SemSorGrid4Env

FP7-223913

Deliverable

D5.1

Specification of high-level application programming interfaces

Kevin R. Page

David C. De Roure, Kirk Martinez, and Jason Sadler

University of Southampton, UK

31/03/2009

Status: Final version

Scheduled Delivery Date: 28/02/2009

Executive Summary

This document defines an Application Tier for the SemsorGrid4Env project. Within the Application Tier we distinguish between Web Applications - which provide a User Interface atop a more traditional Service Oriented Architecture - and Mashups which are driven by a REST API and a Resource Oriented Architecture. A pragmatic boundary is set to enable initial development of Web Applications and Mashups; as the project progresses an evaluation and comparison of the two paradigms may lead to a reassessment of where each can be applied within the project, with the experience gained providing a basis for general guidelines and best practice.

Both Web Applications and Mashups are designed and delivered through an iterative user-centric process; requirements generated by the project case studies are a key element of this approach.

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:

- N/A

Document Information

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

Deliverable	**Number**	5.1	**Name**	Specification of high-level application programming interfaces		
Task	**Number**	5.1	**Name**	Specify high-level application programming interfaces for SemsorGrid4Env		
Work package	**Number**	5				
Date of delivery	**Contractual**		28/02/2009	**Actual**	31/03/2009	
Code name				**Status**	draft ☐	final ☑
Nature	Prototype ☐ Report ☐ Specification ☑ Tool ☐ Other ☐					
Distribution Type	Public ☑ Restricted ☐ Consortium ☐					
Authoring Partner	University of Southampton					
QA Partner	University of Athens					
Contact Person	Kevin R. Page					
	Email	krp@ecs.soton.ac.uk	**Phone**	+44 23 80594059	**Fax**	
Abstract (for dissemination)	This document defines an Application Tier for the SemsorGrid4Env project. Within the Application Tier we distinguish between Web Applications - which provide a User Interface atop a more traditional Service Oriented Architecture - and Mashups which are driven by a REST API and a Resource Oriented Architecture. A pragmatic boundary is set to enable initial development of Web Applications and Mashups; as the project progresses an evaluation and comparison of the two paradigms may lead to a reassessment of where each can be applied within the project, with the experience gained providing a basis for general guidelines and best practice.					
Keywords	Mashup, ROA, SOA, Application Tier, REST					

Project Information

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

Partner	Acronym	Contact
Universidad Politécnica de Madrid (Coordinator)	UPM	Prof. Dr. Asunción Gómez-Pérez Facultad de Informática Departamento de Inteligencia Artificial Campus de Montegancedo, sn Boadilla del Monte 28660 Spain #e asun@fi.upm.es #t +34-91 336-7439, #f +34-91 352-4819
The University of Manchester	UNIMAN	Prof. Carole Goble Department of Computer Science The University of Manchester Oxford Road Manchester, M13 9PL, United Kingdom #e carole@cs.man.ac.uk #t +44-161-275 61 95, #f +44-161-275 62 04
National and Kapodistrian University of Athens	NKUA	Prof. Manolis Koubarakis University Campus, Ilissia Athina GR-15784 Greece #@ koubarak@di.uoa.gr #t +30 210 7275213, #f +30 210 7275214
University of Southampton	SOTON	Prof. David De Roure University Road Southampton SO17 1BJ United Kingdom #@ dder@ecs.soton.ac.uk #t +44 23 80592418, #f +44 23 80595499
Deimos Space, S.L.	DMS	Mr. Agustín Izquierdo Ronda de Poniente 19, Edif. Fiteni VI, P 2, 2º Tres Cantos, Madrid – 28760 Spain #@agustin.izquierdo@deimos-space.com #t +34-91-8063450, #f +34-91-806-34-51
EMU Limited	EMU	Dr. Bruce Tomlinson Mill Court, The Sawmills, Durley number 1 Southampton, SO32 2EJ – United Kingdom #@ bruce.tomlinson@emulimited.com #t +44 1489 860050, #f +44 1489 860051
TechIdeas Asesores Tecnológicos, S.L.	TI	Mr. 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 1

1.1. Scope 1

1.2. Document Structure 1

2. Overview and context within SemsorGrid4Env 2

3. Application Tier 4

3.1. Web Applications 4

3.2. Examination of the Channel Coast Observatory Web Application 6

3.2.1. Map Viewer and Data Catalogue 7

3.2.2. Realtime Data 9

3.3. REST interfaces and mashups 9

3.3.1. REST and Resource Oriented Architecture 9

3.3.2. Examination of the myExperiment REST API 11

3.3.3. REST and Mashups in the Application Tier 15

3.4. Web Applications vs. Mashups 16

4. Evolution of the Application Tier 18

4.1. Comparison of WS-* and REST 18

4.2. Semantic Mashups 18

5. References 19

1. Introduction

1.1. Scope

This document represents the D5.1 of Work Package 5 *High-level Application Programming Interfaces for Semantic Sensor Grids* within the EU project "*Semantic Sensor Grid Rapid Application Development for Environmental Management (SemSorGrid4Env)*".

1.2. Document Structure

Section 2: introduces Work Package 5, sets out its position in the project, and details the relationship and dependencies of the WP – and this deliverable – with the deliverables and future output of this and other project WPs.

Section 3: describes the two aspects of the Application Tier – Web Applications and Mashups. We describe both the traditional Service Oriented Architecture (including OGC standards) and the REST interfaces provided in a Resource Oriented Architecture, and introduce existing examples of both.

