SemSorGrid4Env

FP7-223913

Deliverable

D5.2v2

Implementation and Deployment of a Library of the High-level Application Programming Interfaces

Alex J. Frazer (Editor), David De Roure, Kirk Martinez, and Bart Nagel, Kevin R. Page and Jason Sadler

University of Southampton

24/02/2011

<Status: Final>

<Scheduled Delivery Date: 28/02/2011>

Executive Summary

This deliverable presents the completed implementation of the SemsorGrid4Env high-level Application Programming Interface (HLAPI) to sensor observation data, both from sensor measurement databases and through the SemsorGrid4Env architecture.

The high-level API service is designed to support rapid development of thin web applications and mashups beyond the state of the art in GIS, while maintaining compatibility with existing tools and expectations. It provides a fully configurable API, while maintaining a separation of concerns between domain experts, service administrators and mashup developers. It adheres to REST and Linked Data principles, and provides a novel bridge between standards-based (OGC O&M) and Semantic Web approaches.

This document discusses the background motivations for the HLAPI (including experiences gained from any previously implemented versions), before moving onto specific details of the final implementation, including configuration and deployment instructions, as well as a full tutorial to assist mashup developers with using the exposed observation data.

Note on Sources and Original Contributions

The SemSorGrid4Env consortium is an inter-disciplinary team, and in order to make deliverables self-contained and comprehensible to all partners, some deliverables thus necessarily include state-of-the-art surveys and associated critical assessment. Where there is no advantage to recreating such materials from first principles, partners follow standard scientific practice and occasionally make use of their own pre-existing intellectual property in such sections. In the interests of transparency, we here identify the main sources of such pre-existing materials in this deliverable:

- Several sections build upon experience and details from prior deliverables including [D5.1], [D5.2v1] and [D7.4v1].
- Additional diagrams are taken from [D1.3v1], [D4.3v2] and [D5.2v1], and are cited accordingly in the main text.
- We also include material previously published as [Pag2009] in chapter 2.

Document Information

Contract Number	FP7-223913	**Acronym**	SemSorGrid4Env
Full title	SemSorGrid4Env: Semantic Sensor Grids for Rapid Application Development for Environmental Management		
Project URL	www.semsorgrid4env.eu		
Document URL			
EU Project officer	Antonios Barbas		

Deliverable	**Number**	D5.2v2	**Name**	Implementation and Deployment of a Library of the High-level Application Programming Interfaces		
Task	**Number**	5.2	**Name**	Implement and deploy a library of the High-level Application Programming Interfaces		
Work package	**Number**	WP5				
Date of delivery	**Contractual**		28/02/2011	**Actual**	28/02/2011	
Code name	D5.2v2			**Status**	draft ☑	final ☐
Nature	Prototype ☑ Report ☐ Specification ☐ Tool ☐ Other ☐					
Distribution Type	Public ☑ Restricted ☐ Consortium ☐					
Authoring Partner	SOTON					
QA Partner	UNIMAN					
Contact Person	Kirk Martinez					
	Email	km@ecs.soton.ac.uk	**Phone**	+44 2380 594491	**Fax**	+44 2380 592865
Abstract (for dissemination)	The high-level API service is designed to support rapid development of thin web applications and mashups beyond the state of the art in GIS, while maintaining compatibility with existing tools and expectations. It provides a fully configurable API, while maintaining a separation of concerns between domain experts, service administrators and mashup developers. It adheres to REST and Linked Data principles, and provides a novel bridge between standards-based (OGC O&M) and Semantic Web approaches. This document discusses the background motivations for the HLAPI (including experiences gained from any previously implemented versions), before moving onto specific details of the final implementation, including configuration and deployment instructions, as well as a full tutorial to assist mashup developers with using the exposed observation data.					
Keywords	HLAPI, REST, Linked Data, Mashups, Sensor Networks					

Version log/Date	**Change**	**Author**
0.1 / 21/01/2011	**Initial document planning**	**A. Frazer, R. R. Page**
0.2 / 14/02/2011	**Import from 5.2v1**	**A. Frazer, K. R. Page**
0.3 / 14/02/2011	**Skeleton outline for new sections**	**A. Frazer**
0.31 / 14/02/2011	**Revised skeleton, added text**	**K. R. Page**
0.4 / 16/02/2011	**Added text**	**A. Frazer**
0.5 / 17/02/2011	**Added diagrams**	**A. Frazer**
0.6 / 18/02/2011	**Revised structure**	**K. R. Page**
0.9 / 19/02/2011	**Final document for QA submission**	**A. Frazer**
1.0 / 24/02/2011	**Final document for submission**	**A. Frazer**

Project Information

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number FP7-223913. The Beneficiaries in this project are:

Partner	Acronym	Contact
Universidad Politécnica de Madrid (Coordinator)	UPM	Prof. Dr. Asunción Gómez-Pérez Facultad de Informática Departamento de Inteligencia Artificial Campus de Montegancedo, sn Boadilla del Monte 28660 Spain #e asun@fi.upm.es #t +34-91 336-7439, #f +34-91 352-4819
The University of Manchester	UNIMAN	Prof Norman Paton Department of Computer Science The University of Manchester Oxford Road Manchester, M13 9PL, United Kingdom #e npaton@cs.man.ac.uk #t +44-161-275 69 10, #f +44-161-275 62 04
National and Kapodistrian University of Athens	NKUA	Prof. Manolis Koubarakis University Campus, Ilissia Athina GR-15784 Greece #@ koubarak@di.uoa.gr #t +30 210 7275213, #f +30 210 7275214
University of Southampton	SOTON	Prof. David De Roure University Road Southampton SO17 1BJ United Kingdom #@ dder@ecs.soton.ac.uk #t +44 23 80592418, #f +44 23 80595499
Deimos Space, S.L.	DMS	Mr. Agustín Izquierdo Ronda de Poniente 19, Edif. Fiteni VI, P 2, 2º Tres Cantos, Madrid – 28760 Spain #@agustin.izquierdo@deimos-space.com #t +34-91-8063450, #f +34-91-806-34-51
EMU Limited	EMU	Dr. Bruce Tomlinson Mill Court, The Sawmills, Durley number 1 Southampton, SO32 2EJ – United Kingdom #@ bruce.tomlinson@emulimited.com #t +44 1489 860050, #f +44 1489 860051
TechIdeas Asesores Tecnológicos, S.L.	TI	Dr. Jesús E. Gabaldón C/ Marie Curie 8-14 08042 Barcelona, Spain #@ jesus.gabaldon@techideas.es #t +34.93.291.77.27, #f ++34.93.291.76.00

Table of Contents

Glossary

API	Application Programming Interface
CCO	Channel Coastal Observatory
GeoJSON	Geographic JavaScript Object Notation
GIS	Geographic Information System
GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
IQS	Integration Query Service
JSON	JavaScript Object Notation
O&M	Observations and Measurements
OGC	Open Geospatial Consortium
OWL	Web Ontology Language
REST	Representational State Transfer
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WFS	Web Feature Service
XML	Extensible Markup Language

1. Introduction

The software forming this deliverable is a configurable, reusable service for exposing observations through a High-level API (HLAPI). This includes a documented worked example of exposing the CCO sensor data network as a High-level API.

In this section, we briefly outline the system objectives for the final version of the HLAPI engine, before providing an overview of the document structure.

1.1. System objectives

The objectives for the final version of the SemsorGrid4Env High-level API engine are to:

- Support developers with lightweight, self-descriptive HLAPIs
    - Enable them to quickly build bespoke applications using data in new and previously unforseen ways
- Support data publishers with a configurable API
    - Allow data to be capturing and linked to domain models
    - Establish a separation of concerns and expertise between the system administrator, domain expert, and Web application developer
- Integrate with SemsorGrid4Env architecture

1.2. Document Outline

The document begins with a theoretical background and motivation section, where we discuss the principles on which the current HLAPI engine is based. This includes a review of the previous two SemsorGrid4Env HLAPI implementations, and how their corresponding strengths and weaknesses informed the design of the current HLAPI engine.

This is followed by a discussion of the specific implementation details of the current HLAPI engine, including Use Case and Interaction diagrams for both database- and architecture-driven Use Cases. An overview of the mapping and configuration files used by the HLAPI engine is presented, along with a per-component breakdown of the HLAPI engine itself. The section concludes with an example configuration for setting up and using the HLAPI engine with the CCO sensor network database.

The available source code and dependencies are described next, followed by details of the sample source data used in the HLAPI engine's development. Specific compilation, configuration and installation instructions are given, along with Appendices containing example mapping and configuration files, as well as a tutorial for including the HLAPI observation serialisations in a mashup Web application.

1.2.1. Changes since D5.2v1

The principle change since D5.2v1 is the development of a generalised service to provide High-Level APIs of Observation data (detailed in Section 3). Changes to the HLAPI engine since the original CCO HLAPI implementation described in D5.2v1 are discussed in Section 2.3.1, while changes made since the v.1 application tier libraries introduced in D7.4v1 are discussed in Section 2.3.2.

2. High-level API background and motivation

In this chapter we briefly discuss the underlying principles and practices of our approach to developing the high-level API (HLAPI).

We begin with an overview of the REST and Linked Data principles on which the HLAPI is built, followed by discussion of the various serialised file formats produced by the HLAPI engine and the method of negotiating between them. Previous versions of SemsorGrid4Env REST interfaces are then considered, highlighting the experiences gained from their development with respect to the current HLAPI engine.

2.1. HLAPIs for Observation Data

2.1.1. REST and Linked Data Principles

In [D5.1] we outlined our motivation in developing a REST interface for SemSorGrid4Env: principally in support of domain developers[1] to provide for rapid development of thin applications (web applications and mashups) such as those identified in [D7.1].

To briefly recap, the key principles of REST [Fie2000] are:

- Everything is a resource which is addressable
- Resources have multiple representations
- Relationships between resources are expressed through hyperlinks
- All resources share a common interface with a limited set of operations
- Client server communication is stateless

While a RESTful approach is key to enabling web applications and mashups, we must also remember that the users of our API are *domain* developers and that a rich body of GIS tools and standards (primarily OGC based) are in use that our API must co-exist, if not integrate, with.

Finally, SemSorGrid4Env seeks a *semantic* solution, and our approach in this regard is to provide *Linked Data* (as previewed in [D5.1]) through the high-level API. In the same way that REST identifies and emphasises the defining aspects of web architecture for the document web, so Linked Data prioritises web architecture on the semantic web through four basic design principles [Ber2006]:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names

[1] We distinguish between *domain users* and *domain developers*. In the example of the Flood Use case (WP7) the domain users are those who use the web applications and mashups, such as the emergency planning and decision support web applications described in [D7.1]. Domain developers are users of the high-level API: those who build the web applications and mashups using the high-level API which will then be used by the domain users.

3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs, so that they can discover more things.

While the use of URIs is common throughout the Semantic Web - not least as the basic element of RDF - the requirement to use HTTP URIs sets Linked Data deployment apart. It is a departure from the use of URIs purely as unique identifiers within the graph; in Linked Data they are also a means of retrieving parts of the graph relevant to that resource - the URIs can be dereferenced. The mechanism by which appropriate representations are dereferenced from a URI is detailed in [Pag2009], and explained in the context of the HLAPI in section 2.2.2.

2.2. Requirements

2.2.1. Input requirements

With respect to the SemsorGrid4Env project, there are two required inputs that the HLAPI engine must be able to expose: data stored in sensor measurement databases, and any data exposed through the SemsorGrid4Env architecture.

2.2.1.1. Sensor measurements databases

The HLAPI engine needs to be able to translate any stored data representation of sensor readings into the *observation model* – the ontology developed as part of the SemsorGrid4Env project to express details of observed measurements, and the devices used to measure them (presented in [D4.3v2]). The elements that ultimately make up this observation could be stored in any number of database configurations. Therefore, a configurable mapping is required in order to deal with the unpredictable nature of these data structures.

The CCO stored data service is an example of a sensor measurement database. The CCO stored data service records various observed properties (e.g. wave height, wind speed, air temperature) from a number of different sensing platforms around the south coast of England. A separate database table represents each sensing platform, and each row of a table represents the set of all measurements recorded by the sensing platform at a given time.

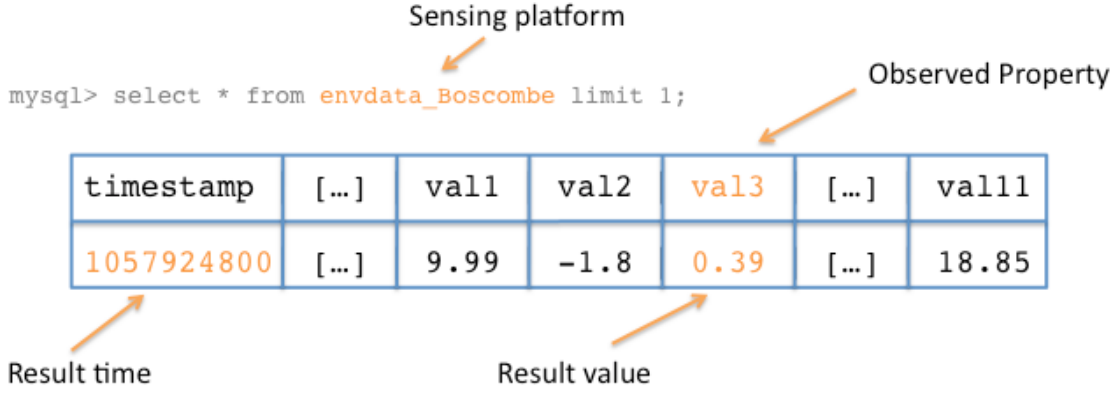

Figure 1 – Structure of the CCO database, highlighting corresponding elements of the observation model

The "*Wunderground*" weather database is another example of stored sensor network data, containing weather data scraped from the "*Weather Underground*" (`http://www.wunderground.com`) website at regular intervals. The HLAPI is also able to transform readings from these databases (i.e. a given measurement from a given time instant for a given sensor) into the observation model, before serialising the output to a configurable location.

