**SemSorGrid4Env**

**FP7-223913**

Deliverable

D1.3v2

# SemSorGrid4Env Architecture – Phase II

Alasdair J G Gray (Editor), Ixent Galpin, Alvaro A A Fernandes, and Norman W. Paton
*University of Manchester*

Kevin Page, and Jason Sadler
*University of Southampton*

Kostis Kyzirakos, and Manolis Koubarakis
*National and Kapodistrian University of Athens*

Jean-Paul Calbimonte, Raúl Garcia-Castro, and Oscar Corcho
*Universidad Politécnica de Madrid*

Jesús E Gabaldón, and Juan José Aparicio
*TechIdeas*

December 17, 2010

Status: Final version
Scheduled Delivery Date: 31 December 2010

# Executive Summary

The goal of the Semantic Sensor Grid Rapid Application Development for Environmental Management (SemSorGrid4Env) project is to provide a software platform that supports the rapid development of applications over data sources, including both sensor networks and traditional database, in which significant value is derived by the use of semantic techniques for service discovery and data integration. The purpose of this deliverable is to specify, design, and validate the software architecture.

The SemSorGrid4Env software architecture comprises a set of Web services. The backbone of the software architecture is provided by the service-oriented services offering the WS-* interfaces defined in this document. These will be supplemented with RESTful resource-oriented interfaces. The SemSorGrid4Env service-oriented Web services can be categorised into three tiers based on the functionality that they provide:

**Application Tier:** Consists of services that provide domain specific support to applications;

**Middleware Tier:** Consists of services that provide extra value through the use of semantic techniques, e.g. service discovery and data integration;

**Data Tier:** Consists of services that provide access to data, which can be published either as a stream (e.g. from a sensor network) or as traditional stored data (e.g. from a database).

This document specifies the interfaces for four service-oriented Web services. The *Stored Data Service* enables traditional data, such as a database, to be published and queried. The *Streaming Data Service* enables continuous publication and querying of data such as that generated by a sensor network. The *Registration and Discovery Service* enables resources, such as data or services, to be registered and later discovered. The *Integration and Query Service* enables the data provided by multiple data sources to be unified and queried through a single data model.

The functionality of the Web services provided by the software architecture will support the use case of the exemplar environmental monitoring application being developed in the SemSorGrid4Env project. The implementation of the Web services defined by this document are the focus of work packages 2-5 of the SemSorGrid4Env project and will be fully described in the deliverables associated with each of them. The deployment, integration, and testing of the Web services is the focus of work package 1 and will be described in deliverables D1.2 and D1.4.

The major changes between this and the previous version of this document are as follows:

**Section 1:**

- The scope of the document has been clarified.

**Section 2:**

- The introductory example has been changed to one from the flood scenario.
- Figure 2.2 has been revised.
- Revision of Section 2.2.1 to reflect changes in WP5.
- Addition of Section 3.1 to discuss metadata.
- Removal of obsolete sections discussing validation and implementation of the architecture.

**Section 3:**

- Revised introduction to reflect changes in metadata policy.
- Service Interface revised for use with semantic property documents.
- Integration Interface revised to allow for the addition and removal of data sources.
- Description of the Query Interface refined to make it self-contained.
- Description of Subscription Interface extended to include usage pattern with the Query Interface.

**Section 4:**

- Section 4.1 now provides descriptions of the services offered in the application tier.
- Description of the Distributed Query Processing Service added in Section 4.3.3.

**Section 5:**

- Old Validation and Implementation sections have been removed and replaced with this section describing some sample service orchestrations in the context of the flood use case.

**Section 6:**

- Updated to reflect current version of the document.

# 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:

- None

# 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** | `http://www.semsorgrid4env.eu/files/deliverables/wp1/D1.3v2.pdf` | | |
| **EU Project Officer** | Antonios Barbas | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Deliverable** | **Number** | D1.3v2 | **Name** | SemSorGrid4Env Architecture – Phase II | | |
| **Task** | **Number** | T1.3v2 | **Name** | Define a Semantic Sensor Grid Architecture | | |
| **Work package** | **Number** | | | WP1 | | |
| **Date of delivery** | **Contract** | 31 December 2010 | **Actual** | 31 December 2010 | | |
| **Code name** | D1.3 | | **Status** | draft ☐ final ☑ | | |
| **Nature** | Prototype ☐ Report ☑ Specification ☐ Tool ☐ Other ☐ | | | | | |
| **Distribution Type** | Public ☑ Restricted ☐ Consortium ☐ | | | | | |
| **Authoring Partner** | UNIMAN | | | | | |
| **QA Parner** | NKUA | | | | | |
| **Contact Person** | Alasdair J G Gray | | | | | |
| | **Email** | A.Gray@cs.man.ac.uk | **Phone** | +44 161 275 6132 | **Fax** | +44 161 275 6213 |
| **Abstract (for dissemination)** | This document specifies, designs, and validates the Semantic Sensor Grid Rapid Application Development for Environmental Management (SemSorGrid4Env) software architecture. The architecture enables the publication and querying of both stored (e.g. database) and streaming (e.g. sensor) data to support the rapid development of applications for environmental monitoring. Significant benefits are provided by the use of semantic technology for service discovery and data integration.<br>The infrastructural backbone of the architecture is provided by four service-oriented services: Stored Data Service for the publication of databases, Streaming Data Service for the publication of sensor data, Registration and Discovery Service to enable resources to found, and Integration and Querying Service to enable multiple data sources to be accessed through a single model. These services will be supplemented with application domain specific services which may offer RESTful interfaces. | | | | | |
| **Keywords** | Architecture, Web service, Service oriented architecture, sensors, stream | | | | | |

| **Version log/Date** | **Change** | **Author** |
|---|---|---|
| 0.1 | Template | A. Ibarrola |
| 0.2 / 27 February 2009 | Initial Structure | A. Gray/A. Fernandes |
| 0.3 / 27 March 2009 | Coalesced contributions from partners | All authors |
| 0.4 / 17 April 2009 | Revised interfaces from partners | A. Gray |
| 0.5 / 8 May 2009 | Restructured document, coalesced interfaces, added validation sections | A. Gray, J. Sadler, and V. Diaz |
| 0.6 / 19 June 2009 | Finalised interfaces, revised implementation section | A. Gray, K. Kyzirakos, I. Galpin, O. Corcho |
| 0.7 / 10 July 2009 | Revised validation sections | A. Gray, J. Sadler |
| 0.8 / 27 July 2009 | Tidied up document, added revised fire validation, added integrator implementation details | A. Gray, I. Liébana, J.P. Calbimonte, O. Corcho |
| 0.9 / 25 August 2009 | Tied up loose ends. Applied QA corrections | A. Gray |
| 1.0 / 28 August 2009 | Converted to final version | A. Gray |
| 1.1 / 21 November 2010 | Initial updating of document | A. Gray |
| 1.2 / 3 December 2010 | Tidied up previous revision. Version sent for QA. | A. Gray |
| 1.3 / 16 December 2010 | Addressed QA comments. Version sent for verification. | A. Gray |
| 2.0 / 17 December 2010 | Corrected document URL. Set to final version. | A. Gray |

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

# Contents

# List of Figures

# List of Tables

# Glossary

**AJAX** Asynchronous JavaScript and XML; Although it is applied more widely to include JSON (JavaScript Object Notation).

**API** Application Programmer Interface. A set of defined method calls.

**BNF** Backus Naur Form.

**DBMS** Database Management System.

**DQP** Distributed query processor.

**GML** Geography Markup Language.

**GSN** Global Sensor Networks.

**HTML** Hypertext Markup Language.

**HTTP** Hypertext Transfer Protocol.

**HTTPS** Hypertext Transfer Protocol Secure. The layering of a cryptographic protocol on top of HTTP.

**JDBC** Java Database Connectivity an API for accessing relational databases from Java applications.

**JSON** Javascript Object Notation.

**Mash-Up** An application that is typically written in a scripting language such as Javascript which is run in a Web browser and that makes asynchronous HTTP calls to enable access to multiple Web services that expose APIs.

**ODBC** Open Database Connectivity an API for accessing relational databases from applications.

**OGC** Open Geospatial Consortium Incorporated.

**OGC-SWE** Open Geospatial Consortium Sensor Web Enablement.

**OGC-WMS** Open Geospatial Consortium Web Map Service.

**OGSA-DQP** Open Grid Services Architecture Distributed Query Processing Services.

**OS** Operating system.

**OWL-S** Semantic Markup for Web Services.

**QName** Qualified name within an XML document.

**$R_2O$** Relational-to-Ontology mapping language.

**RDF** The Resource Description Framework. A W3C standard originally designed for modelling metadata about Web resources.

**RDF/XML** A file containing RDF marked up in XML.

**REST** Representational State Transfer.

**RESTful** A Web service that exposes a Representational State Transfer interface.

**ROA** Resource Oriented Architecture.

**RPC** Remote Procedure Call.

**$S_2O$** Stream-to-Ontology mapping language based on $R_2O$.

**SANY** Sensors Anywhere. European FP6 project.

**SAWSDL** Semantic Annotations for WSDL and XML Schema.

**SMS** Short Message Service.

**SNEEql** Sensor Network Evaluation Query Language for querying streams of data.

**SOA** Service Oriented Architecture.

**SOAP** Simple Object Access Protocol.

**SOS** Sensor Observation Service. Part of the OGC Sensor Web Enablement standards.

**SPARQL** SPARQL Protocol and RDF Query Language for querying RDF documents.

**SPS** Sensor Planning Service. Part of the OGC Sensor Web Enablement standards.

**SQL** Structured Query Language for querying relational databases.

**stSPARQL** Provides spatial and temporal extensions to SPARQL.

**URI** Uniform Resource Identifier.

**WS-*** The group of Web service standards.

**WS-DAI** WS-Data Access and Integration [AAK+06].

**WS-DAI-RDF(S)** The RDF realisation of WS-DAI [GGP08].

**WS-DAI-Streaming** The data stream realisation of WS-DAI [GGFP10].

**WS-DAIR** The relational realisation of WS-DAI [ACK+06].

**WS-DAIX** The XML realisation of WS-DAI [AHK+06].

**WS-N** WS-BaseNotification [GHME06].

**WSDL** Web Services Description Language [CMRW07].

**WSMO** Web Service Modeling Ontology.

**XML** Extensible Markup Language.

**XPath** XML Path Language.

**XQuery** XML Query Language.

# 1. Introduction

This deliverable comprehensively describes the SemSorGrid4Env software architecture at an abstract level.

## 1.1 Context

The main objective of the SemSorGrid4Env project is to specify, design, implement, evaluate and deploy a software architecture that supports the rapid development of applications over data sources, including both sensor networks and traditional databases, in which significant added value is derived by the use of semantic techniques in all aspects of the architecture, but particularly so in the areas of service discovery and data source integration.

The SemSorGrid4Env architecture can be seen as a three-tier architecture insofar as it allows applications to rely on middleware for the semantic discovery and integration of data services that virtualize distributed, autonomous, semantically heterogeneous data sources. In particular, the SemSorGrid4Env architecture can be seen as adhering to an hourglass [NRE94] approach, i.e. a wide variety of applications can benefit from a wide variety of data sources by making use of a much narrower set of middleware services.

From a three-tier perspective, the novelty of the SemSorGrid4Env project lies mainly in deploying a middle tier of semantic-based approaches to service discovery and integration that enables:

1. In the lower, data, tier, not only the use of one-off queries over stored extents, but also continuous queries over streams (irrespective of whether the data originally stems from traditional databases or from sensor networks), and
2. In the upper, application, tier, not only the classical synchronous (RPC-based) and asynchronous (message-based) approaches to activity-oriented distributed computing, but also the more recent, resource-oriented approach based on representational state transfer (REST) [Fie00] built upon a fixed suitable protocol (e.g. HTTP). (For the distinction between activity-oriented and resource-oriented approaches, see [Sne04].)

## 1.2 Scope

This document focuses primarily on the SemSorGrid4Env architecture. It does not aim to specify exhaustively the service implementations that underpin the Web services, this will be focus of deliverables D2.2, D3.3, D4.2, and D5.2. This document also does not consider in detail the technologies that will be used to realize those implementations, or the evaluation and testing of the implementation. This is the focus of deliverables D1.2 and D1.4 respectively.

## 1.3 Objectives

The main objective of the SemSorGrid4Env architecture is to enable the rapid development of thin applications (e.g. web applications, Mash-Ups[1], etc.) that require real-world real-time data

[1]Broadly speaking, a *Web mash-up* is an application that is typically written in a scripting language such as Javascript which is run in a Web browser and that makes asynchronous HTTP calls to enable access to multiple Web services that expose APIs, and presents a unified user interface across the data.

coming from heterogeneous sensor networks, making it possible for sensors to be used for other environmental management purposes than those that they were originally expected to have (hence enabling the reuse of existing sensor networks and potentially spreading the costs) and to combine real-time data with historical data from independent, pre-existing data sources, thereby improving current decision-making procedures in a variety of situations related to environmental management, such as the scientific modelling of physical phenomena with a view towards assessing short-term risks and for emergency response.

The main objectives of this document are:

1. To define the interfaces, by means of the operations and metadata, required to provide the SemSorGrid4Env functionality;
2. To define the Web services, by means of the interfaces that they expose, that will enable the SemSorGrid4Env functionality;
3. To identify the functionality that is assumed to be provided by the underlying Web services framework;

# 2. Overview

Due to the distributed nature of data sources, both sensor networks and existing databases, and of the types of users considered within the SemSorGrid4Env project, the software architecture will be Web service based. As an example of the type of scenario that the SemSorGrid4Env architecture is intended to support, consider a Web mash-up application which aids an emergency planner to detect and co-ordinate the response to a flood. This involves retrieving relevant data from multiple sources (some of which may "integrate" data from other data sources) and presenting it to the user. The interaction of the various Web services needed to support this scenario are shown in Figure 2.1 and described below. Note, this is just an illustrative example showing the types of services required.

1. The mash-up consults a semantically enabled Registry service to find data sources that contain metocean measurements (e.g. wave, tide, and meteorology data) for the south coast of England.
2. The registry uses suitable ontologies (both spatial and meteorological) to interpret the request and identifies an "integrated" data source that presents an ontological view of the data which satisfies the request.
3. The mash-up sends a correlation query over a metocean ontology model to the semantically integrated data source. For example, this might ask for correlations in sensed data about the current wave height with stored data about the height of sea defences in the locality, say the Solent[1]. The goal would be to identify overtopping events, i.e. when a wave goes over the top of a sea wall.
4. The Semantic Integrator rewrites the query over its ontological model into queries over the data sources to extract the required information. Since one of the sources is a stream of information, this will involve a continuous queries, with the stored source being polled through a stream processing engine.
   (a) The Semantic Integrator sends a continuous query to the Stream Processor Data Source to analyse the information stored in the database, e.g. it periodically polls the database for the heights of sea defences in the selected region.
   (b) The Semantic Integrator sends a continuous query to the Sensor Network Source for execution in the network to monitor tide and wave heights.
5. The data sources stream their results back to the integrator (a) and (b) which combines them and converts the result back into the concepts of the ontological view.
6. The Semantic Integrator continuously sends the answer stream to the mash-up.
7. At some point as the mash-up is interpreting the answer stream, it indicates that an over-topping event is predicted. The mash-up application queries a semantically enabled Registry service to discover additional data services that contain information for the affected region (using a spatial ontology).
8. The Registry responds with the relevant data sources.
9. The mash-up sends a query to one of the data sources (e.g. a database from UK Government[2]) to discover the details of schools in the locality.
10. The data source sends the mash-up a response to the query.
11. The mash-up sends the postcodes of the schools to Yahoo!'s Geocoding service.