Section 4: explores the opportunities within SemsorGrid4Env to evaluate Service and Resource Oriented Architectures side-by-side, and their effectiveness at different levels within the overall project architecture. We introduce the possibility of "semantic mashups", applying Semantic Web technology in a Resource Oriented Architecture.

2. Overview and context within SemsorGrid4Env

Work Package 5 encompasses the design and delivery of the *Application Tier* for the SemSorGrid4Env project. This Application Tier is needed to provide the mechanisms – the High-level Application Programming Interfaces (APIs) referenced in the work package and deliverable titles – through which applications can access sensor network components and associated data for presentation and interaction with end-users.

In this document we elaborate upon the nature of these APIs and the Application Tier, considering:

1. the set of reusable software components that interface with the SemSorGrid4Env middleware and are used to build web applications; the APIs an application developer uses to work with the libraries.

2. the resource oriented APIs that expose data (and metadata) from the sensor network, via the web applications, to enable rapid creation of further ad-hoc lightweight applications by independent developers ("mashups").

Both of these aspects support the integration of dynamic sensor network sources and changing (possibly unexpected) application requirements that are fundamental to the SemSorGrid4Env concept, but in the latter case adoption by the external "mashup" developer community is desirable for validation of our approach. This mashup community is associated with a fluid and rapidly advancing body of theory and practice that will doubtless continue to grow apace over the term of the SemSorGrid4Env project; nonetheless it is clear we will need to develop a straightforward REST interface as a minimum to gain developer acceptance.

Past experience in developing web applications and associated REST APIs [DGS2009] has demonstrated that this is best approached in an application-centric manner, with an iterative development model closely involving end-users throughout the process. As such, this document does not set out to explicitly specify an API which might then be implemented as the project progresses – which would run counter to the best practice described above – but instead details the requirements, principles, and paradigms involved.

This document is therefore the starting point for a corpus of best practice and design patterns for the Application Tier which will be built up and reported as the project progresses (deliverable D5.2), incorporating the practical experience gained from development and deployment (deliverable D5.2).

As the principal component for interaction directly with end-users, the Application Tier will be designed and implemented with significant input from the two SemSorGrid4Env cases studies: the Solent Flooding case study (WP7), and the Spanish Fire Prevention case study (WP6). This is especially true given the application-centric, resource oriented, approach taken when developing REST APIs; the recently completed requirements analyses (D7.1v1, D6.1v1) will be instrumental in informing Application Tier development.

We will, at first, approach the two case studies separately, initially focussing on the Flood scenario during implementation stage T5.2v1, before re-applying tools and techniques to the Fire study. This staging will also provide a critical test of our ability to rapidly adapt and deploy our technology.

That is not to say that the Fire case study will be ignored during initial development – its requirements are still crucial to our design. However, we believe the Flood study provides several benefits for immediate progress in WP5:

- an existing, active, sensor network
- existing, readily available, data sets from this sensor network
- management of the sensor network is performed by GeoData, a SemSorGrid4Env partner
- geographic locality of the sensor network and case study end-users to the University of Southampton should encourage closer feedback between users and developers during design and implementation of the Application Tier and ensure resource oriented APIs into the sensors and data is appropriate to the use case

By the time of implementation stage T5.2v2 the other SemSorGrid4Env Work Packages will have advanced considerably, allowing full exposure of their respective components, through the middleware, to the Application Tier. The interfaces and mechanisms for this integration will be included in the forthcoming architecture deliverable (D1.3v1). Similarly the sensor network being constructed for the Fire case study will have been deployed and collecting data for use in the fire development application.

3. Application Tier

Due to differing – but closely related and interlinked – approaches and requirements introduced in the previous section, we define two elements within the Application Tier: Web Applications, and Mashups (utilising RESTful interfaces) (figure 3.1).

