SemSorGrid4Env

FP7-223913

Deliverable

D5.4

Evaluation of the High-level Application Programming Interfaces

Alex J. Frazer (Editor)[1], David De Roure[1], Kirk Martinez[1], Bart Nagel[1], Kevin R. Page[1], Juanjo Aparicio[2] and Josep Rodriguez[2]

[1] University of Southampton

[2] TechIdeas Asesores Tecnológicos, S.L

08/09/2011

<Status: Final>

<Scheduled Delivery Date: 31/08/2011>

Executive Summary

This document describes several evaluations of the HLAPI component from several different perspectives, before identifying areas for future work and drawing conclusions on the component's suitability. Stress tests are performed in order to determine how the component handles an increasingly overloaded server. Performance benchmarking allows us to gauge how well the API scales as more observations are processed, which in turn helps us to decide whether the API meets the requirements for RESTfulness. User evaluations are performed with both domain developers and domain users, to determine whether the HLAPI is useful in developing semantic mashups, and whether those mashups can be useful from a domain user's perspective. Finally, areas for future work are identified and conclusions are drawn.

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:

- Requirements for a RESTful API taken from [D5.1]

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			
EU Project officer	Antonios Barbas		

Deliverable	**Number**	D5.4	**Name**	Evaluation of the High-level Application Programming Interfaces		
Task	**Number**	5.4	**Name**	Evaluate the High-level Application Programming Interfaces		
Work package	**Number**	WP5				
Date of delivery	**Contractual**		31/08/2011	**Actual**	08/09/2011	
Code name	D5.4			**Status**	draft ☐	final ☑
Nature	Prototype ☐ Report ☑ Specification ☐ Tool ☐ Other ☐					
Distribution Type	Public ☑ Restricted ☐ Consortium ☐					
Authoring Partner	SOTON					
QA Partner	TI					
Contact Person	Kirk Martinez					
	Email	km@ecs.soton.ac.uk	**Phone**	+44 2380 594491	**Fax**	+44 2380 592865
Abstract (for dissemination)	The high-level API service is designed to support rapid development of thin web applications and mashups beyond the state of the art in GIS, while maintaining compatibility with existing tools and expectations. It provides a fully configurable API, while maintaining a separation of concerns between domain experts, service administrators and mashup developers. It adheres to REST and Linked Data principles, and provides a novel bridge between standards-based (OGC O&M) and Semantic Web approaches. This document presents results of testing, and evaluates them, before drawing conclusions and providing avenues for future work.					
Keywords	HLAPI, REST, Linked Data, Mashups, Sensor Networks					

Version log/Date	**Change**	**Author**
0.1 / 20/07/2011	**Initial document planning**	**A. Frazer**
0.2 / 01/08/2011	**QA version complete**	**A. Frazer**
0.3 / 30/08/2011	**Stress testing results added**	**A. Frazer**
1.0 / 08/09/2011	**Final draft**	**A. Frazer**

Project Information

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

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

Table of Contents

Glossary

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

1. Introduction

This document describes several evaluations of the HLAPI component from several different perspectives, before identifying areas for future work and drawing conclusions on the component's suitability.

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

1.1. Evaluation objectives

In this deliverable, the HLAPI component is evaluated in several different ways. Stress tests are performed in order to determine how the component handles various system failures. Performance benchmarking allows us to gauge how well the API scales as more observations are processed, which in turn helps us to decide whether the API meets the requirements for RESTfulness. Finally, user evaluations are performed with both domain developers and domain users, to determine whether the HLAPI is useful in developing semantic mashups, and whether those mashups can be useful from a domain user's perspective.

1.2. Document Outline

The document begins with details of component stress tests, carried out to determine how the HLAPI engine copes with drastic system failures. Next, an evaluation of the API itself is presented from several different perspectives: whether it meets the requirements for RESTfulness defined in D5.1; whether a domain developer can use the resulting API succesfully to develop semantic mashups; and whether the affordances of the API are useful to domain users. Finally, areas for future work are identified and conclusions are drawn.