[1]A strait separating the Isle of Wight from the mainland of England.
[2]`http://data.gov.uk` accessed 15 December 2010.

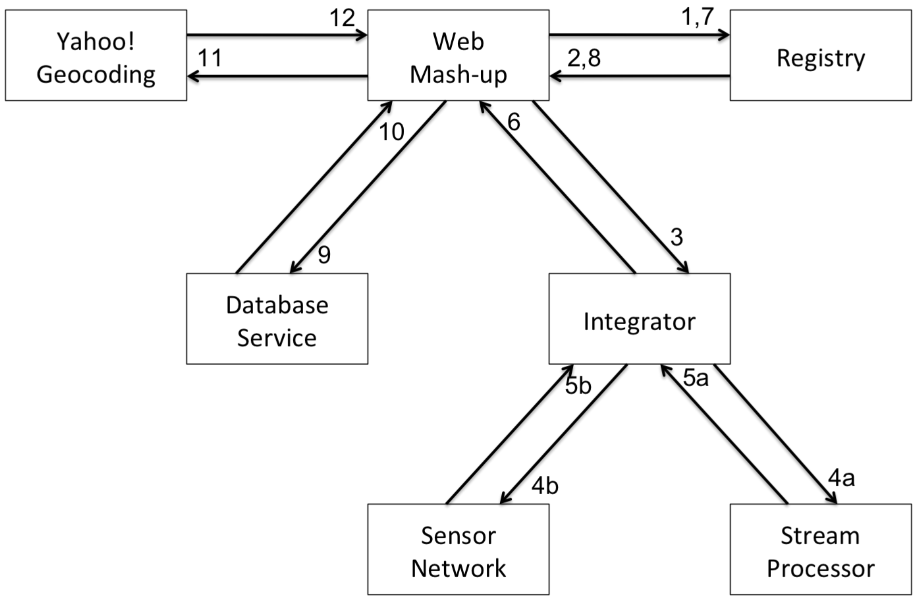


Figure 2.1: The interactions of various Web services to support a Web mash-up application for flood detection and response planning.

12. Yahoo!'s Geocoding service responds with a map containing drawing pins at the latitude and longitude pairs where the schools are located.

This section starts by discussing the type of Web service architectures and their relationship to the SemSorGrid4Env architecture. We will then describe the service tiers to which the majority of the Web services can be seen to belong. The next section identifies the requirements on the underlying Web service infrastructure. Finally, we put the architecture in context with related work.

## 2.1 Web Service Architectures

Two major types of Web service architectures exist: Service Oriented Architecture (SOA) and Resource Oriented Architecture (ROA). For a recent comparison of these two approaches see [PZL08].

### 2.1.1 Service Oriented Architecture

A Service Oriented Architecture (SOA) enables unassociated, loosely-coupled services—each of which provides some self-contained functionality—to be combined through *orchestrations*, calling a sequence of services, to provide some combined functionality. The services communicate by exchanging messages using well-defined, rich, interfaces. The terms used to describe and define service oriented Web service architectures are given in [BHM+04] and [HB04]. A good overview and survey of service oriented Web services, as of 2003, is found in [ACKM04].

In a service oriented architecture, a Web service provides a self-describing interface, defining the messages that it accepts, using the Web Service Description Language (WSDL) [CMRW07]. Other services may send messages, wrapped using SOAP [GHM+07], to invoke the Web service to perform some functionality and, if required, to return a result. Such services often advertise their functionality and interface in a registry so that they can be discovered by potential users.

A large number of Web service standards, commonly referred to as WS-*, have been defined to provide common mechanisms for core functionality. For example, WS-Security [NKMHB06] defines the mechanism by which SOAP messages may be exchanged using end-to-end encryption. These WS-* standards enable feature rich services to be developed by reusing components, and that can inter-operate with other Web services.

### 2.1.2 Resource Oriented Architecture

The concept of a *Resource Oriented Architecture* (ROA) is based on the idea of a set of services and Web applications communicating using the principles of *Representational State Transfer* (REST) [Fie00]. The design principles of REST are:

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

The key principle of REST is the use of *resource* for specific things that a user may wish to reference, and the referencing of these resources using URIs. *Representations* of these resources—encoded in a particular format—are then accessed through the URI, usually using HTTP. REST limits the operations exposed by a Web service to a small, well-defined, standard set—for HTTP these are GET, POST, PUT, and DELETE. This brings both benefits (e.g. compatibility and scalability with standard Web infrastructure) and further constraints (e.g. idempotence becomes desirable across operations to cope with network unreliability).

This constrained set of operations leads to a design process focused on correctly identifying the resources that should be exposed for a service and their representations; while the interface to the resources is simple, the number of resources—every piece of information that could be served—is likely to be many, with a URI for each. Since an application client cannot possibly know of every URI in existence it is important that resources hyperlink to other resources so a client application can navigate around them. However, these are prone to broken links when the resources change, e.g. the underlying data is updated, deleted, etc, and resources that are unreachable through navigation due to no links being created to a new data resource. Any application state that is required to provide a representation of a resource must be completely contained within the request to the server (where the application is the client software processing and modifying the resource representations returned by the service). Transitions in application state are made by moving—"navigating" in a Web sense—to alternate resources provided as URI links in the representation of a resource provided returned by the server.

### 2.1.3 The SemSorGrid4Env Web Service Architecture

The SemSorGrid4Env architecture will combine both architectural approaches for Web services, with services potentially offering both kinds of interfaces to their application logic. The REST service interfaces will be exposed to support application development with minimal *a priori* design such as Web mash-ups. The WS-* service interfaces will be provided to support orchestrations, such as those that semantically integrate data from multiple sources. More discussion of the way in which both kinds of interface will be exposed is provided in the next section.

This document focuses on defining the SOA Web service interfaces. These will provide the "infrastructural backbone" of the deployment of the SemSorGrid4Env architecture and enable the more *ad hoc* Web mash-ups to be developed. This also reflects the natures of the two kinds of Web service architectures since the ROA necessarily focuses on the data content of the services.

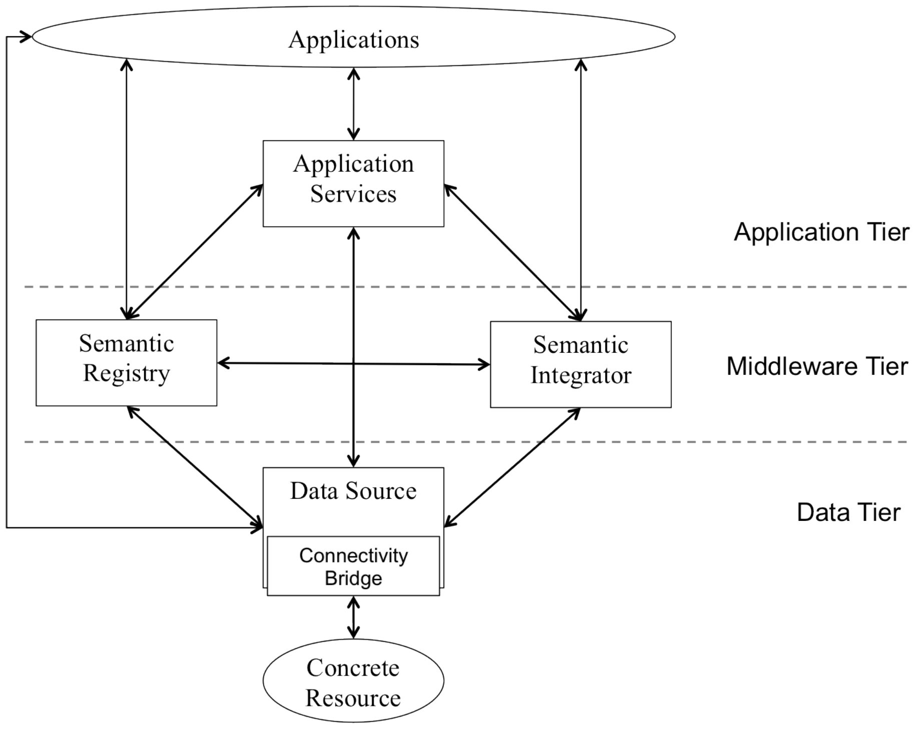


Figure 2.2: The SemSorGrid4Env architecture: the services, their relationships, and the classes that they belong to.

## 2.2 Service Tiers

The SemSorGrid4Env architecture has three major classes of services, which, as highlighted before (Section 1.1), can be seen to define an hourglass [NRE94] (in the sense that, above it, there may be a very large number of applications making use of it and, below it, there may be a very large number of concrete resources that it provides access to), and to instantiate a three-tier distributed system. This document refers to these service classes as characterising (1) an application tier, (2) a middleware tier, and (3) a data tier.

The SemSorGrid4Env architecture offers Web services to form a service-oriented architecture. A service is fully determined by the interfaces it exposes and by the interfaces it expects other services to expose. The interfaces of a service are fully determined by the set of operations it comprises and the metadata it makes available. Each operation is fully determined by the inputs it takes as arguments, the output it returns, and the faults it may raise. Inputs, outputs and faults are grounded on precise type specifications.

Figure 2.2 provides an overview of the SemSorGrid4Env architecture and its relationship to applications and data sources, which are external to it. Boxes denote SemSorGrid4Env services, ovals denote external services/entities (e.g. OGC-SWE services), arrows denote caller-callee relationships. Note that multiplicity is assumed for all services, the applications represented by the cloud at the top, and the many data sources represented by the Data Services and the cloud at the bottom. The diagram presents the services, their relationships, and the class of service that they belong to. The specifics of each tier will be explained in the following subsections. However, the key implications that can be derived are:

- Any service may directly call any other service regardless of which tier they appear in.
- Pre-existing, typically external, services may be called by any service, even directly if the means to carry out the interaction are known to the caller.

- Ultimately, the services in the data tier must either virtualize data sources using a connectivity bridge with specific drivers that ensure interoperability (e.g. most DBMS products offer JDBC drivers) or else resort to services that act as wrappers and are grounded by one or more connectivity bridges (e.g. a WS-DAI implementation wraps many kinds of concrete DBMS products)[3].
- The semantic middleware tier provides additional value to services in the application and data tiers. Broadly speaking, for the latter, it allows them to be federated using semantic integration and query rewriting; for the former, it allows them to find services on the basis of semantic descriptions of these services.

### 2.2.1 Application Tier

The application tier comprises services that provide domain specific functionality to consumers and applications. It is envisaged that the applications supported by the SemSorGrid4Env architecture will lie at or between two extremes of a spectrum. At one extreme lie applications that require agile development, e.g. in response to an unfolding real-world event. At the other extreme lie applications that are driven by stable, long-lived requirements and hence benefit from careful principled design.

In the first case, there may be minimal *a priori* design since information from diverse resources needs to be collected and collated in a rapid manner. For such applications, results would typically be presented as a mash-up by means of juxtaposition, i.e. information items stemming from different sources that are not necessarily automatically related, let alone semantically integrated. However, it is an aim of the SemSorGrid4Env project to provide additional semantic information to enable more intelligent mash-ups for such applications, which may well remain in use only for short periods, as a result of their being intimately associated with a transient real-world event.

For the second case of applications, information items from different sources would often be integrated (as opposed to simply juxtaposed) using semantics-driven mediation middleware to rewrite requests made against a global model into the local model of each contributing source and, conversely, to rewrite results expressed in terms of local models into a result that conforms to a global model. This mediation process projects to applications a unified view of a collection of distributed resources that may be semantically heterogeneous and autonomously conceived, developed and maintained. Such applications may well remain in use for much longer periods as a result of their being intimately associated with stable user requirements.

Finally, to support the full spectrum of applications, i.e. to support those which can only make HTTP calls, the application tier will provide services that enable REST access to the SOA services of the middleware and data tiers.

### 2.2.2 Middleware Tier

The middleware tier provides two main classes of services, viz., semantic service registration and discovery, and semantic integration and query. In this document, the use of the term semantic implies the extensive and intensive use of explicit annotation in formal languages that enable automated reasoning for the purposes of more effective discovery, integration, and (potentially unplanned-for) reuse of services and data sources.

Broadly speaking, a semantic registration and discovery service enables users to express their requests for services in terms of real world concepts, e.g. a data source with data covering the Solent

[3]In the case of sensor networks, the idea of connectivity driver also applies in abstract but is less easy to exemplify (although the work on GSN [GSN07] instantiates some aspects of what the notion of a connectivity driver is intended to mean in this context).

region of the south coast of England, as opposed to the more value-oriented traditional registries, e.g. the longitude and latitude for the Solent would need to be expressed. Analogously, a semantic integration and query processing service offers enriched[4] means, in comparison to classical data integration approaches, to map requests against a global model into requests against local ones.

Note here that in this document, little is said about what might be called lower-level middleware. In particular, the assumption is made that the SemSorGrid4Env project will support both synchronous and asynchronous modes of interaction. Synchronous modes encompass RPC-based approaches, e.g. using SOAP [GHM+07], as well as REST approaches, e.g. using HTTP. Asynchronous modes encompass both paradigms such as publish/subscribe and queue-based mechanisms for decoupling requests and responses. In all such cases, lower-level middleware is involved, e.g. SOAP/REST engines (such as Apache Axis2 [Apa09] and CXF [CXF10]) possibly enriched by other WS-* standards (such as Apache Sandesha [Apa09]for reliable messaging). More details about the underlying infrastructure can be found in Section 2.4 and in deliverable D1.2 [VA09].

### 2.2.3 Data Tier

The data tier comprises data service providers. (In this document, a distinction is only made between data and metadata when there is a specific benefit in doing so, insofar as, often, the same item can be seen as data by one consumer and as metadata by another.)

The main purposes of a data service are to virtualize one or more data sources and to do so in such a way as to reconcile any heterogeneity other than semantic ones. In this respect, a data service acts as a wrapper over a concretely implemented source of data and exposes interfaces that enable every other source of data that is similarly wrapped to be interacted with in a uniform manner.

The virtualization of a source ultimately assumes that a connectivity component (bridge/driver) exists. For classical databases, examples of such connectivity components include ODBC and JDBC, which have specific drivers for specific DBMS/OS combinations. For sensor networks, no such connectivity components exist.

The main classes of data sources of interest in the SemSorGrid4Env project fall into the following broad categories:

1. Repositories containing such documents as are normally created, read, updated and deleted by REST Web services;
2. Repositories of structured data according to a data model (e.g. relational) implemented by a database management system (DBMS); and
3. Sensor networks.

The first kind of resource is virtualized by default, in accordance to the resource-oriented view of Web services that builds on REST interactions using HTTP as an application-level interaction protocol.