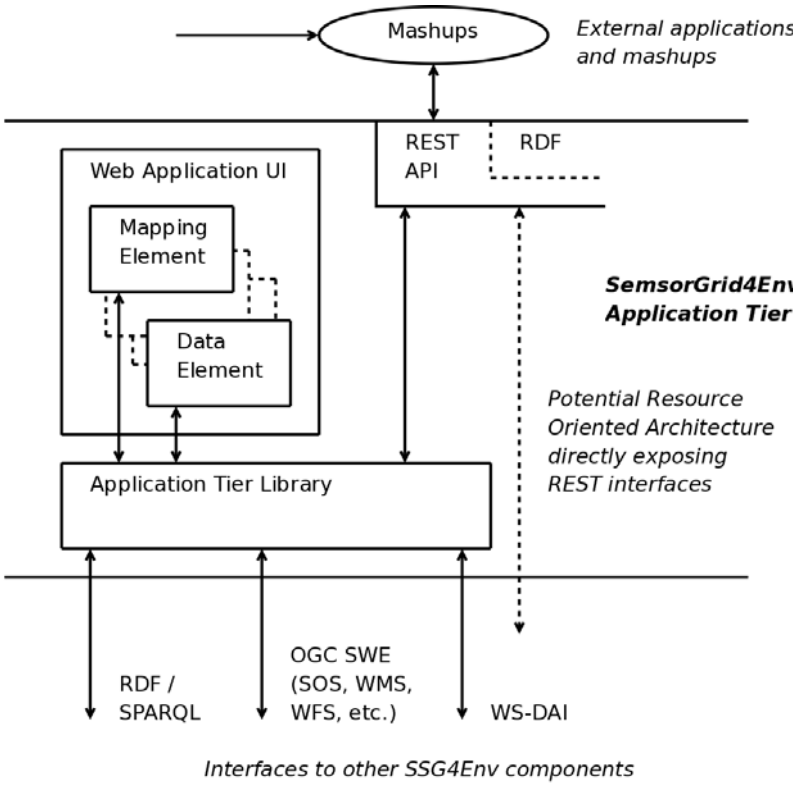

Figure 3.1: SemsorGrid4Env Application Tier

3.1. Web Applications

Web Applications primarily present a User Interface to access, utilise, and manipulate, sensors and associated data provided by systems employing the SemsorGrid4Env architecture.

In terms of the SemsorGrid4Env project the two demonstrator case studies – Fire and Flood – will provide requirements (D6.1, D7.1) that define the interactions an end-user will have with the Web Application, and from these a UI will be designed.

Other SemsorGrid4Env components will be accessed through a middleware. The complete design of the architecture is still in progress (D1.3v1), however some attributes are already known: that it will be a Service Oriented Architecture implemented using Web Services (WS-* standards), that data services are virtualized (e.g. using WS-DAI), and that the middleware will provide a semantic registry and semantic data integration service.

While the two case studies are obviously different, we anticipate several, if not many, of their requirements will be common between them. Given common UI elements, and a common middleware, the Application Tier will include a software library – a toolkit – for easily creating Web Applications atop the SemsorGrid4Env middleware. This will include, for example:

- routines to map between geospatial UI elements (e.g. using OpenLayers) and geospatially constrained queries and results to and from the middleware
- routines to dynamically generate UI elements (e.g. option lists, menus) from middleware queries for available resources and data

The case study Web Applications are also expected to require integration of other data sources external to SemsorGrid4Env (e.g. the Flood use case may require census data for locating centres of dense population, or highways traffic information for potential evacuation routes). As such, the library will also need to:

- retrieve data from external sources that have been included as data services through the SemSorGrid4Env middleware
- retrieve data through external standardised interfaces, such as those defined by the OGC Sensor Web Enablement
- at least initially, retrieve data directly from existing sources (e.g. the Channel Coastal Observatory data for the Solent can be used to to prototype Web Applications until the SemsorGrid4Env architecture is complete)

This final point highlights that we envisage the development process of the Web Applications, and therefore the supporting library, to be an iterative, user-centric, process. To build engagement and trust with our end-users we expect to first build simple Web Applications with limited functionality from which we can gain user feedback. In this aspect we are informed by the Web 2.0 inspired design principles applied to the myExperiment project [DG2009].

This approach ties closely with the user-centric view taken in work-package WP7 – for representative applications that fit real users working within real management systems and driven ultimately by real business cases. As highlighted in the previous section, development of the Application Tier will occur in tandem with development of the user requirements in the Flood case study; this document should be seen within this context as the technical companion to the user requirements described in deliverable D7.1.

Some early software prototypes to start demonstrating and evaluating possible applications and components (dual-map comparison view, temporal sensor data view,

initial REST prototypes) can therefore be found described in deliverable D7.1, and we direct the reader to the companion document at this point.