2. Component Stress Tests

In this chapter we present a series of stress tests, designed to determine how well the HLAPI engine copes with a number of interruptions to its normal operation.

2.1. Stress test process

The web services composing the SemsorGrid4Env middleware have been tested using a stress test tool called *LoadUI*[1]. The SemsorGrid4Env middleware has been installed on a virtual machine guest operating system, while the stress test tool runs from the Host operating system (i.e. the operating system that runs physically on the machine). By doing so, the stress test tool (which is quite demanding in both CPU and RAM usage) won't affect the status of the web service. At the same time, the network won't affect either the process of the test, as it is not used.

To measure the impact that the stress test has created on the virtual machine, a measurement of the memory status will be performed on both machines – host and guest – before and after the stress tests.

The stress test itself will be split into three different phases:

- During phase I, the purpose is to discover the limit of requests per second the server is capable of handling without starting to queue requests.
- During phase II, the server will be suffocated with a quantity of requests per second higher than the limit obtained during phase I, expecting the server to queue requests until some of them are discarded. Typically, a fresh install of Apache Tomcat will need 1000 queued requests before starting to discard incoming connections).
- Phase III will try to discover if the server is able to, after being stressed on phase II, return to normality, by serving all the queued connections.

2.1.1. Requests performed

LoadUI uses a set of predefined test cases to send to the server, and loops them repeatedly to perform the stress tests, so the requests performed are a set of limited and well-known SOAP requests. The test cases tend to represent the most common requests that the server will have to handle and the requests that would have a bigger impact in the server in terms of performance, and memory usage.

The HLAPI, unlike the other SemsorGrid4Env services, provides both a backend (which interacts with the rest of the Semsorgrid4Env middleware) and a frontend (which will be the interface exposed to the users' requests).

Therefore, tests are performed on the frontend while the backend is running in "IQS-driven mode" (as described in [D1.4v2]), continually fetching data from a local instance

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

of the IQS-WS, working with instances of both SNEE-WS and CCO-WS on the same machine. This recreates the typical behavior of a SemsorGrid4Env as accurately as possible.

Finally, the stress tests on the HLAPI are performed as HTTP requests to the RDF representation of the RESTful API, which returns the data obtained from the CCO-WS in a standard RDF format. The complete URL used to stress the HL-API is:

http://rdf.localhost/observations/cco/torbay/latest

2.1.2. Test Platforms

The first platform to be used on the stress test (*Test platform A*) represents a low resource server that could be maintained by a developer intending to deploy the SemsorGrid4Env middleware locally, instead of using the SemsorGrid4Env dedicated servers.

The CPU of the computer used to perform the tests is an AMD Turion™ X2 Dual-Core Mobile RM-72, 3Gb of RAM memory (one completely dedicated to the host OS, resulting in a 2Gb RAM memory). The Guest OS is a 32-bit Ubuntu Natty Narwhal (11.04), whilst the Guest OS is a 64-bit Ubuntu Natty Narwhal (11.04) server edition.

The output of the command `free -m` before starting the tests is:

```
                 total   used   free   shared   buffers   cached
Mem:             2767    2516   251    0        2         109
-/+ buffers/cache:       2404   363
Swap:            8110    285    7825
```

Meanwhile, before starting the tests, the guest OS `free -m` results are:

```
                 total   used   free   shared   buffers   cached
Mem:             998     817    181    0        16        91
-/+ buffers/cache:       709    289
Swap:            1019    74     945
```

The second platform to be tested (*Test platform B*) represents a medium resource server which could be quite affordable to act as a SemsorGrid4Env middleware server.

The CPU of the computer used to perform the tests is an Intel Pentium® Dual-Core E6300@2.80GHz, 8GB of RAM (with two GB dedicated entirely to the Guest OS). The OS used is a 64-bit Ubuntu Natty Narwhal (11.04) on both Host OS and Guest OS, the only difference being that the Host OS is a desktop edition, while the Guest OS is a server edition.