The SemSorGrid4Env architecture virtualizes the latter two kinds of resource using query-processing technology. Thus, for structured data (represented as either tuple sets, as XML documents or RDF triples) the virtualization adopts existing approaches, such as WS-DAI [AKP+06] in which requests are ultimately specified as queries in SQL, XPath/XQuery, or SPARQL. For sensor networks, if the sensed data rests in persistent stored extents and if its temporal aspect is not relevant, then the very same virtualization mechanisms apply. If the temporal aspect is relevant, i.e. if either the data is streamed or (although resting in a persistent stored extent) is best seen as such, then

[4]Enriched in this context means raising the level of abstraction from data model types to conceptualisations of the world.

the SemSorGrid4Env architecture uses a continuous query language, e.g. SNEEql [BGFP08], to specify data requests.

Note that in this document, little is said about what might be called lower-level sensor network software. Broadly speaking, some sensor networks make little or no use of the general-purpose computing power that may be available in a node, and even when they do, there is little reliance on *ad hoc* repurposing (e.g. the code running in the nodes is often unchanged for the duration of a deployment). In contrast, other sensor networks make use of the general-purpose computing power that is available in nodes and effectively construes a sensor network as a distributed computing platform in its own right. In this latter case, queries can be compiled to run inside the network, and it is possible, in principle, to repurpose the network on an as-needed basis (e.g. in response to real-world events as an incident, say an environmental emergency, unfolds over a period of time).

Data sources may also expose their content without providing a query interface. In such a case, the data is made available in such a way that the client can iterate through it. This mechanism can also be used to decouple the posing of a query and the retrieval of the answer to that query. Additionally, a distributed query processing service can be offered which provides query-based access to one or more data services which may or may not offer a query interface themselves.

## 2.3 Service Metadata

Service-oriented Web services are defined by the functional features of the interface that they expose: the operations they support, the parameters of the operations, and the types of the parameters. This provides enough information to enable *well-formed* interactions with the service. However, there is also a need to define the metadata (also called non-functional properties within the Web services community) associated with a service to enable *well-informed* interactions with the service. For example, if two services offer the same interface specification and access to the same dataset, a user will want to differentiate between them based on other features of the service, e.g. the pricing policy associated with using the service or the expected response time. There have been several proposals for providing the non-functional metadata about a service in the interface specification. Approaches such as SAWSDL [FL07], OWL-S [MBH+04], and WSMO [dBBD+05] use semantic annotations to specify the metadata using concepts from ontologies, with SAWSDL embedding the annotations directly in the interface specification.

The metadata of interest for environmental monitoring and emergency response scenarios is focused on the data made available through the service. In particular, users are interested in selecting services based on their spatiotemporal and thematic coverage. Such metadata is orthogonal to the non-funcional properties of the service as described above. The WS-DAI standard recognises the need for metadata about the data content, e.g. the schema, and provides this by associating a property document with each data resource [AAK+06]. The SemSorGrid4Env architecture adopts the WS-DAI approach and extends it to support the expression of the dataset metadata using semantic annotations (Section 3.1). It is anticipated that the metadata associated with the dataset(s) of a service will be stored in a Semantic Registration and Discovery Service (Section 4.2.1), and used to inform service selection.

## 2.4 Technological Infrastructure

The SemSorGrid4Env architecture will be realised as a set of Web services providing a mix of SOA and ROA interfaces defined in subsequent sections of this document. The deployment of these services will rely upon Web service containers such as Apache Axis 2 [Axi08] and Apache CXF [CXF10] which can host both types of services side-by-side. The SemSorGrid4Env architecture makes certain assumptions about the capabilities of these Web service containers and the underlying Web services infrastructure. For clarity, these assumptions and details of their implementation in the infrastructure are summarised below. Further details of the technological infrastructure that will be used in the SemSorGrid4Env project have been defined in deliverable D1.2 [VA09]. Note however that the SemSorGrid4Env architecture is independent of the SemSorGrid4Env technological infrastructure. It only relies on there being a suitable infrastructure that meets the assumptions detailed below.

**Common Message Format.** The SemSorGrid4Env architecture assumes that there is a common format for sending messages between Web services. By assuming a common format, it allows services to concentrate on the functionality that they provide. The SOAP standard [GHM+07] provides this mechanism for SOA Web services, and Web service containers provide tools for parsing these messages. For the ROA Web services the common message format will be HTTP [FGM+99].

**Reliable Messaging.** The SemSorGrid4Env architecture assumes that there is a reliable mechanism for messages to be delivered between services even in the presence of component or network failures. That is, when a Web service sends a message to another service it can assume that the other service receives the message or it will receive an error. The WS-ReliableMessaging standard [DKP+09] provides a protocol by which to offer these guarantees for a SOA architecture. Implementations of the WS-ReliableMessaging standard are available for both Axis and CXF.

**Security.** The SemSorGrid4Env architecture assumes that standard Web service security mechanisms will be employed to provide secure message transfer, e.g. using encrypted messages, and to enable users to authenticate themselves to services for authorization purposes. Current practice is to use WS-Security [NKMHB06] which provides message signing and encryption mechanisms for the secure transfer of certificates, etc. Both Axis and CXF provide implementations of this standard. The ROA Web services will rely on HTTPS [Res00] which provides a security layer over standard HTTP connections.

## 2.5 Related Architectures

The SemSorGrid4Env architecture is not the first software architecture to be developed for making data from sensor networks available on the Web. We will briefly review a few key architectures and identify how our approach differs from or extends these existing mechanisms.

### 2.5.1 Global Sensor Network

Global Sensor Network (GSN) [AHS07] is a middleware platform for the deployment and programming of sensor networks. It allows for the abstraction from the underlying hardware of the sensor nodes and provides query processing capabilities within the middleware. It enables a data-oriented view of sensor networks and the processing of queries over that data.

The key abstraction in GSN is that of a *virtual sensor* which makes a single stream of data available. A virtual sensor can publish the data coming from a sensor network or it may combine multiple sources of data, e.g. readings from actual sensors or the stream from another virtual sensors, and perform some processing to generate its answer stream. However, in GSN the query specification is tied to their virtual sensor definition which separates the sources and time model from the query, which is express in SQL. In the SemSorGrid4Env architecture, it is feasible to use any continuous query processing language, although only support for SNEEql will be developed within SemSorGrid4Env. We also make the data sources available as a set of Web services which goes beyond the capabilities of GSN to combine arbitrary services. As such, it is possible for existing sensor data published as a GSN virtual sensor to be wrapped by a SemSorGrid4Env Web service and made available within the SemSorGrid4Env architecture.

### 2.5.2 Sensor Web Enablement

The Open Geospatial Consortium (OGC) [OGC09] have defined a set of seven standards for enabling the publication of sensor data on the Web, referred to as "Sensor Web Enablement" (OGC-SWE) [BPRD06, BPRD07]. These standards cover the mechanisms for describing sensors and their data (Observation and Measurements Schema, Sensor Model Language, and Transducer Markup Language), the requesting for data and tasking of sensors (Sensor Observation Service, and Sensor Planning Service), and the delivery of data using publish-subscribe mechanisms (Sensor Alert Service, and Web Notification Services). The standards have been developed and implemented as part of the European Union FP6 project, "Sensors Anywhere" (SANY) [BBD+08].

The key components in the SANY architecture are the Sensor Observation Service (SOS) [NP07] and the Sensor Planning Service (SPS) [Sim07]. These provide Web service interfaces for retrieving sensor data and for issuing sensing tasks to sensor devices. However, the data to be retrieved from a sensor can only be filtered on certain attributes: it does not provide the full generality of a declarative query language. Likewise, the sensor planning service relies on a document description of the task rather than a declarative query.

There are three key differences with our approach. First, we allow users to *query* data, including the metadata stored in the registry, using declarative query languages. This goes beyond the simple `attribute op value` mechanisms available in the OGC-SWE standards. Second, we enable the *integration* of data from multiple sources through a single virtual data source. This goes beyond the simple merging of data available in OGC-SWE. Third, we will make extensive use of semantic technologies such as ontologies for the discovery and the integration of data. This enables users to discover data and query it using real world concepts such as "Solent", which refers to a coastal region on the South coast of England, rather than expecting users to provide only co-ordinates for points of interest. However, there is work to enable the use of ontologies on top of the OGC-SWE standards through annotation [SHS08] but not the use of declarative query languages or integration techniques.

Note, that since sensor data is already published using OGC-SWE services, we intend to offer a wrapper service based on the implementation of the Streaming Data Service using the Continuous Query Processor to enable this data to be available for query through the SemSorGrid4Env architecture.

## 2.6 Summary

This section has provided an overview of the SemSorGrid4Env architecture. The following two sections define the Web service interfaces, and the services that will expose these interfaces.

# 3. Interfaces

This section describes the interfaces that are made available by the Web services of the SemSorGrid4Env architecture. The interfaces define the functional properties that are exposed by the Web services. The combination of the functional properties and the metadata (also called non-functional properties within the Web services community) allow *service consumers*, which could be applications or other services, to interact with and distinguish between services. They, therefore, enable service consumers to make *well-formed and well-informed* calls to the interface. The description of the services, their functionality, their dependencies on other services, and which interfaces they make available are described in Section 4.

The terminology used to describe the interfaces is broadly based on the WSDL 2.0[1] approach [CMRW07]. However, we use the term *interface* to be a logical group of operations rather than all of the operations available from a Web service binding as the WSDL proposal does. Our approach allows us to combine interfaces to form a WSDL interface for a Web service rather than respecifying them for each service. In the remainder of this document we will qualify when we refer to a WSDL interface.

Each interface defines a logical set of operations, their types, and the faults that can be generated; these are specified in the following sections. The specification of the interfaces uses the "Pseudo-schemas" notational convention based on the following BNF-style notation to describe attributes and elements:

- '?' denotes an element that may appear zero or once;
- '*' denotes an element that may appear zero or more times;
- '+' denotes an element that may appear one or more times;
- '[' and']' denote a group of elements;
- '|' represents a choice.

The type definitions given in Table 3.1 are assumed for all the interface definitions. Note, when an interface defines a type to be an XML document, the XML schema for these documents is defined in the relevant WS-* standard. Other XML schema documents are available from their respective namespace URI.

Table 3.1: Common type definitions.

| **Types** | |
|---|---|
| EndpointReference | An address conforming to WS-Addressing [GHR06] that specifies the location of a resource as accessed through a Web service, and distinguishes between resources at that Web service location. |
| QName | A String that denotes an XML global element definition. |

## 3.1 Service Interface

The Service Interface allows the metadata associated with the service to be retrieved as a property document. The metadata for the service, captured in the property document, is expressed using semantic annotations as described below. Table 3.2 provides the specification of the interface.

[1]For those familiar with WSDL 1.1, PortType has been renamed to Interface and Port has been renamed to Endpoint.

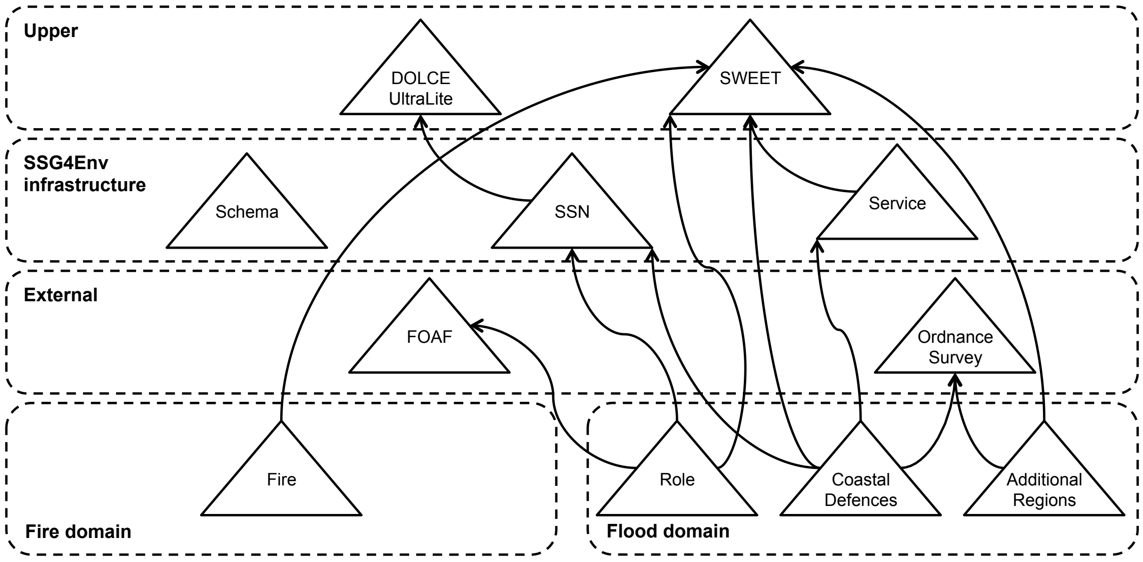


Figure 3.1: The SemSorGrid4Env ontology network for describing service and dataset metadata. Arrows indicate the reuse of ontologies.

Table 3.2: The Service Interface.

| **Types** | |
|---|---|
| PropertyDocument | An XML document that describes a Web service and its dataset(s). |
| **Faults** | |
| ResourceUnavailableFault | The resource is unavailable. |
| **Operations** | |
| GetResourcePropertyDocument | *Allows a requestor to retrieve the property document of the service.* |
| Inputs | |
| Output | resourcePropertyDocument: PropertyDocument |
| Faults | ResourceUnavailableFault |

## Service Metadata

Each service is associated with suitable *metadata* that can be retrieved as a property document through the Service Interface. The metadata contains information about the service e.g. its endpoint reference, supported interfaces, and the service provider, as well as details of the spatiotemporal and thematic coverage of the datasets made available by the service, e.g. the region covered by the data, the time range of the data, and properties such as tide height that are captured in the data.

We make use of semantic annotations, expressed in RDF/XML, to enable the use of ontologies to express the metadata associated with a service and the dataset(s) that it makes available. The network of ontologies used in the SemSorGrid4Env project for describing services and their metadata is depicted in Figure 3.1. The ontology network is composed of different ontologies that can be classified into different layers according to whether the ontology represents domain-specific information, information required for the SemSorGrid4Env services, or upper-level information used to facility interoperability among the other ontologies. For a complete description of the ontologies used, we refer the interested reader to Deliverable D4.3v1 [GCRRH+09] and Deliverable D4.3v2 which will be published in February 2011.

## 3.2 Registration Interface

The Registration Interface allows information pertaining to resources, i.e. service descriptions, ontologies, data set descriptions, etc, to be registered with the service that offers this interface. The registrations may subsequently be updated, to reflect changes in the properties of the registered resource, or deleted if it is no longer required.