2.2.1.2. SemSorGrid4Env architecture streaming data sources

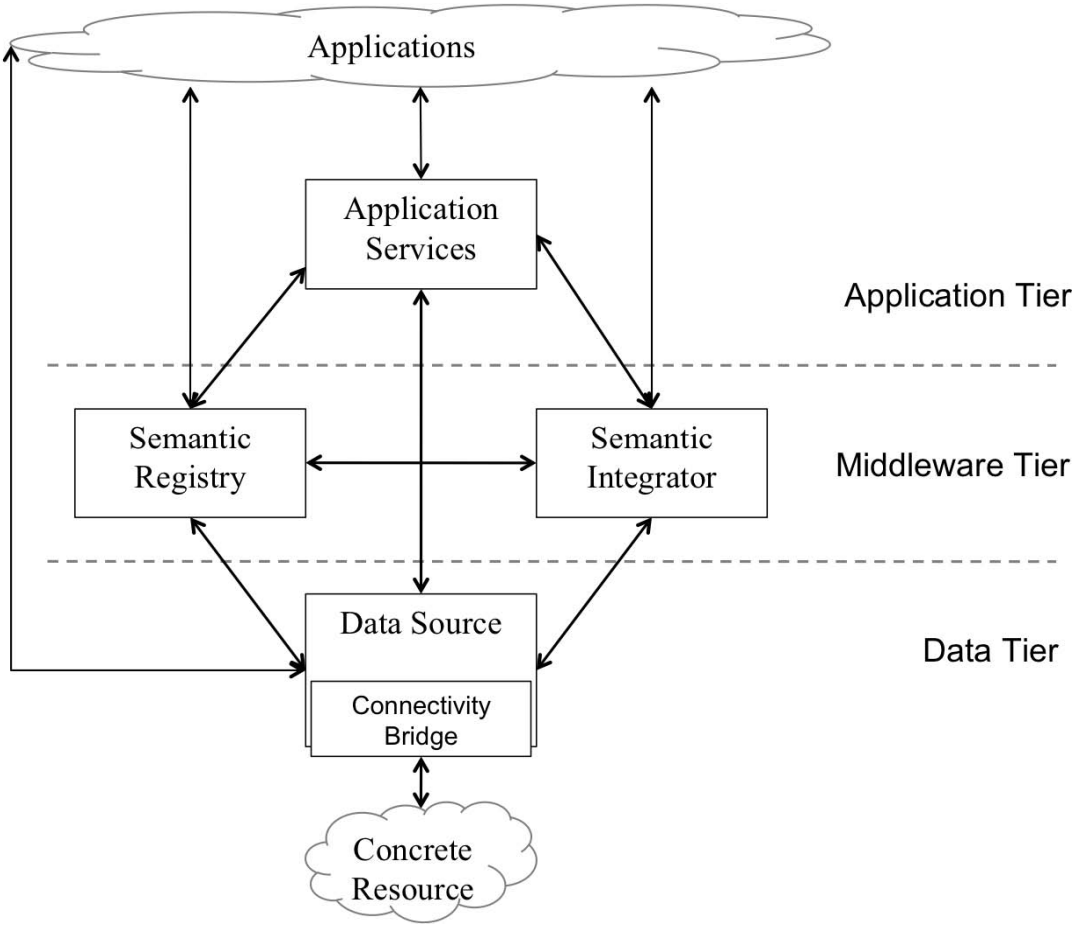

Figure 2 – The SemsorGrid4Env architecture (From [D1.3v1])

The SemsorGrid4Env architecture (represented in Figure 2) allows clients to specify one or more streaming data sources to be integrated into a single resource, and provides interfaces for clients to request the data produced by this resource. The interfaces that can be exposed by a Streaming Data Service are shown in Figure 3. By importing data from the architecture through these interfaces, the HLAPI engine will effectively be able to expose the data sources as a configurable high-level API.

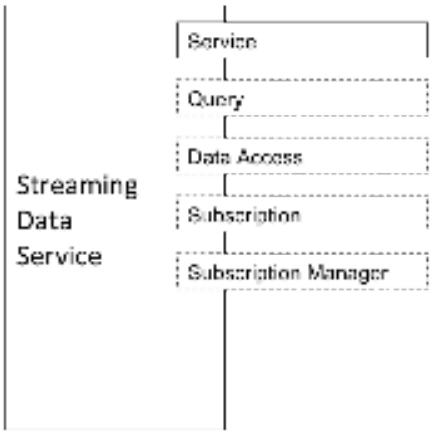

Figure 3 – Interfaces to the Streaming Data Service (from [D1.3v1])

2.2.2. Output requirements

As discussed in Section 2.1, the HLAPI engine should present its output as Linked Data, following REST principles. To meet this requirement, the HLAPI engine outputs observation content in two main ways: as RDF triples contained in a 4Store RDF database, and as serialised RDF/XML, O&M GML, WFS GML and HTML files, made accessible through a Web server.

RDF/XML is the most common way to represent semantic data on the Web, and as such is most likely to be useful in creating semantic mashups – joining multiple semantic datasets together through one or more inferred relationships. Domain developers can use this RDF/XML content to infer new relationships, joining datasets together in order to use them in ways not originally foreseen by their creators. O&M GML [OGC-OM] and WFS GML [OGC-WFS] are typically used in GIS applications. By serialising these formats, domain developers can expose observations data through the tools most familiar to domain users. The HTML Representations will typically be used when any user attempts to access an observation URI through a Web browser, providing a human-readable version of the content from an observation.

Each of the different serialised observation representations is an *information resource*, that is, they each contain information about a particular observation. If more than one of these information resources contains a representation of exactly the same information, they can be grouped under a *common information resource*. This resource implies that all information resources beneath it represent the same information.

Different information resources and common information resources that represent different information about the same observation can all be linked through a *non-information resource*: a URI representing the observation itself. In this way, if the non-information resource for a particular observation is known, any of the information resources associated with it can be negotiated, as in Figure 4.

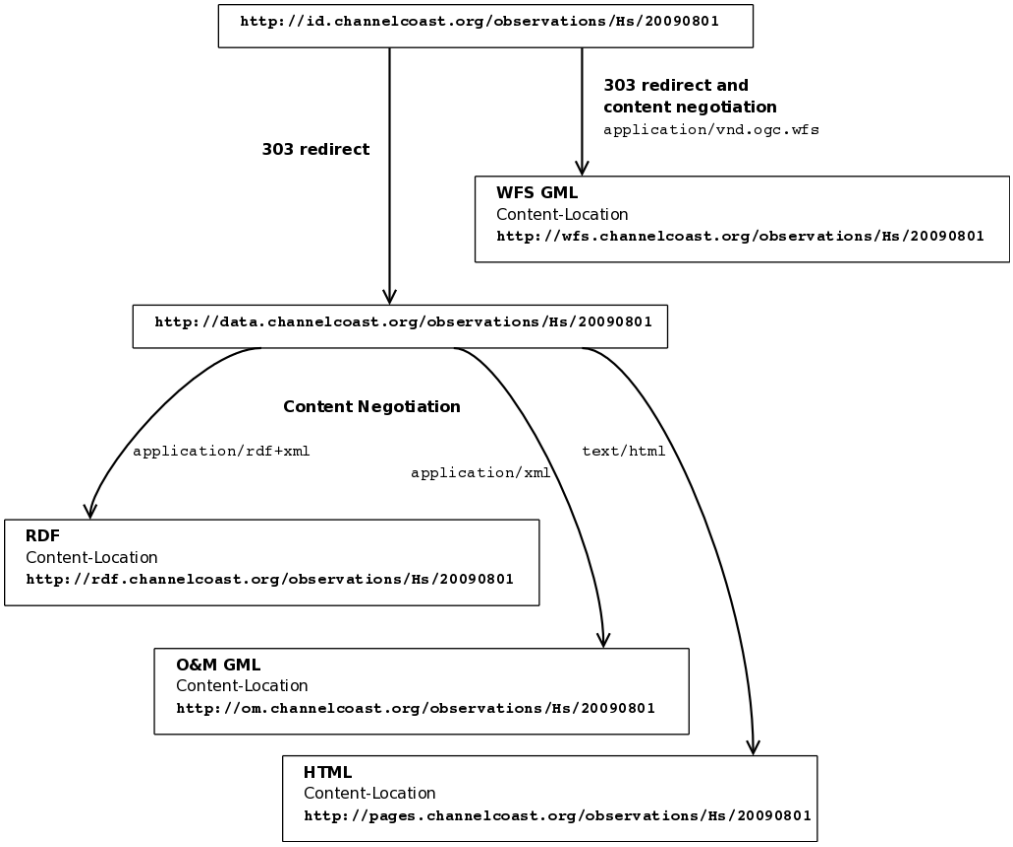

Figure 4 – Content Negotiation between different Information Resources, through a Non-Information Resource and a Common Information Resource (from [D5.2v1])

The main benefit of the 4Store implementation over, for example, the serialised RDF/XML content, is its provision of a searchable data endpoint. When incorporating RDF/XML file content into a Webapp, the user must know the location of the required file (and, indeed, that it even exists at all) before the file content can be used. When using the 4Store endpoint, the user can construct a query using the SPARQL query language in order to retrieve RDF data based on its content, rather than its location on the Web.

2.3. Previous Deployments and Prototypes

2.3.1. CCO HLAPI v.1

The CCO Linked Data REST API takes observations from the CCO database, and exposes them over the Web as RESTful resources. While superficially similar to the outputs of the current HLAPI engine, there are some important differences.

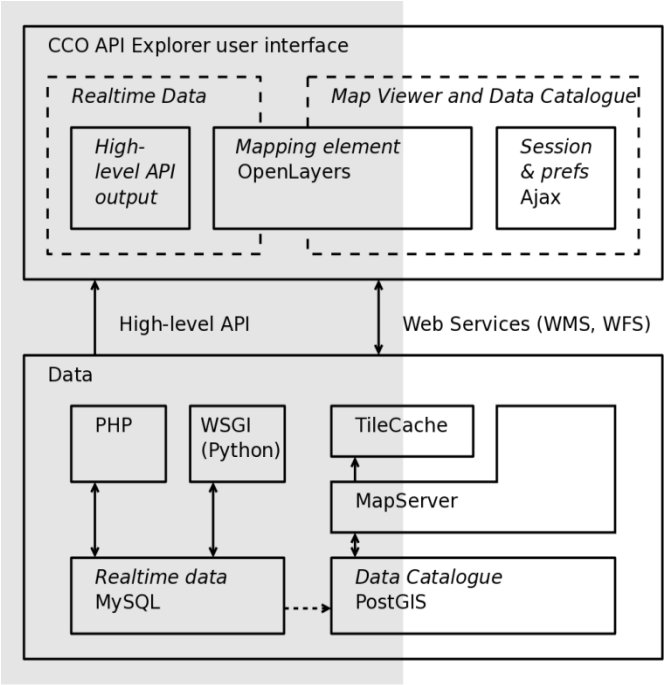

Figure 5 – The CCO HLAPI architecture (from [D5.2v1])

In the current HLAPI engine, a canonical RDF representation is created first, and used to generate the other output formats appropriately. As such, all serialised representations are semantically consistent, and any changes to the underlying RDF will continue to generate valid serialised outputs. In the CCO REST API, an existing GML representation was taken as the canonical representation from which all other serialisations were derived. Because this implementation effectively templated higher-information level resources from an initial lower-information level resource, changes to the GML representation could cause the other representations to change without considering the specifics of each particular format. This could result in different information resource representations for a particular observation no longer being semantically consistent.

In addition, the current HLAPI provides a search interface in the form of a 4Store SPARQL endpoint. The CCO API had no comparable search interface, preventing domain developers from discovering observations based on their semantic content, forcing them instead to rely on the implied semantics of the resource URIs to discover content.

2.3.2. App Tier v.1

The SemsorGrid4Env Application Tier (App Tier) libraries allowed domain developers to interact with the SemsorGrid4Env architecture and registry services, in order to produce RESTful resources for inclusion in Web applications. The libraries were exposed via two Java Web servlets: one to query the registry, the other to query the architecture.