The output of the command `free -m` before starting the tests is:

```
             total   used   free   shared   buffers   cached
Mem:         7875    7784   90     0        371       2010
-/+ buffers/cache:   5402   2472
Swap:        4094    0      4094
```

N.B. the host OS has plenty of free RAM and swap space, so no swapping is expected while performing the tests.

Meanwhile, before starting the tests, the guest OS `free -m` results are:

```
             total   used   free   shared   buffers   cached
Mem:         2008    1300   707    0        39        219
-/+ buffers/cache:   1042   965
Swap:        1019    0      1019
```

2.2. Stress test results

2.2.1. Test platform A

After the stress tests on the HLAPI, the Host OS `free -m` results are:

```
             total   used   free   shared   buffers   cached
Mem:         2767    2672   95     0        9         186
-/+ buffers/cache:   1876   891
Swap:        8110    160    7950
```

While at the guest OS are:

```
             total   used   free   shared   buffers   cached
Mem:         998     991    6      0        14        477
-/+ buffers/cache:   493    505
Swap:        1019    20     999
```

Test summary

- Phase I: during this phase, the HLAPI has been serving 6.95 requests per second, even though it has been able to serve almost 9 requests per second without notably incrementing the number of running requests for a period of time. After 50.5 seconds, it has served a total of 351 requests, without discarding any requests, nor failing at any request.
- Phase II: during this phase, the number of requests per second has been increased to 59.36 which, after 36.5 seconds, forced the server to discard 1049 requests. During this phase, only 3.36 requests per second have been completed by the server, which means that it has reduced its performance to the half of it's initial performance while being stressed.
- Phase III: finally, the HLAPI, has been able to recover from the stress test using 217.5 seconds, and serving 5.05 requests per second, which means a better performance that the one obtained during phase II, but not as good as the one obtained during phase I.

Statistics of the test

Phase	Seconds	Processed requests per second	Completed requests per second
I	50.5	7.0495049505	6.9504950495
II	36.5	59.3698630137	3.3698630137
III	217.5	5.2091954023	5.0574712644

Phase	Queued	Running	Completed	Requests	Failed	Discarded	Processed
I	5	100	351	451	0	0	356
II	995	100	123	123	0	1049	2167
III	0	0	1100	1000	0	33	1133

2.2.2. Test platform B

After the stress tests on the HLAPI, the Host OS `free -m` results are:

```
             total       used       free     shared    buffers     cached
Mem:          2767       2672         95          0          9        186
-/+ buffers/cache:       1876        891
Swap:         8110        160       7950
```

While the guest OS returns the following:

```
             total       used       free     shared    buffers     cached
Mem:           998        991          6          0         14        477
-/+ buffers/cache:        493        505
Swap:         1019         20        999
```

Test summary

- Phase I: during this phase, the HLAPI has been serving 7.83 requests per second, even though, it has been able to serve almost 11 requests per second without incrementing notably the number of running requests for a period of time. After 221.5 seconds, it has served a total of 1735 requests, without discarding, nor failing at, any request.
- Phase II: during this phase, the number of requests per second has been increased to 36.16, which eventually, after 39.5 seconds, has forced the server to discard 131 requests. During this phase, the server has served 7.74 requests per second.
- Phase III: finally, the HLAPI has been able to recover from the stress test using 131.5 seconds, and serving 8.38 requests per second.

Statistics of the test

Phase	Seconds	Processed requests per second	Completed requests per second
I	221.5	7.8735891648	7.8329571106
II	39.5	36.1518987342	7.746835443
III	131.5	8.5931558935	8.3802281369

Phase	Queued	Running	Completed	Requests	Failed	Discarded	Processed
I	9	100	1735	1835	0	0	1744
II	991	100	306	2141	0	131	1428
III	0	0	1102	3143	0	28	1130

3. Component Evaluation

The component evaluation consists of three main sections: performance benchmarking of the HLAPI engine and the API that it generates; the suitability of the API from a domain developer's perspective; and the usefulness of the API from several domain users' perspectives, via the mashups created by the domain developer.

3.1. HLAPI engine performance benchmarking

In order to evaluate the performance of the API generated by the HLAPI engine, several benchmark tests were carried out. The objective of the benchmarking process was to determine how access times for both the RDF triple-store and the serialised HTTP resources would change as the number of serialised observations increased.