Table 3.3: The Registration Interface.

| **Types** | |
|---|---|
| DescriptionDocument | An XML property document that conforms to some XML schema supported by the service (defined in the metadata) which describes a resource, e.g. a Web service, a data source, a sensor, a sensor network, an ontology, etc. |
| UpdateComponent | A pair consisting of a resource property name (QName) and an array of values to be inserted for the property. |
| DeleteComponent | A resource property name (QName). |
| **Faults** | |
| ResourceUnknownFault | The resource is unknown to the service. |
| InvalidResourcePropertyDocumentFault | The property document is not valid. |
| InvalidOntologyDocumentFault | The ontology document is not in a valid form that the service can process. |
| InvalidOntologyURI | The provided URI does not identify to an ontology. |
| UnacceptableTerminationTimeFault | The requested registration period cannot be satisfied by the service. |
| UnableToDeregisterFault | The resource was unable to deregister for some reason. |
| ServiceBusyFault | The service is currently unable to fulfill the request. |
| **Operations** | |
| Register | *Allows a requester to register a description document describing a resource.* |
| Inputs | description: DescriptionDocument,<br>validRegistrationTime?: timestamp:dateTime \|<br>duration:integer |
| Output | |
| Faults | InvalidResourcePropertyDocumentFault,<br>UnacceptableTerminationTimeFault,<br>ServiceBusyFault |
| *Renew (optional)* | *Allows a requester to renew the valid registration period of the specified resource.* |
| Inputs | identifier: EndpointReference,<br>validRegistrationTime?: Timestamp:dateTime \|<br>duration:integer |
| Output | |
| Faults | ResourceUnknownFault,<br>UnacceptableTerminationTimeFault,<br>ServiceBusyFault |
| *Update (optional)* | *Allows a requester to modify the properties registered about a resource.* |
| Inputs | resourceIdentifier: EndpointReference,<br>updates+: UpdateComponent[] |
| Output | |
| *Delete (optional)* | *Allows a requester to delete the property registered about a resource.* |
| Inputs | resourceIdentifier: EndpointReference,<br>deleteProperty: QName |
| Output | |
| Deregister | *Allows a requester to completely remove a registered resource.* |
| Inputs | identifier: EndpointReference |
| Output | |
| Faults | ResourceUnknownFault,<br>UnableToDeregisterFault,<br>ServiceBusyFault |

## 3.3 Discovery Interface

The Discovery Interface allows information pertaining to a resource, i.e. service descriptions, ontologies, data set descriptions, etc, known to the implementing service that offers this interface to be discovered according to their properties without the requirement of using a query.

Table 3.4: The Discovery Interface.

| **Types** | |
|---|---|
| DescriptionDocument | An XML property document that conforms to some XML schema supported by the service (defined in the metadata) which (partially) describes a resource, e.g. a data source, a sensor, a sensor network, etc. |
| **Faults** | |
| ResourceUnknownFault | The resource is unknown to the service. |
| InvalidResourcePropertyDocumentFault | The property document is not valid. |
| ServiceBusyFault | The service is currently unable to fulfill the request. |
| **Operations** | |
| Find | *Allows a requester to discover resources that have been registered that match a partial description document which describes the properties required from the returned resources.* |
| Inputs | description: DescriptionDocument |
| Output | resourceIdentifiers: EndPointReference[] |
| Faults | InvalidResourcePropertyDocumentFault, ServiceBusyFault |
| GetDetails | *Allows a requester to retrieve the description of a resource stored in the registry.* |
| Inputs | identifier: EndpointReference |
| Output | description: DescriptionDocument |
| Faults | ResourceUnknownFault, ServiceBusyFault |

## 3.4 Integration Interface

The Integration Interface provides the mechanisms necessary to access data from multiple resources and potentially to relate the data using mappings, e.g. $S_2O$ [CCG10]. No existing Web service standards exist for this functionality and the current best practice is OGSA-DQP [LMH+09] which only offers a view that consists of the union of relational schemas.

Note that the interface defined in Table 3.5 is consistent with the API offered by OGSA-DQP. As such, the interface can also be used to provide a distributed query processor (DQP) which unifies the schemas of its sources.

Table 3.5: The Integration Interface.

| **Types** | |
|---|---|
| DataResourceAbstractName | The abstract name associated with the data resource(s) represented as a URI (As defined in WS-DAI). |
| MappingDocument | An XML document consisting of a mapping format URI and the encoded mappings between the data sources and the resulting global schema. |
| **Faults** | |
| ServiceBusyFault | The service is already processing a message and concurrent operations are not supported. |
| NotAuthorizedFault | The consumer is not authorized to perform the requested operation or is not authorized to perform the requested operation at this time. |
| InvalidResourceNameFault | The data resource specified is unknown to the service. |
| DataResourceUnavailableFault | The data resource that is the target of the message is currently not available. This could be caused by a temporary fault or could indicate that the data resource has stopped operating permanently. |
| | *Continued overleaf* |

Table 3.5: The Integration Interface. (continued)

| | |
|---|---|
| InvalidMappingDocumentFault | The mapping document specified is not in a valid format or cannot be processed by the service. |
| **Operations** | |
| IntegrateAs | *Creates a data resource which presents the global view over a set of data sources.* |
| Inputs | resourceNames+: DataResourceAbstractName, mappings?: MappingDocument, integratedSourceName?: String |
| Output | integratedDataResource: EndpointReference |
| Faults | InvalidDataResourceNameFault, DataResourceUnavailableFault, InvalidMappingDocumentFault, NotAuthorizedFault, ServiceBusyFault |
| *AddSource (optional)* | *Add one or more data sources to the set of known data sources.* |
| Inputs | resourceNames+: DataResourceAbstractName |
| Output | |
| Faults | InvalidDataResourceNameFault, DataResourceUnavailableFault, NotAuthorizedFault, ServiceBusyFault |
| *RemoveSource (optional)* | *Remove one or more data sources from the set of known data sources.* |
| Inputs | resourceNames+: DataResourceAbstractName |
| Output | |
| Faults | InvalidDataResourceNameFault, DataResourceUnavailableFault, NotAuthorizedFault, ServiceBusyFault |

### Integrate As

The IntegrateAs operation takes an optional parameter of a MappingDocument. If the MappingDocument is omitted, then it is assumed that the resulting virtual resource view is the union of all the source schemas. If a MappingDocument is specified then the XML document consists of a URI specifying the format of the mapping document and the mappings encoded according to the format specified in the URI.

The optional integratedSourceName parameter enables the client to suggest a suitable name for the resulting integrated source.

## 3.5 Query Interface

The Query Interface provides operations to access data from a service using a formal query language to describe the information need. The operations provide two mechanisms for returning data: either directly in the response to the query request message, or indirectly through another data resource which must expose either the Data Access Interface (Section 3.6), or the Subscription (Section 3.7) and Subscription Manager (Section 3.9) Interfaces. The operations use the property documents as the mechanism by which to provide control specific for the data model being accessed, e.g. to control the stream delivery mechanism (push or pull) from a streaming data source.

The Query Interface adopted, and described in summary in Table 3.6, is that of the Web Services Data Access and Integration specification (WS-DAI) [AAK+06]. WS-DAI provides a model independent specification of service interfaces for accessing (querying and updating) data resources, such as a relational database, to support distributed data access. A WS-DAI service can make one or more data resources available including, for example, pre-existing databases. The specification provides details of the metadata and message patterns to be supported, i.e. how to retrieve the schema of a data resource, pass in a query in a formal language, and retrieve the results to the query.

The WS-DAI standard is further specialised by realisations for specific data models: WS-DAIR for relational data [ACK+06], WS-DAIX for XML data [AHK+06], WS-DAI-RDF(S) for RDF data [GGP08], and WS-DAI-Streaming for data streams [GGFP10]. These realisations provide mechanisms specific to the data model for querying, updating, and retrieving data that conform to the patterns defined in the WS-DAI specification. That is, the realisations of the WS-DAI query interface instantiate the GenericQuery and GenericQueryFactory operations with specific methods for the data model of the source. For example, WS-DAIR provides SQLExecute and SQLExecuteFactory as instantiations. Note, however, that the query language supported is not specified in the interface, but in the properties of the data resource, despite the names of the operations in the realisations.

Due to the adoption of the WS-DAI standard, any data source that is made available using an implementation of one of the WS-DAI realisations are consistent with this interface. For further details, we defer to the relevant standard document which also specifies some requirements for associated metadata.

Table 3.6: The Query Interface.

| | |
|---|---|
| **Types** | |
| DataResourceAbstractName | The abstract name associated with the data resource(s) represented as a URI. |
| DatasetFormatURI | The URI of a dataset format. |
| RequestDocument | An XML document that contains the request expression, i.e. the query and its parameters. |
| ResponseDocument | A document consisting of a dataset format URI and the encoded data. |
| ConfigurationDocument | An XML document containing the initial parameters for the indirect access resource. |
| PropertyDocument | An XML document conforming to a defined XML schema to describe the properties of the service. |
| **Faults** | |
| ServiceBusyFault | The service is already processing a message and concurrent operations are not supported. |
| NotAuthorizedFault | The consumer is not authorized to perform the requested operation or is not authorized to perform the requested operation at this time. |
| InvalidResourceNameFault | The data resource specified is unknown to the service. |
| DataResourceUnavailableFault | The data resource that is the target of the message is currently not available. |
| InvalidExpressionFault | The expression given as part of the request contains errors. |
| InvalidLanguageFault | The input dataset (usually the expression component of an incoming request) has an unrecognized language element. |
| InvalidDatasetFormatFault | The dataset format URI specified is not known to the service. |
| InvalidConfigurationDocumentFault | The configuration document specified is not valid. |
| **Operations** | |
| GetDataResourcePropertyDocument | *Returns the core property document values associated with the service implementing this message.* |
| Inputs | resourceName: DataResourceAbstractName |
| Output | properties: PropertyDocument |
| Faults | InvalidResourceName,<br>DataResourceUnavailableFault,<br>NotAuthorizedFault,<br>ServiceBusyFault |
| DestroyDataResource | *Destroy the named data resource; future messages directed at the resource MUST yield an InvalidResourceNameFault.* |
| Inputs | resourceName: DataResourceAbstractName |
| Output | |
| Faults | InvalidResourceName,<br>DataResourceUnavailableFault,<br>NotAuthorizedFault,<br>ServiceBusyFault |
| <GenericQuery> | *Directs a query document to a data resource.* |
| | *Continued overleaf* |

Table 3.6: The Query Interface. (continued)

| | |
|---|---|
| Inputs | resourceName: DataResourceAbstractName, format?: DatasetFormatURI, queryExpression: RequestDocument |
| Output | queryResponse: ResponseDocument |
| Faults | InvalidResourceNameFault, DataResourceUnavailableFault, InvalidDatasetFormatFault, InvalidExpressionFault, InvalidLanguageFault, NotAuthorizedFault, ServiceBusyFault |
| <GenericQueryFactory> | *Creates a relationship between a data resource representing the result of a query and the data access service by which it will be accessed.* |
| Inputs | resourceName: DataResourceAbstractName, responseInterfaceType?: QName, configurationDocument?: ConfigurationDocument, preferredTargetService?: EndPointReference, requestDocument: QueryExpressionDocument |
| Output | resourceList: DataResourceAddressList<EndPointReference> |
| Faults | InvalidResourceNameFault, DataResourceUnavailableFault, InvalidPortTypeQNameFault, InvalidConfigurationDocumentFault, InvalidExpressionFault, InvalidLanguageFault, NotAuthorizedFault, ServiceBusyFault |

### Generic Query Factory

The optional argument ResponseInterfaceType provided to the GenericQueryFactory operation declares the type of interface that should be used to retrieve the data. For example, for the relational realisation this is likely to take the value SQLResponse to indicate that the SQL response interface will be used, a specialisation of the Data Access Interface (Section 3.6). When querying a data stream, the ResponseInterfaceType argument takes on the added semantics of defining if the stream response will be delivered using a push-based mechanism, using the Subscription and Subscription Manager Interfaces (Sections 3.7 and 3.9), or pull-based mechanism using the WS-DAI-Streaming realisation of the Data Access Interface (Section 3.6). If the ResponseInterfaceType argument is omitted, then it is assumed to be contained in the configuration document provided with the query. For further details we refer to the WS-DAI-Streaming specification [GGFP10].

## 3.6 Data Access Interface

The Data Access Interface provides operations for retrieving and iterating over a dataset in a specific data model. Operations will need to be provided by realisations in specific data models, i.e. there will be different realisations for relations, RDF graphs, and data streams. The Data Access Interface uses the WS-DAI mechanisms for data access [AAK+06] and the corresponding realisations: WS-DAIR [ACK+06], WS-DAIX [AHK+06], WS-DAI-RDF(S) [GGP08], and WS-DAI-Streaming [GGFP10].

It should be possible to use the Data Access Interface to operate over data regardless of whether or not it has been generated by a Query Interface interaction, i.e. a dataset may be published that is not the result of a query. Thus, in SemSorGrid4Env we are using this interface in a more generic way than in WS-DAI: It can be used for data sources that were not created by a query. Such a data source may be discoverable through the Registry Service (Section 4.2.1).

The operations defined for the Data Access Interface can be extended with specific messages for the underlying data model, e.g. relational. As an example, WS-DAIR extends this interface with the following operations: GetSQLRowset; GetSQLUpdateCount; GetSQLReturnValue; GetSQLOutputParameter; and GetSQLCommunicationsArea.

Table 3.7: The Data Access Interface.

| | |
|---|---|
| **Types** | |
| DataResourceAbstractName | The abstract name associated with the data resource(s) represented as a URI. |
| DatasetFormatURI | The URI of a dataset format. |
| ResponseDocument | A document consisting of a dataset format URI and the encoded data. |
| PropertyDocument | An XML document conforming to a defined XML schema to describe the properties of the service. |
| **Faults** | |
| ServiceBusyFault | The service is already processing a message and concurrent operations are not supported. |
| NotAuthorizedFault | The consumer is not authorized to perform the requested operation or is not authorized to perform the requested operation at this time. |
| InvalidResourceNameFault | The data resource specified is unknown to the service. |
| DataResourceUnavailableFault | The data resource that is the target of the message is currently not available. |
| **Operations** | |
| GetPropertyDocument | *Retrieves the PropertyDocument for the resource.* |
| Inputs | resourceName: DataResourceAbstractName |
| Output | properties: PropertyDocument |
| Faults | InvalidResourceNameFault, DataResourceUnavailableFault, NotAuthorizedFault, ServiceBusyFault |
| <GetResponseItem> | *Return a specified number of data items.* |
| Inputs | resourceName: DataResourceAbstractName, format: DatasetFormatURI?: URI, position: integer (first item is position 0), count?: integer (0 returns all items) |
| Output | dataset: ResponseDocument |
| Faults | InvalidResourceName, DataResourceUnavailableFault, InvalidDatasetFormatFault, NotAuthorizedFault, InvalidPositionFault, InvalidCountFault, ServiceBusyFault |

## 3.7 Subscription Interface

The Subscription Interface provides the mechanism by which a requester may explicitly subscribe to receive notifications whenever an event occurs. The Web service standard WS-BaseNotification (WS-N) [GHME06] has been adopted for this purpose. This interface corresponds to the NotificationProducer Interface of that standard. The types, faults, and operations of the WS-BaseNotification NotificationProducer interface are summarised in Table 3.8. For further details consult the relevant section in [GHME06].

It is anticipated that a subscription resource will be created when a query is posed using the GenericQueryFactory operation (Section 3.5). Such a subscription resource need not support any mechanisms for further filtering the query results and need not specify any topics. Consumers can use the Subscribe operation to request notification messages from resources that were created by some other user (see the interaction between the Integrator and DQP services in Figure 5.7 for an example of this interaction pattern). The specification of filters and topics in Table 3.8 are stated for compliance with the WS-N standard.

