by their file extensions:

Table 1: File types of Persistors

Function	Flight persistor	Science Persistor
Raw binary data	.dbd	.ebd
short binary data	.tbd	.sbd
medium binary data	.mbd	.nbd
log files	.mlg	.nlg

The files sbdlist.data and tbdlist.dat determine the short binary data to be sent back to the server. The full binary data is usually too big to send over Iridium and is extracted when the glider is recovered. If transmission of a file is interrupted due to loss of communicationss at the surface, the glider takes up transmission where it left off on the next occasion. The glider has a maximum time on surface, set by the pilot which can be extended interactively.

The glider communicates using RUDICS over Iridium with the server via a terminal program, which has a command window for the pilot. A FreeWave radio modem can also be used for short ranges. Data retrieval and other surface behaviour is normally autonomously controlled by scripts, but the pilot can manually intervene to retransmit files, transmit files not on the normal list such as a .mlg log file, dive immediately, abort the mission and restart a new one and so on.

Sensor sampling is decided by the mission file and controlled by sample .ma files, either one for all the sensors together or individual ones where sensor sampling differs. Thus the pilot sets the native sample rate, depth to stop and start at, and profile i.e. diving, climbing, surface or a combination of all possibilities for each sensor.

5.1.3 SEA EXPLORER

The Sea Explorer glider has two ARM9 computers, one for the vehicle, one for the payload. Both are running Linux inside and the main programs are called Seaexplorer and Seapayload. The vehicle and the payload are exchanging information and files through a RS232 line and a zModem protocol. Both computers are accessible without opening the glider using Ethernet for data download, file modification or for upgrading the system.

Both ARM9 computer share the same structure:

- root directory contains the main program and the configuration files
    - The SeaExplorer or Seapayoad binary file
    - sea.cfg = this file describes all the configuration parameters of the glider. This file is extremely important, and can only be modified with caution.
    - sea.msn = this file gives the glider a default configuration for its navigation behavior
- logs directory contains all data acquired during mission
    - seaSSS.M.gli.sub.Y.gz: glider navigation datafiles with information like time, pitch, roll, ballast pumped…
    - seaSSS.M.pldN.dat.Y.gz: complete science datafiles
    - seaSSS.M.pldN.sub.Y.gz: decimated science datafiles this file will be sent over iridium
    - sea.0.gli.evt.1 : Navigation log book which records all events during navigation
    - sea.Arch.XXXXXXXXXX.tar.gz : every time a mission is started, all previously described files are stored included in a zip file and stored on the glider to make sure no information is lost.
- sbin directory contain all additional softwares

Name Description
SSS = vehicle identification number [ex 007, 013, 104]
M = mission number [ex 1, 34 , 943]
Y = Yo number [ex 1, 23, 843]
XXXXXXXXXX= random numbers

N = Payload number [1 or 2]

5.2 Data Management (Quality Assurance)

The following sections provide details on the data quality management workflow of Seaglider, Slocum and SEA EXPLORER AUVs.

5.2.1 Seaglider

When the above files are received, re-assembled and processed by the python routines on the base station, new files appear with a letter 'p" plus integers 'MMM' for the glider serial number and 'NNNN' for the dive number in front of their various file name extensions. The following human-readable files are the result of a successful communication session:

Data File (.dat): This is an ASCII text file which is transmitted to the base station for further processing (e.g. `p1500055.dat` for dive 55 of glider SG150). It contains the raw sensor data and each file covers one dive cycle. This file is made as compact as possible and therefore difficult to read directly by the user. However, the numbers can be interpreted using the column titles line in the header of this file. An example of the first few lines of a data file can be seen in Appendix A.

Log File (.log): This file serves as a summary of what happened during the previous dive (e.g. p1500055.log). It contains the software version, the glider number, the mission number, the dive number, the time that the dive was started, a complete list of parameters and their values (some of them diagnostic like battery voltage and consumption), and some summary information about the dive. An example is in Appendix A.

Capture File (.cap): This file contains detailed information about all the actions the Seaglider performed in the previous dive and is used for debugging (e.g. `p1500055.cap`). This file can become quite large and is usually not transmitted, except for the first few dives or when triggered by a serious error or pilot request. These are occasions when it is recommended to monitor the performance of the Seaglider closely. An example is in Appendix A.

The base station then generates the following more usable files using the above fundamental data files:

ASCII Files (.asc): These files are the reconstituted (uncompressed, reassembled and deferentially summed) versions of the .dat files created by the Seaglider (e.g. `p1500055.asc`).

Engineering File (.eng): In this file the data present in .asc file is restated but converted into engineering units. Along with the p*.log files, these comprise the fundamental data files (e.g. `p1500055.eng`)

netCDF Files (.nc): This file captures all processed files and is self-documenting. The NetCDF files are meant primarily for sharing data between scientific users. There are three versions:

- An up to present mission trajectory file at full resolution (variables are functions of x,y,z,t, and dive)
- An up to present mission vertically-binned trajectory file

- An individual dive file, which contains detailed information on the QC performed by the base station, as well as all entries from .log and .eng files, and a special file on the base station for calibration information (sg_calib_constants.m-see below).

Communication File (comm.log): This is a complete record of the Seaglider's communication with the base station. It is appended with a GPS string immediately upon successful connection, followed by the send/receive dialog.

BaseLog (baselog_YYMMDDHHMMSS): Contains a record of what happens when the base station processes the raw data files into the end data products. The set of python scripts resides on the base station and file manipulation and reprocessing are triggered as soon as the glider logs out of its account.

A few tools are available from Kongsberg Underwater Technology Inc. for analysing dive data, based on Matlab. Other groups have developed similar tools, like the SOCIB toolbox at https://github.com/socib/glider_toolbox.

5.2.2 Slocum

Once uploaded to the server, the .tbd and .sbd files are renamed with the glider name, year, day number, dive number and segment by the server. So for a glider called unit_400 the file name would be `unit_400-yyyy-ddd-m-s.dbd` etc. where m is the dive number and s is the segment number. The day number has leading zeroes but mission and segment number do not. The longer files, .mlg, .dbd and .ebd are copied manually from the glider's flash card to once it has been removed from the glider.

Teledyne have tools to translate the binary files into ASCII, from where it can be imported into environments such as Matlab. Numerous tools to manipulate the data have been developed, in the absence of any from Teledyne, such as the SOCIB toolbox at https://github.com/socib/glider_toolbox.

There are also log files on the server, generated from the surface dialogue between the glider terminal and the glider on each Iridium or FreeWave contact.

5.2.3 SEA EXPLORER

All log files are zipped in*.gz format in order to be sent as binary file during iridium transmissions. Inside the zip files, the format used is csv, every file include the header and the list of data using semi-colon as separator.

The files can be automatically or manually downloaded using Iris land software. The file are stored in the SeaExplorer repository with sub directories per glider and per mission

- SeaExplorer repository
    - /Glider1
        - Mission1
        - Mission2
        - …
    - /Glider 2
        - Mission 1

Onto the iris software

- all communication logs with glider are saved in a log file.
- All communication logs are analysed and stores on a CSV file with all relevant information from all yo of the mission
- A kml file is generated at each surfacing with the first and last surface position.

The SeaExplorer data manipulation is included in the SOCIB toolbox at https://github.com/socib/glider_toolbox.

5.3 Mission planning

Mission planning is not part of the standard preparation for commercial gliders. Each operator has their own technique or preferred method to plan the glider sampling. Basic information about the environment and logistics must be considered in relation to the scientific goals, but no manufacturer tools are available beyond basic self-test tools. These pre-launch tests run the glider through a series of sub-system tests to ensure all components are ready for operation. Many organizations have developed checklists for pre-deployment preparation (GROOM D5.3).

5.4 Control

This section introduces the current approach how gliders are controlled and tasked. The aim of this information is to provide a foundation to derive the requirements for glider control within the BRIDGES Sensor Web infrastructure.

5.4.1 Seaglider

There are four control files that a pilot can edit and leave in the glider home directory to be picked up and replace the versions residing on the glider. The Seaglider downloads the control files, if present, every time it calls the base station. If not present, the copy on the glider is re-used. The command file (cmdfile) is used to modify values of different parameters during the mission. Commands are in capital letters and preceded by $. One of the three directives ($QUIT, $RESUME, $GO) is in the last line of this file. An example from the beginning of a mission is presented below. During a mission, this file could be a single line containing $GO.

```
$C_VBD,2900
$C_PITCH,2788
$C_ROLL_DIVE,2151
$C_ROLL_CLIMB,2050
$D_TGT,45
$T_DIVE,15
$T_MISSION,25
$D_SURF,2
$D_ABORT,60
$SM_CC,690
$MAX_BUOY,150
$NAV_MODE,2
$ALTIM_PING_DEPTH,0
$T_RSLEEP,3
$QUIT
```

Science File provides instructions for how often the sensors are sampled.

```
/depth  time    sample  gcint
200     20      100     300
```

```
1000    60      100     300
```

Targets File provides flight path when navigating by waypoints.

```
UPPER_RIGHT3     lat=3437.83  lon=3305.748  radius=1000  goto=UPPER_RIGHT3
VM_UPPER_RIGHT0  lat=3435.50  lon=3307.0    radius=3852  goto=UPPER_RIGHT3
```

PicoDOS Command Batch File (pdoscmds.bat) contains a list of the commands used only by the glider to do low-level operations, such as re-send a file or change the target.

Every time a glider connects to its account, an archived copy of the cmdfile, targets file, or science file is saved on the base station and renamed to include the dive number. PDOS command files are also saved, but already include the dive number, so they are saved with a serial number. If there are multiple calls on one surfacing, a cmdfile is sent each time, and a serial number is added after the dive number. This allows (in theory) for the reconstruction of pilot command history, although in practice it is difficult because it is not certain if the control file was successfully received, or was just partially sent and the connection lost. Additionally, output of the picoDOS system is returned to the base station in a special .pdos file, which confirms the result of the commands sent.

5.4.2 Slocum

There are several files to control the vehicle during a mission that are user-editable.

.**mi Mission file**. This is the main control file that defines the behaviours the glider executes either intrinsically or via a shorter behaviour file. This file will be set up by the pilot, starting from a stock file (stock.mi) or a previously used mission file. All the dive parameters that a pilot will change during a mission are held in the shorter files that can be uploaded without interrupting the mission. Files are referenced by a number that is read after the behaviour reference. So file sample10.ma would be referenced by the number 10 in “behaviour: sample” in the main mission file.

.**ma mission** segment file. There are 4 of these: the yo file that controls the diving parameters, sample files that control the sensors, the goto file that controls the waypoints and route and the surface file that controls behaviour on the surface.

yoxx.ma. Yo file xx, at the head of the file are a number of sensor values that are set for the dive:

```
sensor: f_max_working_depth 200   # Initial state of glider set to 200m
sensor: u_alt_min_depth(m) 5      # Set depth altimeter comes on at.
sensor: c_alt_time -1             # set number seconds  pings apart.
                                  # <0 = off
sensor: u_min_altimeter(m) 12
sensor: u_max_altimeter(m) 100.0  # the maximum range of the altimeter
sensor: u_max_water_depth_lifetime(yos) 1.0
```

Below this are the arguments that control the dive profile i.e. how many dives before surfacing, depth to dive to and to climb to before inflecting, how much ballast to pump and what pitch to set the glider to and how to do it. Where autoballast is being used, a maximum and minimum vertical velocity is also set. If the thruster is in use, this is controlled from the yo file as well.

```
<start:b_arg>
    b_arg: start_when(enum)             2   # pitch idle (see doco below)
    b_arg: num_half_cycles_to_do(nodim) 2   # Number of dive/climbs to
                                            # perform
                                            # <0 is infinite, i.e. never
                                            # finishes
                                            # arguments for dive_to
    b_arg: d_target_depth(m)     30
    b_arg: d_target_altitude(m)  25
    b_arg: d_use_bpump(enum)     0   # 0  Autoballast/Speed control.
    b_arg: d_bpump_value(X)      400.0   # use_bpump == 0   Total amt of
                                         # ballast, stored as
                                         # C_AUTOBALLAST_VOLUME

    b_arg: d_use_pitch(enum)     3   # 1:battpos  2:setonce  3:servo
                                     # in         rad        rad, <0 dive
    b_arg: d_pitch_value(X)      -0.4528     # -26 deg
    b_arg: d_speed_min(m/s)      0.05   # minimum depth rate for dive
    b_arg: d_speed_max(m/s)      0.2  # arguments for climb_to
    b_arg: c_target_depth(m)      4
    b_arg: c_target_altitude(m)  -1
    b_arg: c_use_bpump(enum)     0   # 0  Autoballast/Speed control.
    b_arg: c_use_pitch(enum)     3   # 1:battpos  2:setonce  3:servo
                                     # in         rad        rad, >0 climb
    b_arg: c_pitch_value(X)      0.4538  # 26 deg
    b_arg: c_speed_min(m/s)      -0.05   # minimum depth rate for climb
    b_arg: c_speed_max(m/s)      -0.2
    b_arg: end_action(enum)      2 # 0-quit, 2 resume
<end:b_arg>
```

Samplexx.ma. This is referenced by the number xx in the sample:behaviour file. By convention, if all sensors are used in the same regime, sample10.ma controls them all. If different sensors are controlled differently, these use sensor 11, 12, 13 etc.

```
<start:b_arg>
   b_arg: sensor_type(enum) 0   #All sensors
    b_arg: state_to_sample(enum) 5   # 0 none
                                     # 1 diving
                                     # 4 climbing
                                     # 5 diving|climbing
    b_arg: intersample_time(s) 0   # if < 0 then off, if = 0 then
                                   # as fast as possible, and if
                                   # > 0 then that many seconds
                                   # between measurements
    b_arg: nth_yo_to_sample(nodim) 1   # Check masterdata
    b_arg: intersample_depth(m) -1   # Check masterdata
    b_arg: min_depth(m) -5   # Check masterdata
    b_arg: max_depth(m) 2000   # Check masterdata
<end:b_arg>
```

The above example from sample10 uses 0 to switch on all sensors in the argument sensor_type. There are up to 60 sensor types that are selected in the case of individual sensors. State_to_sample is self-explanatory, as is intersample_time. Nth yo to sample counts from each surfacing, so this should not exceed num_half_cycles (in the yo file) divided by 2.

goto_lxx.ma. The goto file referenced by xx in the goto_list behaviour. There are a number of variables that are set in the file, followed by the waypoints.

```
<start:b_arg>
   b_arg: start_when(enum) 0
   b_arg: list_stop_when(enum) 007
   b_arg: initial_wpt(enum) -1
   b_arg: num_legs_to_run(nodim) -1
   b_arg: num_waypoints(nodim) 2
   b_arg: list_when_wpt_dist(m) 1000   # used if list_stop_when == 7
<end:b_arg>
<start:waypoints>
   -837.250 4924.132   # comment
   -837.250 4925.260
<end:waypoints>
```

The most important when altering the file is num_waypoints which must be set manually and must be correct.
num_legs_to_run is explicit, or -1 for repeat forever.
initial_wpt -1 is the next one on the list, -2 is the nearest waypoint, or any positive number N is the Nth one on the list.
list_when_wpt_dist is the range to waypoint that the glider goes onto the next.

The format for the waypoints is in degrees and decimal minutes, longitude followed by latitude, negative numbers for south and for west, leading zeroes on degrees are optional i.e.:(DD)DMM.mmm.

surfacexx.ma. behaviour of the glider on the surface referenced by xx in the surface behaviour. The only variable the pilot ever has to change is

```
      b_arg: when_secs(sec) 1800   # Surface every 0.5 hrs for no comms
```

This is set to exceed the expected time under the surface.

5.4.3 SEA EXPLORER

In order to change the glider behavior, the pilot will send commands to the glider during surfacing. The user can also program at any time Iris and Waypoint module softwares to send commands when the glider will surface.