3.1.1. Benchmarking procedure

The procedure involved making a series of requests to resources over time, as more observations were processed and serialised by the HLAPI engine running in "database-driven mode" as described in [D5.2v1]. This would allow us to determine the effect of the total number of serialised resources on the access times of the API.

The following SPARQL queries were posed to the RDF triple-store, and the response times were recorded:

- ```
  SELECT * WHERE {
        ?obs a ssn:Observation.
  } LIMIT 100
  ```

- ```
  SELECT * WHERE {
        ?obs a ssn:Observation;
        ssn:observedBy ?sensor.
  } LIMIT 100
  ```

- ```
  SELECT * WHERE {
        ?obs a ssn:Observation;
        ssn:observedBy ?sensor;
        ssn:observationResultTime ?obsTime.
        ?obsTime time:hasEnd ?time.
  } LIMIT 100
  ```

In addition, a request was made to the serialised RDF resource at the following URI:

```
http://id.api.channelcoast.org/observations/cco/folkestone_met/
      rainfall/20110701
```

Again, the response time was recorded at intervals as more observations were processed and serialised.
```

3.1.2. Results

The results of the benchmarking procedure were as follows:

Number of serialised triples	SPARQL query response time (ms)			HTTP GET request response time (ms)
	Query 1	Query 2	Query 3	
127500	0.0811	0.0536	0.0849	44.33
230865	0.0584	0.0602	0.0985	29.00
310380	0.0443	0.0430	0.0952	28.33
421545	0.0566	0.0784	0.0993	44.33
496785	0.0343	0.0412	0.0799	42.33
3532455	0.0610	0.0710	0.1114	33.67
Correlation coefficient	**0.0715** **No correlation**	**0.4259** **Weak +ve correlation**	**0.7094** **Strong +ve correlation**	**- 0.1928** **No correlation**

3.1.3. Evaluation of benchmarking results

The results shown above suggest that as the number of processed observations (and therefore, the number of triples in the triple-store) increases, the access time to retrieve triples from the triple-store also increases. It also suggests that the increase is exacerbated by the complexity of the SPARQL query posed to the triple-store. However, the data also suggests that there is no correlation between the number of triples processed by the HLAPI engine and the time it takes to access any of the serialised API resources.

As such, the resulting API could be considered to have met the RESTful goal of being scalable. As the API also meets the other requirements of a REST architecture (statelessness, resources as primary objects, URIs to identify these resources, a limited set of exposed operations), the API itself could be considered a true RESTful API. In addition, it could be considered an improvement over accessing the RDF triple-store directly, as its access times will not be affected by the number of resources available in the way that the triple-stores access times will.

3.2. Domain developer evaluation

Discussions with the developer responsible for the semantic mashups suggested that the API exposed by the HLAPI engine was an improvement over typical WS-* interfaces. Depending on the development language involved, previous experiences developing

with WSDL and SOAP involved large amounts of stub code generation, which relied on the WSDL files remaining consistent. If the interfaces changed, the stub code would have to be regenerated again. A SOAP message had to be constructed and a further SOAP response had to be processed in order to extract the data.

With the HLAPI engine's RESTful API, the data can be retrieved easily using standard HTTP libraries, and in some cases, directly through the library used to process the RDF. If the interface (i.e. the set of URIs) changed, the URI requested in the code could simply be changed, and the mashup could function as before.

In addition, the errors thrown when accessing a WS-* Web Services can be from a number of sources (a fault with the SOAP message, an error in processing the data at the server, various network errors). These errors have, in the developer's experience, often been poorly defined, making debugging difficult. Conversely, the errors thrown when accessing the RESTful HLAPI will all come from a well-defined set of HTTP access errors, making it clear what has gone wrong whenever an error occurs.

3.3. Domain user evaluations

As part of the HLAPI evaluation, the mashups developed above were presented to two external agencies: Coastal Defence Partnership, concerned with flood management around the UK; and Associated British Ports, responsible for shipping and port management around the British Isles. In both cases, evaluation focused on the usefulness of exposing data as a high-level, RESTful API, as well as on the usefulness of the mashups themselves.

3.3.1. Coastal Defence Partnership (CDP)

Discussions with CDP primarily focused on the floodgate-closing mashup, with the bird watching and "surf status" mashups shown as examples of unintended data reuse.

