SemSorGrid4Env

FP7-223913

Deliverable

D1.4v2

Reference SemsorGrid4Env Implementation – Phase II

J. Aparicio (editor)

and

J. Rodríguez López

30th April 2011

Status: Final

Scheduled Delivery Date: 30th April 2011

Executive Summary

The objective of this document is to describe the integrated middleware that has been developed in the context of the SemSorGrid4ENV project, which is used in the SSG4env case studies (the Southampton based flood related application and example mashups), and is also able to be used by 3rd parties willing to create their own semantic sensor web applications.

This document provides a brief description of the SSG4env architecture and each of the different components composing the whole system, and how to install, manage, and configure them individually. The deliverable itself consists of the code generated at the end of Phase II, month 32 in the project. It can be found in the file D1.4v2-sources.zip.

Some of the components whose description is included in this document are the subject of separate deliverables. In such cases, the description will point to the external document that provides comprehensive information about the component.

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 preexisting materials in this deliverable:

- Section 1 is an adaptation of the already existing section 1 of the D1.4v1 deliverable and uses the architectural image from the same deliverable.
- Section 2.3. uses an image from the deliverable D3.3v2.
- Section 2.3.2 quotes the deliverable D3.3.v2.
- Section 4.3 uses fragments of the installation instructions of each components, explained deeply in their corresponding deliverable.
- The information exposed at annex A is obtained from the [D4.2v2] sections 3.2.1 and 5.2.1.

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/home.jsp?content=/sew/viewTerm&content=instance.jsp&_sew_var_name=instance&_sew_instance=D1.4+v1&_sew_instance_set=SemSorGrid4Env&_origin=%2Fhome.jsp		
EU Project officer	Antonios Barbas		

Deliverable	**Number**		**Name**			
Task	**Number**	1.4	**Name**	Develop a Reference SemsorGrid4Env implementation		
Work package	**Number**	1		Software Architecture and Middleware for Semantic Sensor Grids		
Date of delivery	**Contractual**		30th April 2011	**Actual**	30th April 2011	
Code name				**Status** draft ☐	final ☑	
Nature	Prototype ☑ Report ☐ Specification ☐ Tool ☐ Other ☐					
Distribution Type	Public ☑ Restricted ☐ Consortium ☐					
Authoring Partner	Techideas					
QA Partner	University of Southampton					
Contact Person	Juanjo Aparicio					
	Email	juanjo.aparicio@techideas.es	**Phone**	+34 902 109 443	**Fax**	+34 932 917 727
Abstract (for dissemination)	This document provides an introduction to the source code generated for phase II of the Reference Semsorgrid4Env implementation, the way in which components are built and how they are tested.					
Keywords	Reference implementation, web services, application, building					

Version log/Date	**Change**	**Author**
0.1 25/03/2011	**First Draft**	**Josep Rodríguez López**
0.2 15/04/2011	**Second Draft**	**J. Aparicio, J. Rodríguez, J. Gabaldón**
0.3 21/04/2011	**Third Draft**	**J. Aparicio, J. P. Calbimonte**
0.4 16/05/2011	**Delivered for QA**	**J. Rodríguez**
0.4 23/05/2011	**First QA**	**J. Sadler**

Project Information

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

Partner	Acronym	Contact
Universidad Politécnica de Madrid (Coordinator)	UPM	Prof. Dr. Asunción Gómez-Pérez Facultad de Informática Departamento de Inteligencia Artificial Campus de Montegancedo, sn Boadilla del Monte 28660 Spain #@ asun@fi.upm.es #t +34-91 336-7439, #f +34-91 352-4819
The University of Manchester	UNIMAN	Prof Norman Paton Department of Computer Science The University of Manchester Oxford Road Manchester, M13 9PL, United Kingdom #@ npaton@cs.man.ac.uk #t +44-161-275 69 10, #f +44-161-275 62 04
National and Kapodistrian University of Athens	NKUA	Prof. Manolis Koubarakis University Campus, Ilissia Athina GR-15784 Greece #@ koubarak@di.uoa.gr #t +30 210 7275213, #f +30 210 7275214
University of Southampton	SOTON	Dr. Kirk Martinez University Road Southampton SO17 1BJ United Kingdom #@km@ecs.soton.ac.uk #t+44 23 80594491, #f +44 23 80595499
Deimos Space, S.L.	DMS	Mr. Agustín Izquierdo Ronda de Poniente 19, Edif. Fiteni VI, P 2, 2º Tres Cantos, Madrid – 28760 Spain #@agustin.izquierdo@deimos-space.com #t +34-91-8063450, #f +34-91-806-34-51
EMU Limited	EMU	Dr. Bruce Tomlinson Mill Court, The Sawmills, Durley number 1 Southampton, SO32 2EJ – United Kingdom #@ bruce.tomlinson@emulimited.com #t +44 1489 860050, #f +44 1489 860051
TechIdeas Asesores Tecnológicos, S.L.	TI	Dr. Jesús E. Gabaldón C/ Marie Curie 8-14 08042 Barcelona, Spain #@ jesus.gabaldon@techideas.es #t +34.93.291.77.27, #f ++34.93.291.76.00