As a general rule, a command:
- starts with '$' and ends with ';'
- includes an argument
- is necessarily followed by <cr><lf>
- a command will be acknowledged by the vehicle ">cmd:"
-

- example: '$alt=20;' the glider is requested to do come back at surface if seabed is mesuared at less than 20 meters

The list of available commands are in the table below:

Table 2: SEA EXPLORER control commands

command	Argument	Note
alt	[5;50]	Altitude in meters (-1 to turn it off)
bd	[-500 ;-100]	Set descent external volume in milliliter relative to neutral position
bs	[1;500]	set absolute volume of surface ballast
bu	[100;500]	Set ascent external volume in milliliter relative to neutral position
cbk	[1;360]	hang up iridium and call back in n minutes
clr	[0,262144]	Alarm code to clear
dst	{0|1}	to hide or print the output of the seadst nmea sentence (will be automaticly disable on next yo)
flr	{0|1}	To turn OFF or ON the flasher
go	N/A	make the glider dive (No argument for this command)
halt	{0|1}	let the glider at surface. Halt must be set up back to 0 before sending a go;
heading	[0;360]	Heading set point in degrees
hpid	[0]	Reset heading regulator, 0 is the only accepted value
log	{0|1}	to hide or print the output of the sealog nmea sentence (will be automaticly disable on next yo)
msk	[0,262144]	create alarm mask to aknowledge warnings, user has to enter alarm code reported in the $seamrs alarm field
pb	[1;100]	linear absolute position for base setting
pd	[0 ;20]	Set descent linear offset in mm relative to neutral position (pu<pd)
pld	[0|1]	Turn off|on payload power
ps	[1;100]	set linear absolute position when glider at surface
pu	[-20;0]	Set ascent linear offset in mm relative to neutral position (pu<pd)
radio	[0|1]	To turn off|on radio permanently (will not be reset on next surfacing)
reboot	N/A	Fully resart and initialize control a command system (use with CAUTIOUS!!!)
sendDataFile	[1;99999]	ask the glider to send the specific science file
sendNavFile	[1;99999]	ask the glider to send the specific navigation file
sr	[0;5]	set surfacing rate, this allow multi-yos
ym	[0;1000]	set maximum number of yo before surfacing in halt mode
zb	[zt;700]	set profile bottom inflection depth in meters
zt	[0;zb]	set profile top inflection depth in meters during multiyos

6 Requirements

This chapter briefly summarizes the requirements for the design of the BRIDGES Sensor Web Architecture. It is structured into different functional as well as non-functional aspects which lead to a first set of design considerations.

Attention is drawn to the fact that further updates of these requirements will be expected throughout the duration of the project. These additional requirements will be addressed by the final version of this document.

6.1 Data Flows

An important aspect concerns the data flows needed for publishing the collected observations. Especially the following aspects need to be covered by the BRIDGES Sensor Web Architecture:

- Flexible data publication mechanism: An approach is needed on how collected observation data can be uploaded to a data repository from where it can be made accessible to users.
- Automatic extraction of relevant meta-information from NetCDF: Many of the collected observation data sets are available as NetCDF files. Thus, metadata for data publication should, as much as possible, be extracted from the NetCDF files (please note: this is rather an implementation requirement than a requirement that influences the Sensor Web architecture.
- Re-use of sensor type descriptions: In order to make the publication of metadata more efficient, it should be avoided that metadata which are the same for every sensor of a specific type need to be repeatedly entered by operators. Instead, sensor type descriptions should be re-used.
- Provision of quality information about observations (e.g. quality flags)
- Guidance on required metadata: Glider operators shall receive guidance which metadata elements need to be provided in order to support the interpretation and discovery of a glider, its instruments and the data sets it generates.
- Use of vocabularies: Wherever possible, identifiers and terms should be referenced as vocabulary entries in order to increase semantic interoperability.

6.2 Security

The developed architecture shall follow the idea of open specifications and standards in order to ensure security. While non-public specifications may lead to security by obscurity, it is not sustainable to follow such an approach. For example, open standards ensure a broader review of specifications so that potential security issues are more likely identified in advance. Furthermore, operators of the glider get a clearer view of the deployed architecture and the involved components so that they are able to replace certain components if security issues are discovered in certain implementations.

6.3 Access Control

Not all data collected by gliders might be intended for public use. Furthermore, plans that shall be carried out by gliders may only be submitted by an authorised researcher. In other cases, some data sets may only be published after quality assurance was carried out. Thus, the following aspects of access control were identified:

- Data access: Only authorised users shall be able to access certain data sets or even individual observations; access criteria comprise of spatial constrains, temporal constrains, instruments, gliders, observed properties, and quality flags
- Data publication: Data may be uploaded to Sensor Web servers only by authorised users
- Tasking: Only authorised users shall be able to submit tasking requests to a glider platform.

6.4 Data Operations

With regards to data operations and functionalities that need to be provided by the BRIDGES Sensor Web infrastructure, the following requirements were identified:

- Publication: A mechanism is needed to publish data collected by gliders. Thus, an upload interface is needed which allows two main functionalities:
    - Upload of observation data
    - Upload of metadata about observation data as well as the gliders and instruments which have generated these data sets

  This publication interface might be used by glider operators to upload collected data or by scientists after validation or processing of data. In the future, also a direct upload from gliders through this interface might become relevant.
- Discovery: The Sensor Web infrastructure shall support the discovery of observation data sets as well as gliders and their instruments
- Retrieval: A download interface is needed that allows users (e.g. scientists) to retrieve the collected observation data sets as well as the corresponding metadata. This interface shall provide several query filters such as:
    - Spatial extend
    - Temporal extend
    - Glider
    - Instrument
    - Observed property
    - Quality flags

6.5 Interoperability

The BRIDGES Sensor Web architecture shall ensure interoperability with other observation data and geospatial information infrastructures. Thus, it shall not comprise isolated specifications but instead it shall re-use existing standards. Important reasons for this requirement are:

- Re-use existing knowledge and experiences
- Conformance with approaches used in other systems and initiatives (e.g. INSPIRE, GEOSS)
- Possibility to re-use existing implementations of the relevant standards

7 Approach

The following sections describe the approach developed to design a service architecture for glider data based on international standards and well-established data formats such as the OGC SWE service suite and NetCDF. An overview of relevant parameters, their metadata and how to encode these is provided.

7.1 Identification of Glider Parameters

A service architecture that focuses on the discovery and provision of glider data requires a strong focus on the metadata of glider platforms, missions, as well as features of interest. This section illustrates the metadata required by a data analyst which need to be exposed by the system.

7.1.1 Calibration

7.1.1.1 Seaglider

All of the Seaglider's sensors, except for the compass, arrive calibrated at the fabrication centre from the sensor manufacturer. The compass is calibrated for the local magnetic conditions with the all of the sensors and batteries in-place and the compass calibration file is stored on the Seaglider. Calibration coefficients and metadata for sensors and for the glider as well as very basic mission characteristics are stored in a human-readable file called sg_calib_constants.m, which is stored in the glider home directory on the base station (example in appendix B). It is used by visualization tools in scripts, as well as dive analysis routines. It has also been used to customize quality control procedures carried out by the base station (spike test parameters, global max/min values, etc.). It is noted that not all of this information resides on the glider. Calibration coefficients and basic mission characteristics are stored as parameters stored in each .log file (written in memory at launch time in a special file). Serial numbers of sensors and detailed mission data are not stored internally. Also important to note is the fact that the sg_calib_constants.m file is not permanently tied to a set of data files or glider, and can easily be overwritten or lost when new missions or new gliders are used.

7.1.1.2 Slocum

As with the Seaglider, only the compass is calibrated by the user on the Slocum. Sensor calibration data for PUCK-capable sensors (e.g. WetLabs PUCK) is stored on the glider, usually within the autoexec.mi file, but otherwise the science data is stored raw and calibrations are applied when the data is uploaded to the data centre.

7.1.1.3 SEA EXPLORER

Only the compass is calibrated by the user. All scientific sensor calibration will be stored in the science bay or in the sensor itself. This will allow the user to change easily payload without moving calibration parameters. Alseamar is engaged in the program FP7 NeXOS so the SeaExplorer will be able to handle SensorML files.

7.1.2 Measured Parameters

All observation parameters of a glider platform are defined in the NetCDF EGO file as data variables. As this file is self-describing, it is very straight-forward to extract these. Based on many existing EGO files, a common set of data variables has been identified. The following table provides an overview (please note: whenever possible a reference to corresponding dictionary entries shall be given; the table below contain vocabulary URIs for the different parameters; for UoM, vocabulary URIs should also be provided in the final version of this document, as available):

Table 3: Overview of a few common Data Variables

Parameter	UoM	Vocabulary URI	Comment
sea_water_pressure	Pa	http://vocab.nerc.ac.uk/collection/P01/current/PPSSPS01/	
sea_water_temperature	deg	http://vocab.nerc.ac.uk/collection/P01/current/TEMPPR01/	
sea_water_electrical_conductivity	S/m	http://vocab.nerc.ac.uk/collection/P01/current/CNDCZZ01/	
sea_water_practical_salinity	[psu]	http://vocab.nerc.ac.uk/collection/P01/current/PSALCU01/	provided as Practical Salinity Unit

The navigational parameter parts of an EGO file are defined by the following variables:

Table 4: Overview of Navigation Parameters

Parameter	UoM	Vocabulary URI	Comment
latitude_gps	deg	http://vocab.nerc.ac.uk/collection/P01/current/ALATGP01/	WGS84 Coordination Reference System
longitude_gps	deg	http://vocab.nerc.ac.uk/collection/P01/current/ALONGP01/	WGS84 Coordination Reference System
vehicle_depth	m	http://vocab.nerc.ac.uk/collection/P09/current/DEPH/	depth from surface
distance_to_sea_floor	m	http://vocab.nerc.ac.uk/collection/P01/current/AHSFZZ01/	
water_depth	m		total depth of the water column
pitch	deg	http://vocab.nerc.ac.uk/collection/P01/current/PTCHEI01/	
roll	deg	http://vocab.nerc.ac.uk/collection/P01/current/ROLLEI01/	
heading	deg	http://vocab.nerc.ac.uk/collection/P01/current/HEADCM01/	
speed	m/s	http://vocab.nerc.ac.uk/collection/P01/current/APSAZZ01/	

7.2 Parameter Mapping

This section provides an overview on the requirements of parameter mappings resulting from the existing approaches. A special focus is given to the possibility to enable discovery of glider data by defining default and required parameters and their definition using Observation & Measurements as well as SensorML.

7.2.1 Glider Data Model for Observations & Measurements

Figure 8 illustrates the general structure of an O&M 2.0 OM_Observation document for usage with glider data. Different aspects that are required for O&M are covered that leverage the discovery of O&M glider data (i.e. a user is able to search for glider data using the different concepts):

Phenomenon Time

This property defines the time frame in which the mission took place. It shall cover the first and last time of measurements.

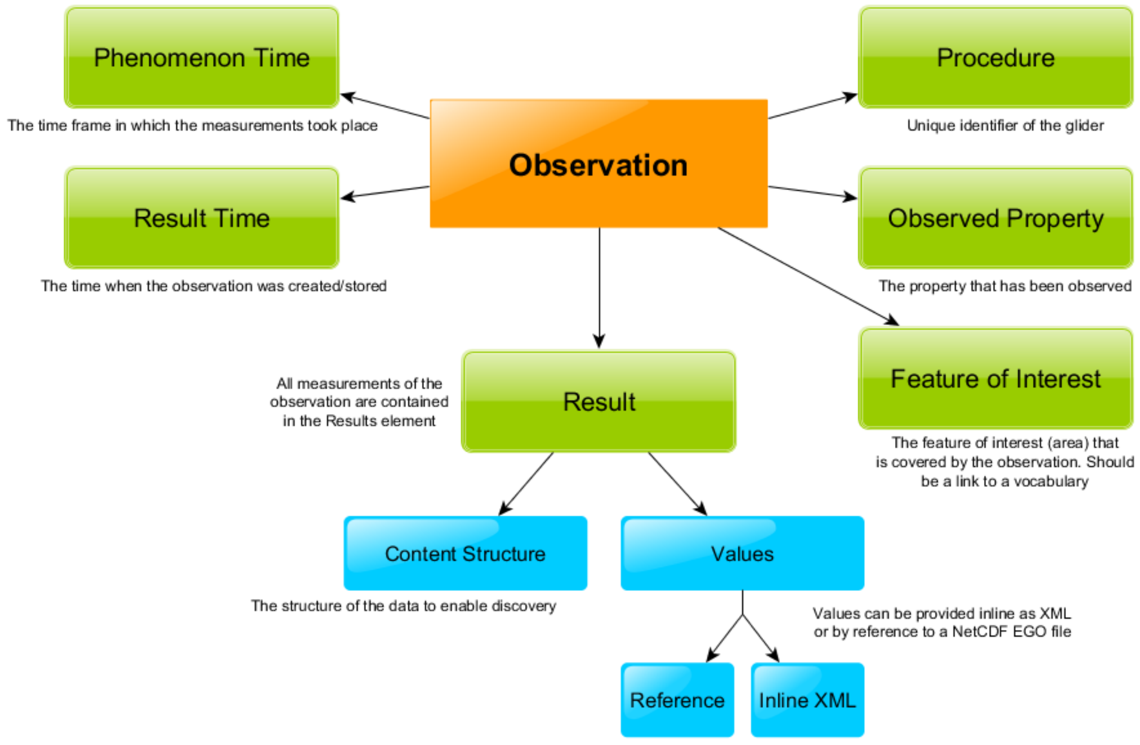

Phenomenon Time
The time frame in which the measurements took place
Observation
Procedure
Unique identifier of the glider
Result Time
The time when the observation was created/stored
Observed Property
The property that has been observed
All measurements of the observation are contained in the Results element
Result
Feature of Interest
The feature of interest (area) that is covered by the observation. Should be a link to a vocabulary
Content Structure
The structure of the data to enable discovery
Values
Values can be provided inline as XML or by reference to a NetCDF EGO file
Reference
Inline XML

Figure 8: Structure of an OM_Observation document for glider data.

Result Time

In contrary to the Phenomenon Time, the Result Time shall define the time when the data set was created and inserted into data storage (e.g. the time of recovery for delayed mode, or time of transmission to the DAC for near real time).

Procedure

The procedure shall link to a unique identifier of the glider instance. The value shall be the same for every mission one glider has conducted.

Observed Property

The observed property shall define the phenomena the glider measured. As a glider does not only measure one property (e.g. sea water salinity) it shall link to composite property (e.g. the science bay) that is itself defined as a SensorML document, thus providing all required information for discovery and analysis.

For the Observed Property we recommend the use of a vocabulary server (i.e. the server operated by NERC[22]). The URI pointing to the corresponding vocabulary entries shall be used as Observable Property identifiers.

Feature of Interest

This property defines the spatial area covered by the mission. It shall link to an entry of an established vocabulary (e.g. the BODC vocabulary[23]). Thus a user can search for missions conducted for a certain area (e.g. the Celtic Sea).

Result

The result contains the actual measurements created during the mission. The different possibilities for structuring the result component are described in section 8.2.1.3.

7.2.2 NetCDF

The NetCDF EGO format (Carval et al., 2013) is the established data format for glider data. This section describes how the EGO data model can be aligned with best practices for O&M 2.0 design.

7.2.2.1 Comparison with Observation & Measurements

To allow storage and provision through OGC SWE services, it is crucial that all parameters required to operate these services are available within the NetCDF EGO format. This section analyses the EGO format and links the existing parameters to O&M 2.0, thus providing guidance on how to establish the connection between the two.

Phenomenon Time

The phenomenon time can directly be derived from the timestamps stored in the NetCDF file. The first and last timestamps shall be considered as the beginning and end of the time period.

Result Time

The EGO NetCDF does not contain information on the creation date itself. A software solution shall take care of setting a reasonable time (e.g. the time when the data was analysed or imported).

Procedure

[22] http://vocab.nerc.ac.uk/
[23] http://www.bodc.ac.uk/data/codes_and_formats/vocabulary_search/

As the procedure shall be a unique identifier for the glider, an approach to define it is to derive the procedure from NetCDF parameters such as GLIDER_OWNER and GLIDER_SERIAL_ID, e.g. of the following form:

- http://glider-instance.bridges-h2020.eu/<GLIDER_OWNER>_<GLIDER_SERIAL_ID>

An exemplary instance would be:

- http://www.bridges-h2020.eu/gliders/NERC_NOC/400

Observed Property

The observed property shall be derived from the procedure. If a glider system stores science and navigation data separately, it shall provide an observed property for each (and also make it available as two separated O&M documents):

- http://www.bridges-h2020.eu/gliders/<GLIDER_OWNER>_<GLIDER_SERIAL_ID>/science
- http://www.bridges-h2020.eu/gliders/<GLIDER_OWNER>_<GLIDER_SERIAL_ID>/navigation

If both are stored in the same NetCDF file, one overall procedure shall be used:

- http://www.bridges-h2020.eu/gliders/<GLIDER_OWNER>_<GLIDER_SERIAL_ID>/sensors

Feature of Interest

NetCDF EGO files do not explicitly define the area of a certain mission. A software solution provides a mechanism to set the area of interest either manually or (semi-) automatically (e.g. via spatial reasoning using the captured coordinates). It shall link to an entry in an established vocabulary, e.g.:

- http://vocab.nerc.ac.uk/collection/C16/current/21a/

Data Quality

Data quality is an important factor for glider data. O&M as well as SensorML provide ways to define the quality of a data set. Devaraju et al. (2015) defines an approach on how to encode quality. NetCDF EGO files are stored as different revisions. The quality checked revision follows a generic approach to define data quality that can be mapped to the O&M model.

