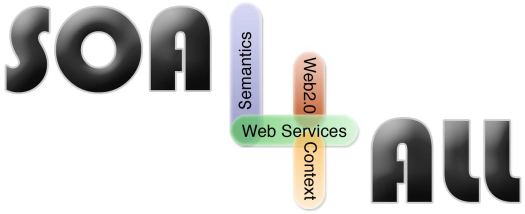

Project Number: **215219**

Project Acronym: **SOA4All**

Project Title: **Service Oriented Architectures for All**

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D2.1.4 Service Provisioning Platform Second Prototype

Activity N: 1	Fundamental and Integration Activities	
Work Package: 2	SOA4All Studio	
Due Date:	31/08/2010	
Submission Date:	31/08/2010	
Start Date of Project:	01/03/2008	
Duration of Project:	36 Months	
Organisation Responsible of Deliverable:	The Open University	
Revision:	1.0	
Author(s):	Maria Maleshkova	OU
	Guillermo Álvaro Rey	iSOCO
	Alex Simov	ONTO
	Bruno Renie	OU
	Dong Liu	OU
Reviewers:	Sven Abels	TIE
	Reto Krummenacher	UIBK

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	21/06/2010	Document structure, table of contents	Maria Maleshkova (OU)
0.2	24/07/2010	Initial version	Maria Maleshkova (OU)
0.3	25/07/2010	Improved draft with screen-shots and more detailed description	Maria Maleshkova (OU)
0.4	09/08/2010	Completed descriptions of the Feedback Framework and the iServe	Maria Maleshkova (OU), Guillermo Álvaro Rey (iSOCO)
0.5	10/08/2010	First version of user interface section	Maria Maleshkova (OU)
0.6	11/08/2010	Complete initial draft	Maria Maleshkova (OU), Guillermo Álvaro Rey (iSOCO), Alex Simov (ONTO), Bruno Renie (OU), Dong Liu (OU)
0.7	12/08/2010	Refined draft	Maria Maleshkova (OU), Guillermo Álvaro Rey (iSOCO), Alex Simov (ONTO)
0.8	16/08/2010	Changes following the reviewers' recommendations and comments	Maria Maleshkova (OU)
1.0	19/08/2010	Final draft	Maria Maleshkova (OU), Guillermo Álvaro Rey (iSOCO), Alex Simov (ONTO), Bruno Renie (OU), Dong Liu (OU)

Table of Contents

List of Figures

Glossary of Acronyms

Acronym	Definition
AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
CSS	Cascading Style Sheets
D	Deliverable
DSB	Distributed Service Bus
EC	European Commission
EXT GWT	Extended GWT
FOAF	Friend Of A Friend
FP	Framework Program
FP7	The 7th Framework Program
GUI	Graphical User Interface
GWT	Google Web Toolkit
hRESTS	HTML format for describing RESTful Services
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IT	Information Technology
RDF	Resource Description Framework
REST	Representational State Transfer
SA-REST	Semantic Annotations for RESTful Services
SOA4All	Service-Oriented Architectures for All
SOAP	Simple Object Access Protocol
SWS	Semantic Web Service
UI	User Interface
URI	Uniform Resource Identifier
WP	Work Package
WS	Web Service
WSDL	Web Service Description Language
XML	eXtended Markup Language
XSLT	XSL Transformations

Executive Summary

Web services have already achieved a solid level of acceptance and play a major role for the rapid development of loosely-coupled component-based systems, within and between enterprises. However, the wider adoption of web service technologies is hindered by the fact that currently most service tasks require extensive manual effort and as a result, web service-based applications suffer from a lack of automation. Research on semantic web services (SWS) has been devoted to reduce the extensive manual effort required for manipulating and using web services. The main idea behind this research is that tasks such as discovery, negotiation, composition and invocation can have a higher level of automation, when services are enhanced with semantic descriptions of their properties. These SWS are amenable to automated reasoning, thus paving the way for the application of knowledge-based algorithms to better support the automation of service-related tasks.

The SOA4All Provisioning Platform supports the creation of semantic web service descriptions by leveraging users as the main source of information, using both direct user input and automated information processing based on prior user-provided input. In addition, it enables users to find relevant services and to put them together in order to compose new, more complex services.

The here presented second Provisioning Platform Prototype focuses on describing the newly implemented functionalities of iServe and the Annotations Recommender. iServe enables the publishing, browsing and searching of semantic web services, while the Annotations Recommender enables the semi-automatic creation of semantic descriptions by suggesting suitable annotations to the user. In addition, we describe the improvements of the implementations of the MicroWSMO and WSMO-Lite Editors, which support users in creating semantic descriptions of RESTful and WSDL-based services. This deliverable also presents the further development work done for the Feedback Framework, which enables users to rate and recommend services based on their experience.

The purpose of this deliverable is twofold. First, it serves as a documentation of the second Provisioning Platform Prototype, providing information about installation and configuration guidelines as well as a description of the main functionalities of the components. Second, it represents a user manual that gives instructions on how to use the different GUI elements and how to complete simple and complex tasks by using the editors, the service browser and the annotations recommender.

1. Introduction

This deliverable introduces the components and features of the second Provisioning Platform Prototype. It describes improvements of existing component implementations, including the MicroWSMO Editor, the WSMO-Lite Editor and the Feedback Framework, and focuses on two new implementation contributions, not previously described in Deliverable 2.1.3: iServe and the Annotations Recommender.

1.1 Purpose and Scope

The purpose of this deliverable is twofold. First, it serves as a documentation of the second Provisioning Platform Prototype, providing information about installation and configuration guidelines as well as a description of the main functionalities of the components. Second, it represents a user manual that gives instructions on how to use the different GUI elements and how to complete simple and complex tasks by using the editors, the service browser and the annotations recommender. It guides the user through the process of creating semantic service descriptions and describes the different tool functionalities.

This deliverable is structured as follows: This section provides a general overview of the second Provisioning Platform Prototype, including a list of the functionalities implemented by each of the prototype components. Section 2 includes installation and configuration guidelines, followed by a detailed documentation of each of the components, including descriptions of each of the GUI elements. Finally, Section 3 provides a short conclusion.

1.2 Service Provisioning Platform Prototype

The design of the Provisioning Platform foresees that it consists of six main components, including:

1. The **Simple SWS Editing Framework**, which supports users in creating and browsing semantic web service descriptions. This includes a service browser, as well as two editors for the creation of MicroWSMO and WSMO-Lite semantic descriptions.
2. The **Annotations Recommender**, which reduces the manual effort required by users, while annotating services, by automatically suggesting suitable annotations.
3. The **Templates and Service Creation Wizards Management Framework**, which supports the reusing of service compositions in the form of templates.
4. The **Feedback Management Framework**, which supports users in deciding, which services to use based on ratings, comments and tags.
5. The **Import Facilities**, for importing existing semantic services and compositions.
6. The **Process Editor**, which enables the user interface-based design of compositions and is developed within Task 2.6. This component is not subject to this deliverable but is rather described in D 2.6.3.

The second Provisioning Platform Prototype is enhanced by two main component implementations, focusing on iServe and the Annotations Recommender. iServe comprises the functionalities of a service browser (1.) and import facilities (5.) and at the same time enables the publishing, search and retrieval of SWS thus representing a fully-fledged semantic web service repository. The Annotations Recommender (2.) assists users in creating semantic annotations by suggesting a classification of the service to the user. The second prototype also includes improvements to the MicroWSMO and WSMO-Lite Editor (1.) as well as further developments of the Feedback Management Framework (4.). The Templates and Service Creation Wizards Management Framework (3.) is realized through

the definition of an application interface and is completed in Task 2.6.