Table 3.8: The Subscription Interface.

| **Types** | |
|---|---|
| Filter | A collection of expressions that evaluate which topics to permit. |
| Policy | A collection of policy statements. |
| NotificationMessage | A notification message as defined in the Notification Interface (Section 3.10). |
| **Faults** | |
| ResourceUnknownFault | The resource is unknown to the service. |
| InvalidFilterFault | The filter was not understood or is not supported by the service. |
| TopicExpressionDialectUnknownFault | The dialect of the topic expression dialect in the filter is unknown to the service. |
| InvalidTopicExpressionFault | The topic expression in the filter is not valid. |
| TopicNotSupportedFault | The topic expression in the filter contains a topic that is not supported. |
| InvalidProducerPropertiesExpressionFault | The filter contained a property expression that is not valid. |
| InvalidMessageContentExpressionFault | The filter contained a message content filter that is not valid. |
| UnrecognizedPolicyRequestFault | The service does not recognize one or more policy request. |
| UnsupportedPolicyRequestFault | The service doe not support one or more of the policy requests. |
| NotifyMessageNotSupportedFault | The service does not support the notify wrapper. |
| UnacceptableTerminationTimeFault | The specified termination time was not acceptable to the service. |
| SubscribeCreationFailedFault | The service failed to process the Subscribe message. |
| MultipleTopicsSpecifiedFault | More than one topic was referenced. |
| NoCurrentMessageOnTopicFault | There are no messages for the referenced topic. |
| **Operations** | |
| Subscribe | *Registers a consumer to receive a subset of the notification messages sent by the service.* |
| Inputs | consumerReference: EndpointReference,<br>filter?: Filter,<br>initialTerminationTime?: dateTime \| duration,<br>subscriptionPolicy?: Policy |
| Output | subscriptionReference: EndPointReference,<br>currentTime?: dateTime,<br>terminationTime?: dateTime |
| Faults | ResourceUnknownFault,<br>InvalidFilterFault,<br>TopicExpressionDialectUnknownFault,<br>InvalidTopicExpressionFault,<br>TopicNotSupportedFault,<br>InvalidProducerPropertiesExpressionFault,<br>InvalidMessageContentExpressionFault,<br>UnrecognizedPolicyRequestFault,<br>UnsupportedPolicyRequestFault,<br>UnacceptableInitialTerminationTimeFault,<br>NotifyMessageNotSupportedFault,<br>SubscribeCreationFailedFault |
| GetCurrentMessage | *Allows a newly registered consumer to retrieve the most recent notify message.* |
| Inputs | topicName: QName |
| Output | message: NotificationMessage |
| Faults | ResourceUnknownFault,<br>TopicExpressionDialectUnknownFault ,<br>InvalidTopicExpressionFault,<br>TopicNotSupportedFault,<br>MultipleTopicsSpecifiedFault,<br>NoCurrentMessageOnTopicFault |

## 3.8 Pull Point Interface

The Pull Point Interface is intended to support consumers of notification messages that are unable to provide an endpoint for receiving notification messages directly and is part of the WS-BaseNotification standard [GHME06]. A service offering the Pull Point Interface can be set up to receive notification messages on behalf of a consumer and from which the consumer can later retrieve the messages or destroy the resource. Notification messages are accumulated on a best effort basis: the implementing service may ignore new messages or it may dispose of old messages in order to receive a new message. The mechanism employed by the service should be expressed in the associated metadata.

Note that a service exposing the Pull Point Interface needs to provide the Notify Interface (Section 3.10) in order to receive notification messages.

Table 3.9: The Pull Point Interface.

| **Types** | |
|---|---|
| NotificationMessage | A notification message as defined in the Notification Interface (Section 3.10). |
| **Faults** | |
| UnableToCreatePullPointFault | It was not possible to create the pull point resource for some reason. |
| ResourceUnknownFault | The resource is unknown to the service. |
| UnableToGetMessagesFault | The service was unable to return messages for some reason. |
| UnableToDestroyPullPointFault | The service is unable to destroy the pull point for some reason. |
| **Operations** | |
| CreatePullPoint | *Allows a requester to create a PullPoint resource.* |
| Inputs | |
| Output | pullPoint: EndpointReference |
| Faults | UnableToCreatePullPointFault |
| GetMessages | *Allows a requester to retrieve notification messages from the Pull Point.* |
| Inputs | maximumNumber?: Integer |
| Output | messages*: NotificationMessages |
| Faults | ResourceUnknownFault, UnableToGetMessagesFault |
| DestroyPullPoint | *Allows a requestor to terminate the Pull Point resource.* |
| Inputs | |
| Output | |
| Faults | ResourceUnknownFault, UnableToDestroyPullPointFault |

## 3.9 Subscription Manager Interface

The Subscription Manager Interface allows a consumer of notification messages to manage their subscription for receiving messages about events with certain characteristics. These operations enable the consumer to alter the filter condition, update the length of time that the subscription is valid, or to end the subscription. The interface definition is taken from the WS-BaseNotification standard [GHME06].

Table 3.10: The Subscription Manager Interface.

| **Types** | |
|---|---|
| **Faults** | |
| ResourceUnknownFault | The resource is unknown to the service. |
| UnacceptableTerminationTimeFault | The specified termination time was not acceptable to the service. |
| UnableToDestroySubscriptionFault | The service was unable to destroy the Subscription for some reason. |
| **Operations** | |
| Renew | *Allows a requester to modify the current lifetime of a subscription.* |
| Inputs | terminationTime: dateTime \| duration |
| Output | terminationTime: dateTime, currentTime?: dateTime |
| Faults | ResourceUnknownFault, UnacceptableTerminationTimeFault |
| Unsubscribe | *Allows a requester to terminate a subscription.* |

*Continued overleaf*

Table 3.10: The Subscription Manager Interface. (continued)

| | |
|---|---|
| Inputs | |
| Output | |
| Faults | ResourceUnknownFault, UnableToDestroySubscriptionFault |

## 3.10 Notification Interface

The Notification Interface allows consumers to receive notification messages in one of two forms: “raw” (application specific) or WS-BaseNotification Notify message. The interface corresponds to the NotificationConsumer Interface in the WS-BaseNotification standard [GHME06].

Table 3.11: The Notification Interface.

| | |
|---|---|
| **Types** | |
| NotificationMessage | Contains the Endpoint Reference for the subscriber and the producer, a QName for the topic of the message, and an XML element for the message body. |
| **Faults** | |
| **Operations** | |
| *Notify (optional)* | *A message containing one or more notifications.* |
| Inputs | notifications+: NotificationMessage |

The notify message does not expect a response and none is defined. Note, other operations expect a response with an empty body, e.g. the Unsubscribe message of the Subscription Manager interface. Also note that there are no faults associated with the Notify message.

The notify message is optional since the consumer of notification messages may expose an EndpointReference for the receipt of “raw” messages.

# 4. Services

This section defines the Web services offered by the SemSorGrid4Env architecture. The services are defined in terms of their dependencies on other services, and the functionality that they provide by the interfaces (Section 3) that they offer. This is similar to the *binding* that occurs in WSDL [CMRW07].

## 4.1 Application Tier Services

The Application Tier offers services that support the applications and provide the high-level Application Programming Interfaces (APIs) [dRSPM09, PSK+09] for accessing sensor network components and associated data for presentation and interaction with end-users. Due to the domain specific nature of application tier services, in this document we limit the scope to those services that provide REST style access to the Semantic Registration and Discovery Service (Section 4.2.1) and the Semantic Integration and Query Service (Section 4.2.2).

### 4.1.1 Registry Access Service

The Registry Access Service enables Web applications to access the SOA style Registration and Discovery Service (Section 4.2.1) to discover resources using a REST interface, i.e. using the HTTP operations `GET`, `POST`, `PUT`, and `DELETE`. The service supports applications in posing a query using HTTP `GET` and returning results as Web resources encoded in some format agreed through content negotiated with the client, e.g. JSON, XML, or HTML.

#### Required Services

The Registry Access Service relies on:

- The Registration and Discovery Service providing the Service Interface (Section 3.1) in order to retrieve the property document describing the registry service;
- The Registration and Discovery Service exposing the Query Interface (Section 3.5) in order to send queries to the service;
- The security mechanisms provided by the underlying infrastructure in order to allow users to authenticate themselves and to ensure that they are only able to discover resources that they are authorized to use.

#### Exposed Interfaces

The Registry Access Service exposes a REST interface.

### 4.1.2 Integrator Access Service

The Integrator Access Service enables Web applications to access the SOA style Integration and Query Service (Section 4.2.1) to enable ontology-based access to integrated data services through a REST interface, i.e. using the HTTP operations `GET`, `POST`, `PUT`, and `DELETE`. The service supports applications in posing a query over an ontological view and periodically retrieving the latest answers to the query. Results are returned as a Web resource encoded in some format agreed through content negotiated with the client, e.g. JSON, XML, or HTML.

**Required Services**

The Integrator Access Service relies on:

- The Integration and Query Service providing the Service Interface (Section 3.1) in order to retrieve the property document describing the integrated resources available at the integration service;
- The Integration and Query Service exposing the Query Interface (Section 3.5) in order to send queries to the service;
- The Integration and Query Service exposing the Data Access Interace and/or the Subscription and Subscription Manager Interfaces to retrieve data values;
- The security mechanisms provided by the underlying infrastructure in order to allow users to authenticate themselves and to ensure that they are only able to discover resources that they are authorized to use.

**Exposed Interfaces**

The Integrator Access Service exposes a REST interface.

## 4.2 Middleware Tier Services

The Middleware Tier of the SemSorGrid4Env architecture provides applications and services a value-added mechanism to interact with data sources. Two types of services will be provided: a Registration and Discovery Service, and an Integration and Query Processing Service. The services of the middleware tier will make extensive use of semantic annotations, provided in formal languages, about the data models in which data sources publish their data. The semantics of the data models of published data sources can be used by automatic reasoners for the purposes of discovery and integration of services and data.

### 4.2.1 Semantic Registration and Discovery Service

The Semantic Registration and Discovery Service, also referred to as the Registry Service or Semantic Registry Service, allows the registration of various resources (e.g. data sources, ontologies, mappings, services, sensors, sensor networks, etc.), which can then be discovered for use in orchestrations by applications and other services. The Semantic Registration and Discovery Service can be seen as a storage and querying service that can be accessed by prospective clients who want to register some resources (e.g. register a data source that provides climatic data from sensors that are located in Spain) or discover resources based on their properties (e.g. discover the data sources that provides access to climatic data).

**Required Services**

The Semantic Registration and Discovery Service relies on:

- Services and resources being able to provide descriptions of themselves in a well defined manner based on the Service Interface (Section 3.1);

- Services and applications potentially exposing the Notification Interface (Section 3.10). This would enable the service or application to receive notification messages when a resource is added or removed from the registry, for example when a new sensor network is registered;
- The security mechanisms provided by the underlying infrastructure in order to allow users to authenticate themselves and to ensure that they are only able to discover resources that they are authorized to use.

**Exposed Interfaces**

The interfaces exposed by the Semantic Registration and Discovery Service are shown in Figure 4.1 and explained below:

**Service Interface:** Enables clients to make well-formed and well-informed interactions with the Registry (Section 3.1);

**Registration Interface:** Enables resources to register a description of themselves and to later renew, update, delete, or deregister that description (Section 3.2);

**Discovery Interface *(optional)*:** Enables clients to discover resources which have certain properties and to retrieve the full description stored in the registry for a specific resource (Section 3.3);

**Query Interface *(optional)*:** Enables clients to query the contents of the Registry in order to discover relevant resources (Section 3.5);

**Data Access Interface *(optional)*:** Enables clients to use an indirect mechanism to retrieve the results of their query (Section 3.6);

**Subscription Interface *(optional)*:** Enables clients to register their interest to receive notifications about new resources with certain characteristics (Section 3.7);

**Subscription Manager Interface *(optional)*:** Enables clients to renew or cancel their subscription to notifications (Section 3.9).

The Discovery and Query Interfaces are optional, although at least one should be provided to enable the functionality to discover *relevant* resources. The Data Access Interface need not be offered even if the Query Interface is exposed since it is possible to get query results directly from the Query Interface. The Subscription and Subscription Manager Interfaces must be provided as a pair, but are only needed if the Registry Service provides support for clients to be notified of new resources.

Details of the Semantic Registration and Discovery Service implementation can be found in [KKK10, KK10].

### 4.2.2 Semantic Integration and Query Service

The Semantic Integration and Query Processing Service, also referred to as the Semantic Integrator Service, will provide mechanisms to create a “virtual” data source. The virtual data source will present a mediated global schema, or ontological view, over the data offered by one or more data sources using semantic technology and annotations. Queries over the global schema of a virtual data source will be rewritten into a set of queries over the local schemas of the data sources, and the results combined to provide the query answer.