For each quality checked parameter an additional parameter flag is introduced. For example:

- TEMP → TEMP_QC

The value of the flag can be one of the following:

- 0 = No quality check performed (no_qc_performed)
- 1 = Good data quality (good_data)
- 2 = Probably goo data quality (probably_good_data)
- 3 = Bad data quality, but probabilities to correct these (bad_data_that_are_potentially_correctable)

- 4 = Bad data quality (bad_data)
- 5 = Adjusted value, e.g. due to inconsistencies (value_changed)
- 8 = Interpolated value (interpolated_value)
- 9 = No data, missing value (missing_value)

Listing 1 illustrates how quality information shall be mapped to O&M files. Following this approach, a software solution can easily show quality information without the need to analyse a NetCDF file. For data sets, quality information should be provided on a meta level, i.e. indicating if quality checks have been performed or not. A value-based approach would only work for inline XML data.

Listing 1: O&M quality encoding of glider data.

```
<?xml version="1.0" encoding="UTF-8"?>
<om:OM_Observation gml:id="glider-0123"
                   xmlns:swe="http://www.opengis.net/swe/2.0"
                   xmlns:gml="http://www.opengis.net/gml/3.2"
                   xmlns:om="http://www.opengis.net/om/2.0"
                   xmlns:xlink="http://www.w3.org/1999/xlink">
   <om:phenomenonTime>...</om:phenomenonTime>
   <om:resultTime>...</om:resultTime>
   <om:procedure xlink:href="http://static-namespace.gliders.eu/glider-0123"/>
   <om:observedProperty xlink:href="http://static-namespace.gliders.eu/glider-
                                    0123/science"/>
   <om:featureOfInterest xlink:href="http://static-namespace.gliders.eu/north-
                                     sea-123"/>
   <!-- Quality information -->
   <om:resultQuality>
      <swe:Text>
         <gml:name>QualityDescriptor</gml:name>
         <swe:value>QC</swe:value>
      </swe:Text>
   </om:resultQuality>
   <om:result>...</om:result>
</om:OM_Observation>
```

For discovery patterns it is sufficient to provide quality information on data set level granularity. A similar approach has been conducted by previous work in this area (e.g. Devaraju et al., 2015). Different quality levels could be:

- `NONE`: no quality assessment has been performed
- `QC`: quality checks have been performed
- `ADJUSTED`: quality checks have been performed and data was adjusted in post-processing

Please note: These quality levels are subject to further discussion so that an update for the final version of this document can be expected.

7.2.3 SensorML

The OGC SensorML 2.0 standard offers a powerful approach to encode a broad range of different metadata about systems (e.g. gliders) and components of such systems (e.g. instruments). Thus, it is ideally suited as a metadata format within BRIDGES. The following two subsections introduce a proposed SensorML model as well as a mapping of certain elements of the EGO NetCDF model to corresponding SensorML elements.

For indicating Observed Property (Inputs) and Classifiers within a SensorML document we recommend the usage of a vocabulary server (i.e. the server operated by NERC[24]).

7.2.3.1 SensorML Model

The following diagrams illustrate how SensorML can be used to describe gliders (types and instances) as well as instruments connected to these gliders (types and instance). If required it is even possible to add a third layer of metadata to also describe the detectors of an instrument (see Figure 9).

It is important to note that SensorML can be used to describe on the one hand types of gliders, instruments and detectors and on the other hand also specific instances. While the type description contain only those metadata which are common to all objects of the same type (e.g. the manufacturer name), the instance descriptions contain only those elements which are specific for a certain object (e.g. the serial number). This avoids redundancies in metadata management and ensures a more efficient provision of metadata.

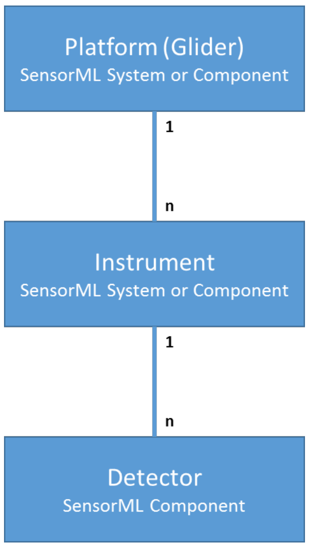


Figure 9: Layers of Proposed SensorML Model

Figure 10 shows the different elements of a description of a glider type. For example it contains references to documentation and to the manufacturer and general characteristics of the glider type.

[24] http://vocab.nerc.ac.uk/

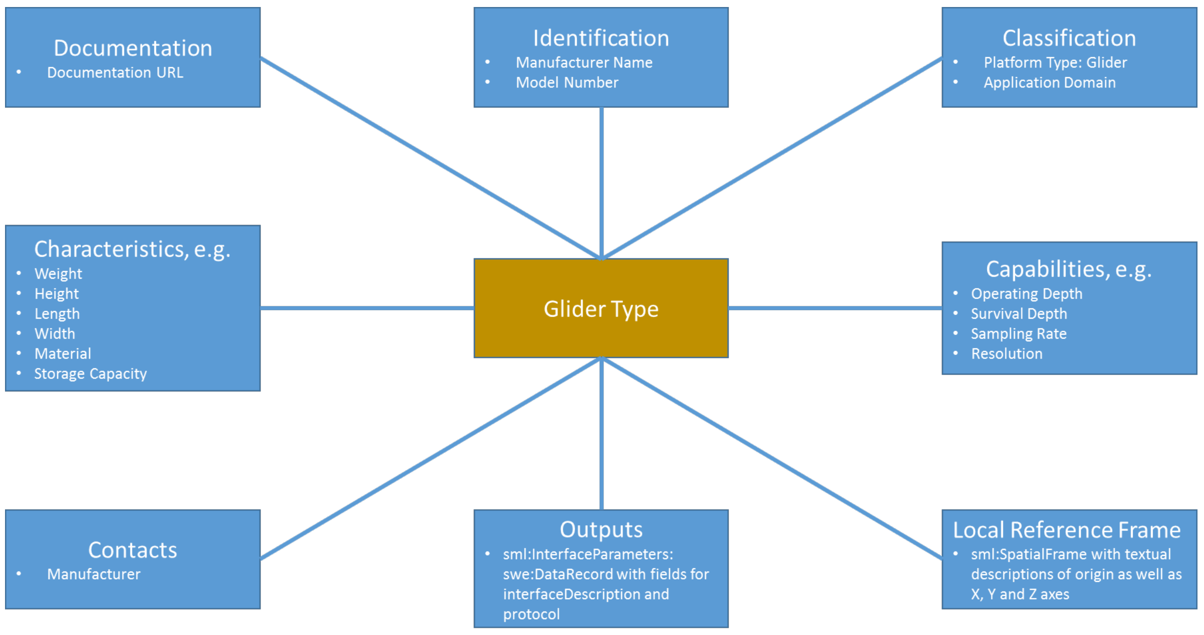


Figure 10: Elements of Glider Type Descriptions

A glider instance (see Figure 11) complements the glider type description with instance specific metadata. This comprises metadata such as individual identifiers, a history of the glider operation (e.g. deployment and maintenance events), configuration parameters, a list of attached instruments, and a reference to the operator.

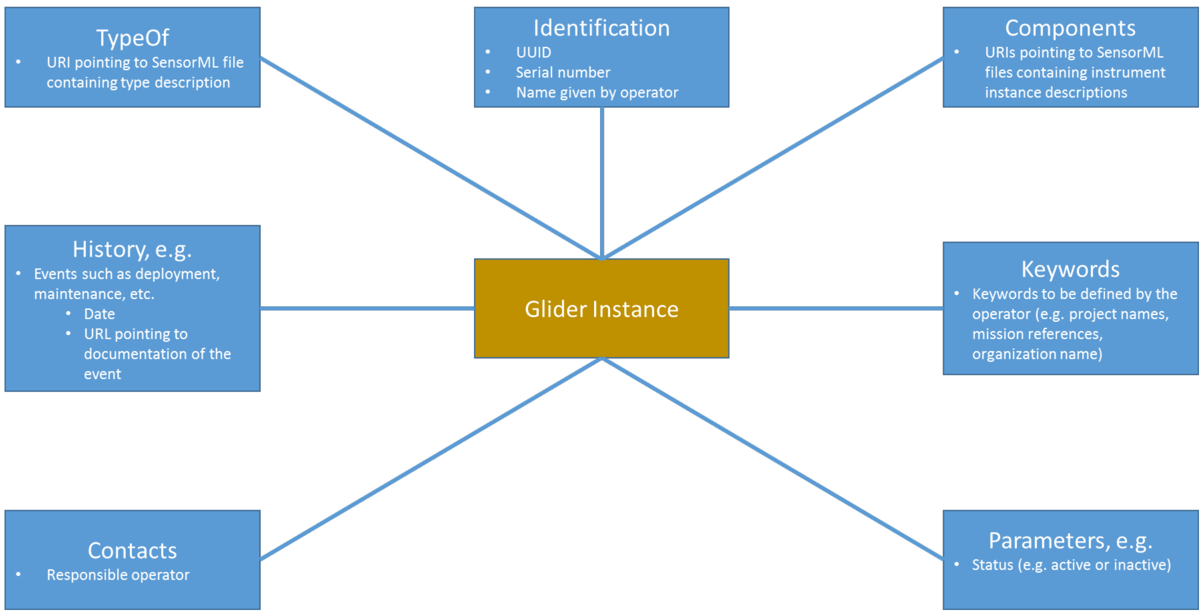

TypeOf
URI pointing to SensorML file containing type description
Identification
UUID
Serial number
Name given by operator
Components
URIs pointing to SensorML files containing instrument instance descriptions
History, e.g.
Events such as deployment, maintenance, etc.
Date
URL pointing to documentation of the event
Glider Instance
Keywords
Keywords to be defined by the operator (e.g. project names, mission references, organization name)
Contacts
Responsible operator
Parameters, e.g.
Status (e.g. active or inactive)

Figure 11: Element of Glider Instance Descriptions

Figure 12 shows a model for the metadata of an instrument type (a detector type description would follow the same approach). The main differences are the description of outputs which contains a list of the observed properties and their units of measurements as well as different information in the capabilities section which are related to the measurement capabilities of the instrument.

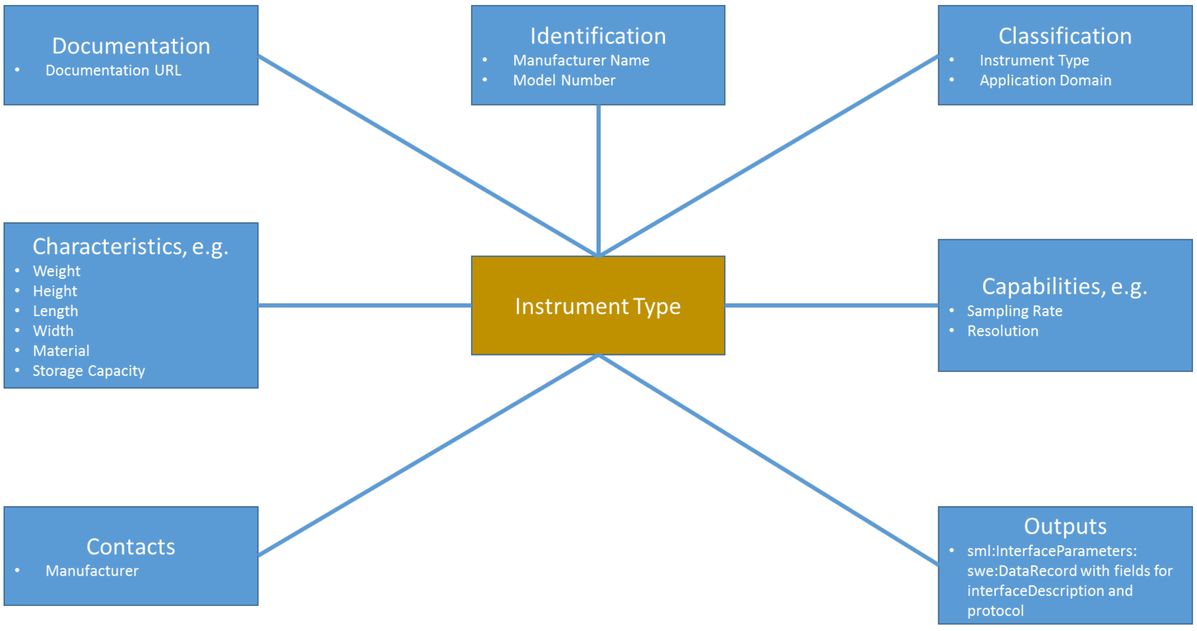


Figure 12: Elements of Instrument Type Descriptions

Finally, Figure 13 shows the structure of an instrument (or detector) instance description. In comparison to the previous examples, it contains an additional element to refer to the glider to which the instrument is attached, a description of the instrument on the glider, as well as a history showing for example calibration events.

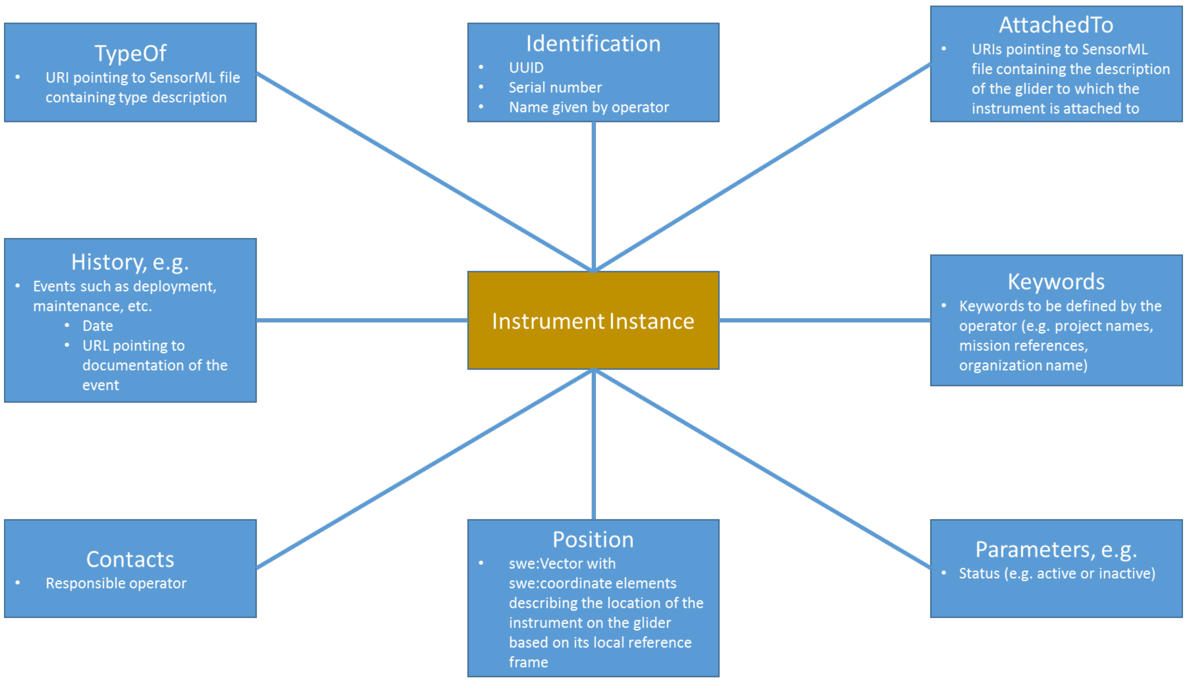

TypeOf
URI pointing to SensorML file containing type description
Identification
UUID
Serial number
Name given by operator
AttachedTo
URIs pointing to SensorML file containing the description of the glider to which the instrument is attached to
History, e.g.
Events such as deployment, maintenance, etc.
Date
URL pointing to documentation of the event
Instrument Instance
Keywords
Keywords to be defined by the operator (e.g. project names, mission references, organization name)
Contacts
Responsible operator
Position
swe:Vector with swe:coordinate elements describing the location of the instrument on the glider based on its local reference frame
Parameters, e.g.
Status (e.g. active or inactive)

Figure 13: Elements of Instrument Instance Descriptions

7.2.3.2 Mappings of NetCDF Attributes to SensorML

From a NetCDF file several metadata elements can be extracted to generate a SensorML description of an instrument or glider. This section shows, based on the EGO profile of

NetCDF, which attributes could be extracted from a NetCDF file to fill specific SensorML fields.