Table of contents

1.Introduction

Deliverable 1.4v2 is the reference implementation of the SemsorGrid4Env architecture. This document accompanies the actual deliverable, which consists of the code available at the project subversion repository. An export of the code has been created and it has been packed for submission in the file *D1.4v2-sources.zip*, which accompanies this document.

The second, and final version of the architecture document [D1.3v2] specifies the services, and their interfaces, required to provide the functionality for a semantic sensor web application. The implementation effort in all technical work packages of the project has produced a functional subset of these services. This section details the mapping

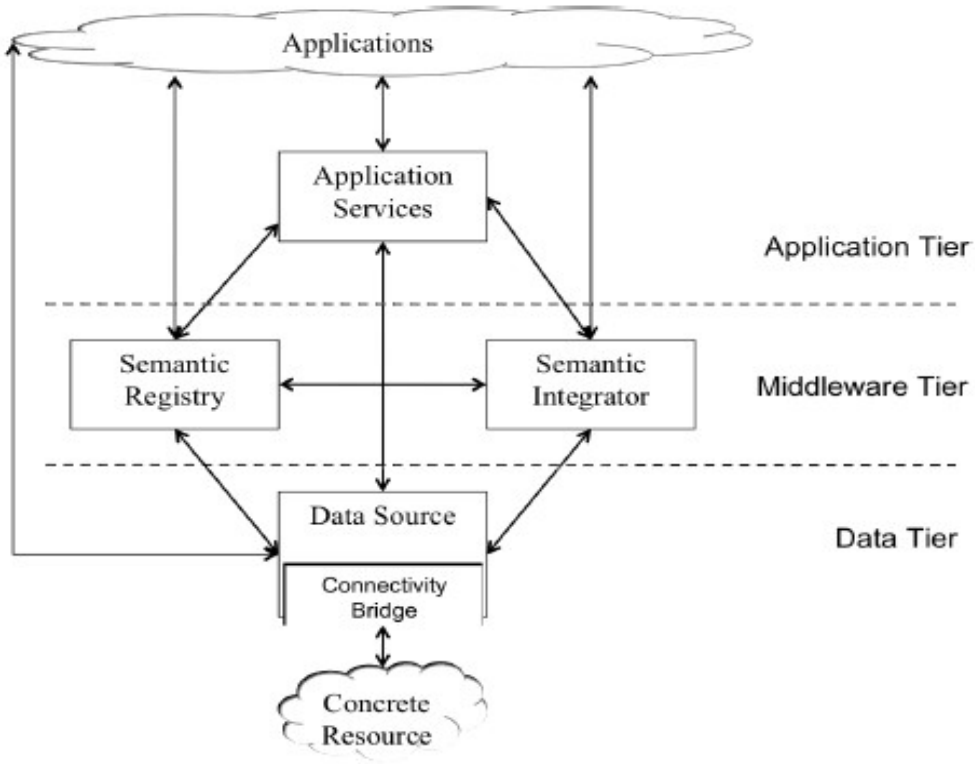

Figure 1: SSG4env Architecture and components [D1.3v2]

between the functional units identified in the architecture document and the implementation components.

The boxes in Figure 1 are mapped to implementation components as follows:

Architecture Component	**Contributing Implementation Components**
Connectivity Bridge	SNEE
Data Source	SNEE, StreamingDataService
Semantic Registry	Strabon, Semantic Registry
Semantic Integrator	Semantic Integrator, IntegrationQueryService
Application Services	High Level API

Usually, the SSG4Env project architecture component are implementated in two different components, one providing the base functionality, and a second one providing its web service interface.