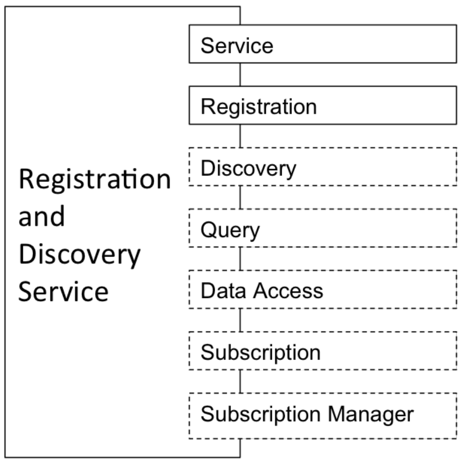


Figure 4.1: Interfaces exposed by the Semantic Registration and Discovery Service

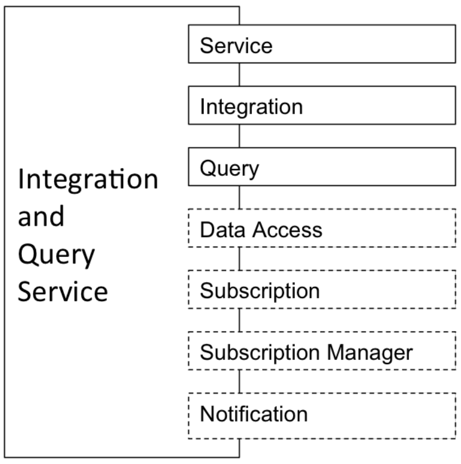


Figure 4.2: Interfaces exposed by the Semantic Integration and Query Service

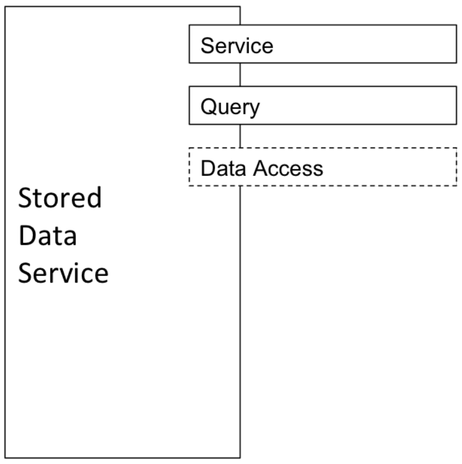


Figure 4.3: Interfaces exposed by the Stored Data Service

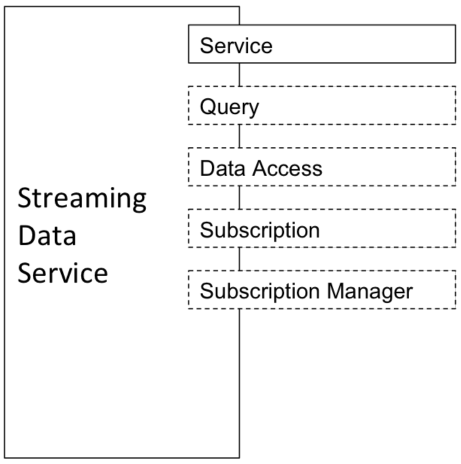


Figure 4.4: Interfaces exposed by the Stream Data Service

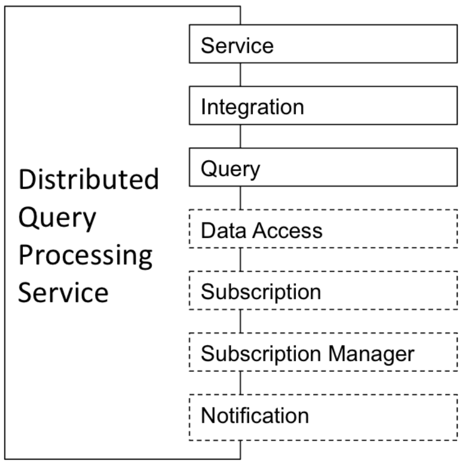


Figure 4.5: Interfaces exposed by the Distributed Query Processing Service

**Required Services**

The Semantic Integration and Query Service requires the following functionality from other services and the underlying infrastructure:

- Data sources providing details of the functional properties and their metadata as defined by the Service Interface (Section 3.1);
- Data sources providing descriptions of their data as defined by the metadata associated with the Query Interface (Section 3.5);
- The ability to query and access data from the underlying data sources using the Query Interface (Section 3.5) and potentially the Data Access Interface (Section 3.6);
- The ability to receive streams of data from the underlying data sources, either using a pull-based mechanism using the Data Access Interface (Section 3.6) or a push-based mechanism using the Subscription Interface (Section 3.7);
- The security mechanisms of the underlying infrastructure in order to access the underlying data sources and to only permit users to access data that they are authorized to use.

**Exposed Interfaces**

The interfaces exposed by the Semantic Integration and Query Service are shown in Figure 4.2 and explained below:

**Service Interface:** Enables clients to make well-formed and well-informed interactions with the Integrator (Section 3.1);

**Integration Interface:** Enables clients to create a virtual data source exposing a global schema that combines the data from one or more data source (Section 3.4);

**Query Interface:** Enables clients to pose a query over a virtual data source, i.e. one that integrates the data from one or more data sources and that has been created through the Integration Interface (Section 3.5);

**Data Access Interface *(optional)*:** Enables clients to retrieve data, for example the results of a query, or to use a pull-based mechanism to retrieve a stream (Section 3.6);

**Subscription Interface *(optional)*:** Enables clients to receive the results of a query over a data stream using the push-based delivery mechanism (Section 3.7);

**Subscription Manager Interface *(optional)*:** Enables clients to renew or cancel their subscription to a push-based delivery stream (Section 3.9);

**Notification Interface *(optional)*:** Enables the Integration Service to receive a push-based delivery stream from a data source (Section 3.10).

If the Integration Service supports queries over streams, then it must expose either the Data Access Interface to support pull-based retrieval of streams, and/or the Subscription Interface to support push-based delivery of streams. If the Subscription Interface is exposed then the Subscription Manager Interface must also be exposed.

Details of the Semantic Integration and Query Service implementation can be found in [CCG10].

## 4.3 Data Tier Services

The Data Tier of the SemSorGrid4Env architecture provides applications and services mechanisms to query and retrieve data from data sources which have been virtualized using query-processing technology. There are two main classes of data sources: stored data sources, such as relational databases, and streaming data sources, such as sensor networks. Note, however, that it is possible to access data of both types without first needing to pose a query using an iterator exposed by the Data Access Interface (Section 3.6).

The objectives of the data tier services are to provide data access services to data consumer services in a virtualized manner, so as to reconcile any heterogeneity other than semantic ones. In this respect, a data service acts as a wrapper over a concretely implemented source of data and exposes interfaces that enable every other source of data that is similarly wrapped to be interacted with in a uniform manner.

### 4.3.1 Stored Data Service

Stored data sources may comprise structured data (e.g. relational tuple-sets) or semi-structured data (e.g. XML documents or RDF graphs). For sensor networks, if the sensed data rests in persistent stored extents and if its temporal aspect is not relevant, then the same virtualization mechanisms apply.

#### Required Services

The Stored Data Service relies on the following functionality from the underlying infrastructure:

- The ability of the underlying infrastructure to enable users to authenticate themselves so that access to the data can be restricted to those who are authorized.

#### Exposed Interfaces

The interfaces exposed by the Stored Data Service are shown in Figure 4.3 and explained below:

**Service Interface:** Enables clients to make well-formed and well-informed interactions with the Stored Data Service (Section 3.1);

**Query Interface:** Enables clients to pose a query over the data source (Section 3.5);

**Data Access Interface *(optional)*:** Enables clients to use an indirect mechanism to retrieve the results of a query (Section 3.6).

Deployments of the stored data service have used the corresponding WS-DAI reference implementation [ACK$^+$06, GGP08].

### 4.3.2 Streaming Data Service

Two classes of streaming data sources are considered:

1. Ones in which the consumer has control over the generation of tuples at the stream source, as is the case with sensor networks which support in-network processing, and whose functionality is determined by the query, i.e. the query specifies the data that is to be collected by the sensor network; and
2. Streaming data sources which emit a pre-determined data stream, in which case the consumer has no control over the generation of tuples at the stream source, and may only pose queries over the data available on the stream. Examples of this type of streaming data source include classical data streams (e.g. a stock-exchange real-time pricing feed) or streams generated by sensor-networks whose functionality is fixed.

Both types of data stream can be queried using a continuous query which is repeatedly evaluated, resulting in a stream of tuples. There are two mechanisms by which a Streaming Data Service may deliver answer streams:

1. Push-based streams in which the data service sends notification messages using the Notification Interface (Section 3.10) as answer tuples are generated, or;
2. Pull-based streams in which the data service stores the answer tuples in a queue that is accessible to the consumer upon request through the Data Access Interface (Section 3.6).

Intuitively, streams lend themselves by default to the push-based approach. Note, a data service may offer both kinds of data delivery mechanism and the client can decide which "best" suits their needs.

#### Required Services

The Streaming Data Service relies on the following functionality from other services and the underlying infrastructure:

- The ability of consumers of push-based streams to be able to receive notification messages by exposing the Notification Interface (Section 3.10);
- The ability of the underlying infrastructure to enable users to authenticate themselves so that access to the data can be restricted to those who are authorized.

#### Exposed Interfaces

The interfaces exposed by the Streaming Data Service are shown in Figure 4.4 and explained below:

**Service Interface:** Enables clients to make well-formed and well-informed interactions with the Streaming Data Service (Section 3.1);

**Query Interface *(optional)*:** Enables clients to pose a query over the data source (Section 3.5);

**Data Access Interface *(optional)*:** Enables clients to use a pull-based mechanism to receive the results of a query (Section 3.6);

**Subscription Interface *(optional)*:** Enables clients to use a push-based mechanism to receive the results of a query (Section 3.7);

**Subscription Manager Interface *(optional)*:** Enables clients to renew or cancel their subscription to a push-based delivery stream. (Section 3.9).

An implementation of the Streaming Data Service, that is able to expose either push-based or pull-based data streams, is described in Deliverable D2.2v1 [GGR+09].

Note that the Query Interface is optional to enable the publication of data streams where there is no mechanism for query, only retrieval of the data. An example of such a service could be a wrapper for an OGC-SWE Sensor Observation Service which is only capable of data access.

### 4.3.3 Distributed Query Processing Service

The Distributed Query Processing Service provides mechanisms to pose queries over a set of distributed data sources as a single "virtual" data source. The virtual data source presents a unified schema of the data sources, and provides query-based access to distributed data services which themselves may only provide the Data Access Interface, or Subscription and Subscription Manager Interfaces. Unlike the Semantic Integration and Query Service, the Distributed Query Processing Service does not resolve semantic heterogeneity between the data sources nor does it attempt to integrate the schemas. Note that the Distributed Query Processing Service does not require query-based access to its data sources. As such, it can be used by the Semantic Integration and Query Service, or any other service or application, as a mechanism to query data sources that do not expose the Query Interface.

#### Required Services

The Distributed Query Processing Service requires the following functionality from other services and the underlying infrastructure:

- Data sources providing details of the functional properties and their metadata as defined by the Service Interface (Section 3.1);
- Data sources providing descriptions of their data as defined by the metadata associated with the Query Interface (Section 3.5);
- The ability to access data from the underlying data sources either through the Query Interface (Section 3.5) or the Data Access Interface (Section 3.6);
- The ability to receive streams of data from the underlying data sources, either using a pull-based mechanism using the Data Access Interface (Section 3.6) or a push-based mechanism using the Subscription Interface (Section 3.7);
- The security mechanisms of the underlying infrastructure in order to access the underlying data sources and to only permit users to access data that they are authorized to use.

#### Exposed Interfaces

The interfaces exposed by the Distributed Query Processing Service are shown in Figure 4.5 and explained below:

**Service Interface:** Enables clients to make well-formed and well-informed interactions with the Distributed Query Processing Service (Section 3.1);

**Integration Interface:** Enables clients to create a virtual data source exposing a unified schema from one or more data source and to add and removed sources (Section 3.4);

**Query Interface:** Enables clients to pose a query over a virtual data source, i.e. one that unifies the data from one or more data sources and that has been created through the Integration Interface (Section 3.5);

**Data Access Interface *(optional)*:** Enables clients to retrieve data, for example the results of a query, or to use a pull-based mechanism to retrieve a stream (Section 3.6);

**Subscription Interface *(optional)*:** Enables clients to receive the results of a query over a data stream using the push-based delivery mechanism (Section 3.7);

**Subscription Manager Interface *(optional)*:** Enables clients to renew or cancel their subscription to a push-based delivery stream (Section 3.9);

**Notification Interface *(optional)*:** Enables the Integration Service to receive a push-based delivery stream from a data source (Section 3.10).

If the Distributed Query Processing Service supports queries over streams, then it must expose either the Data Access Interface to support pull-based retrieval of streams, and/or the Subscription Interface to support push-based delivery of streams. If the Subscription Interface is exposed then the Subscription Manager Interface must also be exposed.

Details of the Distributed Query Processing Service implementation for streaming and stored data sources can be found in [GGR+09]. An alternative implementation for stored relational data sources is provided by OGSA-DQP [LMH+09].

### 4.3.4 Notification Intermediary Service

The Notification Intermediary Service provides a mechanism by which a notification consumer which is unable to directly receive notification messages, e.g. an AJAX Web application running in a browser, can interact with a data source that can only send notification messages, e.g. a sensor network that only supports push-based delivery of streams. The Notification Intermediary Service provides a Notification Endpoint reference for receiving notification messages and a Pull Point Interface (Section 3.8) for the retrieval of the messages. This is a standard mechanism provided by the WS-BaseNotification Standard [GHME06].

#### Required Services

The Notification Intermediary Service does not rely on any other services.

#### Exposed Interfaces

The interfaces exposed by the Notification Intermediary Service are:

**Service Interface:** Enables clients to make well-formed and well-informed interactions with the service. Specifically, it allows clients to discover the services policy on receiving more messages than it can store;

**Notification Interface:** Enables the Notification Intermediary Service to receive notification messages;

**Pull Point Interface:** Enables clients to retrieve notification messages from the service.

# 5. Sample Interaction

This section provides a sample interaction with the SemSorGrid4Env architecture drawn from the flood emergency response planning scenario of work package 7. The scenario is presented in terms of interactions between services which combined together deliver the functionality required by the flood emergency response planner—the user.

## 5.1 Assumptions

The scenario described below makes some assumptions. First, it assumes the deployment of a semantic registry service at a well known location. Second, it assumes the deployment of several data services, and that these have already registered their semantic property documents with the registry service. Third, it assumes the deployment of several services, e.g. a distributed query service, an integration service, and application services, which have already registered their semantic property documents with the registry service. Finally, it assumes that a flood event has been characterised as a query over relevant data sources, and this characterisation has been registered with an application service capable of generating alerts. When the service detects that the conditions are met, it sends an SMS text message or email to the relevant flood response manager.

## 5.2 Flood Emergency Response Scenario Interaction

Upon receiving an alert about an imminent flooding event by text or email, the flood response manager accesses the Web application through the login screen shown in Figure 5.1. When logging into the application, the flood manager selects their role, the region of coast they are responsible for, and the task that they wish to conduct. Note, the options in the drop down selection boxes are populated with terms from the ontology network (Figure 3.1), e.g. the choice of role comes from the concepts of the Role ontology. This process provides an initial definition of the data that is relevant for the current interaction. That is, the values provided by the user in the login process parameterise the queries sent to the registry to discover relevant data sources. For the login selection shown in Figure 5.1, the registry query is parameterised to discover data sources for the Portsmouth area of the Solent for a Coastal Zone Manager who wishes to monitor the current status.

When the login button, hidden in the screenshot by the drop down menu, is pressed several queries are issued to the registry service to discover relevant data sources. The interaction between the Web Application, Application Services, and the Registry Service are shown in Figure 5.3. The Web Application accesses the REST interface of the Application Services to query the Registry Service. The HTTP GET call is parameterised with an stSPARQL query describing the data sources to be returned, the resource name of the registry service, and the format of the data results. In this case, the Web Application expects a JSON array of endpoint references to be returned. The Application Services makes the appropriate SOAP call to the Registry service (`SPARQLExecute`) and returns the endpoint references to the Web Application.

The result of the login process is shown in Figure 5.4. The flood response manager is presented with two map views based on the region selected in the login process, viz. Portsmouth. On the left is a zoomed out map to provide context for the region of interest. On the right is a zoomed in map on the region for which the manager is responsible. Both maps have been superimposed with layers presenting the data from a variety of data sources which satisfy the queries sent to the Registry Service by the Application Services through which the Web Application interacts. The available layers are shown in the Map Layers pop-up window, from which the user can select the layers they wish to be displayed.

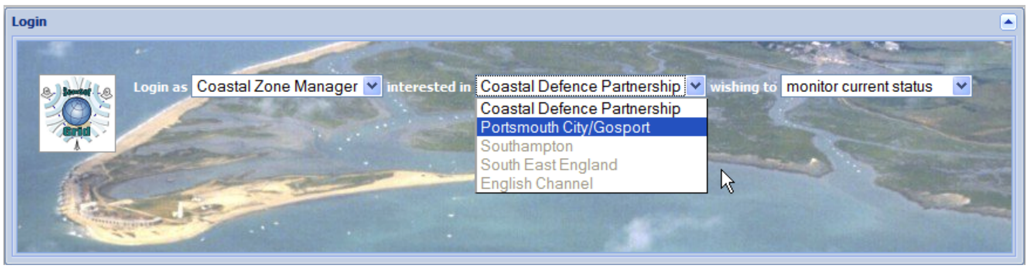


Figure 5.1: Screenshot of the login screen from the flood emergency response Web application.