Overall, the floodgate mashup was seen to be potentially useful. While the integration of street map data was not necessarily useful from CDP employees (who already possess local street knowledge), it has potential use for external contractors who are often employed during emergency situations. The Linked Data offered by the HLAPI could support this functionality in future, by enabling semantic inferences to be made on the data based on a role ontology describing potential users and their requirements.

It was suggested that other agencies (for example, civil contingency planners) would benefit from an augmented version of the mashup, updated to include information on social vulnerability in flood areas. By providing sensor data that obeys Linked Data principles, the HLAPI could link into this data with relative ease, provided that it were published as linked data itself.

In addition, the mashup could be useful to other agencies concerned with different types of flooding beyond the coastal flooding monitored by CCO sensors. By publishing the HLAPI in accordance with a known ontology, any other sensor data published as a HLAPI based on the same ontology could be easily consumed by the mashup with minimal changes.

Moving beyond the data already available as Linked Data, CDP also considered the possibility of incorporating their own (non-Linked) data into the mashup. While they already incorporate data-generating tools into their working practice (such as GPS-enabled digital cameras), this data is not currently recorded or published as Linked Data. There is genuine potential here to publish this data as a RESTful, Linked Data API via the HLAPI engine, allowing the data to be reliably incorporated into semantic mashups.

3.3.2. Associated British Ports (ABP)

ABP were presented with the same set of mashups as CDP, with the addition of a "pilot dashboard" developed around a specific ABP-related use case.

Again, the mashup was considered useful in its current state, although the addition of the position of ships in the vicinity of those assigned to the pilot was seen as desirable. The Linked Data principles employed in publishing the HLAPI would make this relatively trivial on the condition that ship data were also published based on the same principles.

The potential to switch between different data sets for different tasks (e.g. oil spill handling) was seen to be desirable. In the same way as the CDP mashup, the semantic nature of the HLAPI data would enable inferences to be performed based on a role ontology, limiting the information presented through the dashboard in relation to a particular task.

Furthermore, in a similar manner to the GPS-cameras used by CDP, ABP already record shipping data and publish it through their website. The ability to expose this data as a semantic Linked Data API was seen as valuable for future applications, and the HLAPI engine would be capable of generating this API, given the correct configuration and access to the original data.

A robust data schema already exists for representing the kind of data ABP would need. However, it is politically difficult to interfere with this data format, and as such there has been little effort to expose the data for use in mashups. The HLAPI engine's approach to exposing data is perfectly suited to this situation: the original data in its original schema-compliant format can remain untouched, while the linked data is converted to conform with our own ontologies and published entirely separately.

Similarly, much of the data that is useful to ABP is held in centralised databases, with the administrators reluctant to offer direct access to third parties. Again, the HLAPI engine would be ideal to expose this data. Because the observation data is pushed from the original database to the HLAPI engine before it is inserted into the database, the database itself is never actually accessed by mashup and webapp developers. This allows live data to be published in its entirety – in a particularly useful format – with very little overhead for the database administrator. As such, the HLAPI engine could be a helpful tool in negotiating data exposition with reluctant data producers.

3.4. Summary

The API generated by the HLAPI engine has been shown to meet the requirements of a RESTful interface. Combined with the Linked Data formats made available, the resulting API can be seen to satisfy the various requirements discussed in [D5.1].

The API is useful from a domain developer perspective, as it offers a lightweight data interface more suited to agile mashup development than a typical WS-* data interface. The Linked Data output offered by the HLAPI enables development of rich, semantic mashups that offer genuinely useful functionality to domain users. However, this data is of limited use in isolation, and will rely on further related Linked Data sources to make a truly rich – and therefore, truly useful – semantic mashup.

4. Future Work and Conclusions

A number of opportunities for future work exist, extending the HLAPI engine's functionality beyond the scope of the SemsorGrid4Env project.

A number of potential optimisations to the Serialiser component could be implemented. As the Serialiser effectively serialises five separate sets of resources per observation (i.e. one per output format), an obvious next step would involve separating these serialisations into individual threaded processes. This would allow the HLAPI engine to take full advantage of modern, multi-core processors, with particular potential to improve the speed of the HLAPI engine's "batch" mode.