Table 5: Mapping of EGO NetCDF field to SensorML elements

EGO NetCDF Profile Field	SensorML 2.0 Section
TRANS_SYSTEM_ID	Glider Instance → Characteristics
TRANS_FREQUENCY	Glider Instance → Characteristics
POSITIONING_SYSTEM	Glider Instance → Characteristics
PLATFORM_FAMILY	Glider Type → Classification
PLATFORM_TYPE	Glider Type → Classification
PLATFORM_MAKER	Glider Type → Identification
FIRMWARE_VERSION_NAVIGATION	Glider Instance → Characteristics
FIRMWARE_VERSION_SCIENCE	Glider Instance → Characteristics
MANUAL_VERSION	Glider Instance → Characteristics
GLIDER_SERIAL_NO	Glider Instance → Identification
STANDARD_FORMAT_ID	Glider Type → Characteristics
DAC_FORMAT_ID	Glider Type → Classification
WMO_INST_TYPE	Glider Type → Classification
PROJECT_NAME	Glider Instance → Keywords
DATA_CENTRE	Glider Instance → Characteristics
PI_NAME	Glider Instance → Contacts
ANOMALY	Glider Instance → History
BATTERY_TYPE	Glider Instance → Characteristics
BATTERY_PACKS	Glider Instance → Characteristics
SPECIAL_FEATURES	Glider Instance → Characteristics
GLIDER_OWNER	Glider Instance → Contacts
OPERATING_INSTITUTION	Glider Instance → Contacts
CUSTOMIZATION	Glider Instance → Characteristics
DEPLOYMENT_START_DATE	Glider Instance → History
DEPLOYMENT_START_LATITUDE	Glider Instance → History
DEPLOYMENT_START_LONGITUDE	Glider Instance → History
DEPLOYMENT_START_QC	Glider Instance → History

DEPLOYMENT_PLATFORM	Glider Instance → History
DEPLOYMENT_CRUISE_ID	Glider Instance → History
DEPLOYMENT_REFERENCE_STATION_ID	Glider Instance → History
DEPLOYMENT_END_DATE	Glider Instance → History
DEPLOYMENT_END_LATITUDE	Glider Instance → History
DEPLOYMENT_END_LONGITUDE	Glider Instance → History
DEPLOYMENT_END_QC	Glider Instance → History
DEPLOYMENT_END_STATUS	Glider Instance → History
DEPLOYMENT_OPERATOR	Glider Instance → History
SENSOR	Glider Instance → Components
SENSOR_MAKER	Instrument Type → Identification
SENSOR_MODEL	Instrument Type → Identification
SENSOR_SERIAL_NO	Instrument Instance → Identification
SENSOR_PARAMETERS	Instrument Type → Outputs
PARAMETER	Instrument Type → Outputs
PARAMETER_UNITS	Instrument Type → Outputs
PARAMETER_ACCURACY	Instrument Type → Capabilities
PARAMETER_RESOLUTION	Instrument Type → Capabilities

8 System Design

Currently, software for managing glider data focusses on extracting and storing observations in the NetCDF EGO format at recovery time. No real-time data is transferred, nor are chunks of data for a missions provided in near real-time (e.g. each time the glider surfaces after a dive). This section provides a system design using OGC SWE standards that builds upon this setup, making it applicable in today's real world glider applications. In the future, a more sophisticated approach with further integration and usage of OGC SWE standards is desirable.

8.1 Component Architecture

Figure 14 illustrates how the system design extends existing solutions at a high level. The mission and recovery phases form the basis for the extension. The storage and provision phase covers the integration of OGC SWE standards, i.e. SOS, O&M and SensorML.

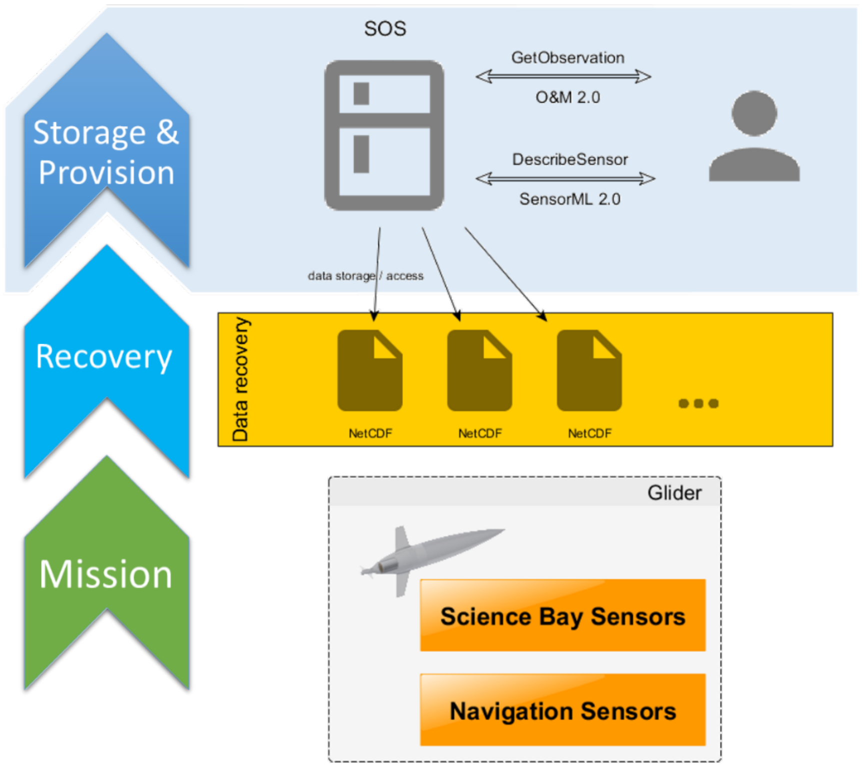


Figure 14: System Architecture Design.

The light-blue shaded area on the top of the illustration marks the contribution of this document. It highlights the fact that the proposed interoperability architecture can be placed on top of existing solutions. An implementation of the architecture should use the EGO NetCDF files to semi-automatically derive required meta-information such as the observed properties and the covered area of interest for a certain mission. A user of this architecture should be able to use the SOS instance as the single entry point to find and get desired information and data. Note that in the case the EGO NetCDF files are delivered in near real time, this system can also operate in near real time.

8.2 Glider Data Management

This sections describes the workflows for managing metadata and data of a glider system. O&M 2.0 and SensorML 2.0 are used to define interoperable encodings.

8.2.1 Sensor Data Workflow

The current de-facto standard data format for gliders is NetCDF, in particular the NetCDF EGO profile. The designed architecture makes the re-use of NetCDF EGO the central approach by wrapping NetCDF EGO in the machine- and human-readable O&M 2.0 model (see section 7.2.1). The following sections define the technical solution for this approach in combination with the Sensor Observation Service.

8.2.1.1 Sensor Observation Service Profile

In order to facilitate interoperability among SOS instances for Glider data, a service profile is required. This section defines the profile and provides some exemplary responses for common requests

Figure 15 shows the SOS interface methods of the Glider profile.

- **GetCapabilities** – this method is the main entry point to an OGC Web Services. A response provides metadata on the services capabilities and its contents. An implementing service shall provide all glider datasets via separate observation offerings. An example of a GetCapabilities response is provided in “Appendix C – SOS Profile – GetCapabilities Response”.

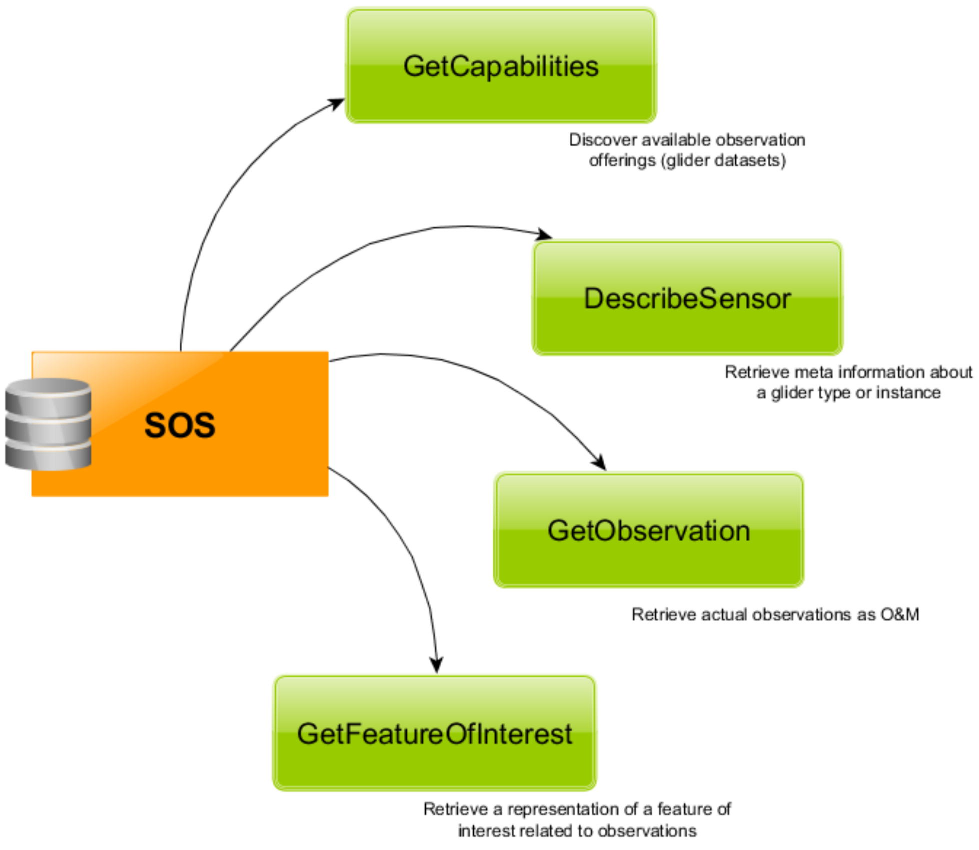

GetCapabilities
Discover available observation offerings (glider datasets)
DescribeSensor
Retrieve meta information about a glider type or instance
SOS
GetObservation
Retrieve actual observations as O&M
GetFeatureOfInterest
Retrieve a representation of a feature of interest related to observations

Figure 15: SOS interface methods of the glider data profile.

- **DescribeSensor** – this method shall return a SensorML 2.0 document as defined in section 7.2.3. The input to this method is the procedure ID of a glider instance or a type
- **GetObservation** – a client can use this method to access the observations provided by an observation offering (= a glider mission)
- **GetFeatureOfInterest** – this method shall return (a reference to) the desired feature of interest (= the area covered by a certain glider mission).

An SOS implementing the Glider profile shall serve the data in the data format

```
http://www.bridges-h2020.eu/sos/om-netcdf-ego/1.0
```

and propagate this format in the Contents section of the Capabilities document (see "Appendix C – SOS Profile – GetCapabilities Response" for an example). The structure of this format is defined in section 8.2.1.3 of this document.

8.2.1.2 Eventing

Eventing can be understood as (near) real-time dissemination of sensor data based on specific rules and patterns. Such rules range from very simple filters (e.g. data from a specific glider) to complex event patterns (e.g. threshold overshoot patterns). In the OGC SWE context eventing mechanisms can be integrated using the OGC PubSub 1.0[25] service specification. This specification defines an extension to existing OGC web services or a standalone web service that is used to implement the Publish/Subscribe paradigm[26]. An interested user can use a filter language such as XPath, OGC Filter Encoding to define the set of desired data.

Not all current gliders provide their measurements in real time in EGO NetCDF. Thus eventing for glider software architecture should focus on data availability. A user could be informed whenever a new dataset on specific observed properties or an area of interest is available. It is possible, that if near real time EGO files are transmitted, eventing could be configured to inform users when a data set is updated.

At the moment, the OGC PubSub specification is a rather new standard which has just been adopted. To better assess the suitability of this standard, 52°North is currently participating to the OGC Testbed-12 which will bring further findings and experiences how this standard could be applied within BRIDGES.

The upcoming revision of this document will cover an architecture approach for such eventing patterns based on the currently ongoing OGC activities.

8.2.1.3 NetCDF Integration

Current established glider platforms and related systems store the captured data in the NetCDF EGO format. In order to reach acceptance and support, the desired system will consider this fact and build additional patterns around the NetCDF EGO storage de-facto standard.

OGC Observations & Measurements 2.0 are the ideal solutions for re-using NetCDF EGO and enriching it by providing metadata in a standardized fashion. Similar approaches have

[25] http://www.opengeospatial.org/projects/groups/pubsubswg
[26] https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern

been proposed by certain INSPIRE Data Specifications (INSPIRE Cross Thematic Working Group on Observations & Measurements, 2014). The usage of O&M leverages the search and discovery patterns established by the use of OGC Catalogue Service for the Web (CSW) and OGC SOS.

The following parts describe general O&M encoding patterns, highlight the issues that occur when applying these to glider data as well as the approach to overcome these issues.

Listing 2 illustrates a common O&M encoding using a data structure analogue to the NetCDF EGO structure. In this Listing, the science bay parts of a glider mission are represented. The Phenomenon Time describes the time window when the actual measurements took place, whereas the Result Time is the time when the data has been processed and stored (e.g. at the recovery phase). The elements Procedure, ObservedProperty and FeatureOfInterest link to corresponding entities as described in section 7.2.

The relevant part for captured data is represented by the Result element. It contains the measured values as a time series, encoded following the definition provided in the Encoding element.

Listing 2: Observation for science data with inline encoded data values.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<om:OM_Observation gml:id="glider-0123"
                   xmlns:swe="http://www.opengis.net/swe/2.0"
                   xmlns:sos="http://www.opengis.net/sos/2.0"
                   xmlns:gml="http://www.opengis.net/gml/3.2"
                   xmlns:om="http://www.opengis.net/om/2.0"
                   xmlns:gco="http://www.isotc211.org/2005/gco"
                   xmlns:xlink="http://www.w3.org/1999/xlink">
   <om:phenomenonTime>
      <gml:TimePeriod gml:id="phenomenonTime-glider-0123">
         <gml:beginPosition>2016-01-01T00:22:00</gml:beginPosition>
         <gml:endPosition>2016-01-01T12:33:00</gml:endPosition>
      </gml:TimePeriod>
   </om:phenomenonTime>
   <om:resultTime>
      <gml:TimeInstant gml:id="resultTime-glider-0123">
         <gml:timePosition>2016-01-01T12:45:00</gml:timePosition>
      </gml:TimeInstant>
   </om:resultTime>
   <om:procedure xlink:href="http://www.bridges-h2020.eu/gliders/vendorA/0123" />
   <om:observedProperty xlink:href="http://www.bridges-
                                    h2020.eu/gliders/vendorA/0123/science" />
   <om:featureOfInterest xlink:href="http://vocab.nerc.ac.uk/collection/
                                     C16/current/21a/"/>
   <om:result>
      <swe:DataArray>
         <swe:elementCount>
            <swe:Count>
               <swe:value>2</swe:value>
            </swe:Count>
         </swe:elementCount>
         <swe:elementType name="Components">
            <swe:DataRecord>
               <swe:field name="time">
                  <swe:Quantity>
                     <swe:label>TIME</swe:label>
                     <swe:uom code="s" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="sea_water_pressure" xlink:href="http://vocab.
                                                           nerc.ac.uk/
```

```
                                                                 collection/
                                                                 P01/current/
                                                                 PPSBPR01/">
                  <swe:Quantity>
                     <swe:label>PRES</swe:label>
                     <swe:uom code="Pa" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="sea_water_temperature" xlink:href="http://vocab.
                                                                   nerc.ac.uk/
                                                                   collection/
                                                                   P02/current/
                                                                   TEMP/">
                  <swe:Quantity>
                     <swe:label>TEMP</swe:label>
                     <swe:uom code="deg" />
                  </swe:Quantity>
               </swe:field>
                <swe:field name="sea_water_electrical_conductivity" xlink:href="
                                                                    http://vocab.
                                                                    nerc.ac.uk/
                                                                    collection/
                                                                    P02/current/
                                                                    CNDC/">
                  <swe:Quantity>
                     <swe:label>CNDC</swe:label>
                     <!--// Siemens per meter //-->
                     <swe:uom code="S/m" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="sea_water_practical_salinity" xlink:href="http://
                                                              vocab.nerc.ac.uk/
                                                              collection/
                                                              P02/current/PSAL/">
                  <swe:Quantity>
                     <swe:label>PSAL</swe:label>
                     <!--// practical salinity units //-->
                     <swe:uom code="[psu]" />
                  </swe:Quantity>
               </swe:field>
            </swe:DataRecord>
         </swe:elementType>
         <swe:encoding>
            <swe:TextEncoding decimalSeparator="."
                              tokenSeparator=","
                              blockSeparator="@@" />
         </swe:encoding>
         <swe:values>
            1451607733,56.22,2.3332,4.801,13@@
            1451607738,56.43,2.3236,4.768,13@@
         </swe:values>
      </swe:DataArray>
   </om:result>
</om:OM_Observation>
```

Listing 3 is very similar to the previous one, but in comparison provides the navigational parts of the glider mission. Still, the structures follows the same encoding principle.

Listing 3: Observation for navigation data with inline encoded data values.