2.SemsorGrid4ENV Components

In this chapter, the most important components of the project will be briefly described.

2.1. High-level Application Programming interfaces

The HLAPI software component takes sensor data from the SemsorGrid4Env Integration Query Service and translates it into observations in accordance with the SemsorGrid4Env ontologies. It uses a configuration written by the service administrator to serialise these observations (and a corresponding set of observation collections) to disk and RDF triplestore. The observations are serialised in a number of different formats (RDF/XML, O&M GML, WFS GML, HTML and GeoJSON), and are accessible to mashup and Web application developers through a RESTful Linked Data observation API (via an Apache Web server and 4Store).

The latest HL-API implementation is fully described at the deliverable [D 5.3v2] and the deliverable [D 5.2v2].

2.2. Integration Query Service

The integration Query Service consists of a library and a web service as explained below.

2.2.1. Semantic Integrator

Formerly known as 'Mapster', the Semantic Integrator is the underlying library used by the Integration Query Service (IQS) web service providing all the needed integration functionalities. It is formed by a group of subcomponents: the query translation, the query processing, and the data translation engines.

The query translation engine parses the SPARQL query, obtaining an abstract syntax tree. Once it has the syntax tree, the stream-to-mapping document, which specifies the relationships between the ontologies used and the underlying streaming data sources, is read, and the SPARQL stream input query is extracted. Once the mappings have been created, it moves along to the translation step. At this time, the Semantic Integrator translates the SPARQL to SNEEQL, so the new query can be posed to the SNEE, which will interact with the integrated Streaming Data Services, obtaining the results of the desired SPARQL query.

Afterwards, the query processing engine of the Semantic integrator processes the previous query using the SNEE Java API.

Once the Query Processing engine returns the results, the Data Translation component translates the data back to the ontological reference of the initial query.

2.2.2. Integration Query Service

The integration Query Service (IQS) is a web service used to integrate data from sources and types.

It provides the the following functionalities:

• Create an integrated data resource exposed through an ontological schema, given a set of streaming data resources and a mapping document.

• Query integrated data resources using a SPARQL extension for streaming data, while the service internally accesses the underlying data sources.

•Provide access to the continuous data queries registered for the integrated data sources.

• Provide metadata for the integrated data resources in the form of property documents.

2.3. Semantic Registry and Discovery Service

The Semantic Registry and Discovery Service (Registry) is composed of a core library and a web service, both described in the following sections.

2.3.1. Strabon

Strabon is the registry underlying storage and query evaluation module for stSPARQL and stRDF, where “st” stands for spatio-temporal. The purpose of the Strabon component is to provide a spatio-temporal approach to SPARQL and RDF, providing, at the same time, the semantic functions that the Registry needs.

As described in the [D3.3v2] deliverable, Strabon is built on top of the RDF store Sesame [45] and extends Sesame's components to be able to manage thematic and spatial metadata that are stored in PostGIS [http://postgis.refractions.net/], [http://www.postgresql.org/].

The first module used is called the Query Engine. It overrides Sesame's parser and evaluator in order to gain the spatio-temporal data comprehension. Moreover, it optimizes and prepares an execution plan benefiting Sesame's optimizer and transaction manager.

A supporting module is PostGIS (http://postgis.refractions.net/), an extension providing spatio-temporal capabilities to the PostgreSQL database management system. The PostGIS module is used by the Storage Manager, responsible for storing the Registry data. Its task is to extend Sesame's RDBMS module.

The Constraint Engine is a module responsible for processing the part of the queries that have to deal with geometries and the conversions that take place during data loading. It implements a Representation Translator component, in order to translate the geometries to other equivalent representations, like OGC Well-Known Text (ISO 19125-1) [http://www.opengeospatial.org/standards/orm] (Open Geospatial Consortium, http://www.opengeospatial.org/). A non-standard geometry format was chosen, because the OGC standard doesn't provides all the functionalities the project needed.

2.3.2. Semantic Registry

The Registry comprises a web service based on “Strabon”, a core library providing a spatio-temporal approach to RDF and SPARQL, and “Atlas 2”, described in the next chapter and the deliverable [D3.3v2] respectively. Its purpose is to provide clients with support for the stSPARQL query language.