Figure 1 provides an overview of the architecture of the Provisioning Platform. iServe, comprising the visualized Service Browser and Import Facilities, and the Annotations Recommender implementations are described in more detail in the following sections, focusing on the provided functionalities and user support. In addition, we describe the improvement of the MicroWSMO Editor, the WSMO-Lite Editor as well as the additional development done for the Feedback Framework.

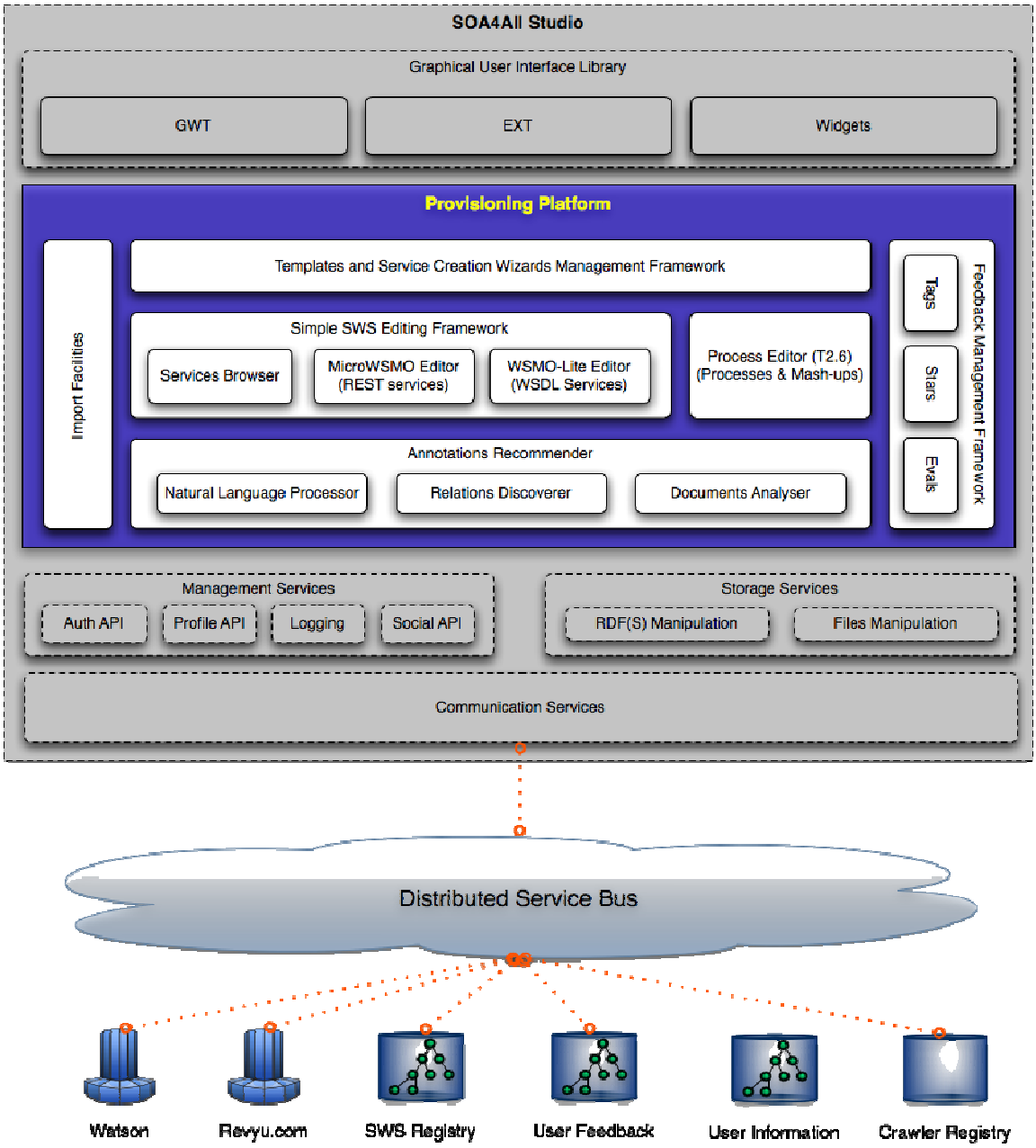

Figure 1: Service Provisioning Platform Architecture

1.2.1 Simple SWS Editing Framework

The main goal of the Simple Semantic Web Services Editing Framework is to support users in providing semantic annotations of existing web services, in order to enable the automation of service discovery and composition. Therefore, the created semantic web services are the basis for further activities and they are discovered, composed and executed by other components provided by the SOA4All dashboard. The first prototype implementation of the Simple SWS Editing Framework focused on providing functionalities of the MicroWSMO and WSMO-Lite editors, while this second prototype implementation includes some extensions and improvements to the editors and describes the newly developed Service Browser, which is part of iServe.

The Simple Semantic Web Services Editing Framework supports two main tasks, including the semantic annotation of web services and the browsing of existing semantic service descriptions. We differentiate between WSDL-based services, whose annotation is supported through the WSMO-Lite editor, and RESTful services, which use the MicroWSMO editor. WSDL service descriptions have a predefined structure and are given in XML, therefore the editor provides main functionalities for XML visualization and Drag&Drop-based XML editing. In contrast, RESTful services are described in plain HTML and as a result, this editor provides functions for marking service properties within the HTML and associating semantic content.