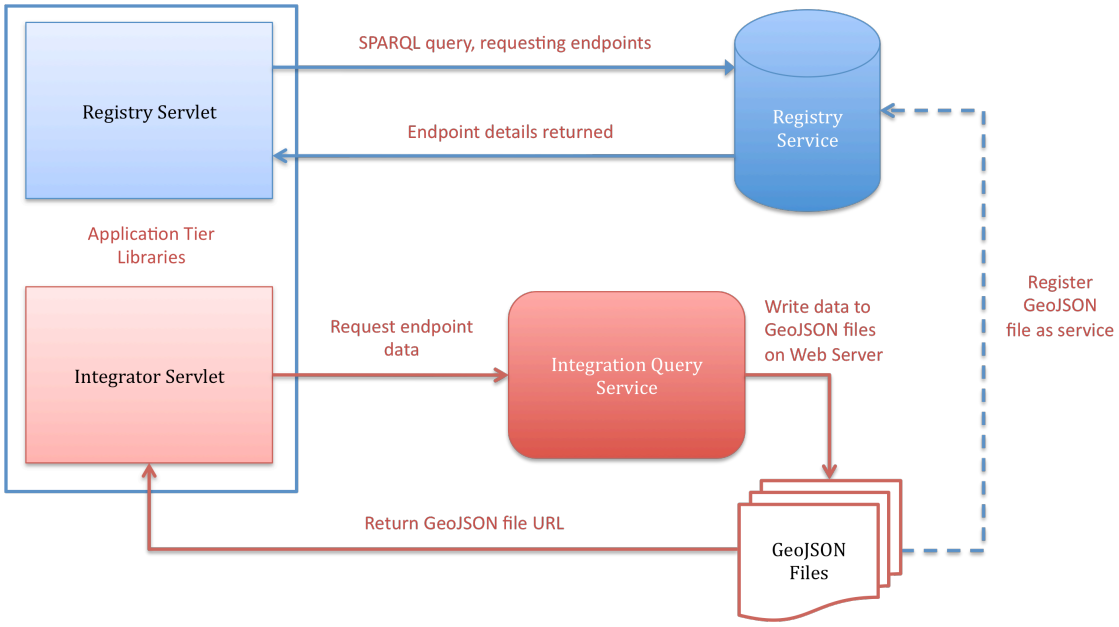

Figure 6 – The App-tier libraries interaction with the SSG4E Registry and IQS components

Domain developers first send a SPARQL query through the registry servlet to discover an architecture source offering the required data. A second query is then sent to an application to poll the discovered architecture service and write the data into a GeoJSON file on the Web server. The URL of this file is returned to the domain developer, to allow the file content to be included in a Webapp. The polling application continues to update the content of the file as long as the architecture continues to produce new data.

Unlike the CCO REST API, this implementation allows domain developers to search for data based on its content, using a SPARQL query interface. However, while the domain developer who constructed the initial request is able to make semantic inferences before initialising the serialised output, this is not true for future developers wishing to use any existing serialised output in novel ways. Because the output is only represented in the GeoJSON format, no further semantic reasoning can be performed on content within these outputs. This point is exacerbated by the lack of API configurability, as this prevents the encoding of any semantics in the representation URIs.

2.3.3. Evolution of the HLAPI service

In moving towards a final design for version 2 of the HLAPI engine, the previous implementations' strengths were combined with further measures to overcome their weaknesses. The previous CCO HLAPI implemented several of the version 2 design requirements, outputting observation measurements as Linked Data, in the form of RESTful resources. In addition, its output was driven by the CCO dataset – a useful example of the kind of generic sensor measurement database that the version 2 HLAPI engine must expose.

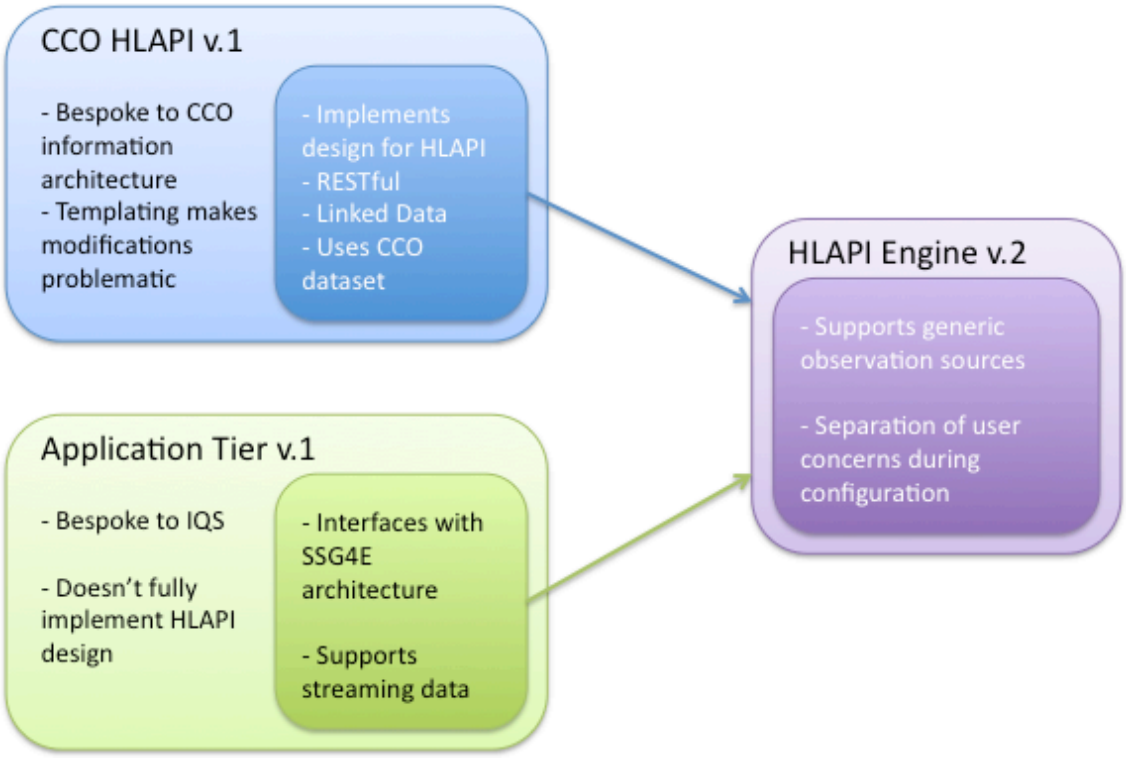

Figure 7 – Pros and cons of previous HLAPI versions, combining to inform the current version

The Application Tier libraries fulfilled the requirement to interface with the SemsorGrid4Env architecture, and to support streaming data sources through this architecture. All of these strengths were combined with support for generic observation data sources, and the separation of user concerns when configuring the engine's input data and output APIs, to arrive at the final design for the version 2 HLAPI engine (see Figure 7).

In the next section, we describe how the requirements and experiences examined in this section have informed the design of the HLAPI service.

3. HLAPI Service Design and Implementation

In this section, we present a broad overview of the HLAPI system, before providing a breakdown of the individual HLAPI engine components. This is followed by an example configuration of the HLAPI engine to expose data from the CCO sensor network, and a tutorial for domain developers to use the serialised outputs in mashup development.

3.1. System Design

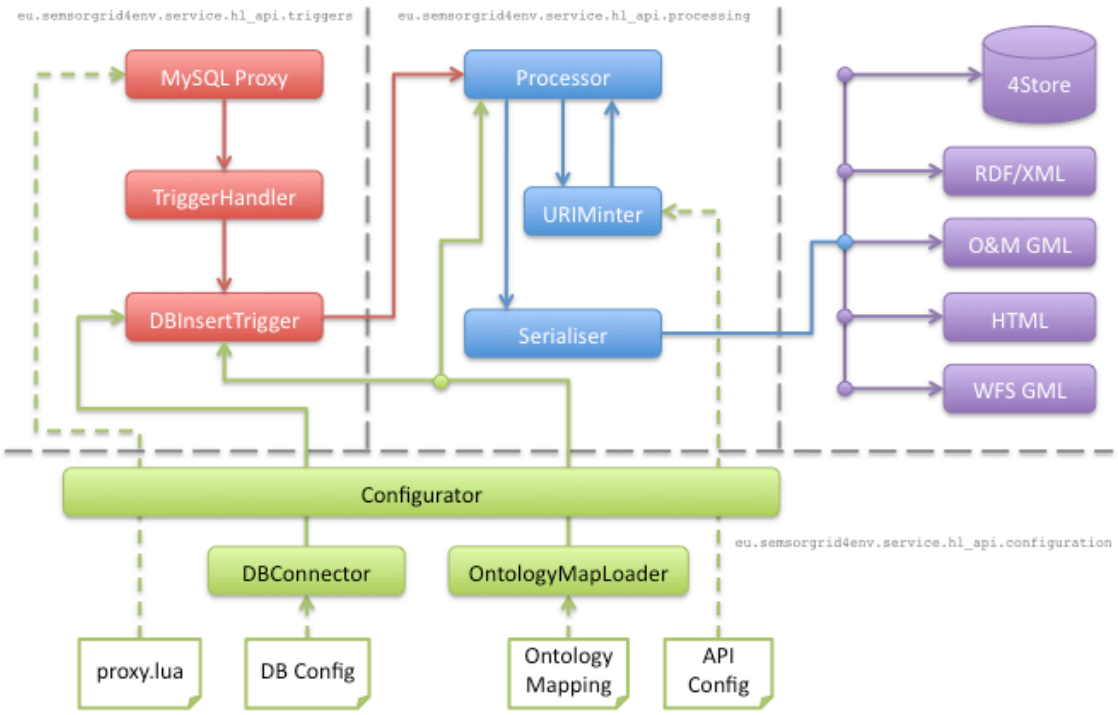

Figure 8 – System overview of the new HLAPI engine

The HLAPI system has been designed to expose the general HLAPI design for generic data sources, as described in Section 2 of this document, as well as [Pag2009]. To achieve this, and achieve tractable configurability, incoming data is transformed into a known observation model (see Figure 9). When data arrives in the system – either through a database insert, or through the SemsorGrid4Env architecture – the corresponding event trigger is activated, and determines what to do with the data. If the data represents an observation that we wish to serialise, the event trigger sends the data to the Processor to be turned into an RDF representation of an observation. If the data does not represent an observation, it is ignored. The generated RDF observation forms the canonical representation of the observation, as it is the most flexible and fully featured representation. All other serialised outputs are lower-information representations, and are therefore derived from the RDF.

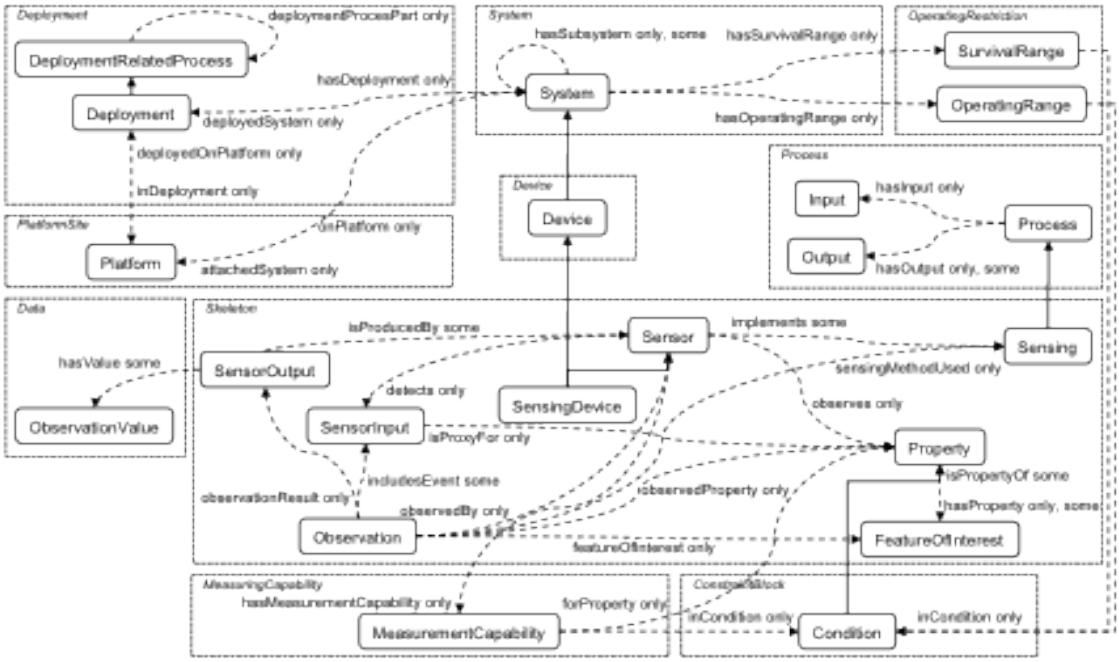

Figure 9 – The SemsorGrid4Env Observation model (from [D4.3v2])

The outputs to be serialised are determined using the API configuration file. This file defines the observation collections that the current observation should appear in, the formats in which to serialise them, and what the corresponding URIs should be. This configuration file is kept separate from the ontology mapping file, in order to separate the administrative concerns of different users; a domain expert is able to configure the mapping of the data source into the observation model, while the system administrator is able to handle to configuration of the exposed APIs.