The Registry plays the “Service Registry” role in a web service architecture. Service providers register their services with the Registry, and, when clients wish to request a service, they will first query the Registry, requesting the web service definition (using the SPARQL query language), and receive an RDF description of the format. The difference between the SSG4env Registry and other web service registries is that the Registry provides a semantic approach to the “Service discovery” function.

Though the Registry is an important back-end component, we have decided not to perform integration tests on it because it will be used as a single component accessed by user application developers. Any developer intending to access a specific web service, whether for use with the HL-API or the Integrator, will be able to pose stSPARQL queries to the Registry, obtaining the web services that will provide the desired data. The same applies in the opposite direction, if a developer has integrated a new data source at the Integrator, the stRDF description of the web service can be passed directly to the Registry.

This way, the Registry will only work as a store and query service used directly by users application developers, without the need for interaction with other components.

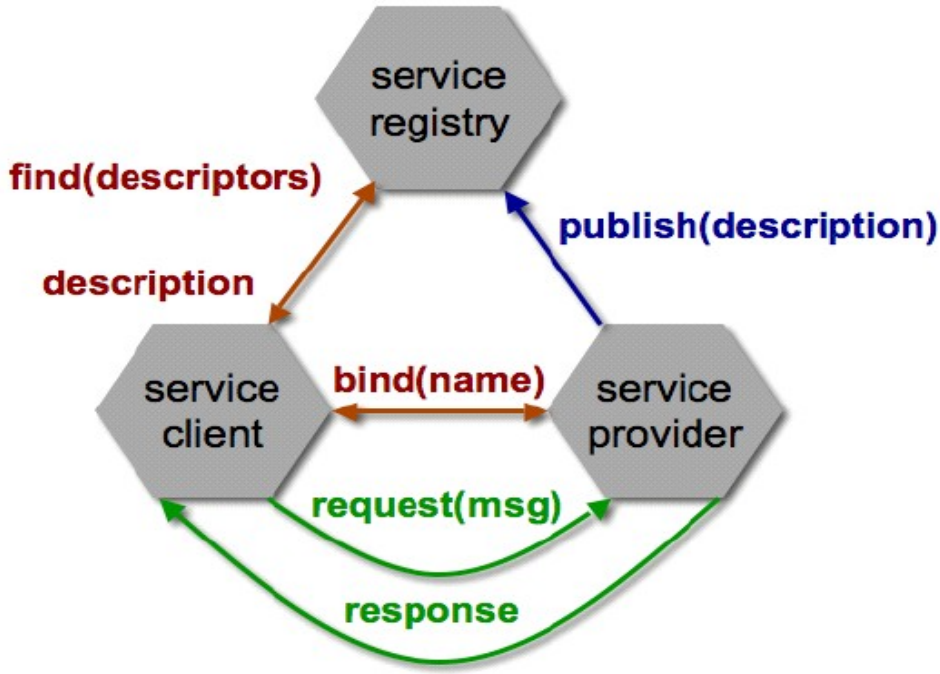

Figure 2: Typical web service architecture, using a service registry

This "Registry" only stores definitions of data source web services.

2.4. SNEE

2.4.1. SNEE Query Engine

The SNEE query processing engine is capable of evaluating SNEEQL queries, explained in detail in deliverable [D2.2v1], either over a local sensor network source, using in-network query processing techniques, or over a set of external streaming and stored data services, using data stream query processing techniques.

The SNEE query engine is able to perform a query by first evaluating it and then performing a peer-to-peer query between all the sensors in the sensor network.

Furthermore, it can also provide access to historic data through a Stored Data Service interface.

2.4.2. SNEE Web Service

The SNEE component provides a web service interface so other components can interact with the SNEE query engine. The web service transforms the queries posed by external clients (usually, the integrator or the HLAPI) into SNEE API method calls.

This way, we can use the SNEE Web Service to pose queries and retrieve long-running query results to and from the SNEE Query Engine.

2.5. Streaming Data Service

The Streaming Data Service (SDS), is a web service interface front-end for the SSG4Env components, setting a template for building SSG4Env web services which will be able to be integrated with the rest of the system.

Instances of the SDS provide web service clients with a continuous stream of data. These will usually be developed to expose data from a sensor network.

An SDS can provide data in both Pull and Push modes, so the client can register to a stream of data, and the SDS will periodically provide the data corresponding to the registered stream.