SWEET

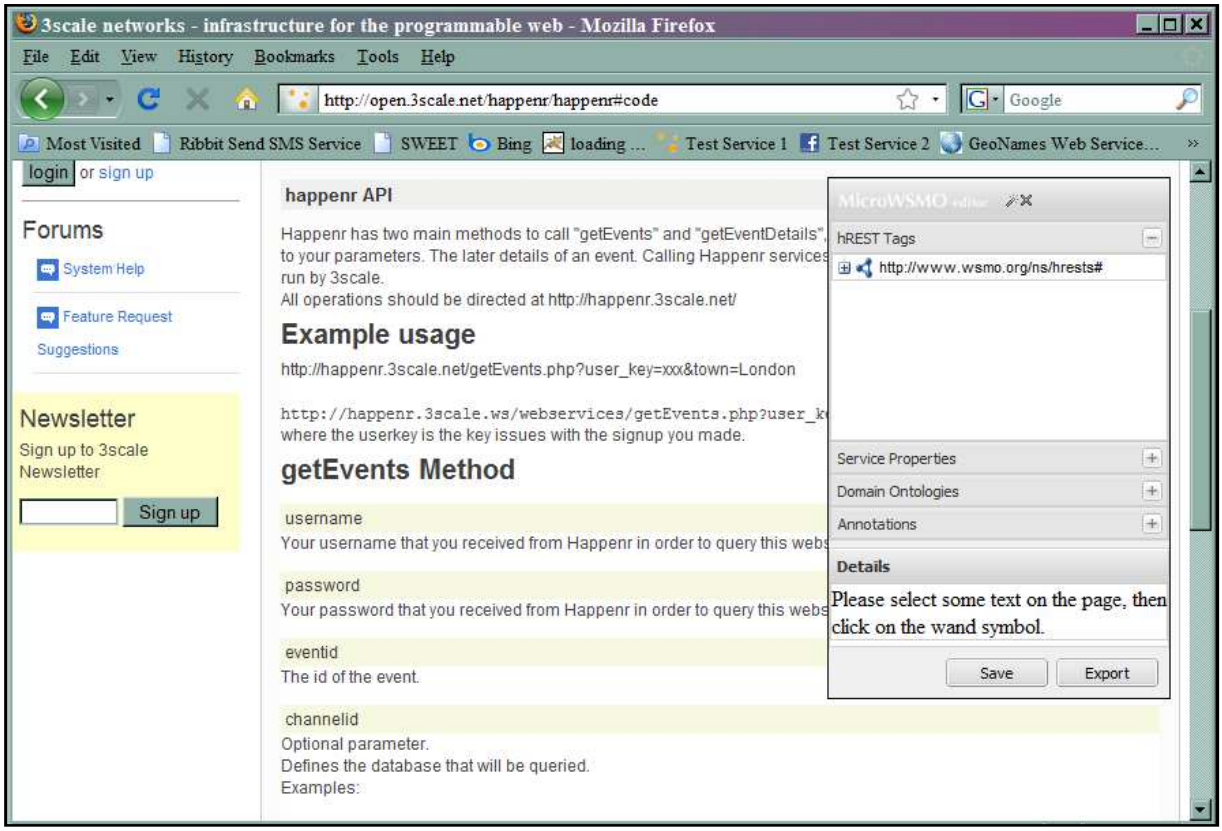

Figure 2: Lightweight MicroWSMO Editor

The first prototype of the MicroWSMO editor, which we decided to call SWEET (Semantic Web sErvice Editing Tool), was implemented in two main versions. The first version takes the form of a vertical widget displayed within a web browser (Figure 2). It is very lightweight and

can be used to directly annotate RESTful service descriptions visualized in the browser window. The second implementation is integrated in the SOA4All dashboard and uses the same layout and technologies as the other components, which are part of the dashboard (Figure 3).

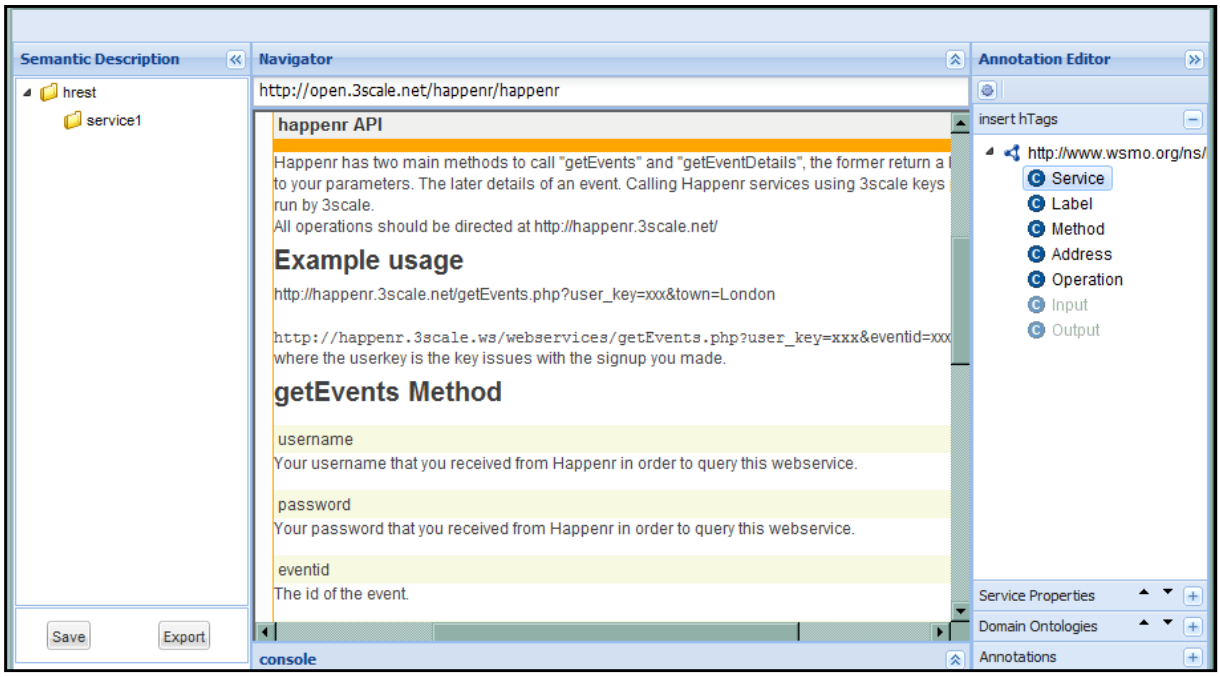

Figure 3: Dashboard MicroWSMO Editor

Both MicroWSMO editor implementations share common main functionalities:

- Insertion of hRESTS (D3.4.3) microformat tags in the HTML service descriptions in order to mark service properties (service, operation, address, HTTP method, input, output and label).
- Integrated ontology search for linking semantic information to service properties.
- Insertion of MicroWSMO (D3.4.3) model reference tags, pointing to the associated semantic meaning of the service properties.
- Saving of semantically annotated HTML RESTful service description.
- Automatic extraction of RDF MicroWSMO service descriptions, based on the annotated HTML, and saving of the resulting RDF.

The second prototype of SWEET is focused on improving and extending the SOA4All dashboard version of the editor. In particular, the main effort was on creating a stable implementation, including a number of useful additional features, such as the direct annotation of service properties or the loading of an ontology by a given URL.

Following the new approach of the SOA4All dashboard for providing a set of decoupled components, which are integrated via APIs but can be used independently (see D2.4.3), the editor component has been decoupled and now can be used as a standalone web application[1]. In addition, following this approach, the editor has been successfully connected

[1] http://sweetdemo.kmi.open.ac.uk/war/MicroWSMOeditor.html

to iServe, enabling the user to publish annotated MicroWSMO files directly in the service repository.

SWEET was presented in a number of conferences and used in hands-on sessions in two summer schools (The Summer School on Service and Software Architectures, Infrastructures and Engineering 2010 SSAIE Summer School[2], Karlsruhe Service Summer School 2010[3]) and as a result, a lot of useful user feedback was gathered. In the process of using the tool, we discovered a few functionalities that were not always performing properly but we also identified some features that would be necessary in order to improve the use of SWEET. We gathered this feedback by directly questioning the users but also by implementing a simple RDF-based events logging, as part of SWEET, which records basic user actions. Every time a user creates a service property or adds a semantic annotation, the events are recorded in RDF and stored in a triple store. The information is completely anonymous, containing no records of the user ID or the IP address, but it provides very valuable insights about the most frequently performed actions, the duration of an annotation process and the places where the annotation fails.

Based on the logged data and the feedback information, we identified that there was a problem with inserting hRESTS tags within the HTML of some Web APIs. This occurred in the cases where the annotated elements were surrounded by <code> or HTML elements. As a result we were able to identify the malfunctioning cases and fix the corresponding SWEET operation. We also improved some further features, such a providing support for customizable style-sheets for highlighting the service annotations, and produced a more stable and reliable version of SWEET.