3.1.1. Use case diagrams

3.1.1.1. Sensor measurement databases

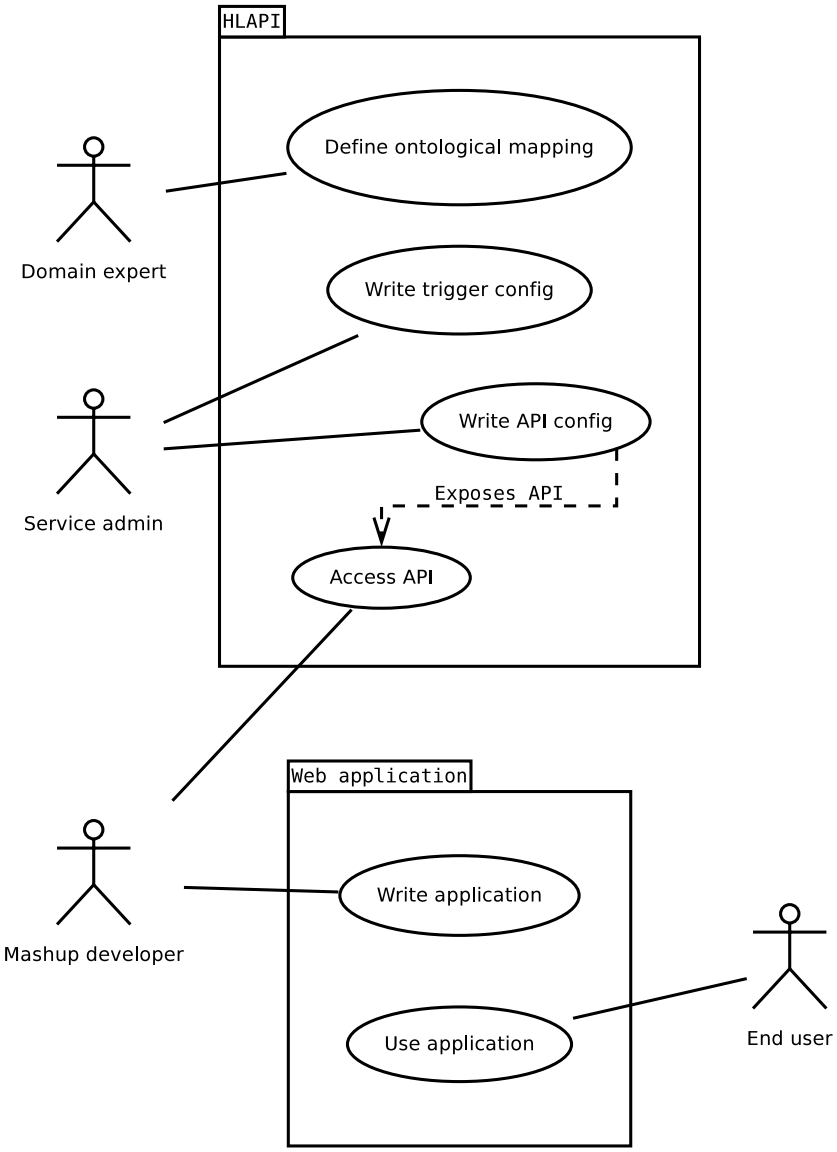

Figure 10 – Use Case diagram for HLAPI based on sensor measurement databases

When using the HLAPI engine to serialise data from a sensor measurement database, configuration concerns can be split between the domain expert and the service administrator. The domain expert is responsible for defining the mapping between the database structure and the observation model. Separately, the service administrator is responsible for defining the API configuration, to determine the format of the exposed APIs. The APIs exposed by this configuration are then used by mashup developers to populate Web applications for end users to interact with.

3.1.1.2. SemSorGrid4Env architecture streaming data sources

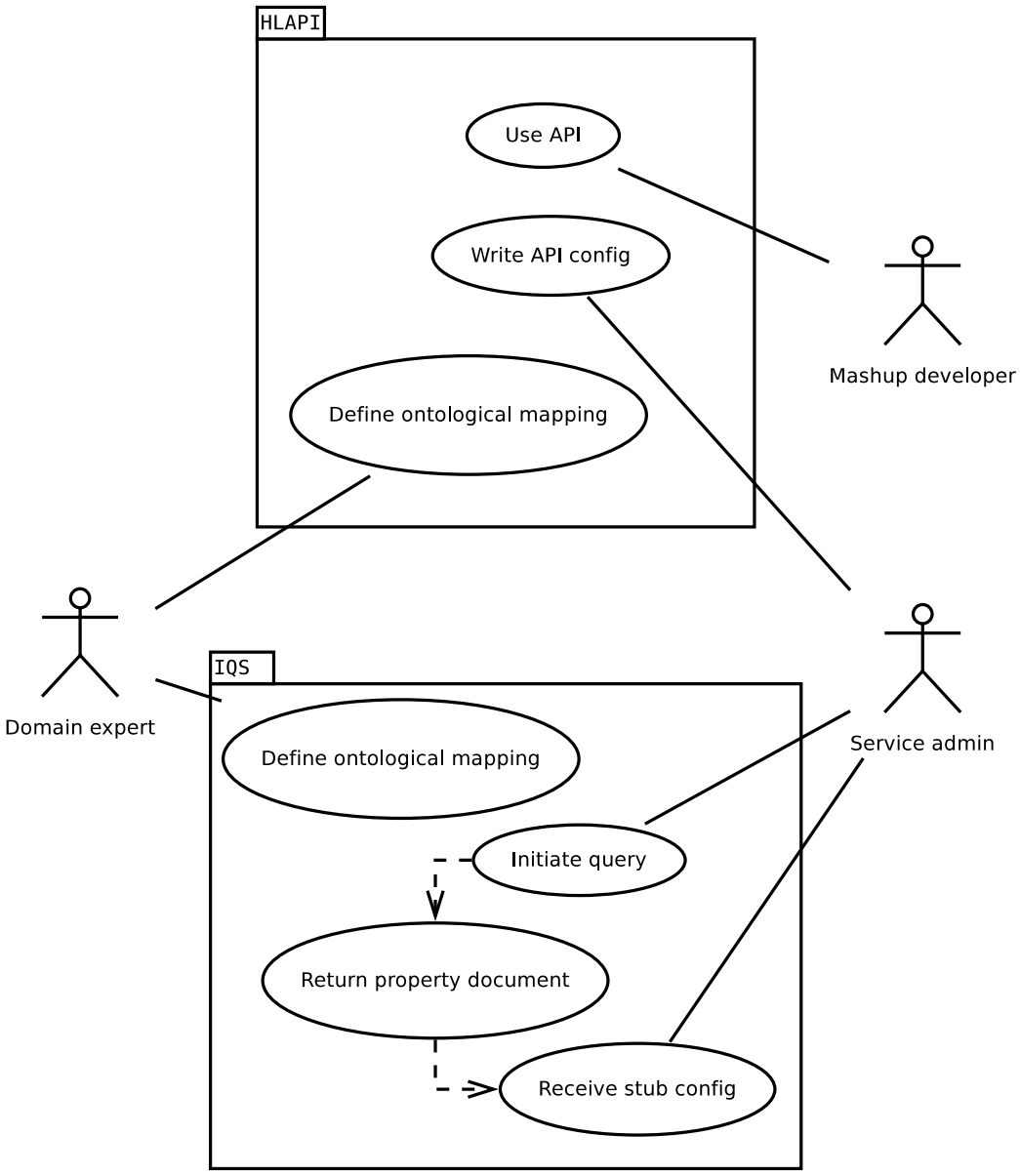

Figure 11 – Use Case diagram for HLAPI based on SSG4E architecture trigger

When serialising data from the SemsorGrid4Env architecture through the HLAPI engine, configuration concerns are once again separated between the service administrator and the domain expert. Here, the domain expert writes two different ontology mappings: one for the HLAPI engine, and one for the architecture streaming data source. In this case, the streaming data source is accessed via the Integration Query Service (IQS) component. The IQS allows a client to pose a semantic query, before returning an integrated data source from which the client can request results. The IQS is discussed further in Section 5.2. The service administrator is able to send the appropriate semantic query to the architecture, which generates a property document describing the service. This document is used to generate a stub configuration file, which the administrator can use to generate an API configuration file similar to the one

used in the sensor database implementation. Once more, the mashup developer is only concerned with the serialised outputs generated by the HLAPI engine, accessing these APIs to populate Web applications.

3.1.2. Interaction diagrams

3.1.2.1. Sensor measurement databases

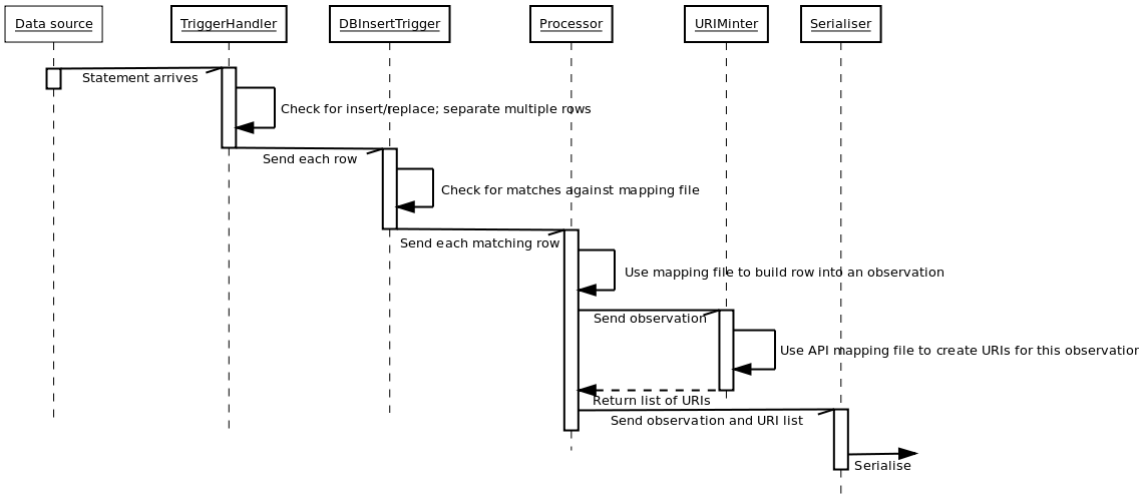

Figure 12 – Interaction diagram for HLAPI based on sensor measurement database

As an `INSERT` statement in sent to the database and caught by MySQL Proxy, it is sent to the *TriggerHandler*, which splits any multiple inserts into individual queries before sending each one to the *DBInsertTrigger*. Here, the data is checked against the ontology mapping file, to determine whether it is an observation we want to expose. If so, the query is sent to the *Processor* to be turned into an observation RDF graph. The observation is sent to the *URIMinter* where the API configuration document is used to generate the URIs used to output the intended APIs. Finally, the observation and the corresponding list of URIs are sent to the *Serialiser* (see section 3.1.2.3 below).

3.1.2.2. SemSorGrid4Env architecture streaming data sources

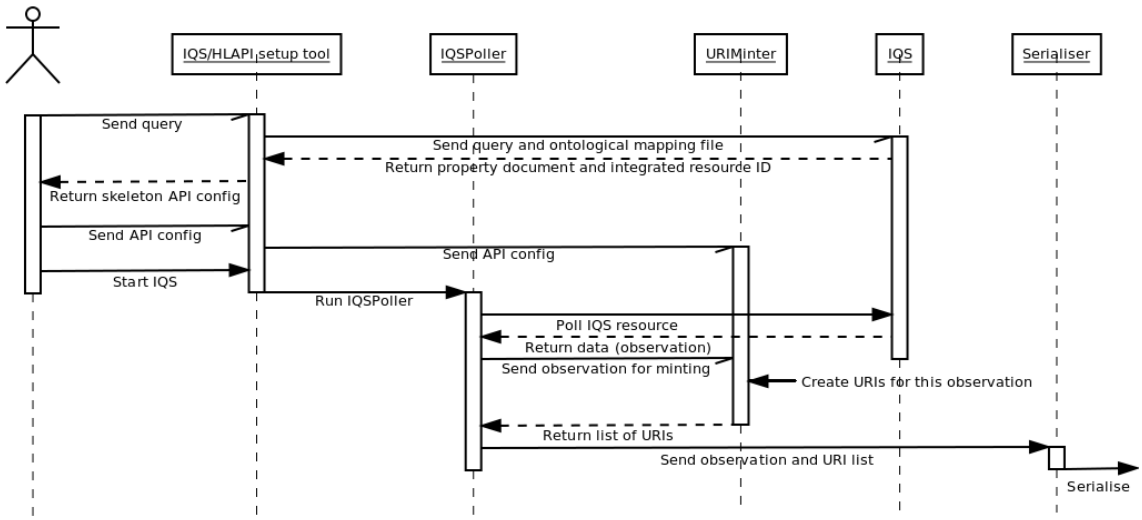

Figure 13 – Interaction diagram for HLAPI based on SSG4E architecture trigger

Interactions with the SSG4E architecture begin by sending a SPARQL query to the *IQS*, through a setup tool. The *IQS* returns a property document and an integrated resource ID to this setup tool, which in turn generates a skeleton API configuration file for the service administrator. This skeleton is completed and sent to the *URIMinter* for later use. Next, the *IQSPoller* is started, polling the *IQS* periodically for new observation data. When observation data is returned, it is sent to the *URIMinter* (as in the case of sensor measurement database data), before being sent to the *Serialiser* where the output files are serialised.