All Streaming Data Service instances should implement the PullStream, PushStream, QueryService and SubscriptionManager interfaces.

3.Building The Framework

This section describes the guidelines to build the components of the SSG4env middleware. This code has been developed and tested using Tomcat 6 and Java 6.

3.1. Obtaining the Code

The code of the SSG4Env project is found at the D1.4v2.zip archive, which this doccument is accompanying.

3.2. Maven build system

The components of the SSG4env middleware make use of Apache Maven as their default build tool. Therefore the installation procedure is the same as for any project built using Maven. That is:

$ mvn install

The SSG4env middleware component web services are developed as *Java* webapps, so a web server with *Java* webapps compatibility is needed (e.g. A*pache tomcat [*http://tomcat.apache.org/*]*).

3.3. Managing dependencies

In order to fetch the project specific dependencies from the repository located at the SSG4Env server, we need to modify the Maven *pom.xml* settings file and add the following lines:

```
<servers>
    <server>
       <id>ssg4env.snapshot</id>
        <username>your-username</username>
        <password>your-password</password>
    </server>
    <server>
       <id>ssg4env.internal</id>
        <username>your-username</username>
        <password>your-password</password>
    </server>
</servers>
```

Code 1: Settings required to fetch dependencies from the SSG4Env repository

In order to obtain a valid username and password at the TechIDEAS server, contact the project administrator using the contact details provided in the project website.

3.4. Specific installation guide for each component

This section assumes that an A*pache tomcat* server is installed on the machine intended to host the component, and also that the code is already under the user's possession.

Integration Query Service:

- Enter the directory on which the project root is located.
- configure the *iqs-ws/src/main/webapp/WEB-INF/config/config.properties* using the references found at the section 5.2.1 of the d4.2v2 directory or Annex A of this document.
- Run *mvn install* at the project's root directory, generating the .war file at the *target/* directory.
- Copy the generated .war archive into your apache tomcat “webapps” folder, and it will be automatically deployed.

Registry:

- You will need a PostGis database accessible from the server that will host the Registry.
- Enter the directory on which the project root is located.
- Modify the *WebContent/WEB-INF/beans.xml* so it has the credentials for your PostgreSQL database.
- Perform a *mvn install* at the project's root directory. The .war file generated will be located at *target/*.
- Copy the generated .war archive into your apache tomcat “webapps” folder, and it will be automatically deployed.
- You will find further instructions about populating the registry with ontologies and service instances at the deliverable [D3.3v2].

SNEE WS:

- Enter the directory on which the project root is located.

- Set the properties you desire at *src/main/webapp/WEB-INF/etc*. Typically, the user might wish to modify the physical-schema.xml to set the SDS direction the SNEE-WS must access when working.

- Run a *mvn install* at the project's root directory, the resulting .war file will be located at the *target/* directory.

- Copy the generated .war archive into your apache tomcat "webapps" folder, and it will be automatically deployed.

HLAPI:

- The jar archive of the component is obtained by running the command mvn install at the project's root directory.

- To get the application running and integrated with the Integration Query Servic, a guide to configure and install the HL-API can be found in its project root directory, in a help file called "Quick Start Guide (IQS)".

4.Testing

The components of the SemsorGrid4Env project have to pass two testing phases apart from the regular Unit Testing, described below.

4.1. Integration Testing

The components of the SSG4env project must interact with each other in order to provide all of their functionalities; so integration tests are required to be performed, checking that the components behave as they should when interacting with each other.

The integration tests have been coded as a Maven project. The tests connect by default to the localhost 8080 port to test the components, so a working copy of the components must be setup on the server performing the tests. If this requirement is not satisfied, errors will occur during the testing. The complete URL for each of the components, in order to fit with the expectations of the tests, must be:

- CCO-WS: http://localhost:8080/CCO-WS
- SNEE-WS: http://localhost:8080/SNEE-WS
- Integration Query Service: http://localhost:8080/IQS
- Registry: http://localhost:8080/Registry

The tests are performed using the JUnit framework, and are executed using the command "*mvn verify*". As of date of this document release, some components might have not achieved all the functionalities required by the project, so some tests may be added during the final phase of the project.