3.2. Examination of the Channel Coast Observatory Web Application

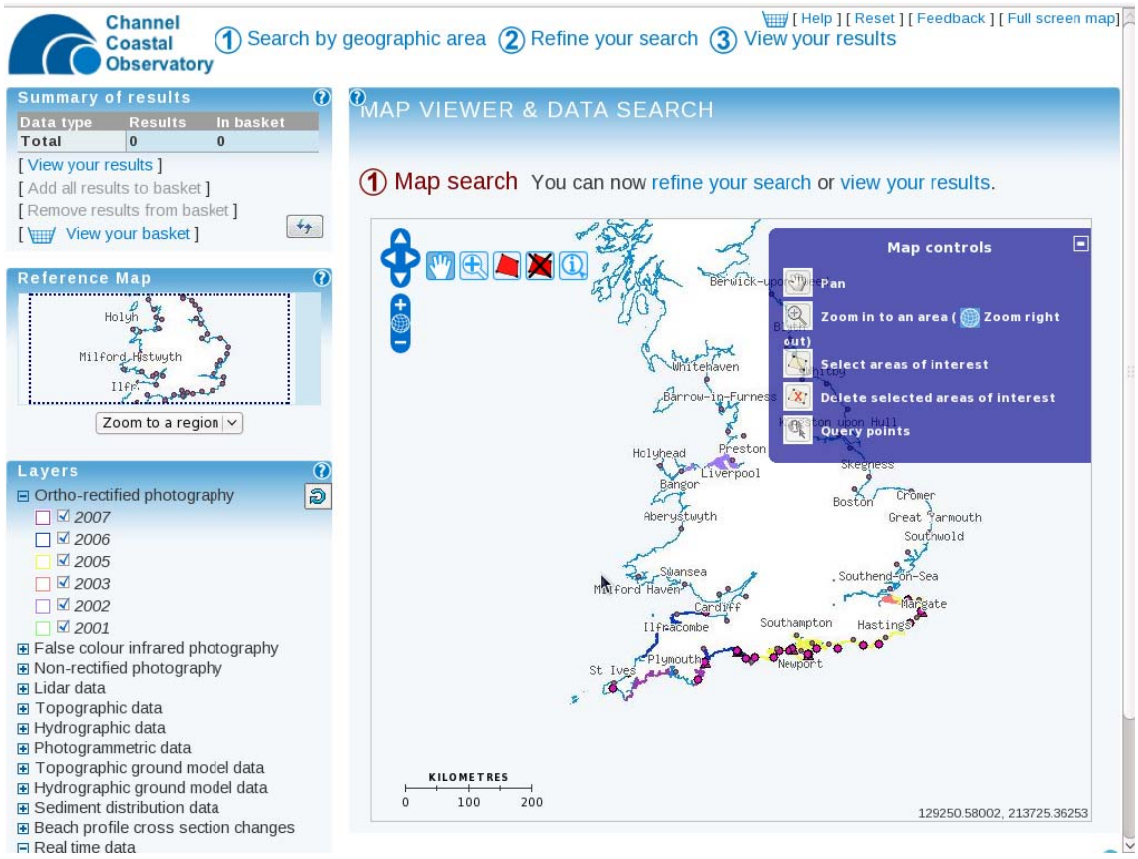

Figure 3.2: CCO Map Viewer and Data Catalogue

The Channel Coastal Observatory[1] (CCO) is the data management centre for the Regional Coastal Monitoring Programmes in South-East England. CCO is hosted by New Forest District Council in partnership with the University of Southampton and the National Oceanography Centre Southampton (NOCS), with technical development and delivery provided by the GeoData Institute.

The CCO website is an example of a Web Application in the domain of interest (environmental sensing) of SemsorGrid4Env; but through GeoData it also provides a key sensor network, historical data set (for modelling), and end user group for the Flooding case study (as described in greater detail in D7.1).

[1] http://www.channelcoast.org/

As such we examine here both the design and interfaces of the website as an example of an existing Web Application – albeit one that cannot take advantage of the as-yet non-existent SemsorGrid4Env semantic infrastructure – and comment (*in italics*) where good practice should influence SemsorGrid4Env Web Application development, and how SemsorGrid4Env might advance the features found on the CCO website.

Two major elements of functionality are found on the CCO website, presented to the user as separate options from the front page: "Realtime Data" and "Map Viewer and Data Catalogue". As shown in figure 3.3 these are implemented independently, principally due to historical development and design decisions.

The SemsorGrid4Env architecture will enable the swift discovery, addition, and integration of new sensor networks and data sources. The Web Application user should expect to be able to draw upon both archived and live data as needed – and the Application Tier must support this.

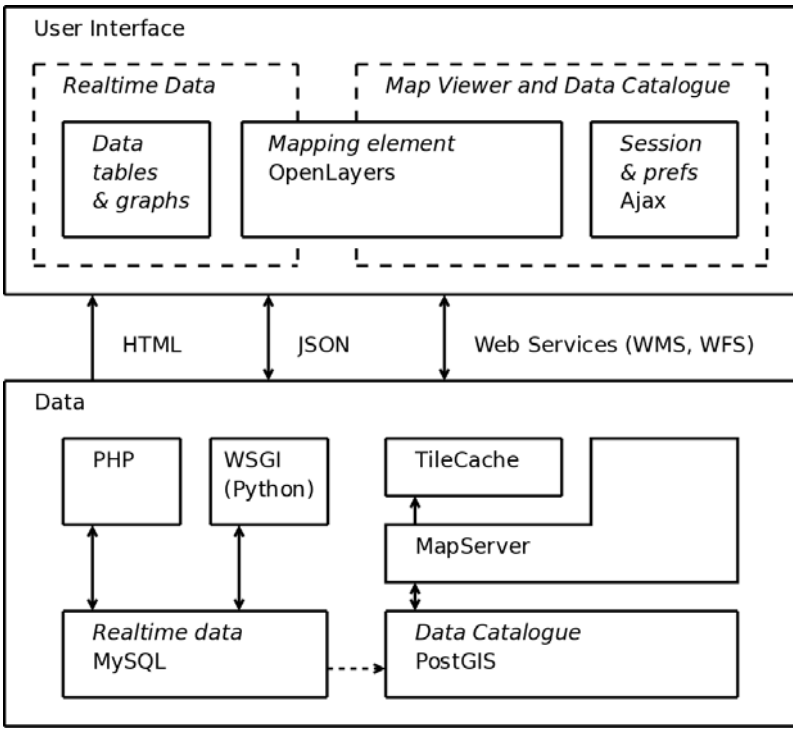

Figure 3.3: Channel Coast Observatory components

3.2.1. Map Viewer and Data Catalogue

The Map Viewer and Data Catalogue interface (figure 3.2) provides a means for the user to select a number of existing data sets for downloading. The user can select a geospatial area within a map viewer component through zooming and panning; further

detail of many types of data (e.g. topographic, hydrographic, infrared, lidar) can be plotted on the map at higher zoom levels by way of adding layers to the map (bottom left hand column of figure 3.2).