```
<?xml version="1.0" encoding="UTF-8"?>
<om:OM_Observation gml:id="glider-0123"
                   xmlns:swe="http://www.opengis.net/swe/2.0"
                   xmlns:sos="http://www.opengis.net/sos/2.0"
                   xmlns:gml="http://www.opengis.net/gml/3.2"
```

```
                    xmlns:om="http://www.opengis.net/om/2.0"
                    xmlns:gco="http://www.isotc211.org/2005/gco"
                    xmlns:xlink="http://www.w3.org/1999/xlink">
    <om:phenomenonTime>
       <gml:TimePeriod gml:id="phenomenonTime-glider-0123">
          <gml:beginPosition>2016-01-01T00:22:00</gml:beginPosition>
          <gml:endPosition>2016-01-01T12:33:00</gml:endPosition>
       </gml:TimePeriod>
    </om:phenomenonTime>
    <om:resultTime>
       <gml:TimeInstant gml:id="resultTime-glider-0123">
          <gml:timePosition>2016-01-01T12:45:00</gml:timePosition>
       </gml:TimeInstant>
    </om:resultTime>
    <om:procedure xlink:href="http://www.bridges-h2020.eu/gliders/vendorA/0123" />
    <om:observedProperty xlink:href="http://www.bridges-h2020.eu/gliders/vendorA/
                                     0123/navigation" />
    <!--// link to well-known vocabulary; instance here: celtic sea //-->
    <om:featureOfInterest xlink:href="http://vocab.nerc.ac.uk/collection/C16/
                                      current/21a/" />
    <om:result>
       <swe:DataArray>
          <swe:elementCount>
             <swe:Count>
                <swe:value>2</swe:value>
             </swe:Count>
          </swe:elementCount>
          <swe:elementType name="components">
             <swe:DataRecord>
                <swe:field name="time">
                   <swe:Quantity>
                      <swe:uom code="s" />
                   </swe:Quantity>
                </swe:field>
                <swe:field name="dead_reckoning_position">
                   <swe:Vector definition="http://www.opengis.net/def/property/
                                           OGC/0/PlatformLocation"
                               referenceFrame="http://www.opengis.net/def/crs/
                                               EPSG/0/4326">
                      <swe:coordinate name="latitude">
                         <swe:Quantity definition="http://dictionary.tbd#Latitude"
                                       axisID="Lat">
                            <swe:uom code="deg"/>
                         </swe:Quantity>
                      </swe:coordinate>
                      <swe:coordinate name="longitude">
                         <swe:Quantity definition="http://dictionary.tbd#
                                                   Longitude"
                                       axisID="Long">
                            <swe:uom code="deg"/>
                         </swe:Quantity>
                      </swe:coordinate>
                   </swe:Vector>
                </swe:field>
                <swe:field name="gps_position">
                   <swe:Vector definition="http://www.opengis.net/def/property/
                                           OGC/0/PlatformLocation"
                               referenceFrame="http://www.opengis.net/def/crs/
                                               EPSG /0/4326">
                      <swe:coordinate name="latitude_gps">
                         <swe:Quantity definition="http://dictionary.tbd#Latitude"
                                       axisID="Lat">
                            <swe:uom code="deg"/>
                         </swe:Quantity>
                      </swe:coordinate>
                      <swe:coordinate name="longitude_gps">
```

```
                        <swe:Quantity definition="http://dictionary.tbd
                                                  #Longitude"
                                      axisID="Long">
                           <swe:uom code="deg"/>
                        </swe:Quantity>
                     </swe:coordinate>
                  </swe:Vector>
               </swe:field>
               <swe:field name="vehicle_depth">
                  <swe:Quantity>
                     <swe:uom code="m" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="distance_to_sea_floor">
                  <swe:Quantity>
                     <swe:uom code="m" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="water_depth">
                  <swe:Quantity>
                     <swe:uom code="m" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="pitch">
                  <swe:Quantity>
                     <swe:uom code="deg" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="roll">
                  <swe:Quantity>
                     <swe:uom code="deg" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="heading">
                  <swe:Quantity>
                     <swe:uom code="deg" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="speed">
                  <swe:Quantity>
                     <swe:uom code="m/s" />
                  </swe:Quantity>
               </swe:field>
            </swe:DataRecord>
         </swe:elementType>
         <swe:encoding>
            <swe:TextEncoding decimalSeparator="."
                              tokenSeparator=","
                              blockSeparator="@@" />
         </swe:encoding>
         <swe:values>
            1451607720,54.9301168,0.2598505,55.0152449,-0.0203009,2455,122,
            2om:r577,185.23,275.11,23.0,1.3@@1451607725,54.9341233,0.2581123,
            55.0152449,-0.0203009,2465,112,2577,183.12,276.34,22.56,1.233
         </swe:values>
      </swe:DataArray>
   </om:result>
</om:OM_Observation>
```

Most of the established software components that are able to process glider EGO data do not have support for the encoding style presented above. In order to benefit from existing software solutions and still provide the interoperable discovery and inter-linking mechanisms, a combination of both NetCDF and O&M is the most suitable solution.

The following Listings provide examples on how to encode an OM_Observation document which links to an external NetCDF EGO file via the “values” element. Listing 4 illustrates the science bay parts.

Listing 4: Out-of-band observation for science data, referring to an external NetCDF file.

```
<?xml version="1.0" encoding="UTF-8"?>
<om:OM_Observation gml:id="glider-0123"
                   xmlns:swe="http://www.opengis.net/swe/2.0"
                   xmlns:sos="http://www.opengis.net/sos/2.0"
                   xmlns:gml="http://www.opengis.net/gml/3.2"
                   xmlns:om="http://www.opengis.net/om/2.0"
                   xmlns:gco="http://www.isotc211.org/2005/gco"
                   xmlns:xlink="http://www.w3.org/1999/xlink">
   <om:phenomenonTime>
      <gml:TimePeriod gml:id="phenomenonTime-glider-0123">
      <gml:beginPosition>2016-01-01T00:22:00</gml:beginPosition>
      <gml:endPosition>2016-01-01T12:33:00</gml:endPosition>
   </gml:TimePeriod>
   </om:phenomenonTime>
      <om:resultTime>
         <gml:TimeInstant gml:id="resultTime-glider-0123">
         <gml:timePosition>2016-01-01T12:45:00</gml:timePosition>
        </gml:TimeInstant>
      </om:resultTime>
   <om:procedure xlink:href="http://www.bridges-h2020.eu/gliders/vendorA/0123" />
   <om:observedProperty xlink:href="http://www.bridges-h2020.eu/gliders/vendorA/
                                     0123/science" />
   <om:featureOfInterest xlink:href="http://vocab.nerc.ac.uk/collection/C16/
                                      current/21a/" />
   <om:result>
      <swe:DataStream>
         <swe:elementType name="Components">
            <swe:DataRecord>
               <swe:field name="time">
                  <swe:Quantity>
                     <swe:label>TIME</swe:label>
                     <swe:uom code="s" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="sea_water_pressure"
xlink:href="http://vocab.nerc.
                                                    ac.uk/collection/P01/current/
                                                    PPSBPR01/">
                  <swe:Quantity>
                     <swe:label>PRES</swe:label>
                     <swe:uom code="Pa" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="sea_water_temperature" xlink:href="http://vocab.
                                                       nerc.ac.uk/collection/
                                                       P02/current/TEMP/">
                  <swe:Quantity>
                     <swe:label>TEMP</swe:label>
                     <swe:uom code="deg" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="sea_water_electrical_conductivity" xlink:href="
                                                                   http://vocab.
                                                                   nerc.ac.uk/
                                                                   collection/
                                                                   P02/current/
                                                                   CNDC/">
                  <swe:Quantity>
                     <swe:label>CNDC</swe:label>
                     <!--// Siemens per meter //-->
                     <swe:uom code="S/m" />
                  </swe:Quantity>
               </swe:field>
               <swe:field name="sea_water_practical_salinity" xlink:href="http://
                                                               vocab.nerc.ac.uk/
                                                               collection/P02/
                                                               current/PSAL/">
                  <swe:Quantity>
                     <swe:label>PSAL</swe:label>
                     <!--// practical salinity units //-->
                     <swe:uom code="[psu]" />
                  </swe:Quantity>
               </swe:field>
                </swe:DataRecord>
         </swe:elementType>
         <swe:encoding>
            <swe:BinaryEncoding byteEncoding="raw" byteOrder="bigEndian">
               <swe:member>
                  <swe:Component ref="application/x-netcdf"
                                 dataType="http://www.unidata.ucar.edu/software/
                                           netcdf/"/>
               </swe:member>
            </swe:BinaryEncoding>
         </swe:encoding>
         <swe:values xlink:href="http://www.ifremer.fr/co/ego/ego/v2/hannon/
                                 hannon_20150908/hannon_mooset00_32_R.nc"/>
      </swe:DataStream>
   </om:result>
</om:OM_Observation>
```

Listing 5 provides the navigational content. Again, the structure is very similar to the science bay parts, and the EGO NetCDF file is linked via the “values” element.

Listing 5: Out-of-band observation for navigation data, referring to an external NetCDF file.

```
<?xml version="1.0" encoding="UTF-8"?>
<om:OM_Observation gml:id="glider-0123"
                   xmlns:swe="http://www.opengis.net/swe/2.0"
                   xmlns:sos="http://www.opengis.net/sos/2.0"
                   xmlns:gml="http://www.opengis.net/gml/3.2"
                   xmlns:om="http://www.opengis.net/om/2.0"
                   xmlns:gco="http://www.isotc211.org/2005/gco"
                   xmlns:xlink="http://www.w3.org/1999/xlink">
   <om:phenomenonTime>
      <gml:TimePeriod gml:id="phenomenonTime-glider-0123">
         <gml:beginPosition>2016-01-01T00:22:00</gml:beginPosition>
         <gml:endPosition>2016-01-01T12:33:00</gml:endPosition>
      </gml:TimePeriod>
   </om:phenomenonTime>
   <om:resultTime>
      <gml:TimeInstant gml:id="resultTime-glider-0123">
         <gml:timePosition>2016-01-01T12:45:00</gml:timePosition>
      </gml:TimeInstant>
   </om:resultTime>
   <om:procedure xlink:href="http://www.bridges-h2020.eu/gliders/vendorA/0123"/>
   <om:observedProperty xlink:href="http://www.bridges-h2020.eu/gliders/vendorA/
                                    0123/navigation"/>
   <!--// link to well-known vocabulary; instance here: celtic sea //-->
   <om:FeatureOfInterest xlink:href="http://vocab.nerc.ac.uk/collection/
                                     C16/current/21a/"/>
   <om:result>
      <swe:DataStream>
         <swe:elementType name="components">
            <swe:DataRecord>
               <swe:field name="time">
                  <swe:Quantity>
```

```
                <swe:uom code="s" />
             </swe:Quantity>
          </swe:field>
          <swe:field name="dead_reckoning_position">
             <swe:Vector definition="http://www.opengis.net/def/property/
                                     OGC/0/PlatformLocation"
                          referenceFrame="http://www.opengis.net/def/crs/
                                          EPSG/0/4326">
                <swe:coordinate name="latitude">
                   <swe:Quantity definition="http://dictionary.tbd#Latitude"
                                 axisID="Lat">
                      <swe:uom code="deg"/>
                   </swe:Quantity>
                </swe:coordinate>
                <swe:coordinate name="longitude">
                   <swe:Quantity definition="http://dictionary.tbd
                                             #Longitude"
                                 axisID="Long">
                      <swe:uom code="deg"/>
                   </swe:Quantity>
                </swe:coordinate>
             </swe:Vector>
          </swe:field>
          <swe:field name="gps_position">
             <swe:Vector definition="http://www.opengis.net/def/property/
                                     OGC/0/PlatformLocation"
                         referenceFrame="http://www.opengis.net
                                         /def/crs/EPSG/0/4326">
                <swe:coordinate name="latitude_gps">
                   <swe:Quantity definition="http://dictionary.tbd#Latitude"
                                 axisID="Lat">
                      <swe:uom code="deg"/>
                   </swe:Quantity>
                </swe:coordinate>
                <swe:coordinate name="longitude_gps">
                   <swe:Quantity definition="http://dictionary.tbd
                                             #Longitude"
                                 axisID="Long">
                               <swe:uom code="deg"/>
                   </swe:Quantity>
                </swe:coordinate>
             </swe:Vector>
          </swe:field>
          <swe:field name="vehicle_depth">
             <swe:Quantity>
                <swe:uom code="m" />
             </swe:Quantity>
          </swe:field>
          <swe:field name="distance_to_sea_floor">
             <swe:Quantity>
                <swe:uom code="m" />
             </swe:Quantity>
          </swe:field>
          <swe:field name="water_depth">
             <swe:Quantity>
                <swe:uom code="m" />
             </swe:Quantity>
          </swe:field>
          <swe:field name="pitch">
             <swe:Quantity>
                <swe:uom code="deg" />
             </swe:Quantity>
          </swe:field>
          <swe:field name="roll">
             <swe:Quantity>
                <swe:uom code="deg" />
             </swe:Quantity>
          </swe:field>
```