A further optimisation could be made to the way in which the resources are serialised. Currently, each time an observation is processed, its relevant files are opened, the new observation data is inserted, the data is written to the file and the file is closed. When the next observation is processed, the same procedure is carried out – potentially using exactly the same files, as observations tend to arrive sequentially. If the need to open and close the same set of files for *each individual observation* could potentially be avoided (perhaps by temporarily storing the current data in memory until a different file were required), the overhead of reading from and writing to a file could be mitigated. Again, this could have particular impact on the "batch" update mode, where high a high volume of observations are serialised, often to the same files consecutively.

Further work could be carried out to integrate the functionality of the HLAPI engine's various support scripts into the engine itself. Currently, a degree of the HLAPI engine's functionality (particularly batch update processing, the generation of collection summaries, and the processing of sensor descriptions) is performed by a set of auxiliary bash scripts. Ideally, this functionality would be rolled into the core functionality of the HLAPI engine, reducing the complexity of its deployment. Further performance benefits could also emerge, due to the degree of statefulness held by the HLAPI engine. This could reduce the need for some of the exhaustive searching necessary in the current post-hoc support scripts.

The processing of sensor descriptions could benefit further from being handled within the HLAPI engine itself. Currently, the URIs (and corresponding locations on disk) of sensors within the API are effectively hard coded in relation to the URIs configured in the API mapping document. This is contradictory to the configurable nature of the rest of the API, and it would be desirable to make the sensors within the API as configurable as the other elements. This would require further functional development of the URIMinter and Serialiser components, as well as further conceptual design and developement of the API mapping document.

On the subject of mapping documents, there exists an opportunity to refactor the Processor subcomponent of the HLAPI engine, in order to receive its ontology mapping files in the S2O format, rather than the current modified version of the D2RQ mapping language. Over the course of the SemsorGrid4Env project, S2O has extended the

original R2O mapping language to include most of the enhancements made to the D2RQ language for use with the HLAPI engine. With the added benefit of being simpler to write, S2O could be more suitable for the HLAPI engine with only a small amount of extra functionality required.

One area of further work concerns the publication of additional APIs. Potential to develop APIs for CDP and ABP have already been discussed in this document, requiring effort to capture their respective existing data sources and convert them into an observation-centric API. Further APIs could also be published based on data scraped from existing Web resources, such as the Wunderground weather reporting service. This would involve comparably less effort, with the majority of work focusing on creating mapping documents and selecting the correct ontologies to express the information as useful Linked Data.

Development of sources of Linked Data could also provide material for future work. The semantic mashups developed using the HLAPI, while successful, were driven by a limited set of Linked Data sources. Exposing the internal data used by ABP and CDP would be beneficial initially. However, making a richer set of genuine Linked Data sources available to link into this data would make the resulting mashups much more robust, as well as providing opportunity for further reuse in future mashups.

4.1. Conclusions

Overall, the HLAPI component can be seen as successful. The HLAPI engine is capable of producing a RESTful API for sensor observations that scales effectively as more observations are published. The resulting API has been shown to be effective both from a developers' point of view, as well as from the perspective of domain users engaging with mashups built on the API.

The areas for future work highlight various ways in which the HLAPI component could be improved and extended. By implementing these changes and building on the current functionality offered by the HLAPI, developers building on the API could produce richer applications, with the resulting added functionality benefitting end users.

References

[D1.4v2] Aparicio, J. and Rodríguez López, J. (2011) "Reference SemsorGrid4Env Implementation – Phase II", Deliverable D1.4v2, SemsorGrid4Env

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

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