3.1.2.3. Observation serialisation

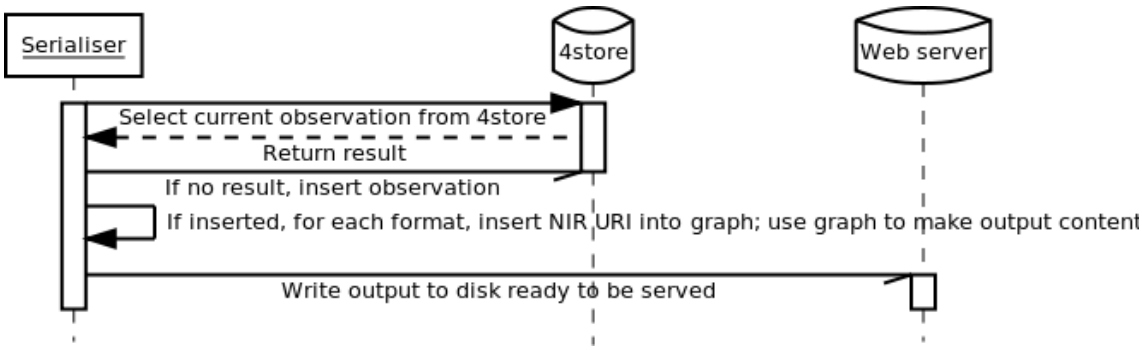

Figure 14 – Interaction diagram for showing HLAPI output serialisation

Once an observation and its corresponding set up URIs is received by the Serialiser, if first checks whether the observation exists in the 4Store database. If it does, serialisation is halted to prevent duplications. If the observation does not exist in the 4Store, it is inserted, and the output formats specified by the API configuration file are written to the appropriate place on disk.

3.2. System implementation

3.2.1. HLAPI configuration

The HLAPI engine requires two main configuration files: the *ontology mapping* file and the *API configuration* file.

3.2.1.1. Ontology mapping

The ontology mapping file is used to translate between database sources and the observation model, in accordance with the observation ontology. It does so by mapping database features (row data, table names, column headings) onto classes that form part of an observation. The domain mapping file is written in the "Turtle" RDF syntax, itself a subset of the "Notation 3" syntax. In the these files, `ClassMap` objects are used to represent classes from the target ontology:

```
map:Folkestone_met_Air_pressure a d2rq:ClassMap;
   d2rq:dataStorage map:database;
   d2rq:class obs:observation;
   d2rq:classDefinitionLabel "observation";
.
```

For each `ClassMap`, a number of `PropertyBridges` are used to express ontology properties of the classes the `ClassMap` represents. In this example, the `d2rq:property` predicate shows that the `PropertyBridge` represents the `time:hasBeginning` property of its parent ClassMap, while the `d2rq:timestamp` variable informs the HLAPI engine how to construct the value of this property:

```
map:Folkestone_met_Air_pressure_dateTimeBeginning a
d2rq:PropertyBridge;
   d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
   d2rq:property time:hasBeginning;
   d2rq:belongsToPropertyBridge
      map:Folkestone_met_Air_pressure_timeInterval;
   d2rq:timestamp "envdata_Folkestone_met.0";
   .
```

While the mapping file format is based on the D2RQ mapping file format, several important changes have been made. D2RQ offered no provision for converting a single database row into multiple instances of the same RDF class. For example, if a database row contains readings for several different observed properties, it would not be possible to convert this data into several different corresponding observations using D2RQ. This is partially overcome by allowing the HLAPI engine to process different `ClassMaps` that map to the same ontology class gracefully. To fully overcome this problem, the d2rq:`belongsToPropertyBridge` property was added, to allow assertion of relational semantics that are not necessarily expressed by the database structure:

```
d2rq:belongsToPropertyBridge
   map:Folkestone_met_Air_pressure_timeInterval;
```

In addition, the event-driven nature of the HLAPI engine required further modification of the mapping language. Features of the database structure, such as column headings and database table names are sometimes useful in determining whether the data being inserted is of interest when deciding whether to serialise an observation. Furthermore, while these values are useful for triggering events, they are not necessarily complete enough for inclusion in the final RDF observation. To overcome the first issue, `d2rq:columnHeading` and `d2rq:tableName` were created, allowing the HLAPI engine to match these features of an `INSERT` statement against the values provided in the mapping file:

```
d2rq:columnHeading "envdata_Folkestone_met.5";

d2rq:tableName "envdata_Folkestone_met";
```

The second problem is overcome via the `d2rq:substitute` variable, which allows a second value (typically a full URI) to be inserted into the observation model when a first value (such as a table name or column heading) is matched:

```
d2rq:substitute
"http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/folkestone_met";
```

The problem is further overcome with the `d2rq:timestamp` and `d2rq:alteredtimestamp` variables, which instruct the HLAPI Processor component to translate the corresponding values from Unix timestamps (typically used in database records) into XML `dateFormat` values, in accordance with the observation ontology:

```
d2rq:timestamp "envdata_Folkestone_met.0";

d2rq:alteredtimestamp "envdata_Folkestone_met.0+1800";
```

3.2.1.2. API Configuration

The API configuration file is used to determine which observations and collections should be serialised, in which file formats, and with which URIs. This is implemented using template URI strings, with various different values to be substituted into them. A configuration for a single serialised collection might look like this: e.g. everything with a place, a property and a time The following example configuration represents a collection containing every observation, with each observation's URI determined by its observed property, the procedure used to measure the property, and the time the observation was made:

```
[serialisation1]
type=canonical
baseuri="http://@@FORMAT.semsorgrid.ecs.soton.ac.uk/"
tail="observations/cco/[obs:procedure]/[obs:observedProperty]/
     [obs:observationResultTime{yMd#Hms}]"
NIR="{FORMAT=id}"
application/rdf+xml="{FORMAT=rdf}"
application/xml="{FORMAT=om}"
application/vnd.ogc.wfs="{FORMAT=wfs}"
text/html="{FORMAT=pages}"
```

The `baseuri` variable represents the URI hostname for the current collection, while the `tail` variable represents the rest of the URI string. Observation properties in square brackets, e.g. `[obs:observedProperty]`, are substituted with the corresponding property value from the observation model to be serialised. Furthermore, if that property value matches one of the print substitution values defined earlier in the configuration file, this value will be substituted instead. Print substitution values are defined as follows:

```
[printsubs]
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/boscombe=boscombe
```

Values beginning with `@@` are substituted on a per-format basis with any corresponding values from a given format's list of properties. For example, in the above collection, the value `@@FORMAT` would be replaced with `om` for the format `application/xml`. As such, different URIs can be constructed for each different serialised format, with the `NIR` format representing the canonical URI of the collection's non-information resource.

3.2.1.3. System Components

DBInsertTrigger

Once a database insert statment is received, the *DBInsertTrigger* first separates out the table name and the individual column values. For each extracted column value, the *DBInsertTrigger* checks the ontology mapping file to determine whether that value is found in any of the observations we are trying to build. If a match is found, the observation's ClassMap is added to the list of observations to be built. Once the table name and all column values have been checked, the list of applicable ClassMaps is sent to the *Processor*.

Processor

On receiving an observation ClassMap ID and the components of a database insert statement, the *Processor* first extracts all of the PropertyBridges that match the required ClassMap. For each PropertyBridge that matches the current ClassMap, the corresponding property is inserted into the appropriate place in the RDF observation graph. The *Processor* extracts the appropriate property value from the database, and in the case of a `timestamp` or `alteredtimestamp` value, manipulates it into the XML `dateTime` format, in accordance with the observation model. Once we have the constructed observation graph (constructed with a temporary "`TEMP_ROOT`" root node), we send it to the *URIMinter* to get an ArrayList of URIs. Finally, the resulting list of URIs, along with the observation graph itself, is sent to the *Serialiser*.

URIMinter

First, the *URIMinter* extracts any print substitution values from the API configuration file and WFS coordinate mappings from the coordinate mapping file. Then, for each subsequent section in the configuration file, the *URIMinter* extracts the appropriate values from the observation – swapping in any available print substitutions – and inserts them into the template URI string. If the observation property is in the XML `dateTimeFormat` format, the result is transposed into `YYYYMMDD#hhmmss` notation, as defined by the corresponding pattern in the configuration file. Any format-specific substitutions are then passed into the URI tail or hostname. At this stage, any available WFS coordinates from the current sensor are added to the URI tail. Finally, all of the generated URIs are added to the ArrayList/HashMap container, and returned to the *Processor*.

Serialiser

First, the current observation's non-information resource URI is substituted into the observation graph's temporary root node. Next, a check is performed to determine whether this observation has already been added to the 4Store: if it has already been added, serialisation is stopped to prevent duplicate entries; otherwise, the observation is assumed to represent new data, and the graph is added to the 4Store. Then, for each format specified in the API mapping file, we serialise the observation to disc, in the location specified at startup, combined with the URI tail in the current format's HashMap entry

3.3. Configuring the HLAPI Service

In this section, we provide a worked example of how to configure the HLAPI, using the CCO database as a data source. This process involves three separate phases: configuration of the MySQL Proxy layer, mapping between the data source and the observation model, and configuration of the generated APIs.

3.3.1. MySQL Proxy

In order to intercept the `INSERT` statements being sent to the CCO database, an instance of MySQL Proxy is needed. As the name implies, this application runs as a proxy between a database and anyone wishing to interact with the database. As statements are

sent to the database, MySQL Proxy – configured using a script written in the "Lua" scripting language – can alter, reject, or divert the statements to other applications. In this case, we require MySQL Proxy to divert any statements to the HLAPI engine, where they will be processed as explained above.

The example Lua script for the CCO database is generic enough that it will work for any database into which data rows are inserted. It simply removes any backtick symbols from the query (to prevent their contents from being executed as shell commands), and sends the resulting string to a shell script, "hlapi.sh":

```
function read_query(packet)
if packet:byte() ~= proxy.COM_QUERY then return end
local query = packet:sub(2)
      query=string.gsub(query, "`", "")
      os.execute('/usr/local/src/hlapi.sh "' .. query .. '" &')
end
```

The shell script itself is used to build the Java classpath for the HLAPI engine, before passing this classpath, the MySQL statement, and the locations of any configuration files to the HLAPI Java process:

```
#!/bin/sh
cd /usr/local/src/HLAPI

THE_CLASSPATH=

for i in `ls ../HLAPI_libs/*.jar`
do
  THE_CLASSPATH=${THE_CLASSPATH}:${i}
done

java -cp ".:${THE_CLASSPATH}"
   eu/semsorgrid4env/service/hl_api/triggers/TriggerHandler "$1"
      "doc/onto_map.n3" "doc/api_map.ini" "doc/database.n3"
```

The HLAPI engine itself also requires a short database configuration file, to enable connections to the database through the JDBC connection libraries:

```
@prefix map:<file:/Users/user/Desktop/cco.n3#>.
@prefix d2rq:<http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#>.
@prefix jdbc:<http://d2rq.org/terms/jdbc/>.

map:database a d2rq:Database;
      d2rq:jdbcDriver "com.mysql.jdbc.Driver";
      d2rq:jdbcDSN "jdbc:mysql://127.0.0.1/CCO";
      d2rq:username "cco";
      d2rq:password "**********";
      jdbc:autoReconnect "true";
      jdbc:zeroDateTimeBehavior "convertToNull";
      .
```

3.3.2. Ontology Mapping

When exposing a database through the HLAPI engine, a mapping file is required in order to map between the items in the database and the classes and properties in the observation ontology. As explained above, this file is written in an extended version of the D2RQ mapping language, with one "ClassMap" required for each mapping between a single observed property in the database and an observation RDF model.

The following example shows the ClassMap and a selection of corresponding PropertyBridges for the property "*Hs*" (representing wave height) from the `envdata_Boscombe` database table:

```
map:Boscombe_Hs a d2rq:ClassMap;
   d2rq:dataStorage map:database;
   d2rq:class obs:observation;
   d2rq:classDefinitionLabel "observation";
   .

[...]

map:Boscombe_Hs_dateTimeBeginning a d2rq:PropertyBridge;
   d2rq:belongsToClassMap map:Boscombe_Hs;
   d2rq:property time:hasBeginning;
   d2rq:belongsToPropertyBridge map:Boscombe_Hs_timeInterval;
   d2rq:timestamp "envdata_Boscombe.0";
   .

[...]

map:Boscombe_Hs_hasValue a d2rq:PropertyBridge;
   d2rq:belongsToClassMap map:Boscombe_Hs;
   d2rq:property obs:hasValue;
   d2rq:column "envdata_Boscombe.12";
   d2rq:belongsToPropertyBridge
      map:Boscombe_Hs_observationResult;
   .
```

For further, complete examples of CCO ontology mapping ClassMaps, see Appendix A.

Note that the values for any `d2rq:timestamp`, `d2rq:alteredtimestamp`, `d2rq:column` or `d2rq:columnHeading` property each refer to the corresponding database table (in this case, "*envdata_Boscombe*"), followed by the required database column.

3.3.3. API Configuration

The remaining configuration file is the API configuration file, used to determine which collections should be serialised for a given observation, which file formats they should be serialised in, and what URIs should be used for these serialised collections.

The API configuration file comprises two main sections: print substitution strings, and URI minting and serialisation details.

Print substitutions are used to replace full URIs from the observation RDF with more sensible strings for inclusion in a URI. These will typically be used to replace the URIs of observed properties and sensing devices. The full list of print substitutions required for the CCO database can be found in Appendix A, with the following excerpt providing an example of substitutions for both sensors and observed properties:

```
[printsubs]
# sensors
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/hornsea=hornsea
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/looebay=looebay
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/milford=milford
# observed properties
http://marinemetadata.org/2005/08/manly#Hmax=Hmax
http://marinemetadata.org/2005/08/ndbc_waves#Swell_Period=Tz
http://marinemetadata.org/2005/08/ndbc_waves#Wind_Wave_Height=Hs
```

The following sections are used to define collections, URI formats and serialised file formats. The value of the `type` variable can be either `canonical` (to represent the canonical representation of an observation) or `collection` (to represent a collection of `canonical` results). The `baseuri` variable represents the hostname for this observation's URIs, and the `tail` variable represents the rest of the URI string – both before any substitution with observation values or file format abbreviations. The remaining lines define the output formats to be serialised, along with any values to be substituted into the URI string that are specific to each given output format. The CCO HLAPI deployment includes two example serialisations. One of these serialisations, shown below, represents the canonical version of every timestamped observation, for every observed property, from every location:

```
[serialisation1]
type=canonical
baseuri="http://@@FORMAT.semsorgrid.ecs.soton.ac.uk/"
tail="observations/cco/[obs:procedure]/[obs:observedProperty]/
      [obs:observationResultTime{yMd#Hms}]"

NIR="{FORMAT=id}"
application/rdf+xml="{FORMAT=rdf}"
application/xml="{FORMAT=om}"
application/vnd.ogc.wfs="{FORMAT=wfs}"
text/html="{FORMAT=pages}"
```

This collection and a second containing every date-stamped observation for every different observed property can be found in the example API config file in Appendix A.