The integration tests are performed between a pair of components, even though, some other components might be involved to provide a data source. The pairs of components identified are the following:

1. Streaming Data Service ↔ SNEE-WS. The SDS used in the tests will be the CCO-WS, but as far as the CCO-WS implements the SDS interface, any SDS passing the unit tests will be correctly integrated with the SNEE-WS, if these integration tests are passed succesfully. This test proves that:
    - A CCO-WS pull data stream can be accessed from the SNEE-WS using the pull interface. It is proved by the test SNEEPullStreamFromCCOTest; In this test, a Pull Stream is created in the SNEE-WS using the Query Interface.

Once this new Pull Stream is created, the newest item is prompted using the getStreamNewestItem method, and it is compared to the return of the same function of the CCO-WS.

- A query can be posed to the CCO-WS from the SNEE-WS. It is performed using the test SNEEQueryToCCOTest; A new PullStream is created at the SNEE-WS using the QueryInterface and the function genericQueryFactory. The SNEEql query used uses a CCO-WS resource. Furthermore, at the SNEEPullStreamFromCCOTest the query created is tested as a PullStream.

2. IQS ↔ SNEE-WS The test class performing this tests is SNEEIQSITTest, from the package eu.semsorgrid4env.sneeiqsItTests, the execution of this test is as follows: first, a new Integrated stream is obtained, merging the PullStream from the CCO-WS and the one located at the SNEE-WS (the sources integrated can be modified using a resource configuration to fit your local installation, initially, the integrated services will point to the web services specified at the beginning of this section), after that, a query is posed to the virtual integrated stream, obtaining a newly created PullStream. Finally, this PullStream is fetched and compared to the stream generated at the SNEE-WS during the execution of the test (created by the query the IQS posed to the SNEE-WS). This test can be configured, modifying the SPARql query, the sources used when integrating a new virtual Integrated Stream, and the mappings used. What has been proven with this test is that:

   - A SNEE-WS data stream can be accessed from the IQS using the pull interface. This is proved when the comparison between the IQS PullStream and the SNEE-WS PullStream is correct.

   - A query can be posed to the SNEE-WS from the IQS. We can affirm this when the test poses a query to a virtual integrated stream which uses a source from the SNEE-WS correctly.

3. HL-API ↔ IQS The test class of this tests is called IQSHLAPIITTest, located at the package eu.semsorgrid4env.iqshlapiItTests, it checks the following integrated features between the HLAPI and the IQS:

   - Check that a Integrated Stream can be created at the IQS using the HL-API.

   - Check that a SPARQL query can be posed to the IQS from the HLAPI.

   - Check that the aforementioned virtual data stream can be accessed using the pull interface.

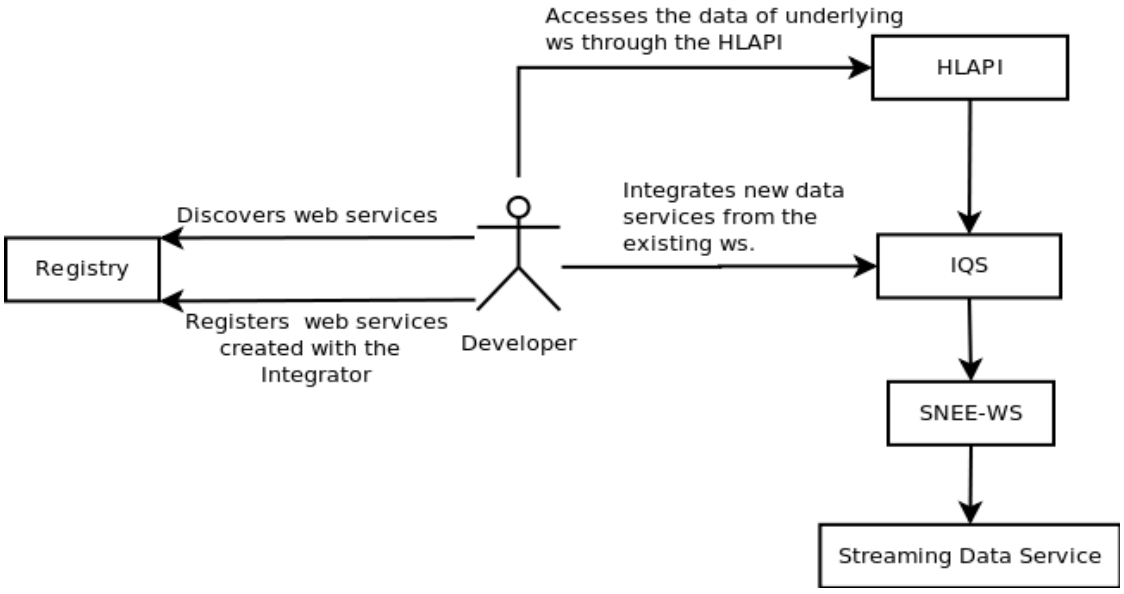

Figure 3: A schema of the relation between different components and the developer