The User Interface comprises several visual components, which are populated with data retrieved using several different service interfaces (figure 3.3). *These visual components and services are examples of the elements we plan to implement in the Application Tier software library; while SemsorGrid4Env applications will be more wide-ranging, generic components with this kind of functionality will be re-usable across them. The underlying SemsorGrid4Env components and middleware will provide semantically enriched data and adapt to integrate new sensor sources, so future components will reflect and take advantage of this (e.g. dynamic generation/re-generation of the layers), but should also expose and access this more advanced functionality – where possible – using existing standard interfaces.*

The map viewing component is implemented using OpenLayers[2], a Javascript library for building web based geospatial applications. OpenLayers can present and integrate map data provided through several services and formats, including KML, Google Maps, Yahoo! Maps, and – as used by the CCO site – the Web Map Service (WMS) [OGC-WMS] and Web Feature Service (WFS) [OGC-WFS].

Session information – including data selection, preferences, and the "shopping basket" (which collates user selected sets data for ultimate download) – is handled by bespoke Ajax and server-side elements; the overall page is composed using the elements by PHP on the server.

MapServer[3] is used to feed map data to the OpenLayers component using exposed WMS and WFS services. The MapServer instance is backed by a PostGIS[4] database storing the map and feature data (PostGIS spatially enables PostgreSQL through additional support of geographic objects).

WMS and WFS are web-based interoperability service specifications published by the Open Geospatial Consortium (OGC). They are used to transfer data, typically used in a map view, from the server where the data is stored to the client where the map is presented to the user, or from one map server to another.The biggest difference between two services is that WMS returns raster formats (PNG, GIF, JPEG) whilst WFS uses the Geography Markup Language (GML – the OGC XML Schema for geographic features) to return data which is rendered at a later stage (typically by the client).

Images generated by MapServer are split by geographic area into "tiles"; these tiles can be pre-generated, and are cached by TileCache[5] (which implements a proposed WMS profile called WMS-C).

[2] http://www.openlayers.org/

[3] http://www.mapserver.org/

[4] http://postgis.refractions.net/

[5] http://www.tilecache.org/

3.2.2. Realtime Data

The CCO website also allows users to view "live" (last recorded) and recent data that has been received from the network of wave, tide, and meteorological sensors. Through a series of robust bespoke software components, data is uploaded from the sensors when they have network connectivity to a collating server, and from here the data is stored in a MySQL database.

Selecting the Live Data section of the CCO website, a user is presented with a mapping element that plots the locations of the sensors for which data is available. From here – or the list of sensor sites below the map element – a user is taken to a page for the selected sensor. Here tables and plots of the data are generated from the MySQL database and combined on the page using PHP.

The CCO live data feed system was developed before the advent of the OGC Sensor Observation System (SOS) [OGC-SOS]; if re-designed now this would be an obvious interface to implement as it complements the WMS and WFS already in use by the CCO and other GIS systems worldwide (indeed a partial implementation of SOS is included in recent versions of MapServer).

The SOS – an API for managing deployed sensors and retrieving sensor data and specifically "observation" data – is one of a series of futher OGC standards that comprise the Sensor Web Enablement (SWE) suite [OGC-SWE]. Other relevant SWE specifications include the Sensor Planning Service (SPS), Sensor Alert Service (SAS), and Web Notification Services (WNS) – together they can be used to implement a SOA for a sensor web with interoperable interfaces.

3.3. REST interfaces and mashups

The core of the SemSorGrid4Env WS-* architecture is based on tried and tested technologies, which are very suitable for a planned approach to designing the different components and the interfaces between them.

To enable the rapid development of web based thin-applications – mashups – we also adopt the representational state transfer (REST) paradigm and Resource Oriented Architecture pioneered by the web mashup community.

3.3.1. REST and Resource Oriented Architecture

Representational state transfer (REST) is a set of design principles which have been popularly and successfully adopted in many ("RESTful") web services, and is typically framed as an alternative to "heavyweight" web services, including as the WS-* family.

REST aims to capture the features of the Web which allow it to scale so successfully:

- everything is a resource which is addressable
- resources have multiple representations

- relationships between resources are expressed through hyperlinks
- all resources share a common interface with a limited set of operations
- client server communication is stateless

REST is not, however, defined in terms of, nor limited to, the web [Fie2000] (though HTTP meets the REST criteria) and while there have been attempts to clarify the application of REST to web services through definition of a Resource Oriented Architecture [RR2007] the term is still often loosely, sometimes incorrectly, applied.

In this document we refer to REST exclusively in the context of its application to Web Services, and so use the terms REST and Resource Oriented (Architecture) interchangeably.

The key principle of REST is the use of *resources* for specific things that we 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 contrasts with a potentially expansive set of operators (for RPC style web services) or message contracts (for SOA style web services). It also means HTTP is retained as an application layer protocol as per its originally design, rather than being re-purposed as a transport layer, e.g. for SOAP; 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.

A Resource Oriented Architecture also requires statelessness – that each HTTP operation is totally separate from any other. As such, any state the service has must also be exposed as a resource; an application client enters that state by accessing the URI for that resource; to enter another state a will use another URI.

Any application state a service requires 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.

3.3.2. Examination of the myExperiment REST API

myExperiment[6], jointly developed by the Universities of Manchester and Southampton, is a collaborative environment where scientists can safely publish their workflows and experiment plans, share them with groups and find those of others. Workflows, other digital objects and collections can now be swapped, sorted and searched in a manner inspired by Web 2.0 sites for e.g. photos and videos (fig 3.4).