Both example serialisations define outputs in all four formats supported by the HLAPI engine: RDF/XML, O&M GML, WFS GML and HTML. The `NIR` format is used to represent the non-information resource, from which the serialised information resources are negotiated.

3.3.4. WFS coordinate mapping

An additional mapping file can be used to provide geographical coordinates for a given sensor URI. This is useful when generating WFS representations (which require

coordinates) of observations where the corresponding sensor does not supply its own coordinates. The format of this file is similar to the print substitution section of the API configuration file. For sensors associated with the CCO data source, an example set of coordinates could be defined as follows:

```
[coords]
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/boscombe=
      "-1.83888,50.71111"
```

3.4. Using the HLAPI

Appendix B contains a tutorial describing how a mashup developer would use the serialised HLAPI observations to populate a Web application.

4. Source Code

The source code for the HLAPI engine is available from the SemsorGrid4Env Subversion repository, at:

```
https://ssg4env.techideas.net/repos/HLAPI/trunk/hlapi
```

4.1. Code Structure

The HLAPI engine uses Maven 2.1 to compile its class files and manage any external dependencies. The system components are collected under the package `eu.semsorgrid4env.service.hl_api`, with the following sub-packages:

- `configuration`, which contains classes concerned with the loading and management of mapping files;
- `triggers`, which contains classes that determine whether incoming data is relevant to the required API output, and that organise this data into a format that the Processor can use;
- `processing`, which contains classes that build observations, mint their URIs, and generate serialised outputs.

4.2. External application dependencies

The HLAPI engine has the following external dependencies, all of which are resolved through the Maven build process:

To fulfill an overall dependency on D2R, in order to read ontology mapping files, interact with databases, and generate RDF models using the Jena RDF library:

- `antlr-2.7.5.jar`
- `arq.jar`
- `commons-lang-2.3.jar`
- `commons-logging-1.1.jar`
- `concurrent.jar`
- `d2r-server-0.7.jar`
- `d2rq-0.7.jar`
- `icu4j-3.4.jar`
- `iri.jar`
- `jakarta-oro-2.0.8.jar`
- `jena.jar`
- `jetty-6.1.10.jar`
- `joseki.jar`
- `json.jar`
- `junit-4.5.jar`
- `log4j-1.2.12.jar`
- `mysql-connector-java-5.1.7-bin.jar`
- `postrgresql-8.2dev-503.jdbc3.jar`
- `servlet-api-2.5-6.1.10.jar`
- `slf4j-api-1.5.6.jar`
- `slf4j-log4j12-1.5.6.jar`

- `velocity-1.5.jar`
- `xercesImpl.jar`
- `xml-apis.jar`

To interact with the 4Store RDF database:

- `cp-common-fourstore-0.3.1.jar`
- `cp-common-utils-1.0.1.jar`
- `cp-common-openrdf-0.2.1.jar`
- `openrdf-sesame-2.3.0-onejar.jar`

To process `.ini` configuration files:

- `commons-configuration-1.6.jar`

4.3. Configuration files

The HLAPI engine is supplied with examples of the following required configuration files:

- `/doc/onto_map.n3` – used to define the mapping between the incoming data source and the observation model
- `/doc/database.n3` – contains access information for connecting to the source database
- `/doc/api_map.ini` – defines the API collections to be serialised, and their associated URI patterns
- `/doc/coords.ini` – contains any missing sensor coordinates, used to serialise WFS representations
- `/doc/webroot.ini` – defines the path on disk to begin writing serialised outputs
- `/conf/resources.ini` – describes the relationship between the various Information, Non-Information and Common-Information Resources in the API

4.4. Additional Scripts

The following additional scripts are supplied to assist the configuration and running of the HLAPI engine:

- `/scripts/generate-apache-conf.sh` – used to generate stub Apache domain redirect configurations, in order to negotiate content for the various serialised output formats
- `/scripts/batch-import.sh` – imports data from an existing database as a batch import, to bring the serialised outputs into line with the sensor database content
- `/scripts/hlapi.sh` – utility script to run the HLAPI engine with its associated mapping and configuration files
- `/scripts/pipe.lua` – loaded into MySQL Proxy, in order to intercept incoming database `INSERT`/`REPLACE` statements

5. Source Data

The HLAPI engine is designed to be extensible enough to handle any data source that could be represented through the observation model. Its configurable, event-driven nature means that provided the correct mappings are in place to convert the source data into observation RDF, the HLAPI will be able to expose the data as serialised output.

During development of the HLAPI engine, two main data sources have been used, namely the CCO sensor network database, and the SemsorGrid4Env Integration Query Service.

5.1. CCO Data

The Channel Coastal Observatory (CCO) is the data management centre for the Regional Coastal Monitoring Programmes of England. Over a period of more than 5 years, the GeoData Institute has designed, built from the top-down, and operated the data management infrastructure to run this programme. This includes software to manage and transmit real-time data from the largest network of coastal sensors in the UK; a data management infrastructure to manage data and metadata for over 65,000 environmental surveys of different types amounting to terabytes of storage; and a website to deliver real time and surveyed data to a public audience through highly complex dynamic map and data visualisation interfaces, serving over a million hits per month.

Initial development of the API in [D5.2v1] has focused on publishing data from the CCO network of marine and coastal sensors monitoring:

- wave height
- sea surface temperature
- wave period
- wave spread
- wave direction
- tide height.

5.2. SemsorGrid4Env Integration Query Service

The SemsorGrid4Env Integration Query Service (IQS) allows a number of streaming or stored data services to be exposed as a single integrated resource. A semantic query is sent to the IQS, along with a list of data sources to integrate. Subsequent requests to the resulting *integrated resource* will return the collated set of results from all data sources that comprise that integrated resource. When creating an integrated resource, a mapping file is provided to define the output format of any requests made to the resource. In this way, the IQS can return data in the form of an RDF graph complying with the observation ontology. This graph can then be passed to the HLAPI *Processor* component, ready for its URIs to be minted, and its serialised output formats to be written.

6. Installation and Execution

In this chapter, we provide instructions on how to compile, configure and deploy the HLAPI engine, as well as details of any additional software components required by the system.

6.1. Setup

Before compiling and deploying the HLAPI engine, several software dependencies need to be met. In addition, a degree of configuration needs to be completed before the HLAPI engine will operate.

6.1.1. Requirements

The HLAPI engine has the following external software dependencies.

- Web Server: Apache 2.2 – `http://projects.apache.org/projects/http_server.html`
- SQL Database: MySQL 5 – `http://mysql.com/downloads/mysql/#downloads`
- MySQL Proxy – `http://forge.mysql.com/wiki/MySQL_Proxy`
- Java SDK: JDK 1.6 – `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
- RDF Database: 4Store – `http://4store.org/`

Installation instructions for each component can be found on the respective organisation's website

6.1.2. Configuration

Having checked the HLAPI code out of SVN (as described in Section 4), the following configuration should be completed before running the HLAPI engine:

- Edit the example ontology mapping file (`onto_map.n3`), corresponding database access file (`database.n3`), API mapping file (`api_map.ini`) and WFS coordinate mapping file (`coords.ini`), found in the `/doc` subdirectory of the main project directory. Details of these files are provided in Section 3.3, with full examples provided in Appendix A.
- The `hlapi.sh` file, found in the `/scripts` subdirectory of the main project directory, should point to the `TriggerHandler` class in the `/target/eu/semsorgrid4env/service/hl_api/triggers/` subdirectory, as well as the mapping files found in `/doc`. If you need to move the contents of

the `/target` or `/doc` subdirectories for any reason, ensure the paths in `hlapi.sh` point to the corresponding new locations.

- The `pipe.lua` file, found in the `/scripts` subdirectory of the main project directory, should point to hlapi.sh in the same directory. Again, if you need to move the hlapi.sh script for any reason, ensure the path in pipe.lua points to its new location.

- Create an "observations" graph in 4Store, using the following command:
  `4s-backend-setup observations`

- Edit the example `resources.conf` file in the `/conf` subdirectory of the main project directory, to describe the Information Resources, Common Information Resources, and Non-Information Resources for the intended API. Next, run the `generate-apache-conf.sh` script from the `/scripts` subdirectory, to generate the stub code needed to implement API content negotiation. Check this stub code is correct, and add it to the appropriate section of your Apache configuration file.

- Edit the example `webroot.ini` file in the `/doc` subdirectory, to point to the Web server document root into which the API output files should be written.

6.2. Compiling and Running

First, compile the HLAPI engine Class files by navigating to the base project directory and running the Maven `compile` command:

```
host: repos/HLAPI/trunk> mvn compiler:compile
```

The compiled Class files can be found in the `target` subdirectory of the main project directory.

Next, Run the 4Store http SPARQL interface with the following command:

```
4s-backend observations
4s-httpd -p 8000 observations
```

Start MySQL Proxy as a background process, using the previously configured `pipe.lua` as an input parameter. Be sure to run MySQL Proxy as a user who has write-access to the directory into which the HLAPI engine will output its serialised data (in this case, the proxy is run using `sudo`):

```
sudo mysql-proxy –proxy-lua-script=/path/to/pipe.lua &
```

Make sure the data source responsible for inserting readings into the database is now pointing at the proxy (port 4040 by default), rather than at the MySQL database itself (typically port 3306). As the data source inserts results into the database, the HLAPI engine will insert the resulting observations into 4Store, and write the serialised output formats to disk. The 4Store status page can be checked to confirm the triples are being stored correctly (default URL: `http://hostname:8000/status`).

7. Test Strategy

The HLAPI testing strategy follows the recommended SemsorGrid4Env testing strategy, incorporating both individual unit tests and overall system integration tests.

7.1. Unit Testing

The HLAPI engine Java classes are accompanied by corresponding unit tests, written using the *JUnit* testing framework. The unit tests are found in the `src/test` subdirectory of the main project directory, further separated according to their corresponding Java package. The tests can be run as part of the main Maven build process (using the `mvn compile` command), or individually (using `mvn test`).

7.2. Integration Testing

Integration testing of the HLAPI engine components involves successfully detecting any appropriate data, converting it to the observation model, minting an appropriate set of URIs for any configured collections, and serialising these collections to disk.

To implement this workflow with sufficient data to perform reliable tests, the `batch-import.sh` script – found in the `scripts` directory (see Section 4.4) – is used. While this script is primarily used to bring HLAPI output up-to-date with the corresponding data stored in the database, it can also be used as a mechanism to test the full HLAPI engine workflow with a high volume of data. Combined with the appropriate HLAPI engine configuration files, this script can be used to verify that the correct serialised outputs are being generated, while any inappropriate data is ignored.