4.2. Stress/Load testing

The stress/load tests are performed using LoadUI (http://www.loadui.org/), an Eviware application providing a fast and easy manner to stress test web services, providing the results of the test and some relevant statistics.

The main goal of these tests is to discover the limits of the components. For this purpose, we attack the components with LoadUI creating a large number of connections, so we can measure how many connections per second the server is able to serve without collapsing, while, obviously, serving the correct responses. Another aspect that must be tested, is the recovery capacity of the server after the stress tests.

Predictably, the performance of the server will be strongly related to the hardware that it is running on, so the results of the tests are always related to a specific configuration, and the tests results will vary if performed on other machines.

5.Annex A: IQS Configuration Instructions

The configuration file of the IQS is found at:

WEB-INF/classes/config/config.properties

The format of this file is:

VAR_NAME = VAR_VALUE

The most common variables used are the following ones (a further explanation can be found on [D4.2v2] at the sections 3.2.1 and 5.2.1):

- integrator.queryexecutor.adapter: Used to specify the streaming data processor, typically, it will be snee or ssg4e. If ssg4e is chosen, the following variables must be declared at the configuration file:
  - integrator.queryexecutor.adapter.ssg4e.pull: The pull service URL, i.e. the SNEE-WS Pull service end point reference.
  - Integrator.queryexecutor.adapter.ssg4e.query: The query service URL, i.e. the SNEE-WS query service end point reference.
- Integrator.repository.provider: Specifies the type of storage of the integrator resources. (e.g memory-only, file-system, etc.). If the option “file” (file system) is selected, the following configuration must be declared:
  - integrator.repository.url: The url of the folder containing the storage of the resources used by the integrator.

6. Bibliography

[D1.3v2] - “SemSorGrid4Env Architecture - Phase II”, Alasdair J G Gray, 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).[D7.4v1] – “Application (Coastal and Estuarine Flood Warning in Southern UK) - Phase I prototype” - Alex J. Frazer, Craig W. Hutton, Kevin R. Page (Editor), Jason D. Sadler (University of Southampton).

[D1.4v1] - “Reference SemsorGrid4Env Implementation - Phase I”, J. Aparicio and A. Gray, K. Kyzirakos, J. Sadler, J.P. Calbimonte.

[D2.2v2] - “Implementation and Deployment of Data Management and Analysis, and the Query Processing Components - Phase II”, Alasdair J G Gray, Ixent Galpin, Alvaro A A Fernandes, and Norman W Paton (University of Manchester), Alexis Kotsifakos, George Valkanas, and Dimitrios Gunopulos (National and Kapodistrian University of Athens).

[D3.3v2] - “Implementation and deployment of the registry services - Phase II” - Manos Karpathiotakis, Kostis Kyzirakos, Zoi Kaoudi, Vissarion Fisikopoulos, Iris Miliaraki, Manolis Koubarakis, Yannis Ioannidis, and Michael Hatzopoulos (National and Kapodistrian University of Athens), Vana Kalogeraki (Athens University of Economics and Business).pasar

[D4.2v2] - “Implementation and Deployment of the Ontology-based Data Integration Service v2”, Jean-Paul Calbimonte, and Oscar Corcho (Universidad Politécnica de Madrid), Alasdair J G Gray (University of Manchester).

[D5.1] - “Specification of high-level application programming interfaces” - Kevin R. Page David C. De Roure, Kirk Martinez, and Jason Sadler (University of Southampton).

[D5.2v2] - “Implementation and Deployment of a Library of the High-level Application Programming Interfaces ” - Alex J. Frazer (Editor), David De Roure, Kirk Martinez, and Bart Nagel, Kevin R. Page and Jason Sadler .

[D5.3v1] - “Programming patterns and development guidelines for Semantic Sensor Grids ” - David C. De Roure, Alex J. Frazer, Kirk Martinez, Kevin R. Page