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix Services: <http://www.semsorgrid4env.eu/ontologies/ServiceOntology.owl#>
prefix RegistryOntology:
  <http://www.semsorgrid4env.eu/ontologies/RegistryOntology.owl#>
prefix CoastalDefences:
  <http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#>
prefix time: <http://sweet.jpl.nasa.gov/2.0/timeExtent.owl#>
prefix AdditionalRegions:
  <http://www.ordnancesurvey.co.uk/ontology/v1/AdditionalRegions#>
prefix OSHydrology:
  <http://www.ordnancesurvey.co.uk/ontology/Hydrology/v2.0/Hydrology.owl#>
select distinct ?ENDPOINT
where {
  ?SERVICE rdf:type Services:WebService .
  ?SERVICE Services:hasEndpointReference ?ENDPOINT .
  ?SERVICE Services:hasServiceType Services:StreamingDataService .
  ?SERVICE Services:hasDataset ?DATASET .
  ?DATASET time:hasTemporalExtent ?TIME .
  filter(?TIME contains "[NOW,NOW-12]"^^RegistryOntology:TemporalInterval) .
  ?DATASET Services:includesPropertyType ?PROPERTYTYPE .
  filter((str(?PROPERTYTYPE) =
      "http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#TideHeight") ||
    (str(?PROPERTYTYPE) =
      "http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#WaveHeight")).
  ?DATASET Services:coversRegion ?SERVICEREGION .
  ?SERVICEREGION RegistryOntology:hasGeometry ?SERVICEREGIONGEO .
  AdditionalRegions:SolentModelledArea RegistryOntology:hasGeometry ?SOLENTGEO .
  filter(?SERVICEREGIONGEO covers ?SOLENTGEO) .
}
```

Figure 5.2: Sample stSPARQL query sent by the Web Application to the Registry Service to discover the endpoint reference for streaming data services containing tide and wave height measurements over the last 12 hours for the Solent region of the south coast of England.

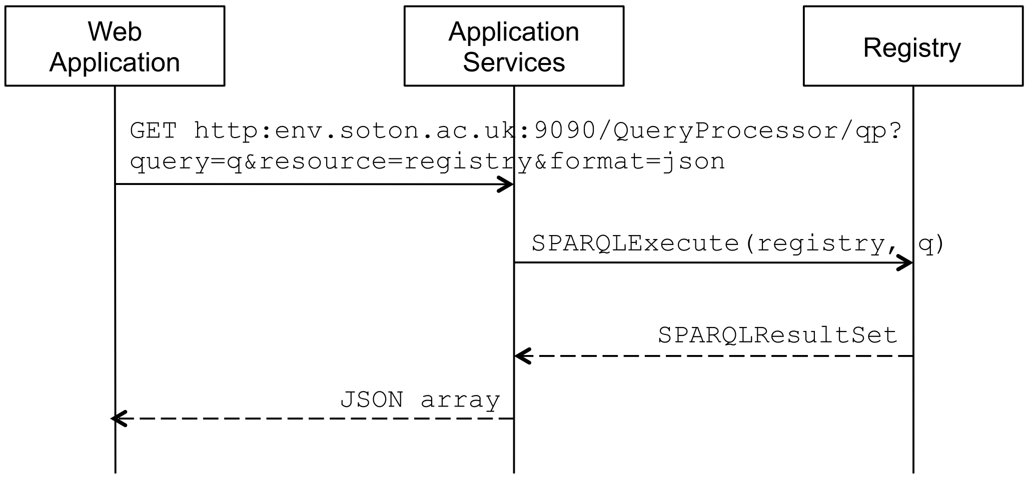

Web Application
Application Services
Registry
GET http:env.soton.ac.uk:9090/QueryProcessor/qp?query=q&resource=registry&format=json
SPARQLExecute(registry, q)
SPARQLResultSet
JSON array

Figure 5.3: Interaction between the Web Application, Application Services, and the Registry Service to discover relevant data sources. A sample query *q* is shown in Figure 5.2.

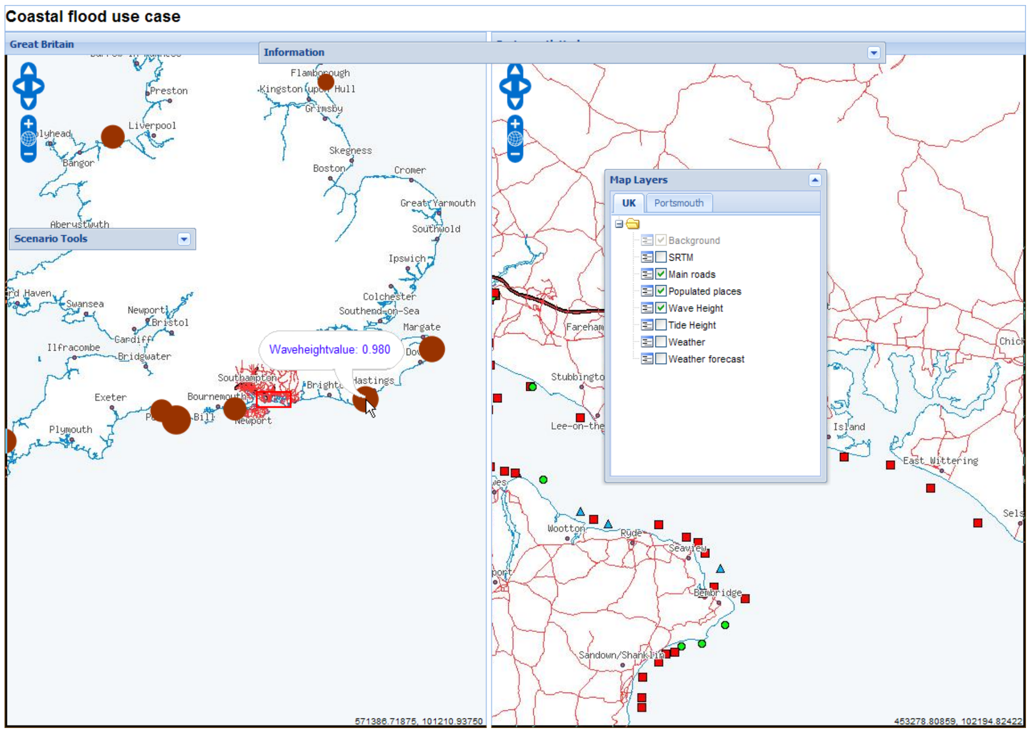
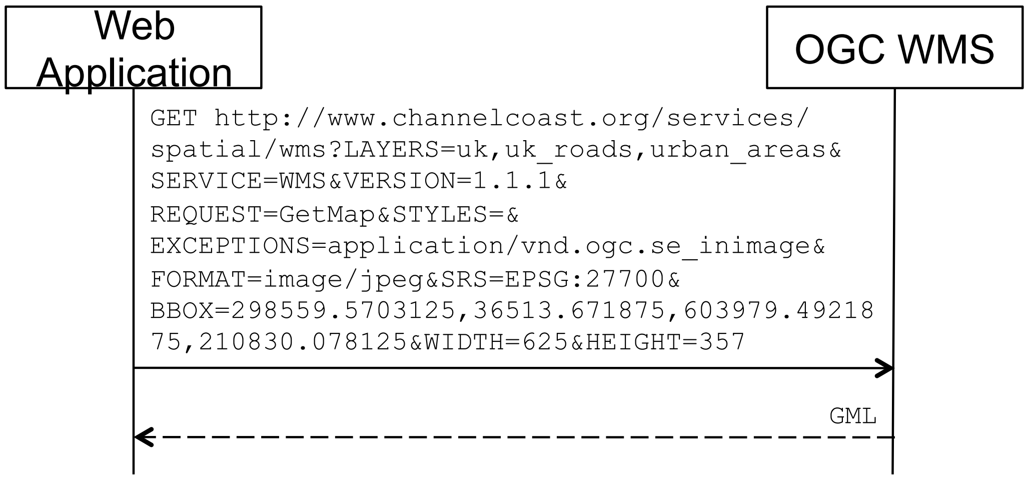


Figure 5.4: Screenshot of the flood emergency response Web application.



```
GET http://www.channelcoast.org/services/
spatial/wms?LAYERS=uk,uk_roads,urban_areas&
SERVICE=WMS&VERSION=1.1.1&
REQUEST=GetMap&STYLES=&
EXCEPTIONS=application/vnd.ogc.se_inimage&
FORMAT=image/jpeg&SRS=EPSG:27700&
BBOX=298559.5703125,36513.671875,603979.49218
75,210830.078125&WIDTH=625&HEIGHT=357
```

Figure 5.5: Interaction between the Web Application and an OGC-WMS.

In the example shown three layers have been selected for display. Two of these layers—showing the main roads and the populated areas—have been retrieved from OGC-WMS services [de 06], which are external to the SemSorGrid4Env architecture. OGC-WMS provide data as GML which can be natively retrieved and displayed by the Web Application, thus the interaction, shown in Figure 5.5, is made directly by the Web Application without the use of the Application Services.

Juxtaposed on top of the layers retrieved from the OGC-WMS are the current locations where overtopping events are taking place. These are displayed as red circles: the size of the circle being controlled by the value of the sensor reading, i.e. the larger the circle the more severe the overtopping event. An overtopping event is defined as taking place when a sensor's wave height reading is greater than some threshold value for the location. This requires retrieving the sensor values from the Channel Coastal Observatory Streaming Data Service (CCO-WS) and integrating it with the associated threshold value available in a Stored Data Service (CCO-Stored). The integrated data source is available as a virtual data source in the Semantic Integration and Query Service. The Semantic Integration and Query Service uses a Distributed Query Processing Service (SNEE-WS) to retrieve the various sensor feeds from the Streaming Data Services (CCO-WS) and to query the Stored Data Service. The interactions between the services is shown in Figure 5.7. Note that the Distributed Query Processing Service uses a pull mechanism offered through the Data Access Interface (Section 3.6) to retrieve the data from the Streaming Data Service while the Semantic Integrator relies on a subscription mechanism (Section 3.7) offered by the Distributed Query Processing Service.

## 5.3 Summary

This section has provided some sample orchestrations for the flood use case scenario that are made possible through the SemSorGrid4Env architecture. Note that the functionality offered by the architecture exceeds that available from either GSN or OGC-SWE in terms of its ability to support data integration and the use of domain terminology. In particular, we have shown the ability to

- Discover data sources based on their spatiotemporal and thematic coverage through a registry service that stores the semantically annotated property documents of the data sources;
- Support the mashup of a variety of data sources, including those external to the SemSor-Grid4Env architecture;