Appendix A: Example Mapping Files

Ontology mapping

Folkestone air pressure

```
map:Folkestone_met_Air_pressure a d2rq:ClassMap;
        d2rq:dataStorage map:database;
        d2rq:class obs:observation;
        d2rq:classDefinitionLabel "observation";
        .
map:Folkestone_met_Air_pressure_observationResultTime a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
        d2rq:property obs:observationResultTime;
        .
map:Folkestone_met_Air_pressure_timeInterval a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
        d2rq:property time:Interval;
        d2rq:belongsToPropertyBridge
map:Folkestone_met_Air_pressure_observationResultTime;
        .
map:Folkestone_met_Air_pressure_dateTimeBeginning a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
        d2rq:property time:hasBeginning;
        d2rq:belongsToPropertyBridge map:Folkestone_met_Air_pressure_timeInterval;
        d2rq:timestamp "envdata_Folkestone_met.0";
        .
map:Folkestone_met_Air_pressure_dateTimeEnd a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
        d2rq:property time:hasEnd;
        d2rq:belongsToPropertyBridge map:Folkestone_met_Air_pressure_timeInterval;
        d2rq:alteredtimestamp "envdata_Folkestone_met.0+1800";
        .
map:Folkestone_met_Air_pressure_procedure a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
        d2rq:property obs:procedure;
        d2rq:tableName "envdata_Folkestone_met";
        d2rq:substitute
"http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/folkestone_met";
        .
map:Folkestone_met_Air_pressure_observedProperty a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
        d2rq:property obs:observedProperty;
        d2rq:columnHeading "envdata_Folkestone_met.5";
        d2rq:substitute "http://marinemetadata.org/2005/08/cf#air_pressure_at_sea_level";
        .
map:Folkestone_met_Air_pressure_featureOfInterest a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
        d2rq:property obs:featureOfInterest;
        d2rq:constantValue "http://www.eionet.europa.eu/gemet/concept?cp=7495";
        .
map:Folkestone_met_Air_pressure_observationResult a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
        d2rq:property obs:observationResult;
        .
map:Folkestone_met_Air_pressure_ValueThing a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
        d2rq:property obs:ValueThing;
        d2rq:column "envdata_Folkestone_met.5";
        d2rq:belongsToPropertyBridge map:Folkestone_met_Air_pressure_observationResult;
        .
```

Rhyl Flats mean wave height

```
map:RhylFlats_Hs a d2rq:ClassMap;
        d2rq:dataStorage map:database;
        d2rq:class obs:observation;
        d2rq:classDefinitionLabel "observation";
        .
map:RhylFlats_Hs_observationResultTime a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:RhylFlats_Hs;
```

```
        d2rq:property obs:observationResultTime;
        .
map:RhylFlats_Hs_timeInterval a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:RhylFlats_Hs;
        d2rq:property time:Interval;
        d2rq:belongsToPropertyBridge map:RhylFlats_Hs_observationResultTime;
        .
map:RhylFlats_Hs_dateTimeBeginning a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:RhylFlats_Hs;
        d2rq:property time:hasBeginning;
        d2rq:belongsToPropertyBridge map:RhylFlats_Hs_timeInterval;
        d2rq:timestamp "envdata_RhylFlats.0";
        .
map:RhylFlats_Hs_dateTimeEnd a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:RhylFlats_Hs;
        d2rq:property time:hasEnd;
        d2rq:belongsToPropertyBridge map:RhylFlats_Hs_timeInterval;
        d2rq:alteredtimestamp "envdata_RhylFlats.0+1800";
        .
map:RhylFlats_Hs_procedure a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:RhylFlats_Hs;
        d2rq:property obs:procedure;
        d2rq:tableName "envdata_RhylFlats";
        d2rq:substitute "http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/rhylflats";
        .
map:RhylFlats_Hs_observedProperty a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:RhylFlats_Hs;
        d2rq:property obs:observedProperty;
        d2rq:columnHeading "envdata_RhylFlats.12";
        d2rq:substitute "http://marinemetadata.org/2005/08/ndbc_waves#Wind_Wave_Height";
        .
map:RhylFlats_Hs_featureOfInterest a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:RhylFlats_Hs;
        d2rq:property obs:featureOfInterest;
        d2rq:constantValue "http://www.eionet.europa.eu/gemet/concept?cp=7495";
        .
map:RhylFlats_Hs_observationResult a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:RhylFlats_Hs;
        d2rq:property obs:observationResult;
        .
map:RhylFlats_Hs_uom a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:RhylFlats_Hs;
        d2rq:property obs:uom;
        d2rq:constantValue "urn:ogc:def:uom:OGC:m";
        d2rq:belongsToPropertyBridge map:RhylFlats_Hs_observationResult;
        .
map:RhylFlats_Hs_ValueThing a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:RhylFlats_Hs;
        d2rq:property obs:ValueThing;
        d2rq:column "envdata_RhylFlats.12";
        d2rq:belongsToPropertyBridge map:RhylFlats_Hs_observationResult;
        .
```

API configuration

```
# substitutions for more concise URI content
[printsubs]

# procedures
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/folkestone_met=folkestone_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/arunplatform_met=arunplatform_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/deal_met=deal_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/hernebay_met=hernebay_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/looebay_met=looebay_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/sandownpier_met=sandownpier_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/swanagepier_met=swanagepier_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/teignmouthpier_met=teignmouthpier_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/weymouth_met=weymouth_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/worthing_met=worthing_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/lymington_met=lymington_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/westbaypier_met=westbaypier_met
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/perranporth=perranporth
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/weymouth=weymouth
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/sandownbay=sandownbay
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/bidefordbay=bidefordbay
```

```
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/boscombe=boscombe
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/bracklesham=bracklesham
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/chesil=chesil
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/folkestone=folkestone
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/goodwin=goodwin
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/haylingisland=haylingisland
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/hornsea=hornsea
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/looebay=looebay
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/milford=milford
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/minehead=minehead
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/penzance=penzance
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/pevenseybay=pevenseybay
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/rustington=rustington
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/rye=rye
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/seaford=seaford
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/startbay=startbay
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/torbay=torbay
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/westonbay=westonbay
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/westbay=westbay
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/rhylflats=rhylflats
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/hernebay_tide=hernebay_tide
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/sandownpier_tide=sandownpier_tide
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/teignmouthpier_tide=teignmouthpier_tide
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/westbaypier_tide=westbaypier_tide
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/swanagepier_tide=swanagepier_tide
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/deal_tide=deal_tide
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/lymington_tide=lymington_tide

# observedProperties
http://marinemetadata.org/2005/08/manly#Hmax=Hmax
http://marinemetadata.org/2005/08/ndbc_waves#Dominant_Wave_Period=Tp
http://marinemetadata.org/2005/08/ndbc_waves#Mean_Wave_Direction=Dirp
http://marinemetadata.org/2005/08/noaa-
waves#Directional_Spread_of_the_Dominant_Wave=Sprp
http://marinemetadata.org/2005/08/ndbc_waves#Swell_Period=Tz
http://marinemetadata.org/2005/08/ndbc_waves#Wind_Wave_Height=Hs
http://marinemetadata.org/2005/08/cdip#SST=Tsea
http://marinemetadata.org/2005/08/cf#air_pressure_at_sea_level="Air_pressure"
http://marinemetadata.org/2005/08/cf#wind_speed="Wind_speed"
http://marinemetadata.org/2005/08/cf#wind_to_direction="WDir"
http://marinemetadata.org/2005/08/cf#rainfall_amount="rainfall"
http://marinemetadata.org/2005/08/cf#tropopause_air_temperature="TAir"

# each of the following sections defines a single canonical resource or collection
[serialisation1]
type=canonical
baseuri="http://@@FORMAT.semsorgrid.ecs.soton.ac.uk/"
tail="observations/cco/[obs:procedure]/[obs:observedProperty]/[obs:observationResultTime
{yMd#Hms}]"
# each subsequent line in the section represents a serialisation format, along with its
substitution properties
NIR="{FORMAT=id}"
application/rdf+xml="{FORMAT=rdf}"
application/xml="{FORMAT=om}"
application/vnd.ogc.wfs="{FORMAT=wfs,coords=true}"
text/html="{FORMAT=pages}"

[serialisation2]
type=collection
baseuri="http://@@FORMAT.semsorgrid.ecs.soton.ac.uk/"
tail="observations/cco/[obs:observedProperty]/[obs:observationResultTime{yMd}]"
# each subsequent line in the section represents a serialisation format, along with its
substitution properties
NIR="{FORMAT=id}"
application/rdf+xml="{FORMAT=rdf}"
application/xml="{FORMAT=om}"
application/vnd.ogc.wfs="{FORMAT=wfs,coords=true}"
text/html="{FORMAT=pages}"
```

WFS coordinate substitutions

```
[coords]
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/boscombe="-1.83888,50.71111"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/chesil="-2.52293,50.60370"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/folkestone="1.12874,51.06258"
```

```
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/folkestone_met="1.12874,51.06258"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/goodwin="1.48341,51.25078"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/haylingisland="-0.95703,50.73244"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/milford="-1.61445,50.71256"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/minehead="-3.47365,51.22851"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/penzance="-5.50325,50.11382"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/perranporth="-5.17417,50.35265"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/pevenseybay="0.41662,50.78295"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/rustington="-0.49456,50.73440"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/sandownbay="-1.13116,50.65146"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/seaford="0.07592,50.76631"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/startbay="-3.61568,50.29228"
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/westbay="-2.74929,50.69254"
```

Appendix B: Mashup Development Tutorial

As an example of using the HLAPI, this section describes how a "surf status" mashup application was built.

The purpose of this mashup is to take wave height data from the HLAPI for one or more areas, plot this data on a graph, and at the same time pick up related information from other sources such as a map showing the location and lists of nearby amenities.

Scripting language and libraries

This example uses the PHP[2] scripting language. For Sparql queries and RDF manipulation it uses the Arc2[3] library and, for ease of coding and readability, Graphite[4]. The Google Chart API[5] is used for charts, and the Google Static Maps API[6] and Openlayers[7] for mapping.

Another useful tool is an RDF browser such as the Q&D RDF Browser[8].

First we load in the Arc2 and Graphite libraries and set up Graphite with a list of namespaces for coding simplicity.

```
require_once "arc/ARC2.php";
require_once "Graphite.php";
$graph = new Graphite();
$graph->ns("id-semsorgrid", "http://id.semsorgrid.ecs.soton.ac.uk/");
$graph->ns("ssn", "http://purl.oclc.org/NET/ssnx/ssn#");
$graph->ns("DUL", "http://www.loa-cnr.it/ontologies/DUL.owl#");
$graph->ns("time", "http://www.w3.org/2006/time#");
```

This continues for other useful namespace prefixes. The `id-semsorgrid` prefix is added for further code brevity.

[2] http://php.net

[3] http://arc.semsol.org/

[4] http://graphite.ecs.soton.ac.uk/

[5] http://code.google.com/apis/chart/

[6] http://code.google.com/apis/maps/documentation/staticmaps/

[7] http://openlayers.org/

[8] http://graphite.ecs.soton.ac.uk/browser/

Displaying a map of all wave height sensors

One of the observation serialisations available from the CCO deployment of the HLAPI is a GeoJSON format. This serialisation, which shows the locations of all wave height readings made in a particular time frame, can be rendered by various mapping engines including Openlayers.

The markup to display the map, given the path to an OpenJSON file, is very simple and fully documented by Openlayers.

Depending on how the HLAPI is configured, the OpenJSON representation of wave height readings for a particular hour may be at

```
http://geojson.semsorgrid.ecs.soton.ac.uk/observations/cco/Hs/20110215/00
```

Given this URL a map such as the following may be generated:

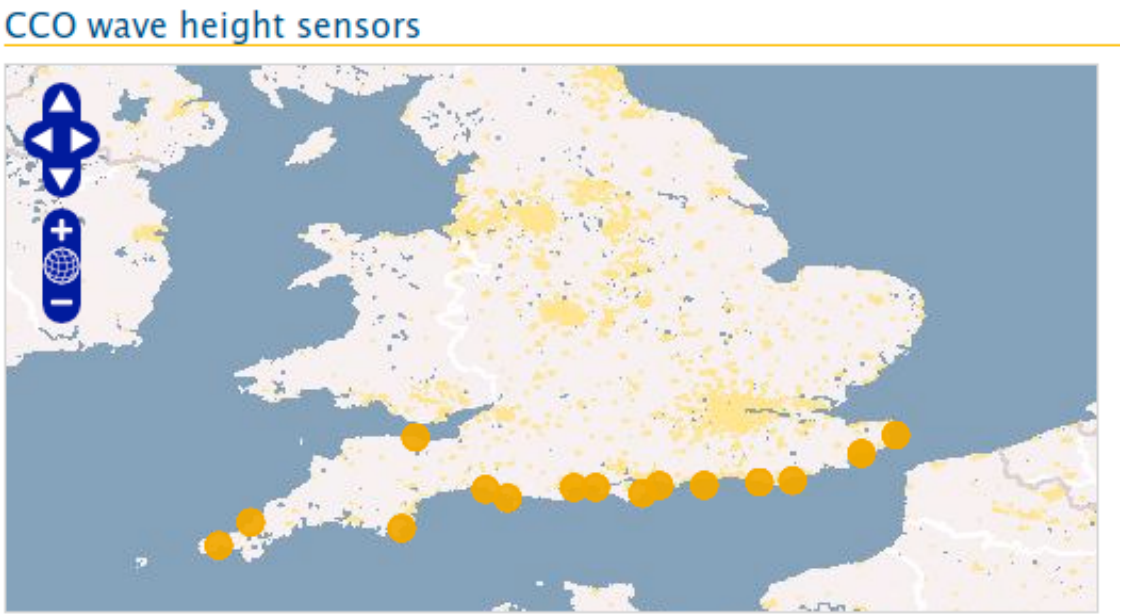

Figure 15 – Example representation of sensor locations via an OpenLayers map interface

Getting the day's wave height readings and the sensor metadata

Ideally we would programmatically find our way to the collection of observations we want to represent, but for brevity we'll assume we already know the URI of the collection. In the case of the CCO deployment, to get wave height data for the Boscombe sensor the day's readings are identified by

```
id-semsorgrid:observations/cco/boscombe/Hs/YYYYMMDD
```

Using PHP's `date` function we can complete the URI with today's date and direct Graphite to load the resources into a graph -- Graphite and the HLAPI will automatically negotiate a content type which can be used.

```
$observationsURI = "id-semsorgrid:observations/cco/boscombe/Hs/" .
date("Ymd"); $graph->load($observationsURI);
```

Graphite allows the graph to be rendered directly as HTML to quickly visualise what is available. The same can be achieved by using a dedicated RDF browser.

```
echo $graph->dump();
```

The beginning of the output is something like the following:

```
id-semsorgrid:observations/cco/boscombe/Hs/20110215 -> rdf:type ->
   DUL:Collection -> DUL:hasPart ->
      id-semsorgrid:observations/cco/boscombe/Hs/20110215#000000,
      id-semsorgrid:observations/cco/boscombe/Hs/20110215#003000,
      id-semsorgrid:observations/cco/boscombe/Hs/20110215#010000

id-semsorgrid:observations/cco/boscombe/Hs/20110215#000000 ->
   rdf:type -> ssn:Observation -> ssn:observedBy ->
   id-semsorgrid:sensors/cco/boscombe -> ssn:featureOfInterest ->
   http://www.eionet.europa.eu/gemet/concept?cp=7495 ->
   ssn:observedProperty ->
   http://marinemetadata.org/2005/08/ndbc_waves#Wind_Wave_Height ->
   ssn:observationResult -> _:arce2d5b1 ->
   ssn:observationResultTime -> _arce2d5b3 <- is DUL:hasPart of
   <- id-semsorgrid:observations/cco/boscombe/Hs/20110215
```

The bnodes are also shown, and their IDs can be traced to see which properties are available on each node.