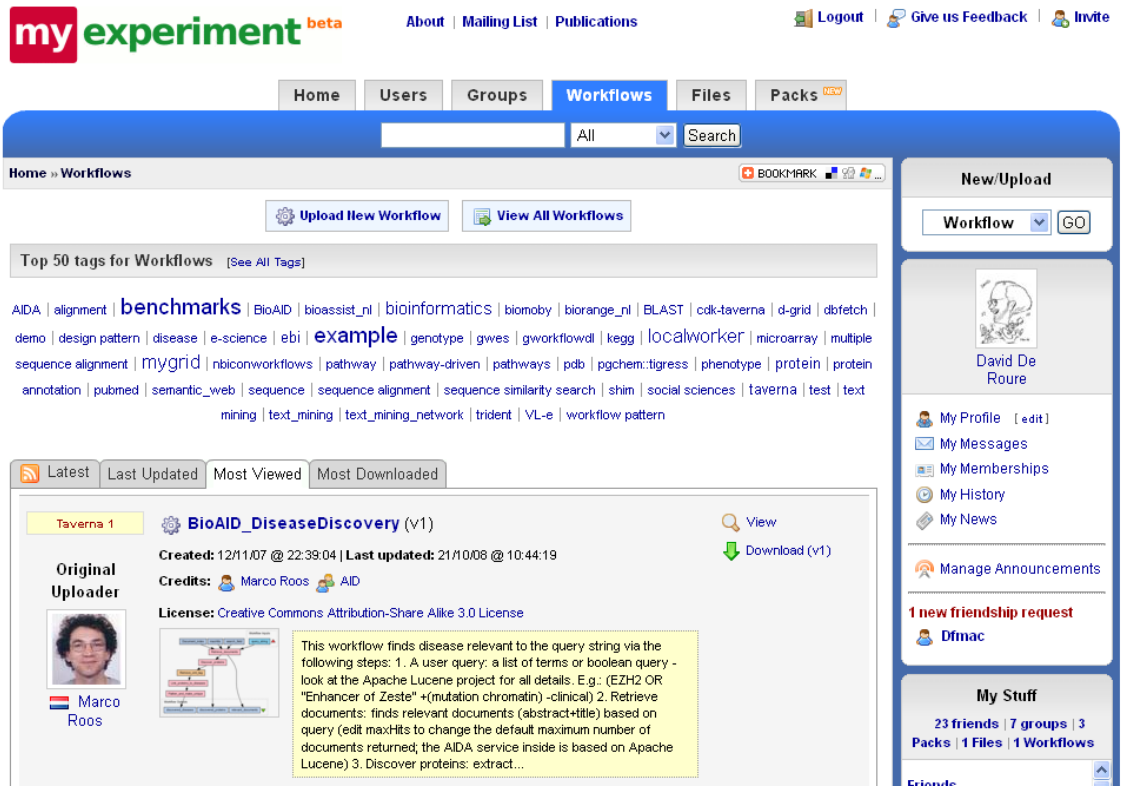

Figure 3.4: myExperiment workflows page

As well as the Web Application, myExperiment exposes its functionality through a REST API[7]; we explore a few of the interfaces from this API below. We do not highlight the myExperiment API as a perfect REST implementation (indeed some elements might be considered non-RESTful by some in the community), but because it is an API exposed by a Web Application and developed under the principles outlines above – the lessons and experience of developing the myExperiment Web Application and REST API will feed into the development of the SemsorGrid4Env Application Tier.

6 http://www.myexperiment.org/

7 http://wiki.myexperiment.org/index.php/Developer:API

For a detailed description of a fully RESTful web service see chapter 3 of Richardson and Ruby [RR2007], where the example is the Amazon S3 web storage service.

myExperiment provides a number of read only index representations. For example, workflows in myExperiment are available from the URI:

http://www.myexperiment.org/workflows.xml

Example output returned might be:

```
<?xml version='1.0' encoding='UTF-8'?>

<workflows>
        <workflow uri='http://www.myexperiment.org/workflow.xml?id=4'
        resource='http://www.myexperiment.org/workflows/4'>EBI
        InterProScan</workflow>

        <workflow uri='http://www.myexperiment.org/workflow.xml?id=5'
        resource='http://www.myexperiment.org/workflows/5'>DOI2PMID</workflow>

        <workflow uri='http://www.myexperiment.org/workflow.xml?id=6'
        resource='http://www.myexperiment.org/workflows/6'>Genome          annotation
        pipeline demonstrator workflow for Nucleic Acids Research</workflow>

        <workflow uri='http://www.myexperiment.org/workflow.xml?id=7'
        resource='http://www.myexperiment.org/workflows/7'>Fetch Dragon images from
        BioMoby v2</workflow>

        <workflow uri='http://www.myexperiment.org/workflow.xml?id=10'
        resource='http://www.myexperiment.org/workflows/10'>HUMAN   Microarray   CEL
        file to candidate pathways</workflow>

  ...

</workflows>
```

The server returns a list of workflows, for each detailing the resource of each workflow, a URI from which a representation of the resource can be retrieved, and a description of the workflow.