One of the main improvements of SWEET was focused on enabling the better search and use of ontologies. In order to be able to use further sources for domain ontologies, besides Watson, we defined an API for searching for suitable ontologies for service annotation. Based on this API, we are able to use Watson or switch to Cupboard[4] or any other repository for ontologies. In addition, we implemented a new functionality, which enables users to load their own ontology into the tool and use it for the semantic annotation. By providing the URL of the ontology, users can view its classes, properties and instances and use them in the same way, in which ontologies retrieved by searching Watson, were used.

Finally, we extended SWEET with functionality for adding SAWSDL model reference annotations directly to the different service elements. Until now, it was only possible to annotate input and output properties with semantic entities from ontologies. The new prototype allows users also to add model reference to any URL, for example, pointing to a classification ontology.

SOUR

The WSMO-Lite Editor, which we decided to call SOUR, is a visual component of the SOA4All dashboard. The primary task of the editor is to facilitate the manual annotation of WSDL service descriptions with semantic information, following the WSMO-Lite service ontology specification and using the SAWSDL annotation mechanism. The first prototype of the tool provided the main functionalities required for creating annotations, namely the user is able to create new annotations on existing description and also modify or remove already created annotations. Informative tooltip balloons and auxiliary windows provide additional

[2] http://www.ssaie.eu

[3] http://www.service-summer.org

[4] http://cupboard.open.ac.uk

detailed information on demand.

The editor supports the following set of annotation functionalities:

- Insertion of reference annotations in XML Schema elements to classes from the information model ontology.
- Insertion of transformation annotations in XML Schema elements, providing the proper mapping (lifting and lowering) between service specific XML data and semantic model data.
- Insertion of functional annotations (capabilities and categories) for WSDL interfaces, services and operations to appropriate functional and behavioural descriptions.
- Insertion of non-functional description annotations, specifying any details related to the service implementation or the running environment.
- Removal of all kinds of annotations.

The second prototype of the WSMO-Lite Editor emphasizes on improved graphical interface usability, namely completely new layout style aligned with the SOA4All Studio, progress indicators for longer lasting processes, more informative tooltips.

The editor component has been decoupled from the Studio's dashboard and can now be used as a standalone web application[5]. The flexible decoupling mechanism allows for maintaining a single component both, as part of the SOA4All dashboard and independently.

From integration point of view, the editor has been successfully integrated with iServe, enabling the user to publish annotated (SA)WSDL files directly in the service repository. The management of the *Storage Services (D.2.4.2)* data has been improved as well.

SWEET and SOUR provide support for annotating both WSDL-based and RESTful services. However, in order to be able to support the automation of web service execution, in addition to these two annotation editors, there is also an editor for the creation of lifting and lowering schemas for WSDL services[6] (D.3.4.4 v2). In this way, users do not have to define the transformation between the implementation level and the semantic level manually, but can rather use the graphical tool.

The final component of the Simple Semantic Web Services Editing Framework is the Service Browser that was developed as part of iServe, described in detail in the next section.

1.2.2 iServe[7]

The Service Browser and the Importing Facilities, as described by the initial architecture design of the Provisioning Platform (Figure 1), were implemented as part of iServe, a platform for the seamless publication and discovery of semantic web services. iServe addresses the publication of services from a novel perspective based on lessons learned from the evolution of the Web of Data[8]. iServe transforms service annotations expressed in a variety of formats into what we refer to as Linked Services– linked data describing service– that can directly be interpreted by state of the art semantic web technologies for their

[5] http://stronghold.ontotext.com:8080/wsmoliteeditor/

[6] http://www.soa4all.eu/docs/D3.4.4_DATA_GROUNDING_COMPONENT_V2.PDF

[7] http://iserve.kmi.open.ac.uk

[8] Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. International Journal on Semantic Web and Information Systems (IJSWIS) (2009).

discovery and further processing. The iServe is integrated with SWEET and SOUR, so that users can create semantic annotations over services and directly publish them to iServe, where they can be search and retrieved.

iServe enables the importing of a variety of semantic web service formats, including SAWSDL annotations, WSMO-Lite annotations, MicroWSMO annotations of Web APIs and OWL-S Semantic Web Services descriptions and converts them into RDF, which abstracts away from the original approach used for annotating the services. As a result, both WSDL services and RESTful services are published in the same repository, enabling unified search and discovery. Taking the original descriptions, iServe automatically generates the appropriate RDF statements according to the Minimal Service Model[9] and expose them as linked data, thus providing simple means for publishing semantic web services in a manner that is suitable for the description and interlinking of services, people and data.

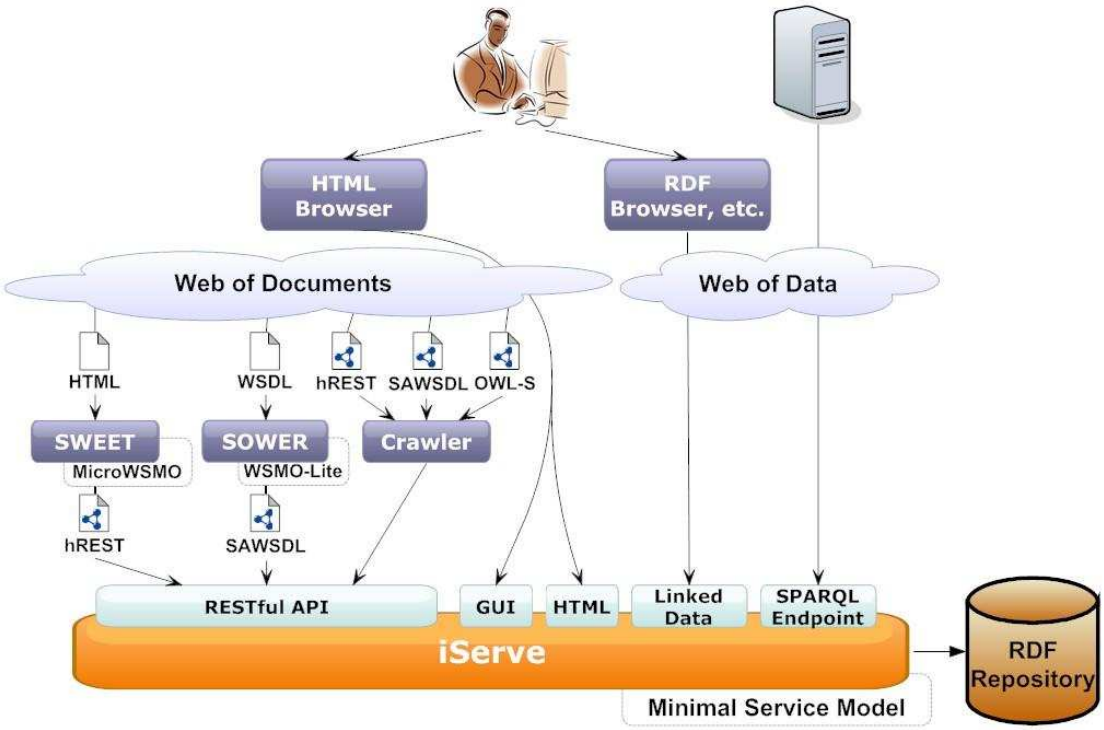

Figure 4: iServe Architecture

Figure 4 shows iServe, its connection to SWEET and SOUR, and the different ways of accessing it. In order to facilitate the consumption and manipulation of the published semantic service descriptions, iServe provides three interfaces:

- A Web-based application, called iServe Browser[10], allowing users to browse, query and upload services to iServe. This application realizes the functionalities foreseen for the Service Browser component in the Simple Semantic Web Services Editing Framework.
- A SPARQL endpoint, where all the data hosted in iServe can be accessed and

[9] Pedrinaci, C., Liu, D., Maleshkova, M., Lambert, D., Kopecky, J., and Domingue, J.: iServe: a Linked Services Publishing Platform, Workshop: Ontology Repositories and Editors for the Semantic Web at 7th Extended Semantic Web Conference (2010).