```
PREFIX cd: <http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#>
PREFIX sb: <http://www.w3.org/2009/SSN-XG/Ontologies/SensorBasis.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?windspeedvalue ?windtimestamp ?lon ?lat
FROM STREAM <http://www.semsorgrid4env/ccometeo.srdf>
WHERE {
  ?WaveObs a cd:Observation;
    cd:observationResult ?windspeedvalue;
    cd:observationResultTime ?windtimestamp;
    cd:observationResultLatitude ?lat;
    cd:observationResultLongitude ?lon;
    cd:observedProperty ?windProperty;
    cd:featureOfInterest ?windFeature.
  ?windFeature a cd:Feature;
    cd:locatedInRegion cd:SouthEastEnglandCCO.
  ?windProperty a cd:WaveHeight.
}
```

Figure 5.6: Sample SPARQL$_{Stream}$ query sent by the Web Application to the Integration Service.

- Correlate data through data integration techniques offering a reconciled ontological view over a variety of data sources including streaming data from sensors and traditional stored data from databases.

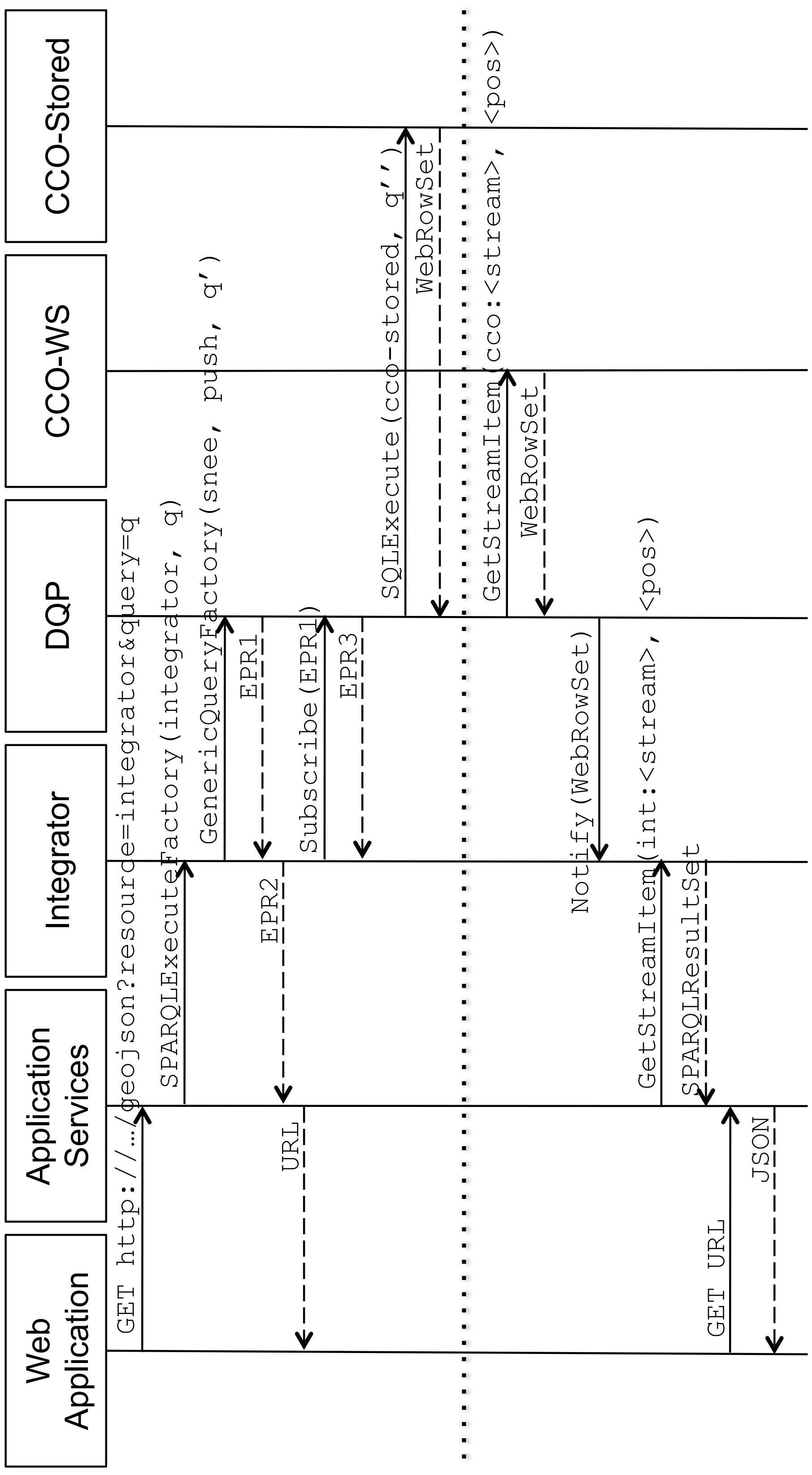

Web Application
Application Services
Integrator
DQP
CCO-WS
CCO-Stored
GET http://…/geojson?resource=integrator&query=q
SPARQLExecuteFactory(integrator, q)
GenericQueryFactory(snee, push, q')
EPR1
Subscribe(EPR1)
EPR3
EPR2
URL
SQLExecute(cco-stored, q'')
WebRowSet
GetStreamItem(cco:<stream>, <pos>)
WebRowSet
Notify(WebRowSet)
GetStreamItem(int:<stream>, <pos>)
SPARQLResultSet
GET URL
JSON

Figure 5.7: Interaction between the services to retrieve semantically integrated sensor data. Operations above the dotted line show the interactions in setting up a continuous query, while those below the dotted line are repeated interactions for sending query answers between the services. An example query $q$ is shown in Figure 5.6.

# 6. Conclusions

This document has presented the SemSorGrid4Env Web services software architecture. The need for the software architecture arises from the SemSorGrid4Env exemplar application [HSP$^+$10] that demonstrates a scenario in which agile application development for environmental monitoring can be supported. The use case considers situations that make use of data from distributed sources, both streaming data (e.g. from a sensor network) and stored data (e.g. from a database), and integrates the data to provide a semantically rich unified view of disparate sources.

The infrastructural backbone of the architecture consists of the specification of five core service-oriented Web services: the *Registration and Discovery Service*, the Integration and Querying Service, the *Stored Data Service*, the *Streaming Data Service*, and the *Distributed Query Processing Service*. These core services will be supplemented with additional domain specific services and RESTful interfaces.

Details of the initial SemSorGrid4Env deployment of the software architecture can be found in Deliverable D1.4v1 [AGK$^+$10]. The final deployment will be described in D1.4v2 due 30 April 2011. Specifications of the implementations of the Web services developed as part of the SemSorGrid4Env project can be found as follows:

**High-level API:** Initial implementation and deployment are detailed in deliverable D5.2v1 [PSK$^+$09]. The final deployment will be specified in D5.2v2 due 28 February 2011.

**Semantic Registration and Discovery Service:** Initial implementation and deployment are detailed in deliverable D3.3v1 [KKK09] and the following publications [KKK10, KK10]. The final deployment will be specified in D3.3v2 due 28 February 2011.

**Semantic Integration and Query Service:** Initial implementation and deployment are detailed in deliverable D4.2v1 [CC09] and the following publication [CCG10]. The final deployment will be specified in D4.2v2 due 28 February 2011.

**Streaming Data Query Service:** Initial implementation and deployment are detailed in deliverable D2.2v1 [GGR$^+$09] and the following publication [GBG$^+$]. The final deployment will be specified in D2.2v2 due 28 February 2011.

**Distributed Query Processing Service:** Initial implementation and deployment are detailed in deliverable D2.2v1 [GGR$^+$09]. The final deployment will be specified in D2.2v2 due 28 February 2011.

Note, the Stored Data Service uses the WS-DAI reference implementations [ACK$^+$06, GGP08].

# Bibliography

[AAK+06] M. Antonioletti, Malcolm Atkinson, A. Krause, S. Laws, S. Malaika, Norman W. Paton, D. Pearson, and G. Riccardi. Web services data access and integration - the core (WS-DAI) specification, version 1.0. Recommendation GFD.74, Open Grid Forum, 5 September 2006.

[ACK+06] Mario Antonioletti, Brian Collins, Amy Krause, Simon Laws, James Magowan, Susan Malaika, and Norman W. Paton. Web services data access and integration - the relational realisation (WS-DAIR) specification, version 1.0. Recommendation GFD.76, Open Grid Forum, 20 July 2006. `http://www.ogf.org/documents/GFD.76.pdf`.

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer, 2004.

[AGK+10] Juan José Aparicio, Alasdair J. G. Gray, Kostis Kyzirakos, Jason Sadler, and Jean-Paul Calbimonte. Reference semsorgrid4env implementation – phase i. Deliverable D1.4v1, SemSorGrid4Env, April 2010.

[AHK+06] Mario Antonioletti, Shannon Hastings, Amy Krause, Stephen Langella, Steven Lynden, Simon Laws, Susan Malaika, and Norman W. Paton. Web services data access and integration – the XML realization (WS-DAIX) specification, version 1.0. Recommendation GFD.75, Open Grid Forum, 2 August 2006. `http://www.ogf.org/documents/GFD.75.pdf`.

[AHS07] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *Proceedings of 8th International Conference on Mobile Data Management (MDM 2007)*, pages 198–205, Mannheim (Germany), May 2007. IEEE Computer Society.

[AKP+06] Mario Antonioletti, Amy Krause, Norman W. Paton, Andrew Eisenbrg, Simon Laws, Susan Malaika, Jim Melton, and Dave Pearson. The WS-DAI family of specifications for web service data access and integration. *SIGMOD Record*, 35(1):48–55, March 2006.

[Apa09] Apache web services project, 17 November 2010 2005-2009. `http://ws.apache.org/` accessed 17 November 2010.

[Axi08] Apache Axis2, 2004-2008. `http://ws.apache.org/axis2/` accessed 17 November 2010.

[BBD+08] Mihai Bartha, Thomas Bleier, Pascal Dihé, Denis Havlik, Désirée Hilbring, Marco Hugentobler, Ionut Iosifescu Enescu, Siegbert Kunz, Patrick Jacques, Sascha Scholbinski, Ingo Simonis, Jörg Stumpp, Thomas Usländer, and Kym Watson. Specification of the sensor service architecture v1. Project Deliverable D2.3.2, Sensors Anywhere, August 2008. `http://sany-ip.eu/publications/2420`.

[BGFP08] Christian Y. A. Brenninkmeijer, Ixent Galpin, Alvaro A. A. Fernandes, and Norman W. Paton. A semantics for a query language over sensors, streams and relations. In *Proceedings of 25th British National Conference on Databases (BNCOD 25)*, volume 5071 of *LNCS*, pages 87–99, Cardiff (UK), July 2008. Springer.

[BHM+04] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard (Eds). Web services architecture. Working group note, W3C, 11 February 2004. `http://www.w3.org/TR/ws-arch/`.

[BPRD06] Mike Botts, George Percivall, Carl Reed, and John Davidson. OGC® sensor web enablement: Overview and high level architecture. In *Proceedings of Second International Conference on GeoSensor Networks - GSN 2006*, volume 4540 of *LNCS*, pages 175–190, Boston (MA, USA), October 2006. Springer.

[BPRD07] Mike Botts, George Percivall, Carl Reed, and John Davidson (Eds). OGC® sensor web enablement: Overview and high level architecture. White Paper 3, Open Geospatial Consortium Inc., 28 December 2007. http://portal.opengeospatial.org/files/?artifact_id=25562.

[CC09] Jean-Paul Calbimonte and Oscar Corcho. Implementation and deployment of the ontology-based data integration service – phase 1. Deliverable D4.2v1, SemSorGrid4Env, December 2009.

[CCG10] Jean-Paul Calbimonte, Oscar Corcho, and Alasdair J. G. Gray. Enabling ontology-based access to streaming data sources. In *Proceedings of 9th International Semantic Web Conference (ISWC 2010)*, LNCS, Shanghai, China, November 2010. Springer.

[CMRW07] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana (Eds). Web services description language (WSDL) version 2.0 part 1: Core language. Recommendation, W3C, 26 June 2007. http://www.w3.org/TR/wsdl20.

[CXF10] Apache cxf, 17 November 2010. http://cxf.apache.org/ accessed 17 November 2010.

[dBBD+05] Jos de Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Uwe Keller, Michael Kifer, Birgitta König-Ries, Jacek Kopecký, Rubén Lara, Holger Lausen, Eyal Oren, Axel Polleres, Dumitru Roman, James Scicluna, and Michael Stollberg. Web service modeling ontology (WSMO). Member submission, W3C, June 2005. http://www.w3.org/Submission/WSMO/.

[de 06] Jeff de la Beaujardiere (Ed). OpenGIS® web map server implementation specification. Standard Specification 06-042, Open Geospatial Consortium Inc., 15 March 2006.

[DKP+09] Doug Davis, Anish Karmarkar, Gilbert Pilz, Steve Winkler, and Ümit Yalçinalp (Eds). Web services reliable messaging (WS-ReliableMessaging) version 1.2. Standard, OASIS, 2 February 2009. http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.html.

[dRSPM09] David de Roure, Jason Sadler, Kevin Page, and Kirk Martinez. Specification of high-level application programming interfaces. Deliverable D5.1v1, SemSorGrid4Env, March 2009.

[FGM+99] Roy Thomas Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and Tim Berners-Lee. Hypertext transfer protocol – HTTP/1.1. Standard RFC2616, Internet Engineering Task Force, June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, Information and Computer Science, University of California, Irvine, California, USA, 2000.

[FL07] Joel Farrell and Holger Lausen (Eds). Semantic annotations for WSDL and XML schema. Recommendation, W3C, August 2007. http://www.w3.org/TR/sawsdl/.

[GBG+] Ixent Galpin, Christian Y. A. Brenninkmeijer, Alasdair J. G. Gray, Farhana Jabeen, Alvaro A. A. Fernandes, and Norman W. Paton. SNEE: A query processor for wireless sensor networks. *Distributed and Parallel Databases*. To appear in a special issue on Query Processing in Sensor Networks.

[GCRRH+09] Raúl García-Castro, Gabriel Rucabado-Rucabado, Chris Hill, Agustín Izquierdo, and Óscar Corcho. Sensor network ontology suite. Deliverable D4.3 Version 1.0, SemSorGrid4Env, December 2009.

[GGFP10] Alasdair J. G. Gray, Ixent Galpin, Alvaro A. A. Fernandes, and Norman W. Paton. SemSorGrid4Env streaming data service interface. Technical report, University of Manchester, February 2010.

[GGP08] Miguel Esteban Gutiérrez and Asunción Gómez-Pérez. Web services data access and integration – the RDF(S) realization (WS-DAI-RDF(S)) ontology specification. Working Draft Version 0.8, Open Grid Forum, 22 February 2008.

[GGR+09] Alasdair J G Gray, Ixent Galpin, Varadarajan Rajagopalan, Alvaro A A Fernandes, Norman W Paton, Alexis Kotsifakos, Dimitris Kotsakos, and Dimitrios Gunopulos. Implementation and deployment of data management and analysis, and the query processing components – phase 1. Deliverable D2.2v1, SemSorGrid4Env, December 2009.

[GHM+07] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon (Eds). SOAP version 1.2 part 1: Messaging framework (second edition). Recommendation 1.2, W3C, 27 April 2007.

[GHME06] Steve Graham, David Hull, Bryan Murray, and Eds. Web services base notification 1.3 (WS-BaseNotification). Standard, OASIS, 1 October 2006. `http://docs.oasis-open.org/wsn/`.

[GHR06] Martin Gudgin, Marc Hadley, and Tony Rogers (Eds). Web services addressing 1.0 – core. Recommendation, W3C, 9 May 2006. `http://www.w3.org/TR/ws-addr-core/`.

[GSN07] GSN: Middleware for sensor networks, 2005-2007. `http://gsn.sourceforge.net/` accessed 5 May 2009.

[HB04] Hugo Haas and Allen Brown (Eds). Web services glossary. Working group note, W3C, 11 February 2004. `http://www.w3.org/TR/ws-gloss/`.

[HSP+10] Craig Hutton, Jason Sadler, Kevin Page, Mike Clark, Robin Newman, and Samantha Roe. Flood user requirements specification. Deliverable D7.1 Version 2, SemSorGrid4Env, September 2010.

[KK10] Manolis Koubarakis and Kostis Kyzirakos. Modeling and querying metadata in the semantic sensor web: The model strdf and the query language stsparql. In *Proceedings of 7th Extended Semantic Web Conference (ESWC 2010) Part I*, LNCS, pages 425–439, Heraklion, Crete, Greece, June 2010. Springer.

[KKK09] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Implementation and deployment of the registry services – phase i. Deliverable D3.3v1, SemSorGrid4Env, December 2009.

[KKK10] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Developing registries for the semantic sensor web using stRDF and stSPARQL (short paper). In *3rd International Workshop on Semantic Sensor Networks*, Shanghai, China, November 2010.

[LMH+09] Steven Lynden, Arijit Mukherjee, Alastair C. Hume, Alvaro A. A. Fernandes, Norman W. Paton, Rizos Sakellariou, and Paul Watson. The design and implementation of OGSA-DQP: A service-based distributed query processor. *Future Generation Computer Systems*, 25(3):224–236, March 2009.

[MBH+04] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic markup for web services. Member submission, W3C, November 2004. `http://www.w3.org/Submission/OWL-S/`.

[NKMHB06] Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker. Web services security: Soap message security 1.1. Standard specification, OASIS, 1 February 2006.

[NP07] Arthur Na and Mark Priest (Eds). Sensor observation service. Implementation Standard OGC 06-009r6, Open Geospatial Consortium Inc., October 2007.

[NRE94] NRENAISSANCE Committee, National Research Council. *Realizing the Information Future: The Internet and Beyond*. The National Academies Press, 1994. `http://www.nap.edu/catalog.php?record_id=4755`.

[OGC09] Open geospatial consortium, incorporated, 1994-2009. `http://www.opengeospatial.org/` accessed 10 July 2009.

[PSK+09] Kevin R. Page, Jason Sadler, Oles Kit, David C. De Roure, and Kirk Martinez. Implementation and deployment of a library of the high-level application programming interfaces – phase i. Deliverable D5.2v1, SemSorGrid4Env, December 2009.

[PZL08] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big" web services: making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web - WWW 2008*, pages 805–814, Beijing (China), April 2008. ACM.

[Res00] E. Rescorla. HTTP over TLS. Informational RFC2818, Internet Engineering Task Force, May 2000. `http://tools.ietf.org/html/rfc2818`.

[SHS08] Amit P. Sheth, Cory Henson, and Satya Sanket Sahoo. Semantic sensor web. *IEEE Internet Computing*, 12(4):78–83, July-August 2008.

[Sim07] Ingo Simonis (Ed). OpenGIS® sensor planning service implementation specification. Implementation Specification OGC 07-014r3, Open Geospatial Consortium Inc., August 2007.

[Sne04] James Snell. Resource-oriented vs. activity-oriented web services. a quick look at the relationship of rest-style and soap-style web services, 12 October 2004. `http://www.ibm.com/developerworks/webservices/library/ws-restvsoap/` accessed 1 May 2009.

[VA09] Miguel Vidal and Juan José Aparicio. Deployment of technological infrastructure. Deliverable D1.2v2, SemSorGrid4Env, December 2009.