(Note that in this example content negotiation – i.e. to return XML – is encoded in the URI)

A GET on a specific workflow, e.g.

http://www.myexperiment.org/workflow.xml?id=173

might output:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<workflow uri="http://www.myexperiment.org/workflow.xml?id=173"
        resource="http://www.myexperiment.org/workflows/173" version="2">

  <title>Unique tags</title>

  <description>This workflow takes a comma separated list of tags and removes
        duplicate entries. Tags may have multiple words in them. An example string
        is "carrots,handbags,carrots,cheese".</description>

  <uploader uri="http://www.myexperiment.org/user.xml?id=22"
        resource="http://www.myexperiment.org/users/22">Don Cruickshank</uploader>

  <created-at>Tue Mar 11 16:52:42 +0000 2008</created-at>


        <preview>http://www.myexperiment.org/workflow/image/173/unique_tags_18054_2.
        png</preview>

 <svg>http://www.myexperiment.org/workflow/svg/173/unique_tags_18054_2.svg</svg>

  <license-type>by-sa</license-type>

        <content-uri>http://www.myexperiment.org/workflows/173/download/unique_tags_
        18054.xml</content-uri>

  <content-type>application/vnd.taverna.scufl+xml</content-type>

  <tags>

        <tag uri="http://www.myexperiment.org/tag.xml?id=555"
        resource="http://www.myexperiment.org/tags/555">example</tag>

        <tag uri="http://www.myexperiment.org/tag.xml?id=450"
        resource="http://www.myexperiment.org/tags/450">scampi</tag>

        <tag uri="http://www.myexperiment.org/tag.xml?id=760"
        resource="http://www.myexperiment.org/tags/760">design pattern</tag>

  </tags>

</workflow>
```

Here the returned XML confirms the resource and representation, also flagging the version of the workflow, the time it was created, a title, and a more verbose description. The creator of the workflow is provided (as a resource and description), with links by which the client can obtain a representation of the creator. Details about the workflow itself are given, including the application type, a URI to the workflow data, images that can be used to preview the workflow (as is done in myExperiment) and the license the workflow is released under.

The above examples are read-only. A POST of a workflow to:

http://www.myexperiment.org/workflow.xml

(i.e. the myExperiment service generates the new resource) with the input workflow data:


```
<?xml version="1.0"?>

<workflow>

  <title>Cove NetCDF visualization</title>

  <description>The Cove workflow reads oceanographic NetCDF data, does a simple
        processing step, writes the data out, then sends a web service message to
        the COVE visualization tool. It is a four step, sequential workflow.

  </description>

  <license-type>by-sa</license-type>

  <content-type>application/xaml+xml</content-type>

  <content encoding="base64" type="binary">
        PhdmOlNlcXVlbnRpYWxXb3JrZmxvd0FjdGl2aXR5DQogIHhtbG5zPSJodHRw
        Oi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3Jlc2VhcmNoLzIwMDcvU2NpZW50
        aWZpY1dvcmtmbG93Ig0KICB4bWxuczp3ZjOiaHR0cDovL3NjaGVtYXMubWlj

        ...

        eSBCYXkiIExpbms9Int3ZjpBY3Rpdml0eUJpbmQgV3JpdGUsUGF0aD1MaW5r
        fSIgSG9zdE5hbWU9ImxvY2FsaG9zdDoxMTIyMyIvPg0KPC93ZjpTZXF1ZW50
        aWFsV29ya2Zsb3dBY3Rpdml0eT4NCg==

  </content>

  <preview encoding="base64" type="binary">

         iVBORw0KGgoAAAANSUhEUgAAAo8AAADACAIAAADqYx68AACJHOlEQVR4nOyd
        B3gVxdqAZ+vZO89J772QQu9deq9SFBDsKF6xge23XRWvDXtBsQA2QIoiIAhI
        7ySUhJqQ3uup28s/exIQlatiEgyXfTkPz8lmM7s738xXZme+QRFBn8V5S+f

   ...

         k+u2DMpAiPL8t7/F0YD1/PuY8IBq7R3706uhVHvNGXmRonhPTbwKr7lUvApv
        uW7RSlvppXhNIWVVTkW07YOotZfcDwjvB10qiLukmikKwsupSr38P/JToTLY
        1NlZAAAAAElFTkSuQmCC

  </preview>

</workflow>
```


This contains similar elements to our previous example, but of course this a new workflow being uploaded, and here the workflow is encoded in a binary format.

After POSTing, the server would return the output:

```
<?xml version="1.0" encoding="UTF-8"?>

<workflow uri="http://www.myexperiment.org/workflow.xml?id=635"
        resource="http://www.myexperiment.org/workflows/635" version="1">

  <title>Cove NetCDF visualization</title>

  <description>The Cove workflow reads oceanographic NetCDF data, does a simple
        processing step, writes the data out, then sends a web service message to
        the COVE visualization tool. It is a four step, sequential
        workflow.</description>

  <uploader uri="http://www.myexperiment.org/user.xml?id=22"
        resource="http://www.myexperiment.org/users/22">Don Cruickshank</uploader>

  <created-at>Wed Jan 21 15:56:19 +0000 2009</created-at>

  <preview>http://www.myexperiment.org/workflow/image/635/preview.png
        </preview>

  <license-type>by-sa</license-type>

  <content-uri>http://www.myexperiment.org/workflows/635/download/
        cove_netcdf_visualization_88240</content-uri>