[10] http://iserve.kmi.open.ac.uk/browser.html

queried.
- A RESTful API that enables creating, retrieving and querying for services directly from applications.

These three main access points provide the direct use of iServe's main functionalities:

- Importing service annotations in a range of formalisms (e.g., SAWSDL, WSMO-Lite, MicroWSMO) that cover both WSDL services and Web APIs. This provides the functionalities of the Importing Facilities, as foreseen by the Provisioning Platform Architecture.
- Providing means for publishing semantic annotations of services, which are automatically assigned a resolvable HTTP URI.
- Supporting content negotiation so that service annotations can be returned in HTML for human users, or in RDF for machine interpretation.
- Providing a SPARQL endpoint allowing advanced querying over the service annotations.
- Offers a read/write REST API so that services can easily be retrieved and published from remote applications.
- Automatically generates links between the published service annotations and additional documents on the Web such as the original service description or documentation so that users and machines can easily discover more information.

In summary, iServe is a novel and open platform for publishing semantic annotations of services based on a direct application of linked data principles to publish service annotations expressed in terms of a simple vocabulary for describing services of different kinds (e.g., WSDL and Web APIs) with annotations in diverse formalisms (e.g., OWL-S, WSMO-Lite). It directly contributes to achieving the SOA4All goal of making services on the Web more accessible and usable for everyone.

1.2.3 Feedback Framework

Following the decoupled approach for the whole SOA4All Studio, the Feedback Framework component, which deals with user-generated feedback in the form of ratings, comments and tags, and which used to be included as a server-side module within the dashboard, has been enabled as a separated RESTful service itself. This way, the service can be easily integrated within different platforms and third-party applications.

Additionally, and also following the approach of iServe, the new version of the Feedback Framework places a special emphasis on the Linked Data paradigm, exposing the semantic user-generated data about ratings, comments and tags as Linked Data. In this line, the service has been rebranded as **Linked User Feedback (LUF)** in the public website[11] where general information about the service is given.

The LUF RESTful service exposes an API that can be accessed by third-party applications in order to store new feedback information, as well as to retrieve the available data. The API is described in the same website, and general information is also given in Section 3.2.2 of this document.

[11] http://soa4all.isoco.net/luf/about

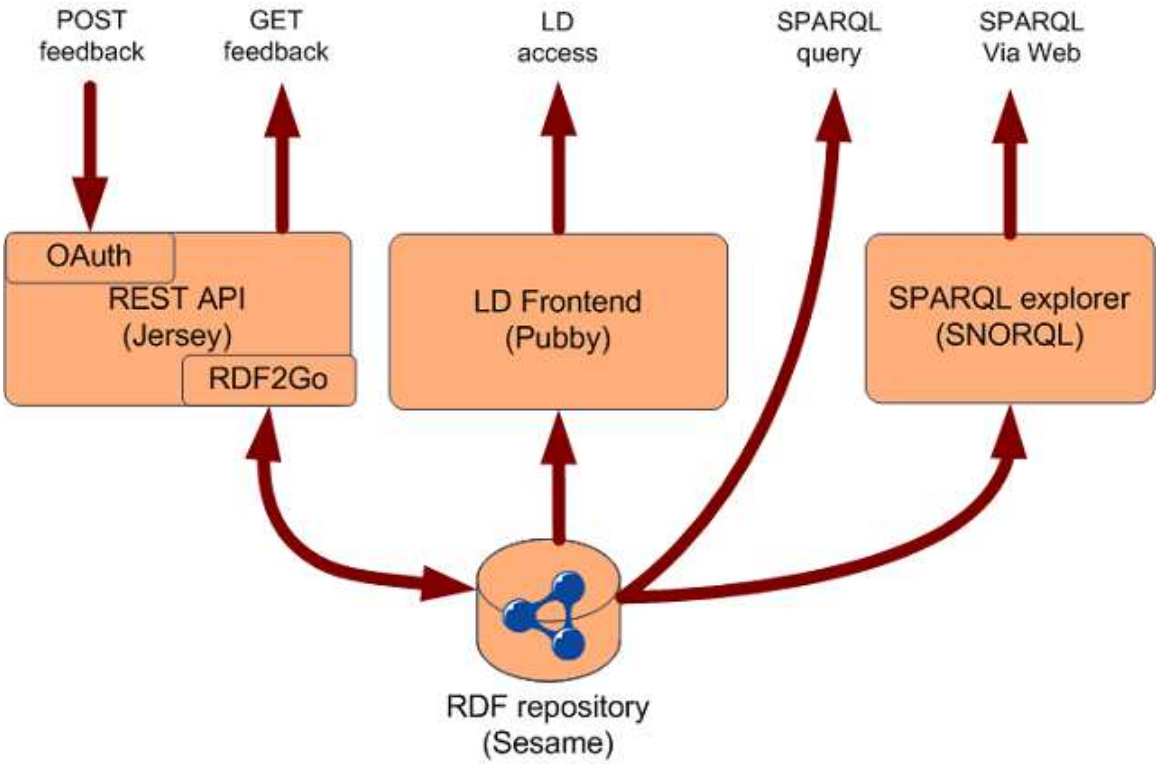

Figure 5: Linked User Feedback Internal Architecture

In addition to the API (developed with Jersey[12], Jersey OAuth[13] and RDF2Go[14]) and the Linked Data access (enabled on top of the Sesame[15] semantic repository making use of Pubby[16]), a SPARQL endpoint for querying is available, and also a direct Web access (by making use of SNORQL[17]). Figure 5 depicts the different methods for accessing the service.

The vocabularies used to semantically represent the feedback of the users on services has already been described in the previous deliverable D2.1.3: the Review Schema[18] and the Tag Ontology[19]. In, addition, the Provenance vocabulary[20] is being considered as a way to efficiently deal with the different origins of the feedback information. An example of a rating, a comment and a tagging is illustrated below:

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rev:     <http://purl.org/stuff/rev#> .
@prefix tags:    <http://www.holygoat.co.uk/owl/redwood/0.1/tags/> .

<http://iserve.kmi.open.ac.uk/resource/services/123456789#example>
```

[12] https://jersey.dev.java.net

[13] http://wikis.sun.com/display/Jersey/OAuth

[14] http://rdf2go.semweb4j.org

[15] http://www.openrdf.org

[16] http://www4.wiwiss.fu-berlin.de/pubby/

[17] http://github.com/kurtjx/SNORQL

[18] http://purl.org/stuff/rev#

[19] http://www.holygoat.co.uk/owl/redwood/0.1/tags/

[20] http://purl.org/net/provenance/

```
        rev:hasReview <http://soa4all.isoco.net/luf/ratings/xxx> ;
        rev:hasReview <http://soa4all.isoco.net/luf/comments/yyy> ;
        tags:tag <http://soa4all.isoco.net/luf/taggings/zzz> .