```
                <swe:field name="heading">
                   <swe:Quantity>
                      <swe:uom code="deg" />
                   </swe:Quantity>
                </swe:field>
                <swe:field name="speed">
                   <swe:Quantity>
                      <swe:uom code="m/s" />
                   </swe:Quantity>
                </swe:field>
             </swe:DataRecord>
          </swe:elementType>
          <swe:encoding>
             <swe:BinaryEncoding byteEncoding="raw" byteOrder="bigEndian">
                <swe:member>
                   <swe:Component ref="application/x-netcdf"
                                  dataType="http://www.unidata.ucar.edu/software/
                                            netcdf/"/>
                </swe:member>
             </swe:BinaryEncoding>
          </swe:encoding>
          <swe:values xlink:href="http://www.ifremer.fr/co/ego/ego/v2/hannon/
                                  hannon_20150908/hannon_mooset00_32_R.nc"/>
       </swe:DataStream>
    </om:result>
 </om:OM_Observation>
```

The main difference between the first and second two Listings are the Encoding and Values elements. The Encoding elements defines a raw byte encoding and related meta information:

- The Endianness is defined in the byteOrder attribute. This should always be **bigEndian** as this is the common value for NetCDF files.
- The ref attribute of the Component element defines the MIME type that shall be used: **application/x-netcdf**
- The datatype attribute of the Component element defines the general type of the data, e.g. a reference to a vocabulary. For NetCDF the value http://www.unidata.ucar.edu/software/netcdf/ shall be used.

The Values element shall only contain one attribute, xlink:href. This is the link where a user or software can obtain the actual NetCDF EGO file.

This approach allows the reuse of existing NetCDF files and also enables the provision of glider data via OGC services such as SOS 2.0 implementations.

The parameters used in the swe:DataRecord element all refer to an entry in a vocabulary. In order to establish interoperability among O&M instances, the following entries shall be used, for example:

- sea_water_pressure:
  `http://vocab.nerc.ac.uk/collection/P01/current/PPSBPR01/`
- sea_water_temperature:
  `http://vocab.nerc.ac.uk/collection/P02/current/TEMP/`
- sea_water_electrical_conductivity:
  `http://vocab.nerc.ac.uk/collection/P02/current/CNDC/`
- sea_water_practical_salinity:
  `http://vocab.nerc.ac.uk/collection/P02/current/PSAL/`

- latitude: `http://vocab.nerc.ac.uk/collection/P01/current/ALATZZ01/`
- longitude: `http://vocab.nerc.ac.uk/collection/P01/current/ALONZZ01/`

8.2.2 Metadata Workflow

With regard to metadata workflows the SOS interface described in section 8.2.1.1 provides already the necessary functionality.

If a glider is deployed or if data collected by a glider (or an instrument on a glider) shall be published, it is first necessary to publish the metadata of the glider/instrument. For this purpose the InsertSensor operation of the SOS 2.0 interface shall be used. The input parameter of this operation is a SensorML document as described in section 7.2.3. Similarly, it may become necessary to update the metadata of a glider or instrument (e.g. if a calibration was performed). In this case the SOS UpdateSensor operation shall be used (in this case the updated SensorML document needs to be provided as input parameter).

For retrieving the metadata of a glider or instrument, the SOS DescribeSensor operation shall be used. This operation requires the identifier of the glider or instrument as input parameter and returns the corresponding SensorML document. As future versions of EGO NetCDF grow to include a richer vocabulary of parameters (in particular metadata and navigational parameters which may not have standard names yet), this workflow will still provide the necessary functionality.

8.3 Glider Tasking

This section provides an interoperable approach for glider tasking, or piloting, using the OGC Sensor Planning Service (SPS) standard.

The OGC SPS provides a comprehensive interface for controlling and tasking sensors. This does not only comprise the submission of certain tasks (e.g. tracks that shall be followed) but also functionality for managing tasks and for checking if a certain task is feasible (e.g. if the glider is not blocked by another task).

From the operations described in the SPS standard, the following operations were identified as required elements of the BRIDGES SPS profile:

- GetCapabilities: Access to a description of the SPS server, including an overview of supported operations, available sensors, and supported tasks.
- DescribeTasking: Access to detailed descriptions of individual tasks and the required parameters.
- Submit: Submission of requests (included task parameters) to execute a certain task.
- GetStatus: Check the execution status of a previously submitted task.
- GetTask: Retrieve a complete description of a previously submitted tasks (including all submitted parameters).
- DescribeResultAccess: As the SPS offers only an interface submitting tasks and not for retrieving the resulting observations, this operation allows to query which server (e.g. SOS server) provides access to the collected data.
- DescribeSensor: Access to the detailed metadata about a certain sensor that can be tasked through the SPS server.
- GetFeasibility: Checking if a task with certain parameters can be executed (e.g. availability of the sensor, technical feasibility of the requested task

- Update: Updating the parameters of a previously submitted task (may not be supported by all platforms or in all situations)
- Cancel: Cancelling the execution of a specific task (may not be supported by all platforms or in all situations)

Besides these mandatory operations, SPS servers may also support two additional operations for enabling the reservation of tasks (e.g. blocking a glider for a certain tasks with a later confirmation if it shall be executed).

During the remainder of the BRIDGES projects, this profile will be further refined and improved. While the set of operations that shall be supported will remain constant, the main goal will be to provide more detailed guidance how certain types of tasks may be handled through a SPS server. Based on further interactions with operators and glider manufacturers, the aim is to develop a profile (including vocabulary extensions) how certain tasks and their parameters may be encoded. Ultimately, this profile will allow the glider pilots to be “platform free” in their thinking and command execution, as fleets of gliders, possibly of different make, can be tasked simultaneously with a standard interface.

9 Implementation Patterns

This section provides a discussion, on which level the different components of the presented Sensor Web architecture will be deployed and how they may be distributed.

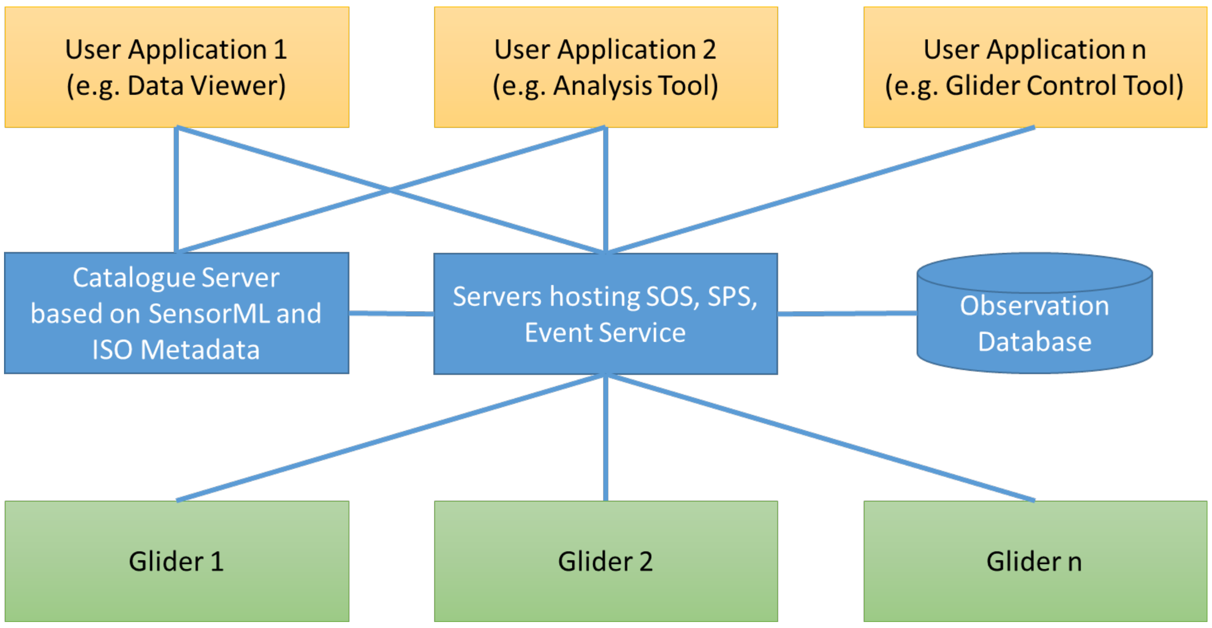


Figure 16: Overview of the components covered by the Sensor Web architecture

The green boxes in Figure 16 show the gliders and the instruments they are carrying. On the gliders software is running which ensures the functioning of the gliders and manages, among other functionalities, the collection of the data. From the gliders the collected data needs to be transferred to one or more servers for persisting and publishing the collected data on the Web. This link between the Gliders and the Sensor Web servers, may be achieved either by file transfer (e.g. transferring NetCDF files after recovery or in near real time from their base stations) or in the future by direct satellite communication links.

The Sensor Web components covered by this deliverable are marked with blue colour in Figure 16. On the one hand there needs to be a data archive consisting of an observation database and a SOS server providing interoperable access to the content of this archive. On the other hand this may be complemented by SPS or eventing server which provide additional functionality (command transmission and event notification). Furthermore, this document specifies also a SensorML profile for providing metadata about gliders, their instruments and the collected data sets). These metadata records can be used for creating registries/catalogues which allow a client to discover available data sets but also gliders and instruments that are/were used.

Based on the Sensor Web components, it is possible to build user applications (marked yellow). These components can rely on catalogues/registries for discovering the resources of interest and subsequently use SOS, SPS, and Event Services to download observation data, request certain observation tasks, or to subscribe to notifications if certain measurements become available.

During the remainder of the BRIDGES project, the content of this section will be further refined. While this version of the document provides a general overview of the architecture, the final version will provide more detailed guidance on typical implementation patterns (e.g.

how to load data from a glider into the SOS server) based on prototypical experiments. In addition, the possible role of existing user applications, such as the French GFCP (Glider Fleet Control Panel) or the SOCIB glider toolbox, will be described.

10 Conclusion

Within this document the current state of glider data acquisition and management has been documented. Based on this status quo an architecture for interoperable data discovery and access has been developed. Here, the focus was laid on well-established international standards such as the OGC Sensor Web Enablement standards suite.

The compatibility with existing solutions played an important role in the development of the architecture. The approach reuses established workflows and data formats such as the EGO NetCDF profile to maximize the probability for adoption. Analysing existing glider solutions and the corresponding requirements also supports this approach.

The result of this work is an interoperable architecture. An application encoding for Observation & Measurements 2.0 has been developed that supports the reuse of existing NetCDF infrastructures. In order to provide detailed information on specific glider systems a SensorML 2.0 profile has been developed in collaboration with related oceanographic research projects. This profile allows the definition of metadata such as calibration parameters and quality descriptors for gliders and their deployed sensors. A dedicated profile for the Sensor Observation Service 2.0 allows users and clients to discover glider datasets using a standardised interface.

In the upcoming revision of this document an approach for managing glider missions (e.g. using the OGC Sensor Planning Service 2.0) will be considered. In addition, feedback received will be gathered and incorporated in order to make the system design more applicable.

11 Future Work

The proposed interoperability architecture developed and described within this document will be circulated and discussed within relevant communities during the remainder of the BRIDGES project timeline. Especially the following groups will be targeted:

- Scientists, e.g.:
  - What are typical tools and applications that need to access the Sensor Web infrastructure?
  - Is the content of the proposed SensorML metadata profile sufficient or are further metadata elements needed?
  - Is further functionality required?
  - Discuss the integration and enhancements of vocabularies
- Developers, e.g.:
  - Which further guidance is needed to implement the proposed Sensor Web components?
  - Are there further requirements which should be considered when refining the Sensor Web architecture?
  - Which (open source) building blocks could help to facilitate the implementation of server or client components?
- Glider operators, e.g.:
  - Is there additional functionality which should be considered?
  - What are further requirements on glider tasking/control for which the tasking specification in this document should provide more details?
- Glider manufacturers, e.g.:
  - Are there further aspects which should be considered in the Sensor Web architecture?
  - How could the transfer of data between the glider and the Sensor Web servers be further facilitated?

The upcoming revision of this document will feature recommendations and updates based on the feedback and discussions received from the community.

Besides the updates based on the feedback received, the work until the final version of this document should consider the following topics:

- Patterns for automatic data integration: This shall provide further guidance how the transfer of data between the gliders (e.g. during the recovery phase) can be achieved.
- Establishment of vocabularies for glider parameters within the community: For this purpose existing vocabularies will be analysed and recommendations on potential enhancements will be provided.
- Real-time dissemination of glider data using interoperable architectures: This concerns especially push based data distribution so that data consumers receive new observation as soon as they are available (e.g. OGC PubSub 1.0 compliant services)
- Glider planning architectures: In this first revision of the document the focus was put on managing and distributing observation data. The tasking of gliders and the SPS was also addressed but with a lower level of detail. As the collection of requirements

for this functionality is a continuous process, a more detailed specification will be provided in the final version.

- Access control: As the standardisation of access control mechanisms for OGC service interfaces is still a very dynamic process, the decision was taken to consider this functionality for the final version of this document.
- Prototypical testing in order to validate selected elements of the presented Sensor Web architecture.

12 Recommendations

To gain as much benefit as possible from the efforts spent on designing the architecture described in this document, a set of recommendations has been captured. A next step should be to discuss these recommendations with the BRIDGES project partners (e.g. during a General Assembly or a dedicated workshop) and possibly a larger audience such as the EGO network.

The following topics have been described within this documented in detail. For the matter of convenience, a short summary is provided:

- **Alignment between research projects** – In order to ensure a harmonised approach it is important to organise an exchange with partners of the related projects (e.g. COMMON SENSE, NeXOS, SCHeMA). Besides establishing a common Sensor Web Enablement architecture, such an alignment could focus on the development of a marine SensorML profile which can be re-used across these projects. The inter-project work has already started, the intermediary results have been presented at the Oceanology Internal 2016 Conference[27] and the EGU General Assembly 2016[28]. During the coming month, the partners will focus on finalizing the SensorML work and establish best practice solutions.
- **Establish open source software** – within BRIDGES it will be possible to build upon the open source developments resulting from NeXOS in order to build prototypes for testing the BRIDGES Sensor Web specifications. In order to create a robust set of software and tools, the availability of open source components should be communicated and promoted (both within BRIDGES – e.g. for WP4, 5 and 6 – and externally).
- **Develop NetCDF integration best practice** – An implementation of the proposed SWE architecture should use the EGO NetCDF files to semi-automatically derive required meta-information such as the observed properties and the covered area of interest for a certain mission. A user of this architecture should be able to use the SOS instance as the single entry point to find and get desired information and data. A best practice paper will be developed that helps partners and data providers to understand and implement the described concepts. The contents of this paper will be presented at the next BRIDGES General Assembly.
- **Encourage the usage of PUCK-capable sensors** – for BRIDGES, PUCK is relevant for facilitating the link between instruments and Gliders, providing a harmonized access to metadata and data, but not for the communication between Gliders and stations on the shore. In particular, WP5 and WP6 shall consider using PUCK-capable sensors whenever possible.

[27] OI 2016 - Workshop Sensor Web Enablement (SWE): http://www.oceanologyinternational.com/Whats-On/Associated-Events/Workshop-Sensor-Web-Enablement-SWE/

[28] EGU GA 2016, ESSI1.1 Session: http://presentations.copernicus.org/EGU2016-14690_presentation.pdf

Appendix A – File Contents Examples (Seaglider)

The following files are stored on Seaglider:

Bathymetry Map Files: This files provide the Seaglider with geographic environmental information. Seaglider can carry up to 999 bathymetry maps.

Battery File: This file is used to keep track of the power consumption.

```
Pitch_motor 1193.670
Roll_motor 3216.083
VBD_pump_during_apogee 264191.594
VBD_pump_during_surface 2941.014
VBD_valve 0.000
Iridium_during_init 2273.357
Iridium_during_connect 3526.789
```

Compass Cabilration File: Created when the compass is calibrated in factory or when new batteries or sensors are added.

```
"tcm2mat.150, Sparton SP3003D, sn 165, cal 20-July-10"
-0.0231 -0.7585  1.8698  0.0061
-0.0127  0.9539  0.0654 -0.0050
 0.9697 -0.0009 -0.0120 -0.0046  1.0472 -0.0163  0.0165  0.0233
1.0427 27.8139  5.9382 -33.9168
```

The following files are created and stored on the base station:

Data File (.dat): This is an ASCII text file which is transmitted to the base station for further processing. It contains the raw sensor data and each file covers one dive of information. This file is made as compact as possible and therefore difficult to read directly by the user. However, the numbers can be interpreted using the column titles line in the header of this file. An example of the first few lines of a data file can be seen below:

```
version: 66.11
glider: 150
mission: 11
dive: 168
start: 8 1 115 10 25 39
columns:
rec,elaps_tms,depth,heading,pitch,roll,AD_pitch,AD_roll,AD_vbd,GC_pha
se,mag.x,mag.y,mag.z,sbect.TempFreq,sbect.CondFreq,sbe43.O2Freq,wlbb2
f.redCount,wlbb2f.blueCount,wlbb2f.fluorCount$
data:
0 23586 185 2963 -661 10 440 1933 1096 2 419 268 35 175899 127035
271671 72 74 51 543
1 10010 9 -118 11 40 0 -1 249 0 -22 23 40 539 -13 N N N N N
1 9986 -3 -89 -25 0 0 0 248 0 -15 12 23 -404 2 N N N N N
1 9996 37 43 1 -52 0 0 238 0 8 -9 6 -76 -6 N N N N N
1 10011 64 102 -23 -19 0 1 249 0 19 -13 -53 -137 29 N N N N N
1 10003 48 101 7 -35 0 -1 278 0 14 -14 -28 1061 62 N N N N N
1 13576 116 8 -28 43 367 1 156 -1 6 -11 -33 799 472 278486 78 75 55
543….
```

Log File (.log): This file serves as a summary of what happened during the previous dive. It contains the software version, the glider number, the mission number, the dive number, the time that the dive was started, a complete list of parameters and their values, and some summary information about the dive. An example:

```
version: 66.11
glider: 150
mission: 11
dive: 168
start: 8 1 115 10 25 39
data:
$ID,150
$MISSION,11
$DIVE,168
$N_DIVES,0
$D_SURF,2
$D_FLARE,3
$D_TGT,990
….