A lot of useful information such as the sensor's coordinates is attached to the sensor's URI, which is linked from each `ssn:Observation` node. It's easy to get the URI, simply by getting `ssn:Observation` nodes, and then collecting the first found `ssn:observedBy` property of any of them. It's important to handle the case where there are not yet any results.

```
$sensor = $graph->allOfType("ssn:Observation")->
   get("ssn:observedBy")-> distinct()->current();
if ($sensor->isNull())
   die("No results yet today");
$sensorURI = $sensor->uri;
```

To get the sensor's coordinates we ask Graphite to dereference the sensor's URI and load its triples, then traverse the expanded graph to fetch the required values. The traversals here can once again be visualised by first dumping the graph or exploring the graph in any RDF browser.

```
$graph->load($sensorURI);
$location = $graph->resource($sensorURI)->get("ssn:hasDeployment")->
   get("ssn:deployedOnPlatform")->get("sw:hasLocation");
$coords = array(floatVal((string) $location-> get("sw:coordinate2")->
   get("sw:hasNumericValue")), floatVal((string) $location->
      get("sw:coordinate1")-> get("sw:hasNumericValue")),);
```

To collect all wave height observations we query the graph for all nodes of type `ssn:Observation` and skip over those whose `ssn:observedProperty` property is not

that which we are looking for (just in case we have other observation types in our graph).

Each observation corresponds to a particular time interval so we need to collect the time (in this example we'll associate the end of the time interval -- `time:hasEnd` -- with the reading) as well as the wave height observation itself. The code snippet below also skips any observations whose `ssn:observationResultTime` property doesn't point to a node of type `time:Interval`, but it would be trivial to also parse nodes of different time classes.

Finally in this snippet the array of observations is sorted by time.

Again, to see how the traversal is built up it is easiest to inspect the graph visually.

```
$observations = array();
foreach ($graph->allOfType("ssn:Observation") as $observationNode) {
   if ($observationNode->get("ssn:observedProperty") !=
     "http://marinemetadata.org/2005/08/ndbc_waves#Wind_Wave_Height")
   continue;
$timeNode = $observationNode->get("ssn:observationResultTime");
if (!$timeNode->isType("time:Interval"))
   continue;
$time = strtotime($timeNode->get("time:hasEnd"));
$observations[$time] = floatVal((string) $observationNode->
   get("ssn:observationResult")->get("ssn:hasValue")->
      get("DUL:hasDataValue")); }
ksort($observations, SORT_NUMERIC);
```

Visualising the data

The array resulting from the code above can be used to produce a chart of the wave heights. Explaining the snippet below is beyond the scope of this document, but it uses the Google Chart API to produce a line graph of wave height against time.

```
// organise data
$keys = array_keys($observations);
$start = array_shift($keys);
$end = array_pop($keys);
$period = $end - $start;
$datax = $datay = array();
$maxheight = ceil(max($observations) * 10 * 1.2) / 10;
foreach ($observations as $time => $height) {
   $datax[] = ($time - $start) * 100 / $period;
   $datay[] = $height * 100 / $maxheight;
}

// x axis labels
$axisx = array();
for ($time = $start; $time <= $end; $time += $period / 6)
   $axisx[] = date("H:i", $time);

// parameters for Google Chart API
$chartparams = array(
"cht=lxy", //line x-y
"chs=340x200", //size
"chco=0066cc", //data colours
```

```
"chm=B,99ccff,0,0,0", //fill under the line
"chd=t:" . implode(",", $datax) . "|" . implode(",", $datay), //data
"chxt=x,y,x", //visible axes
"chxr=0,0,100|1,0," . $maxheight, //x and y axis ranges
"chxl=0:|" . implode("|", $axisx) . "|2:|Time", //custom labels for
axes, evenly spread, also axis titles
"chxp=2,50|3,50", //positions of axis titles
"chf=bg,s,ffffff00", //transparent background );

// output chart
echo '<img src="http://chart.apis.google.com/chart?' . implode("&",
$chartparams) . '">';
```

It's easy to show a map with the sensor's position highlighted, too: the following uses the Google Static Maps API to do this.

```
echo '<img
src="http://maps.google.com/maps/api/staticmap?size=300x200&center=' .
$coords[0] . ',' . $coords[1] .
'&zoom=8&maptype=hybrid&sensor=false&markers=' . $coords[0] . ',' .
$coords[1] . '">';
```

Fetching related data from other data sources

We can get the name of a nearby place and the nearest post code from the web services provided by Geonames[9]. Geonames returns XML that is easy to parse with PHP. Again, explaining how the external API call works is beyond the scope of this document.

```
// get nearby place name $placenameXML =
simplexml_load_file("http://ws.geonames.org/findNearbyPlaceName?lat={$
coords[0]}&lng={$coords[1]}"); $placename = array_shift($placenameXML-
>xpath('/geonames/geoname[1]/name[1]'));  // get nearby postcode
$postcodeXML =
simplexml_load_file("http://ws.geonames.org/findNearbyPostalCodes?lat=
" . $coords[0] . "&lng=" . $coords[1]); $postcode =
array_shift($postcodeXML->xpath('/geonames/code[1]/postalcode[1]'));
```

The postcode is used in the surf status mashup to fetch the British region name from Ordnance Survey, which in turn is used to fetch population and traffic accident data from Eurostat.

Data is also collected from Linked Geodata[10] to get the whereabouts of nearby facilities. For instance, to get parking facilities within five kilometres of the sensor, its Sparql endpoint is queried as follows:

```
$store = ARC2::getRemoteStore(array("remote_store_endpoint" =>
   "http://linkedgeodata.org/sparql/"));
$rows = $store->query("
   PREFIX lgdo: <http://linkedgeodata.org/ontology/>
   PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
```

[9] http://www.geonames.org/

[10] http://linkedgeodata.org/

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE {{ ?place a lgdo:Parking . } UNION
   { ?place a lgdo:MotorcycleParking . } UNION
   { ?place a lgdo:BicycleParking . }
   ?place a ?type ;
   geo:geometry ?placegeo ;
   rdfs:label ?placename .
   FILTER(<bif:st_intersects>
   (?placegeo, <bif:st_point> ($coords[1], $coords[0]), 5)). }",
"rows");
```

The returned results include the coordinates of each parking facility (`placegeo`), from which the distance to the sensor can be calculated.

Similar queries can be used to get data on other types of nearby amenities -- the surf status mashup also locates nearby pubs, cafés and shops.

Finished mashup

The finished mashup, once styled, looks something like the screenshot shown (with only three readings so far that day).

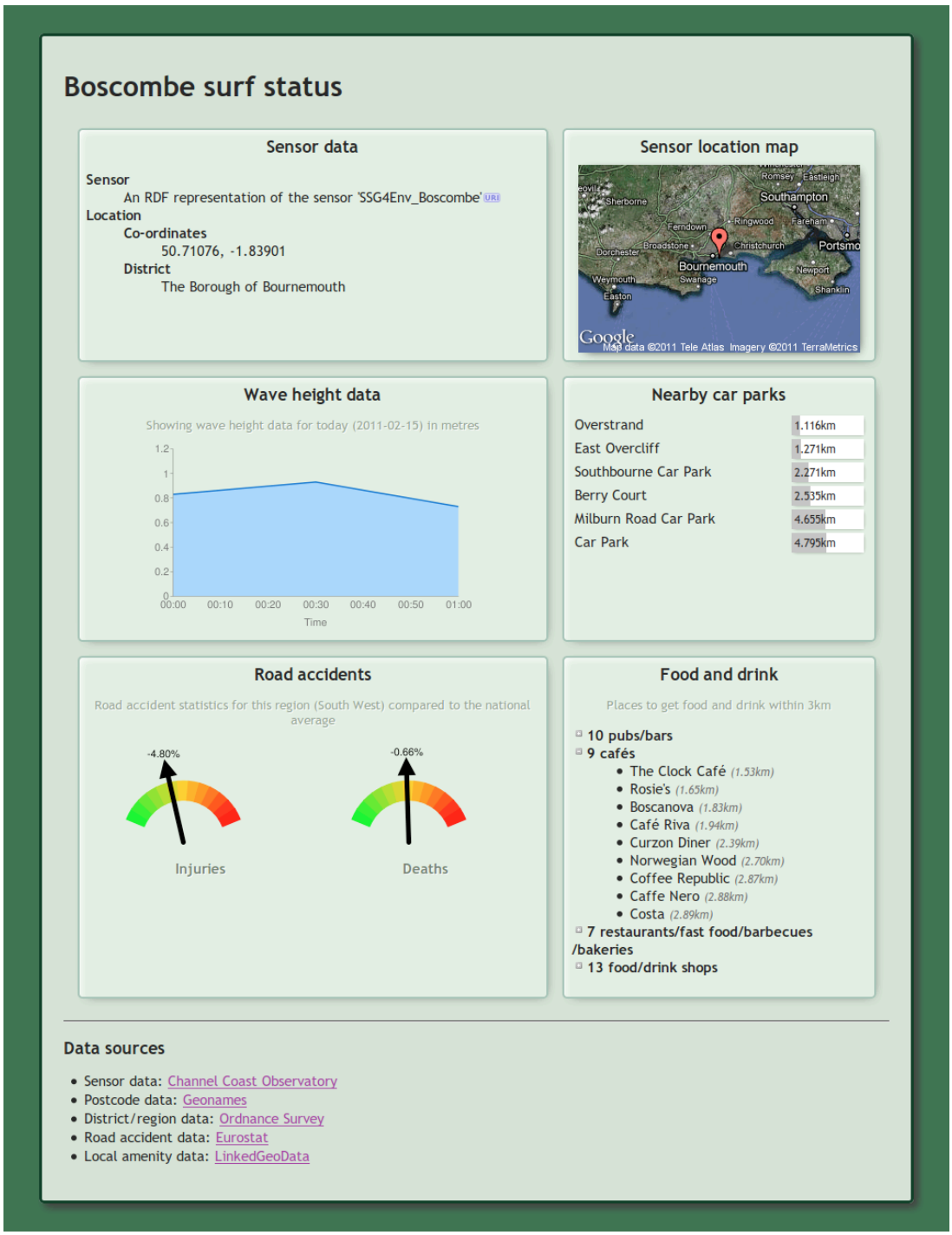

Figure 16 – Example mashup using HLAPI serialised data sources

References

[Ber2006] Berners-Lee, T. (2006) "Linked Data, Design Issues", http://www.w3.org/DesignIssues/LinkedData.html

[D1.3v1] Gray, A. J. G., Galpin, I.,Fernandes, A. A. A., Paton, N. W., Page, K., Sadler, J., Koubarakis, M., Kyzirakos, K., Calbimonte, J., Corcho, O., Garcia, R., Diaz, V., Liebana, I. (2009) "SemSorGrid4Env architecture – phase I", Deliverable D1.3v1, SemSorGrid4Env

[D2.2v1] Gray, A. J. G., Galpin, I., Rajagopalan, V., Fernandes, A. A. A., Paton, N. W., Kotsifakos, A., Kotsakos, D., and Gunopulos, D. (2010) "Implementation and Deployment of Data Management and Analysis, and the Query Processing Components – Phase 1", Deliverable 2.2v1, SemSorGrid4Env

[D4.3v2] Castro, R. G., (2011) "Sensor Network Ontology Suite v2", Deliverable D4.3v2, SemSorGrid4Env

[D5.1] Page, K. R., De Roure, D.C., Martinez, K., and Sadler, J. (2009) "Specification of high-level application programming interfaces", Deliverable D5.1, SemSorGrid4Env

[D5.2v1] Page, K. R., Sadler, J., Kit, O., De Roure, D. C., and Martinez, K. (2009) "Implementation and Deployment of a Library of the High-level Application Programming Interfaces – Phase 1", Deliverable D5.2v1, SemSorGrid4Env

[D7.1] Clark, M., Hutton, C., Sadler, J., and Roe, S. (2009) "Flood user requirements specification", Deliverable D7.1, SemSorGrid4Env

[Fie2000] Fielding, R. T. (2000). "Architectural Styles and the Design of Network- based Software Architectures", PhD thesis, Information and Computer Science, University of California, California, USA, 2000

[OGC-OM] Cox, S. (2007) "Observations and Measurements – Part 1 – Observation schema", OGC 07-022r1

[OGC-SOS] Na, A. and Priest, M. (2007) "OpenGIS Sensor Observation Service 1.0", OGC 06-009r6

[OGC-WFS] Vretanos, P. A. (ed.) (2005) "OpenGIS Web Feature Service (WFS) Implementation Specification 1.1.0", OGC 04-094

[Pag2009] Page, K. R., De Roure, D. C., Martinez, K., Sadler, J. D., and Kit, O. Y. (2009) "Linked Sensor Data: RESTfully Serving RDF and GML", Proceedings of the 2nd International Workshop on Semantic Sensor Networks