<http://soa4all.isoco.net/luf/ratings/xxx>
        rdf:Type rev:Review ;
        rev:rating "3" ;
        rev:minRating "1" ;
        rev:maxRating "5" ;
        rev:reviewer <http://example.com/users#user1> ;
        rev:createdOn "2010-06-22T12:29:28+0200"^^xsd:dateTime .

<http://soa4all.isoco.net/luf/comments/yyy>
        rdf:Type rev:Review ;
        rev:text "This is a comment" ;
        rev:reviewer <http://example.com/users#user1> ;
        rev:createdOn "2010-06-22T10:59:01+0200"^^xsd:dateTime .

<http://soa4all.isoco.net/luf/taggings/zzz>
        rdf:Type tags:Tagging ;
        tags:associatedTag "one tag" , "other tag" ;
        tags:taggedBy <http://example.com/users#user1> ;
        tags:taggedOn "2010-06-22T13:03:05+0200"^^xsd:dateTime .
```

As stated before, the LUF service can easily be integrated within external applications. This can be demonstrated by the fact that both the iServe browser and SPICES (the service consumption platform described in D2.2.3) already integrate the LUF service through its API. Also, the LUF website retrieves the latest generated feedback by accessing the SPARQL endpoint.

1.2.4 Annotations Recommender

Currently, using the annotation editors involves some amount of manual work. In addition to marking service properties, users are also required to choose the appropriate semantic annotation from the list of available ontologies, as provided by the ontology search engine (Watson). The results returned by the search engine are base on some very specific API attributes and parameters, not on the context of the whole web page.

The purpose of the Annotations Recommender is to assist the user in the process of adding semantic annotations to web services. By analyzing the textual content of the web pages describing WSDL-based and RESTful services, it is able to suggest a category by using instance-based learning techniques. Recent experiments based on test data collected from ProgrammableWeb[21] and training data from the Open Directory Project[22] show that the service classifier component is able to determine the correct category among three different classes with a 70% accuracy.

[21] http://www.programmableweb.com

[22] http://www.dmoz.org/about.html

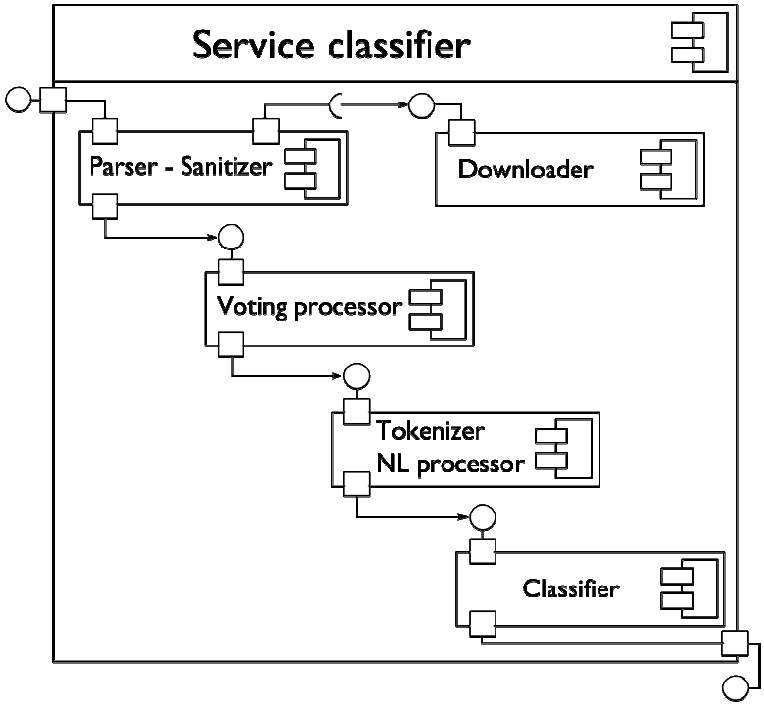

Figure 6: Service Classifier Component

Determining the type of functionality of the service and its domain are one of the main tasks that need to be completed, when doing annotations recommendation. Knowing the type of a service enables easier discovery but also eases the search for suitable domain ontologies. Therefore, the main component of the Annotations Recommender is devoted to classifying the type of service. The Service Classifier architecture is shown on Figure 6. The workflow starts with the downloader component, its sequence diagram can be found on Figure 7. Its task is to fetch a web page given its URL, return the content and report encountered fetch errors. The fetched pages are processed and cached for better performance to avoid fetching the same page multiple times.

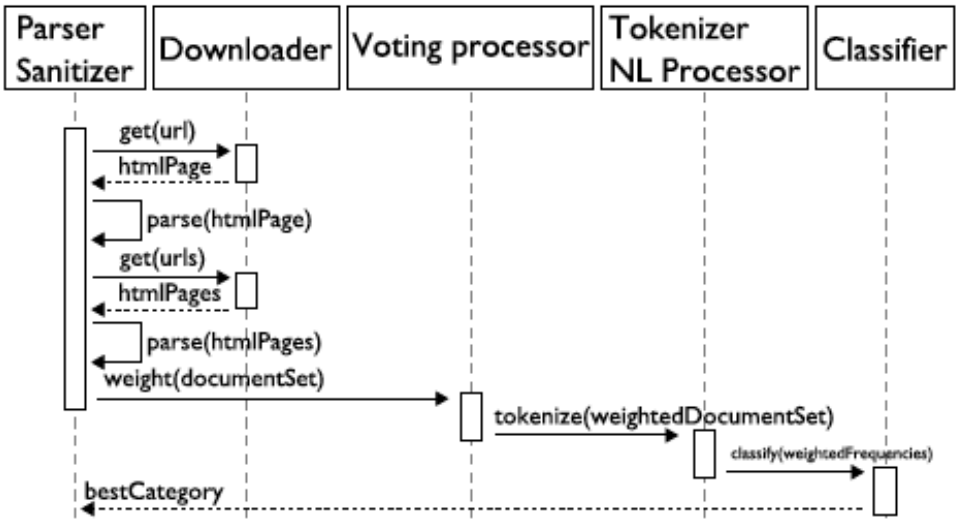

Figure 7: Service Classifier Component Communication

When the main page describing the service is fetched and cached, it is sent to the parser

and sanitizer component. The sanitizer removes unnecessary elements from the HTML page, mainly JavaScript elements and flash objects (if any). The parser analyses the different links of the page and requests related pages from the downloader. Based on experiments, fetching 5 of the sub-pages linked from the main page increases the accuracy already by 10%. Therefore, retrieving related pages, in addition to the main service page has an important effect on the accuracy of the component results.

Once a document and the set of related pages are collected and sanitized, they are given to the voting processor, which was initially supposed to convert the HTML pages to a weighted list of phrases and sentences: a sentence inside an <h1> HTML element would be given a bigger weight than a sentence inside a <p> element. However, this step has been merged with the next one: the tokenizer directly splits the textual content of the HTML page into a list of tokens, which are the input data for the classifier.

The classifier component takes the tokenized text and finds the best matching category based on the data it has been trained with. Once fully trained and setup, the classifier has a very simple API:

```
public String classify(URL url);
public String[] classify(URL url, int matches);
```

The input parameter is the URL of the web page describing a service, and the output is the best match found by the classifier. Optionally the classifier could return a list of categories ordered by relevance instead of a single category.

1.3 Customisation Done for This Deliverable

The here described second Provisioning Platform Prototype and its components are directly contributing to the functionalities of the final version of the Provisioning Platform. The current state of the implementation reflects almost completely the expected end result and there are a few improvements, which will be made by the time this deliverable is submitted. There are no major assumptions or customizations made in this version of the components.

1.4 Roadmap for Future Plans

The Provisioning Platform Prototype documented in this deliverable represents the final version of the implemented components. Therefore, there is no roadmap for future work. Still, minor improvements and fixes will be carried out until the end of the project, even though, there are no more planned deliverables.

2. Prototype Documentation

This section of the deliverable provides practical information about using the second Provisioning Platform prototype. It provides installation and configurations instructions as well as guidelines about the functionalities of the different elements of the components. The prototype documentation also includes step-by-step descriptions of how to use the service browser of iServe, how the invoke the LUF RESTful API and how to perform service classification. We also describe the changes made to SWEET and SOUR.

2.1 Installation and Configuration

The second Provisioning Platform prototype is following the approach of decoupled components. Each of the tools is exposed through a Web API or through a URL, where it can be invoked, and there are no fixed connections or hard-coded links between the components. As a result, the SOA4All dashboard is realized by combining these detached components and third-party users can choose to include some of the components as part of their applications, in a similar way. Therefore, there are no particular installation or configuration requirements, since the parts of the Provisioning Platform are available as services.

Currently all components, which are part of the Provisioning Platform, are completely integrated in the SOA4All dashboard. Therefore, there are no additional installation tasks, which need to be completed and no dependent software components, which need to be configured. The installation and configuration guidelines of the SOA4All dashboard are described in D2.4.4. It is recommended to use the Firefox[23] browser, version 3.5 or later, for viewing GWT UI components, such as SWEET.

2.2 How to use the Prototype

This section describes in detail how each of the prototype components can be used. Each implementation is described in terms of the functionalities of the GUI and a step-by-step instruction how to complete some common tasks. This section includes descriptions of the prototypes of the MicroWSMO Editor, the WSMO-Lite Editor, iServe, the Feedback Framework and the Annotations Recommender.

2.2.1 Simple SWS Editing Framework

SWEET

The current version of SWEET has the same user interface, as the previous version, since the main implementation effort was focused on achieving stability and fixing faults of the previous version. Still, there are two new functionalities that have been implemented. First, the user is able to add SAWSDL model reference annotation to any service property. This can be done very easily by right-clicking on a node of the service annotation three in the *Semantic Description* panel and inputting the URI for the model reference.

In order to ease the annotation tasks, we have discovered that sometimes users want to use their own ontologies to describe the service, instead of relying only on the ones provided by Watson. Therefore, similarly to SOUR, we now provide a button for loading an ontology, given its URL. After loading the ontology, the user can directly use it to make annotations.