$MEM,227036
$DATA_FILE_SIZE,143266,2714
$CAP_FILE_SIZE,336428,0
$CFSIZE,260165632,103530496
$ERRORS,0,1,0,0,0,0,0,0,0,0,0,0,111,0,0,0
$GPS,010815,180653,3326.589,3200.869,38,1.6,39,3.5
```

Capture File (.cap): This file contains detailed information about all the actions the Seaglider performed in the previous dive and is used for debugging. This file can become quiet large and is usually not transmitted, except for the first few dives, where it's recommended to monitor the performance of the Seaglider closely. An example:

```
12204.920,SSYS,N,uploading complete capture due to CAPUPLOAD
Compressing THISDIVE.KAP to sg0059kz.x...
12227.604,SSYS,N,Capture file opened
Compressing THISDIVE.DAT to sg0059dz.x...
Compressing THISDIVE.LOG to sg0059lz.x...
12254.633,HCF8,N,file 'sg0059dz.x00' opened...
12257.033,HCF8,N,file 'sg0059dz.x00' has 8192 bytes, closed...
12258.648,HCF8,N,file 'sg0059dz.x01' opened...
….
12301.273,HBATT,N,24V batt pack voltage = 24.57V
12301.411,HBATT,N,10V batt pack voltage = 10.91V
12301.661,SDIVE,N,Measuring depth & angle for 1 sec... done.
12305.524,SDIVE,N,Measured depth: 1.80m angle: -63.13deg
12305.868,SGLMALLOC,N,glheap_walk: 228300 bytes free, 17 blocks free,
129308 bytes alloc, 16 blocks alloc
12306.000,SSURF,N,Trying call 0...
12306.047,SSURF,N,Calling phone number: 12062217301
12308.184,HPHONE,N,initializing PSTN connection
12339.062,HPHONE,N,Phone registered
….
```

The base station then generates the following files using data provided by the Seaglider:

ASCII Files (.asc): These files are the reconstituted (uncompressed, reassembled and deferentially summed) versions of the Data Files created by the Seaglider. An example:

```
version: 66.11
glider: 150
mission: 11
```

```
dive: 168
basestation_version: 2.8
start: 08 01 115 10 25 39
columns:
rec,elaps_tms,depth,heading,pitch,roll,AD_pitch,AD_roll,AD_vbd,GC_pha
se,mag.x,mag.y,mag.z,sbect.TempFreq,sbect.CondFreq,sbe43.O2Freq,wlbb2
f.redCount,wlbb2f.blueCount,wlbb2f.fluorCount$
data:
0 23586 185 2963 -661 10 440 1933 1096 2 419 268 35 175899 127035
271671 72 74 51 543
1 33596 194 2845 -650 50 440 1932 1345 2 397 291 75 176438 127022
NaN NaN NaN NaN NaN
2 43582 191 2756 -675 50 440 1932 1593 2 382 303 98 176034 127024
NaN NaN NaN NaN NaN
3 53578 228 2799 -674 -2 440 1932 1831 2 390 294 104 175958 127018
NaN NaN NaN NaN NaN
4 63589 292 2901 -697 -21 440 1933 2080 2 409 281 51 175821 127047
NaN NaN NaN NaN NaN
5 73592 340 3002 -690 -56 440 1932 2358 2 423 267 23 176882 127109
NaN NaN NaN NaN NaN
6 87168 456 3010 -718 -13 807 1933 2514 1 429 256 -10 177681 127581
278486 78 75 55 543
….
```

The following are base station log files:

Communication File (comm.log): This is a complete record of the Seaglider's communication with the base station.

```
Connected at Tue Jul 7 00:06:31 PDT 2015
logged in
1:1:2:0:11:25:-2:435:1880:1130:-55.70:1.12:11.01:24.75:9.51:39.68
GPS,070715,070253,3457.302,3400.156,7,1.7,7,3.8
EOP_CODE=CONTROL_FINISHED_OK
RECOV_CODE=QUIT_COMMAND
ver=66.11,rev=3836M,frag=8,launch=070715:053649
Iridium bars: 3 geolocation: 3443.491211,3359.635498,020508,203003
Tue Jul  7 00:06:44 2015 [sg150] Sending 143 bytes of cmdfile
Tue Jul  7 00:06:44 2015 [sg150] Sent 143 bytes of cmdfile
1:1:2:0:11:25:1 logout
Disconnected at Tue Jul 7 00:07:09 PDT 2015
….
```

BaseLog (baselog_YYMMDDHHMMSS): Contains a record of what happens when the base station tries to process the raw data files to the end data products.

```
1:23:44 27 Jan 2016 UTC: INFO: BaseLog.py(92): Process id = 4710
21:23:44 27 Jan 2016 UTC: INFO: Utils.py(670): Python version 2.7.6
21:23:44 27 Jan 2016 UTC: INFO: Utils.py(679): Numpy version 1.8.2
21:23:44 27 Jan 2016 UTC: INFO: Utils.py(688): Scipy version 0.13.3
21:23:44 27 Jan 2016 UTC: INFO: Base.py(1346): Invoked with command
line [/usr/local/basestation/Base.py --mission_dir
/home/pilot/repro/sg150_009 --verbose --make_dive_profiles --
make_dive_bp$
21:23:44 27 Jan 2016 UTC: INFO: Base.py(1348): PID:4714
21:23:44 27 Jan 2016 UTC: INFO: Base.py(1390): Started processing
21:23:44 27 Jan 2016 UTC
21:23:44 27 Jan 2016 UTC: INFO: Base.py(1418): Instrument ID = 150
21:23:44 27 Jan 2016 UTC: INFO: Base.py(911): No .pagers file found -
skipping .pagers processing
```

```
21:23:44 27 Jan 2016 UTC: INFO: Base.py(1450): Processing
comm_merged.log
21:23:44 27 Jan 2016 UTC: INFO: Base.py(1470): Finished processing
comm_merged.log
21:23:45 27 Jan 2016 UTC: INFO: Base.py(1569): Processing seaglider
selftests
21:23:45 27 Jan 2016 UTC: INFO: Base.py(1592): No new selftests to
processed
21:23:45 27 Jan 2016 UTC: INFO: Base.py(1594): Processing pdoscmd.bat
logs
21:23:45 27 Jan 2016 UTC: INFO: Base.py(1610): No pdos logfiles found
to process
21:23:45 27 Jan 2016 UTC: INFO: Base.py(1617): Processing dive(s)
21:23:45 27 Jan 2016 UTC: WARNING: Base.py(1628): No fragment size
found for 0 - using 8192 as default
21:23:45 27 Jan 2016 UTC: INFO: Base.py(280): fragment_size = 8192
21:23:45 27 Jan 2016 UTC: INFO: Base.py(383): Processing
/home/pilot/repro/sg150_009/sg0001dz
21:23:45 27 Jan 2016 UTC: INFO: Base.py(702): Checking fragment
/home/pilot/repro/sg150_009/sg0001dz.1a.x00
….
```

Engineering File (.eng): In this file the data present in .asc file is restated but converted into engineering units.

```
%version: 66.11
%glider: 150
%mission: 11
%dive: 168
%basestation_version: 2.8
%start: 08 01 115 10 25 39
%columns:
elaps_t_0000,elaps_t,depth,head,pitchAng,rollAng,pitchCtl,rollCtl,vbd
CC,rec,GC_phase,sbect.condFreq,sbect.tempFreq,sbe43.O2Freq,wlbb2f.red
Count,wlbb2f.blueCount,wlbb2f.fluorCount,wlb$
%data:
37562.586 23.586 185.000 296.300 -66.100 1.000 -10.741 0.933 397.901
0.000 2.000 8029.283 5798.782 3754.541 72.000 74.000 51.000 543.000
419.000 268.000 35.000
37572.596 33.596 194.000 284.500 -65.000 5.000 -10.741 0.905 335.352
1.000 2.000 8030.105 5781.068 NaN NaN NaN NaN NaN 397.000 291.000
75.000
37582.582 43.582 191.000 275.600 -67.500 5.000 -10.741 0.905 273.054
2.000 2.000 8029.979 5794.335 NaN NaN NaN NaN NaN 382.000 303.000
98.000
37592.578 53.578 228.000 279.900 -67.400 -0.200 -10.741 0.905 213.269
3.000 2.000 8030.358 5796.838 NaN NaN NaN NaN NaN 390.000 294.000
104.000
….
```

netCDF Files (.nc): This file captures all processed files and is self-ducumenting. NetCDF file is meant primarily for sharing data between scientific users.

Appendix B – Base station calibration file example (Seaglider)

```
%sg_calib_constants.m
% values for basestation calculations, diveplot.m, etc.
% last edited 23-Feb-15, F.Stahr

% basic glider and mission params
id_str      =     '150' ;

%mass       =     53.652      ; % in kg, for PortSusan
mass  =     53.912      ; % for open ocean waters

mission_title =   '20150707_009';
%mission_title    = 'Univ of Cyprus';

%rho0       =     1022.8      ; % in kg/m3 for Puget Sound
rho0  =     1029.2      ; % for open ocean waters

%volmax     =     52859 ; % projected for Puget Sound
%volmax =    52904.3;    ; % from Puget Sound regression, 23-Feb-15
volmax      =     52935 ; % projection for ocean ballast

% regressed from PS 23-Feb-15, hydrodynamic model params
hd_a = 1.69279e-03;
hd_b = 1.29775e-02;
hd_c = 1.04316e-05;

pitchbias = 0;    % pitch reference in regressions

% software motor limits - not used by Base 2.08
pitch_min_cnts    =     459   ;
pitch_max_cnts    =     3719  ;
roll_min_cnts     =     160   ;
roll_max_cnts     =     3940  ;
vbd_min_cnts      =     480   ;
vbd_max_cnts      =     3960  ;
vbd_cnts_per_cc = -3.9809 ; %DRH 20150708 -4.0767     ;


% CT sensors cal constants
calibcomm =   'SN 0071 cal 14-Jan-15';    % SN and cal date
t_g   =     4.36930384E-03    ;
t_h   =     6.36992015E-04    ;
t_i   =     2.57456023E-05    ;
t_j   =     2.83939021E-06    ;
c_g   =     -1.00037548E+01 ;
c_h   =     1.13004296E+00    ;
c_i   =     -1.12554014E-03   ;
c_j   =     1.76934797E-04    ;
cpcor       =     -9.5700000E-08    ;
ctcor       =     3.2500000E-06     ;
sbe_cond_freq_min =     2.8   ; % kHz, from cal for 0 salinity
sbe_cond_freq_max =     8.5   ; % kHz, est for greater than 32.5
sbe_temp_freq_min =     2.8   ; % kHz, from cal for 1 deg T
sbe_temp_freq_max =     7.0   ; % kHz, from cal for 32.5 deg T

%below splits apply at 500m level by default
QC_cond_spike_shallow=0.005; % was .006 20160128, #
```

```
0.15/ARGO_sample_interval_m, # [S/ml/m] Carnes 0.02
QC_cond_spike_deep=0.0009;  %was .001 20160128, #
0.025/ARGO_sample_interval_m, # [S/ml/m] Carnes 0.01
QC_temp_spike_shallow = 0.10;
QC_temp_spike_deep = 0.050; % below 500m; %added 20150309 DRH (J.
Bennett suggestion .01 for both)
QC_cond_spike_depth = 600;
QC_temp_spike_depth = 600;
QC_spike_comm = 'Spike defined as (abs(v2-0.5*(v3+v1)) - 0.5*abs(v3-
v1)) / 0.5*abs(d3-d1)';
QC_median_comm = 'Median Filter on C and T as follows: marked
probably bad if a point falls outside 2 std of the local 9 point
window';

QC_temp_min=10; %':-2.5, # [degC] Carnes, compare global Schmid -2.5
(labsea?) MDP -4.0
QC_temp_max=40; %':43.0, # [degC] Carnes, compare global Schmid 40.0
QC_salin_min=35.0; %':19.0, # [PSU] was 2.0 per Carnes; ditto Schmid
but we can't fly in waters that fresh
QC_salin_max=41.0; %':45.0, # [PSU] Carnes, compare global Schmid
41.0

% SBE oxygen cal constants
comm_oxy_type= 'SBE 43f';
calibcomm_oxygen = 'SN 129, 04-May-11'; % SN and cal date
Soc   = 2.4977E-04      ;
Foffset = -8.1950E+02   ;
o_a = -9.9371E-04;
o_b = 1.3160E-04;
o_c = -1.4794E-06;
o_e = 3.60E-02;
PCor = 0;   % used as flag to force usage of new algorithm

% this glider also carries WET Labs BB2F-VMG SN 450, last cal 15-Jun-
11
calibcomm_scatterometer =     'WET Labs BB2F-VMG SN 450, 07-Jun-11'; %
SN and cal date
scale_470 = 1.28e-5; %(m^-1 sr^-1)/counts
dark_counts_470 = 51;
resolution_counts_470 = 1.0;
%calibration to be implemented in processing:
%Vol_scatter_470 = scale_470*(OUTPUT - dark_counts_470); % (m^-1 sr^-
1)

scale_700 = 3.250e-6; %(m^-1 sr^-1)/counts
dark_counts_700 = 49;
resolution_counts_700 = 2.2;
%calibration to be implemented in processing:
%Vol_scatter_700 = scale_700*(OUTPUT - dark_counts_700); % (m^-1 sr^-
1)

scale_fluor = 0.0161; %(micrograms / liter)/count
dark_counts_fluor = 56;
resolution_counts_fluor = 2.2;
max_counts_fluor = 4121;
%calibration to be implemented in processing:
%Chl_conc_fluor = scale_fluor*(OUTPUT - dark_counts_fluor); %
(micrograms/liter)
```

Appendix C – SOS Profile – GetCapabilities Response

```
<?xml version="1.0" encoding="UTF-8"?>
<sos:Capabilities xmlns:sos="http://www.opengis.net/sos/2.0"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:ows="http://www.opengis.net/ows/1.1"
                  xmlns:xlink="http://www.w3.org/1999/xlink"
                  xmlns:fes="http://www.opengis.net/fes/2.0"
                  xmlns:swes="http://www.opengis.net/swes/2.0"
                  xmlns:gml="http://www.opengis.net/gml/3.2"
                  version="2.0.0"
                  xsi:schemaLocation="http://www.opengis.net/fes/2.0
                          http://schemas.opengis.net/filter/2.0/filterAll.xsd
                          http://www.opengis.net/swes/2.0
                          http://schemas.opengis.net/swes/2.0/swes.xsd
                          http://www.opengis.net/sos/2.0

http://schemas.opengis.net/sos/2.0/sosGetCapabilities.xsd
                          http://www.opengis.net/gml/3.2
                          http://schemas.opengis.net/gml/3.2.1/gml.xsd
                          http://www.opengis.net/ows/1.1
                                http://schemas.opengis.net/ows/1.1.0/owsAll.xsd">
   <ows:ServiceIdentification>
      <ows:Title xml:lang="eng">SOS for EGO Glider Data</ows:Title>
      <ows:Abstract xml:lang="eng">
         52North Sensor Observation Service - Data Access for EGO Glider Data
      </ows:Abstract>
      <ows:ServiceType>OGC:SOS</ows:ServiceType>
      <ows:ServiceTypeVersion>2.0.0</ows:ServiceTypeVersion>
      <ows:Profile>
         http://www.opengis.net/spec/OMXML/2.0/conf/countObservation
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/OMXML/2.0/conf/geometryObservation
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/OMXML/2.0/conf/measurement
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/OMXML/2.0/conf/samplingPoint
      </ows:Profile>
      <ows:Profile>
            http://www.opengis.net/spec/OMXML/2.0/conf/spatialSampling
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/core
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/daRetrieval
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/foiRetrieval
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/insertionCap
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/json
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/kvp-core
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/obsInsertion
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/pox
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/rest</ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/resultInsertion</ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/resultRetrieval
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/sensorInsertion
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/soap
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SOS/2.0/conf/updateSensorDescription
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SWE/2.0/conf/core
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SWE/2.0/conf/general-encoding-rules
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SWE/2.0/conf/text-encoding-rules
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SWE/2.0/conf/uml-block-components
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SWE/2.0/conf/uml-record-components
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SWE/2.0/conf/uml-simple-components
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SWE/2.0/conf/uml-simple-encodings
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SWE/2.0/conf/xsd-block-components
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SWE/2.0/conf/xsd-record-components
      </ows:Profile>
      <ows:Profile>
         http://www.opengis.net/spec/SWE/2.0/conf/xsd-simple-components
      </ows:Profile>
      <ows:Profile>
http://www.opengis.net/spec/SWE/2.0/conf/xsd-simple-encodings
      </ows:Profile>
      <ows:Fees>NONE</ows:Fees>
      <ows:AccessConstraints>NONE</ows:AccessConstraints>
   </ows:ServiceIdentification>
   <ows:ServiceProvider>
      <ows:ProviderName>52North</ows:ProviderName>
      <ows:ProviderSite xlink:href="http://52north.org/swe"/>
      <ows:ServiceContact>
         <ows:IndividualName>TBA</ows:IndividualName>
         <ows:PositionName>TBA</ows:PositionName>
         <ows:ContactInfo>
            <ows:Phone>
               <ows:Voice>+49(0)251/396 371-0</ows:Voice>
            </ows:Phone>
            <ows:Address>
               <ows:DeliveryPoint>Martin-Luther-King-Weg 24</ows:DeliveryPoint>
               <ows:City>Münster</ows:City>
```