  <content-type>application/xaml+xml</content-type>

  <tags/>

</workflow>
```

Here the server has taken the uploaded workflow, assigned it a resource, and returned the details for the newly assigned resource as it did for an existing resource in our earlier example, i.e. the client is told that the workflow has been accepted and where representations are to be found.

myExperiment also exposes and RDF API[8].

3.3.3. REST and Mashups in the Application Tier

For purposes of this document, when we use the term mashup, we refer to a third party web application which integrates data from two or more separate sources, at least one of which is external to the web application's internal data structure, is accessed using an API, and is processed to generate greater user value than its original form .

[8] http://wiki.myexperiment.org/index.php/Developer:RDF

SemsorGrid4Env aims to promote rapid development and deployment of web applications utilising the project components - where the SemsorGrid4Env data is a mashup external source.

To enable mashup development we will expose SemsorGrid4Env data and functionality via REST API – the prevalent mechanism used and accepted by the mashup community.

This will build upon the data structures composed for, and by, the Web Applications – the Web Applications will gather data from services through the middleware and "add value", which is then accessed through the REST API. The REST API exposes functionality and data from the Web Application, not (directly) from the entire SemsorGrid4Env architecture.

Other parts of SSG4Env are designed to promote rapid adaptability and flexibility too, e.g. deployment and reconfiguration of sensors and data sources, so these facets must also be exposed to deliver an end-to-end solution using SemsorGrid4Env components (though as the Web Applications will make use of these features, the REST API should include them as a matter of course).

Development of the - resource centric - REST API will a process of identifying the resources required to encapsulate the application concepts, and structuring the hyperlinks between their representations.

We expect this to be a domain specific exercise based on the application and its requirements. While there are likely to be shared concepts between the two case studies, this is a manifestation of their overlapping domains (geo-location, sensors, measurement etc.) and there will also be resource concepts unique to each case study (and application).

Code to generate and structure resource representations will be within the Application Tier libraries – but again, while the library will be generic and enable re-use of functionality, both Web Applications and their REST APIs will be specific to some degree.

For example, the initial REST examples in deliverable D7.1 have resources structured by their environment (e.g. aquatic) and function (e.g. wave height).

While the purpose of the REST API is to expose SemsorGrid4Env to external developers for lightweight rapid application development and integration, we expect the development of the first mashups to take place within the project, to act as demonstrators and proof-of-concept (e.g. Alert Information System, D7.1).

3.4. Web Applications vs. Mashups

We have deliberately discussed Web Applications and Mashups separately, and it is worthwhile expanding upon why we have introduced this distinction.

The Web Applications utilise and fuse data from several sources, and build on a semantic integration service through the middleware – why are the these Web Applications not “mashups”?

In some senses this is just a terminology distinction we are imposing. Mashup development is a new, perhaps immature, and rather ill-defined field, and over the course of the SemsorGrid4Env project it is inevitable that further innovation and change will occur – possibly including the definition of the terms we use. Witness the introduction of “Enterprise Mashups”, for instance, which offer data fusion in a more traditional Service Oriented Architecture - “mashups as a service”, and would probably not be approved of by REST purists.

So more accurately the distinction is drawn between two approaches: the WS-* architecture exposed by the middleware to the Application Tier, and the REST architecture exposed by the Application Tier to mashups.

4. Evolution of the Application Tier

4.1. Comparison of WS-* and REST

The placing of WS-* and REST within the project architecture is due to their appropriateness: the middleware for bringing together the constituent lower level components into a cohesive system; and a REST API for rapid development of lightweight applications.

While there are differences in these approaches, they are not incompatible, and the SemsorGrid4Env project provides an interesting environment in which to study, compare, and evaluate the two with each other.

Nor do we see the division between WS-* and REST as fixed and permanent, merely an initial architecture which, from a pragmatic point of view, is required to move forward with software development. While WS-* is not currently the favoured approach within the mashup community, the APIs exposed by the middleware could be used directly (or through the Application Tier library) for mashup-style applications (perhaps in the spirit of the Enterprise mashups mentioned above).

Similarly, while we initially expect the REST APIs to be application specific - rather than exhaustively expose the complete set of SemsorGrid4Env functionality and data through a single REST API - it would be an instructive experiment to try and deliver a REST API for services currently exposed through the middleware – and to evaluate the effectiveness of these techniques.

We expect to report on the effectiveness and applicability of the two approaches in deliverable D5.3v2.

4.2. Semantic Mashups

It is clear that there is a very strong common element between REST and RDF – the primacy of resources. There is also a semantic thread running through all the components in SemsorGrid4Env, so it is a natural extension to enhance mashups, as well as our Web Applications, with these semantics.

The overlap between the REST principles and architectural style, and the Semantic Web's foundation upon RDF, is a topic we will explore further and in depth. A first step will be returning RDF from our REST API.

We will also investigate exposing SemsorGrid4Env data through the Linking Open Data community, where several sets of geospatial data are already included, centred around the GeoNames database.

5. References

[DGS2009] De Roure, D., Goble, C. and Stevens, R. (2009) The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows. Future Generation Computer Systems 25, pp. 561-567.

[DG2009] De Roure, D. and Goble, C. (2009) "Software Design for Empowering Scientists," IEEE Software, vol. 26, no. 1, pp. 88-95, January/February 2009

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

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

[OGC-SWE] Botts, M., Percivall, G., Reed, C., and Davidson, J. (2007) “OGC Sensor Web Enablement: Overview and High Level Architecture”, version 3, OGC 07-165

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

[OGC-WMS] de la Beaujardiere, J. (ed.) (2006) “OpenGIS Web Map Service (WMS) Implementation Specification 1.3.0” OGC 06-042

[RR2007] Richardson, L. and Ruby, S. (2007) “RESTful Web Services, O'Reilly & Associates, Sebastopol, California, May 2007, ISBN 0-596-52926-0.