```
                <ows:AdministrativeArea>
                   North Rhine-Westphalia
                </ows:AdministrativeArea>
                <ows:PostalCode>48155</ows:PostalCode>
                <ows:Country>Germany</ows:Country>
                <ows:ElectronicMailAddress>
                   info@52north.org
                </ows:ElectronicMailAddress>
             </ows:Address>
          </ows:ContactInfo>
       </ows:ServiceContact>
    </ows:ServiceProvider>
    <ows:OperationsMetadata>
       <ows:Operation name="GetCapabilities">
          <ows:DCP>
             <ows:HTTP>
                <ows:Get xlink:href="http://bridges.demo.52north.org/EGO-glider-
                                    sos/service/kvp?">
                   <ows:Constraint name="Content-Type">
                      <ows:AllowedValues>
                         <ows:Value>application/x-kvp</ows:Value>
                      </ows:AllowedValues>
                   </ows:Constraint>
                </ows:Get>
                <ows:Post xlink:href="http://bridges.demo.52north.org/EGO-glider-
                                     sos/service/pox">
                   <ows:Constraint name="Content-Type">
                      <ows:AllowedValues>
                         <ows:Value>application/xml</ows:Value>
                         <ows:Value>text/xml</ows:Value>
                      </ows:AllowedValues>
                   </ows:Constraint>
                </ows:Post>
             </ows:HTTP>
          </ows:DCP>
          <ows:Parameter name="AcceptFormats">
             <ows:AllowedValues>
                <ows:Value>application/xml</ows:Value>
             </ows:AllowedValues>
          </ows:Parameter>
          <ows:Parameter name="AcceptVersions">
             <ows:AllowedValues>
                <ows:Value>2.0.0</ows:Value>
             </ows:AllowedValues>
          </ows:Parameter>
          <ows:Parameter name="Sections">
             <ows:AllowedValues>
                <ows:Value>All</ows:Value>
                <ows:Value>Contents</ows:Value>
                <ows:Value>FilterCapabilities</ows:Value>
                <ows:Value>InsertionCapabilities</ows:Value>
                <ows:Value>OperationsMetadata</ows:Value>
                <ows:Value>ServiceIdentification</ows:Value>
                <ows:Value>ServiceProvider</ows:Value>
             </ows:AllowedValues>
          </ows:Parameter>
          <ows:Parameter name="updateSequence">
             <ows:AnyValue/>
          </ows:Parameter>
       </ows:Operation>
       <ows:Operation name="DescribeSensor">
          <ows:DCP>
             <ows:HTTP>
                <ows:Get xlink:href="http://bridges.demo.52north.org/EGO-glider-
                                    sos/service/kvp?">
                   <ows:Constraint name="Content-Type">
                      <ows:AllowedValues>
                         <ows:Value>application/x-kvp</ows:Value>
```

```
               </ows:AllowedValues>
            </ows:Constraint>
         </ows:Get>
         <ows:Post xlink:href="http://bridges.demo.52north.org/EGO-glider-
                               sos/service/pox">
            <ows:Constraint name="Content-Type">
               <ows:AllowedValues>
                  <ows:Value>application/xml</ows:Value>
                  <ows:Value>text/xml</ows:Value>
               </ows:AllowedValues>
            </ows:Constraint>
         </ows:Post>
      </ows:HTTP>
   </ows:DCP>
   <ows:Parameter name="procedure">
      <ows:AllowedValues>
         <ows:Value>
            http://static-namespace.gliders.eu/glider-0123
          </ows:Value>
      </ows:AllowedValues>
   </ows:Parameter>
   <ows:Parameter name="procedureDescriptionFormat">
      <ows:AllowedValues>
         <ows:Value>http://www.opengis.net/sensorML/1.0.1</ows:Value>
      </ows:AllowedValues>
   </ows:Parameter>
   <ows:Parameter name="validTime">
      <ows:AnyValue/>
   </ows:Parameter>
</ows:Operation>
<ows:Operation name="GetFeatureOfInterest">
   <ows:DCP>
      <ows:HTTP>
         <ows:Get xlink:href="http://bridges.demo.52north.org/EGO-glider-
                              sos/service/kvp?">
            <ows:Constraint name="Content-Type">
               <ows:AllowedValues>
                  <ows:Value>application/x-kvp</ows:Value>
               </ows:AllowedValues>
            </ows:Constraint>
         </ows:Get>
         <ows:Post xlink:href="http://bridges.demo.52north.org/EGO-glider-
                               sos/service/pox">
            <ows:Constraint name="Content-Type">
               <ows:AllowedValues>
                  <ows:Value>application/xml</ows:Value>
                  <ows:Value>text/xml</ows:Value>
               </ows:AllowedValues>
            </ows:Constraint>
         </ows:Post>
      </ows:HTTP>
   </ows:DCP>
   <ows:Parameter name="featureOfInterest">
      <ows:AllowedValues>
         <ows:Value>
            http://vocab.nerc.ac.uk/collection/C16/current/21a/
         </ows:Value>
      </ows:AllowedValues>
   </ows:Parameter>
   <ows:Parameter name="observedProperty">
      <ows:AllowedValues>
         <ows:Value>
            http://vocab.nerc.ac.uk/collection/P01/current/PPSBPR01/
         </ows:Value>
         <ows:Value>
            http://vocab.nerc.ac.uk/collection/P02/current/TEMP/
         </ows:Value>
```

```
               <ows:Value>
                  http://vocab.nerc.ac.uk/collection/P02/current/CNDC
               </ows:Value>
               <ows:Value>
                  http://vocab.nerc.ac.uk/collection/P02/current/PSAL/
               </ows:Value>
            </ows:AllowedValues>
         </ows:Parameter>
         <ows:Parameter name="procedure">
            <ows:AllowedValues>
               <ows:Value>
                  http://static-namespace.gliders.eu/glider-0123
               </ows:Value>
            </ows:AllowedValues>
         </ows:Parameter>
         <ows:Parameter name="spatialFilter">
            <ows:AllowedValues>
               <ows:Range>
                  <ows:MinimumValue>0.0 -122.6819</ows:MinimumValue>
                  <ows:MaximumValue>52.0464393 13.72376</ows:MaximumValue>
               </ows:Range>
            </ows:AllowedValues>
         </ows:Parameter>
      </ows:Operation>
      <ows:Operation name="GetObservation">
         <ows:DCP>
            <ows:HTTP>
               <ows:Get xlink:href="http://bridges.demo.52north.org/EGO-glider-
                                    sos/service/kvp?">
                  <ows:Constraint name="Content-Type">
                     <ows:AllowedValues>
                        <ows:Value>application/x-kvp</ows:Value>
                     </ows:AllowedValues>
                  </ows:Constraint>
               </ows:Get>
               <ows:Post xlink:href="http://bridges.demo.52north.org/EGO-glider-
                                     sos/service/pox">
                  <ows:Constraint name="Content-Type">
                     <ows:AllowedValues>
                        <ows:Value>application/xml</ows:Value>
                        <ows:Value>text/xml</ows:Value>
                     </ows:AllowedValues>
                  </ows:Constraint>
               </ows:Post>
            </ows:HTTP>
         </ows:DCP>
         <ows:Parameter name="featureOfInterest">
            <ows:AllowedValues>
               <ows:Value>
                  http://vocab.nerc.ac.uk/collection/C16/current/21a/
               </ows:Value>
            </ows:AllowedValues>
         </ows:Parameter>
         <ows:Parameter name="observedProperty">
            <ows:AllowedValues>
               <ows:Value>
                  http://vocab.nerc.ac.uk/collection/P01/current/PPSBPR01/
               </ows:Value>
               <ows:Value>
                  http://vocab.nerc.ac.uk/collection/P02/current/TEMP/
               </ows:Value>
               <ows:Value>
                  http://vocab.nerc.ac.uk/collection/P02/current/CNDC/
               </ows:Value>
               <ows:Value>
                  http://vocab.nerc.ac.uk/collection/P02/current/PSAL/
               </ows:Value>
               <ows:Value>op_2</ows:Value>
            </ows:AllowedValues>
         </ows:Parameter>
         <ows:Parameter name="offering">
            <ows:AllowedValues>
               <ows:Value>http://www.52north.org/test/offering/1</ows:Value>
            </ows:AllowedValues>
         </ows:Parameter>
         <ows:Parameter name="procedure">
            <ows:AllowedValues>
               <ows:Value>
                  http://static-namespace.gliders.eu/glider-0123
               </ows:Value>
            </ows:AllowedValues>
         </ows:Parameter>
         <ows:Parameter name="responseFormat">
            <ows:AllowedValues>
               <ows:Value>http://www.opengis.net/om/2.0</ows:Value>
            </ows:AllowedValues>
         </ows:Parameter>
         <ows:Parameter name="spatialFilter">
            <ows:AllowedValues>
               <ows:Range>
                  <ows:MinimumValue>0.0 -122.6819</ows:MinimumValue>
                  <ows:MaximumValue>52.0464393 13.72376</ows:MaximumValue>
               </ows:Range>
            </ows:AllowedValues>
         </ows:Parameter>
         <ows:Parameter name="temporalFilter">
            <ows:AllowedValues>
               <ows:Range>
                  <ows:MinimumValue>2008-10-29T00:00:00.000Z</ows:MinimumValue>
                  <ows:MaximumValue>2015-12-09T13:45:15.000Z</ows:MaximumValue>
               </ows:Range>
            </ows:AllowedValues>
         </ows:Parameter>
      </ows:Operation>
      <ows:Parameter name="service">
         <ows:AllowedValues>
            <ows:Value>SOS</ows:Value>
         </ows:AllowedValues>
      </ows:Parameter>
      <ows:Parameter name="version">
         <ows:AllowedValues>
            <ows:Value>2.0.0</ows:Value>
         </ows:AllowedValues>
      </ows:Parameter>
   </ows:OperationsMetadata>
   <sos:filterCapabilities>
      <fes:Filter_Capabilities>
         <fes:Conformance>
            <fes:Constraint name="ImplementsQuery">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsAdHocQuery">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsFunctions">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsResourceId">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsMinStandardFilter">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsStandardFilter">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsMinSpatialFilter">
               <ows:NoValues/>
               <ows:DefaultValue>true</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsSpatialFilter">
               <ows:NoValues/>
               <ows:DefaultValue>true</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsMinTemporalFilter">
               <ows:NoValues/>
               <ows:DefaultValue>true</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsTemporalFilter">
               <ows:NoValues/>
               <ows:DefaultValue>true</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsVersionNav">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsSorting">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsExtendedOperators">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsMinimumXPath">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
            <fes:Constraint name="ImplementsSchemaElementFunc">
               <ows:NoValues/>
               <ows:DefaultValue>false</ows:DefaultValue>
            </fes:Constraint>
         </fes:Conformance>
         <fes:Spatial_Capabilities>
            <fes:GeometryOperands>
               <fes:GeometryOperand xmlns:ns="http://www.opengis.net/gml/3.2"
                                    name="ns:Envelope"/>
            </fes:GeometryOperands>
            <fes:SpatialOperators>
               <fes:SpatialOperator name="BBOX">
                  <fes:GeometryOperands>
                     <fes:GeometryOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:Envelope"/>
                  </fes:GeometryOperands>
               </fes:SpatialOperator>
            </fes:SpatialOperators>
         </fes:Spatial_Capabilities>
         <fes:Temporal_Capabilities>
            <fes:TemporalOperands>
               <fes:TemporalOperand xmlns:ns="http://www.opengis.net/gml/3.2"
                                    name="ns:TimeInstant"/>
               <fes:TemporalOperand xmlns:ns="http://www.opengis.net/gml/3.2"
                                    name="ns:TimePeriod"/>
            </fes:TemporalOperands>
            <fes:TemporalOperators>
               <fes:TemporalOperator name="Before">
                  <fes:TemporalOperands>
```

```
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="After">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="Begins">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="Ends">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="EndedBy">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="BegunBy">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="During">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                          name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="TEquals">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="TContains">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="TOverlaps">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="Meets">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="MetBy">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
               <fes:TemporalOperator name="OverlappedBy">
                  <fes:TemporalOperands>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimeInstant"/>
                     <fes:TemporalOperand
xmlns:ns="http://www.opengis.net/gml/3.2"
                                         name="ns:TimePeriod"/>
                  </fes:TemporalOperands>
               </fes:TemporalOperator>
            </fes:TemporalOperators>
         </fes:Temporal_Capabilities>
      </fes:Filter_Capabilities>
   </sos:filterCapabilities>
   <sos:contents>
      <sos:Contents>
         <swes:procedureDescriptionFormat>
```

```
             http://www.opengis.net/sensorML/1.0.1
          </swes:procedureDescriptionFormat>
          <swes:observableProperty>
             http://vocab.nerc.ac.uk/collection/P01/current/PPSBPR01/
          </swes:observableProperty>
          <swes:observableProperty>
             http://vocab.nerc.ac.uk/collection/P02/current/TEMP/
          </swes:observableProperty>
          <swes:observableProperty>
             http://vocab.nerc.ac.uk/collection/P02/current/CNDC/
          </swes:observableProperty>
          <swes:observableProperty>
             http://vocab.nerc.ac.uk/collection/P02/current/PSAL/
          </swes:observableProperty>
          <swes:offering>
             <sos:ObservationOffering xmlns:ns="http://www.opengis.net/sos/2.0">
                <swes:identifier>
                   http://www.52north.org/test/offering/1
                </swes:identifier>
                <swes:procedure>
                   http://static-namespace.gliders.eu/glider-0123
                </swes:procedure>
                <sos:observedArea>
                   <gml:Envelope>
                      <gml:lowerCorner>54.9301168 1.2598505</gml:lowerCorner>
                      <gml:upperCorner>52.5654455 0.2231123</gml:upperCorner>
                   </gml:Envelope>
                </sos:observedArea>
                <sos:phenomenonTime>
                   <gml:TimePeriod gml:id="phenomenonTime_1">
                      <gml:beginPosition>2016-01-01T00:22:00</gml:beginPosition>
                      <gml:endPosition>2016-01-01T12:33:00</gml:endPosition>
                   </gml:TimePeriod>
                </sos:phenomenonTime>
                <sos:resultTime>
                   <gml:TimePeriod gml:id="resultTime_1">
                      <gml:beginPosition>2016-01-01T00:22:00</gml:beginPosition>
                      <gml:endPosition>2016-01-01T12:33:00</gml:endPosition>
                   </gml:TimePeriod>
                </sos:resultTime>
                <sos:responseFormat>
                   http://www.bridges-h2020.eu/sos/om-netcdf-ego/1.0
                </sos:responseFormat>
             </sos:ObservationOffering>
          </swes:offering>
       </sos:Contents>
    </sos:contents>
 </sos:Capabilities>
```