[23] http://www.mozilla-europe.org/en/firefox/

Finally, SWEET has been integrated with iServe and now the *Save* and *Export* buttons have an option *Save To Service Repository* and *Export To Service Repository* for publishing the semantic RESTful service descriptions directly to iServe.

SOUR

The current version of the WSMO-Lite Editor contains extended GUI support for data management of the *Storage Services*. This includes uploading local file system resources to the storage space (Figure 8) as well as preview or removal of storage data (Figure 9). As a result, this functionality turns into a key *Storage Services* visual front-end not only for the editor but for all dashboard components as well.

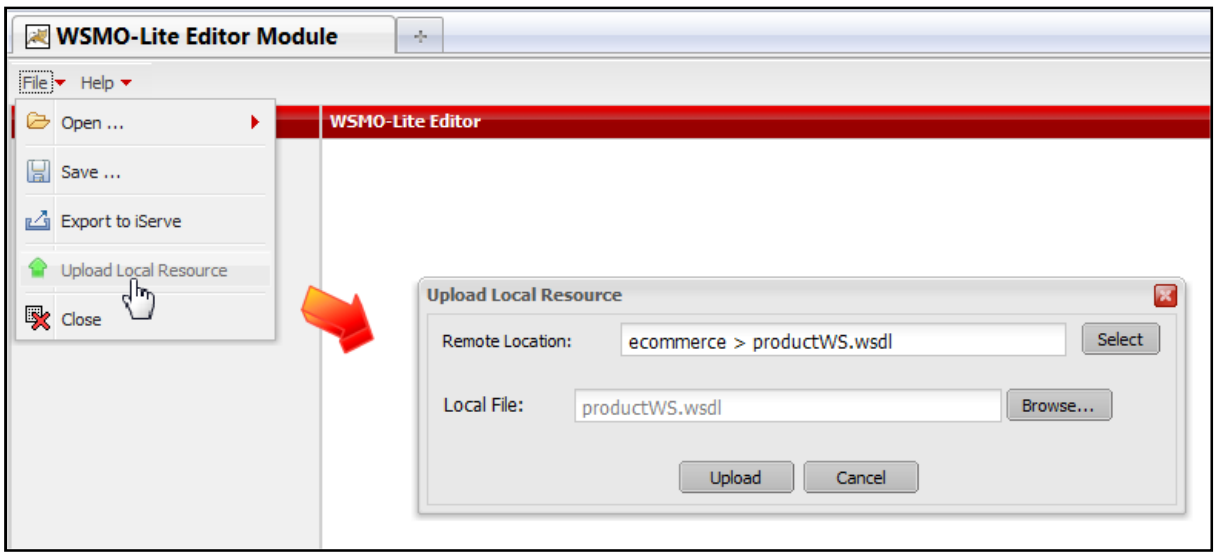

Figure 8: WSMO-Lite Editor – Local Resources Upload

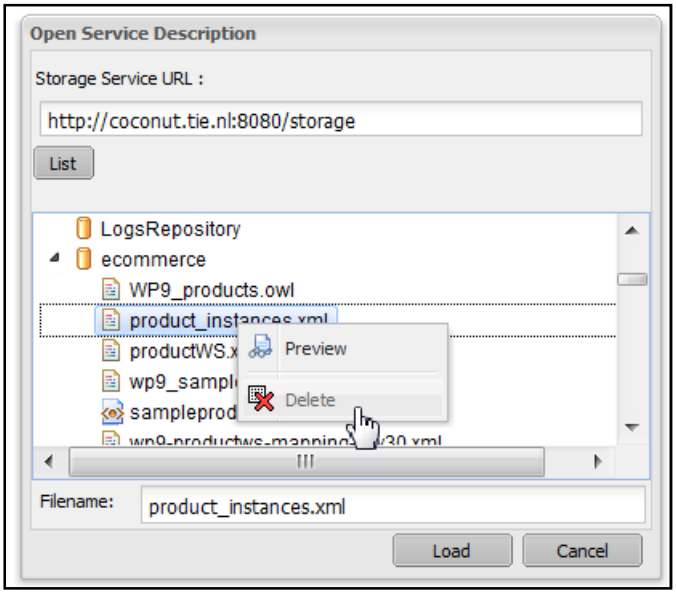

Figure 9: WSMO-Lite Editor – Removal of Resources

As part of the resource management functionalities, the editor now offers loading resources (ontologies and service descriptions) not only from the *Storage Services* but from any (publicly) accessible URL.

The service descriptions publishing functionality was achieved by integration with the iServe system[24]. Having in advance a valid iServe account, the user can send annotated service descriptions (SAWSDL) directly from the WSMO-Lite Editor (Figure 10).

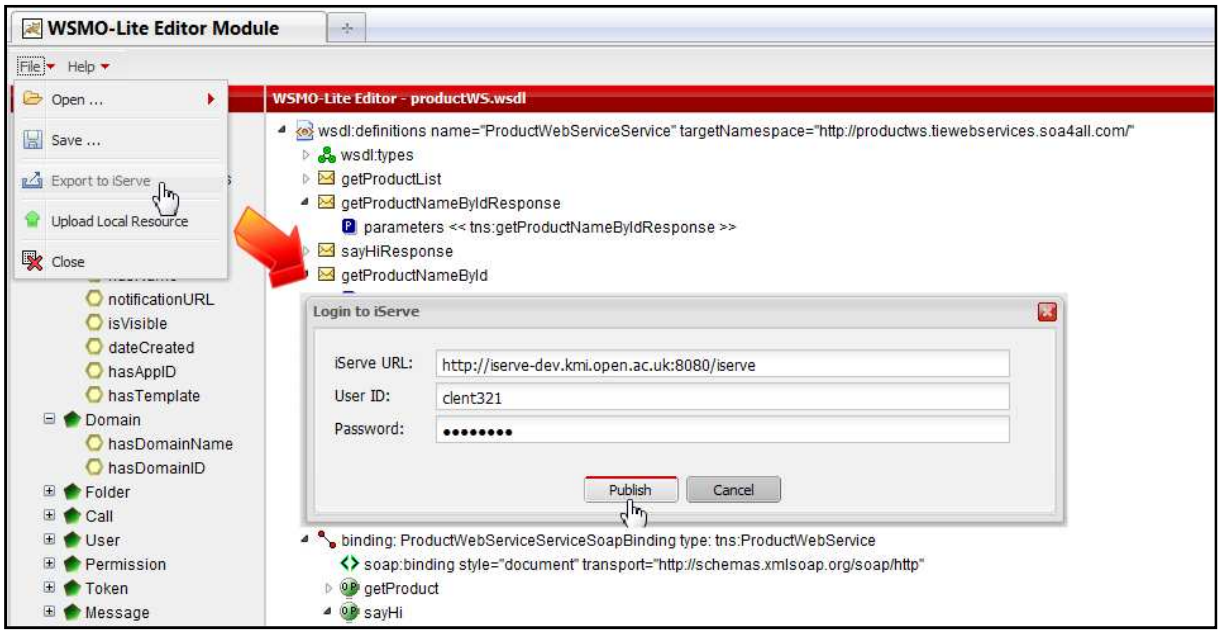

Figure 10: WSMO-Lite Editor – iServe Integration

2.2.2 iServe

This section describes the main functionalities offered by the iServe Service Browser, since this is the only component of iServe that has a user interface. The Service Browser supports the following tasks:

- Logging in with an existing OpenID account, creation of a user profile and editing of user profile.
- Browsing services per category, using a predefined or a custom taxonomy.
- Loading a taxonomy.
- Searching for services by keyword.
- Viewing service details.
- Adding a new service.
- Removing a service (available only for the creator of the service).

The following sections describe each of these tasks in more detail.

In order to add and remove services, the user needs to login to iServe with an OpenID account. Clicking on the login button located in the top right corner of iServe browser, opens a pop-up dialog box for inputting the user OpenID. For MyOpenID or Yahoo OpenID (http://openid.yahoo.com), first, choose the provider from the list, and only input the short ID. The whole URI of your OpenID will be automatically generated and put into the nethermost text box. For using OpenID gotten from other providers, unfortunately, the complete URI

[24] http://iserve.kmi.open.ac.uk

needs to be typed in. After inputting the OpenID and clicking on the login button, the Web browser will possibly be redirected to the corresponding OpenID provider to do the authentication. Then, iServe will verify the result of authentication sent by the OpenID provider, and allow the login to iServe. A known limitation is that currently Google OpenID (http://openid-provider.appspot.com) is not supported. Still, no login is required for only browsing and viewing the services in the repository.

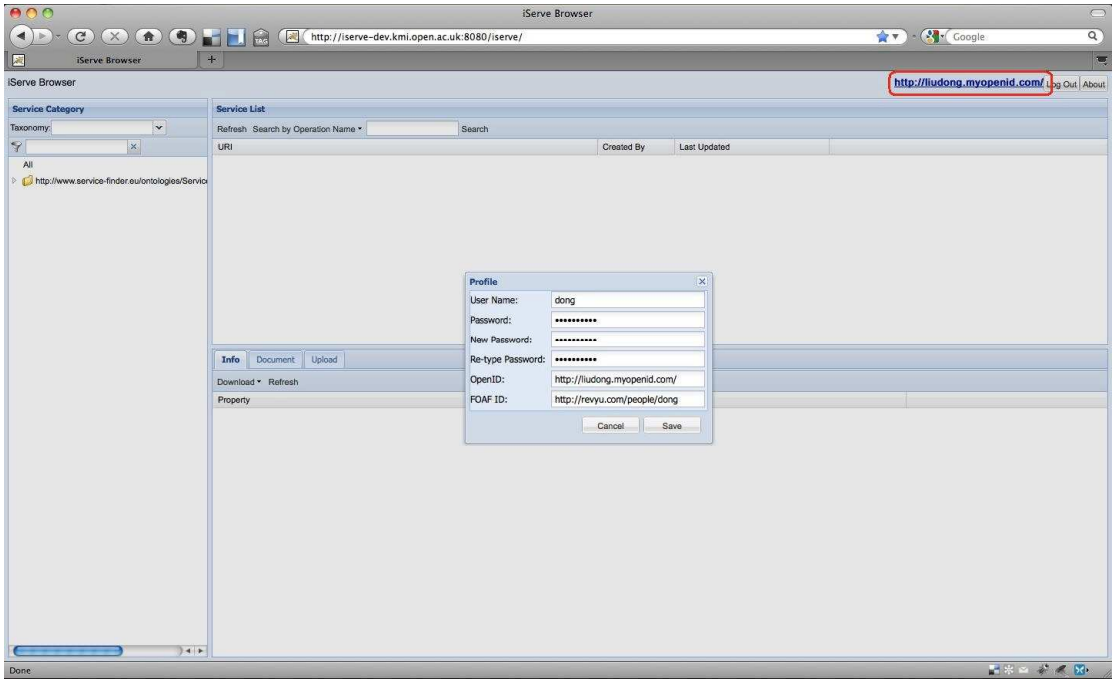

Figure 11: Modifying User Profile

When a user logs in the first time, he/she is requested to complete the profile, or, at least, provide a valid FOAF ID. User name and password are used for identification, when accessing iServe trough the RESTful API. That is, the user name and password must be sent together with the HTTP request. The initially created user profile can be modified by clicking on the OpenID used to login, which is displayed in the upper-right corner of the iServe browser (Figure 11).

One of the main functionalities of iServe is browsing services by category. As shown in Figure 12, the service categorisation tree is shown on the left hand side of the iServe browser. By selecting a category, you will get services under the selected category listed on the right-hand side. In addition, iServe also provides a simple text-based filter, so that when the user types some text, the categorization tree shows only categories whose name or parent category name contain the inputted text.

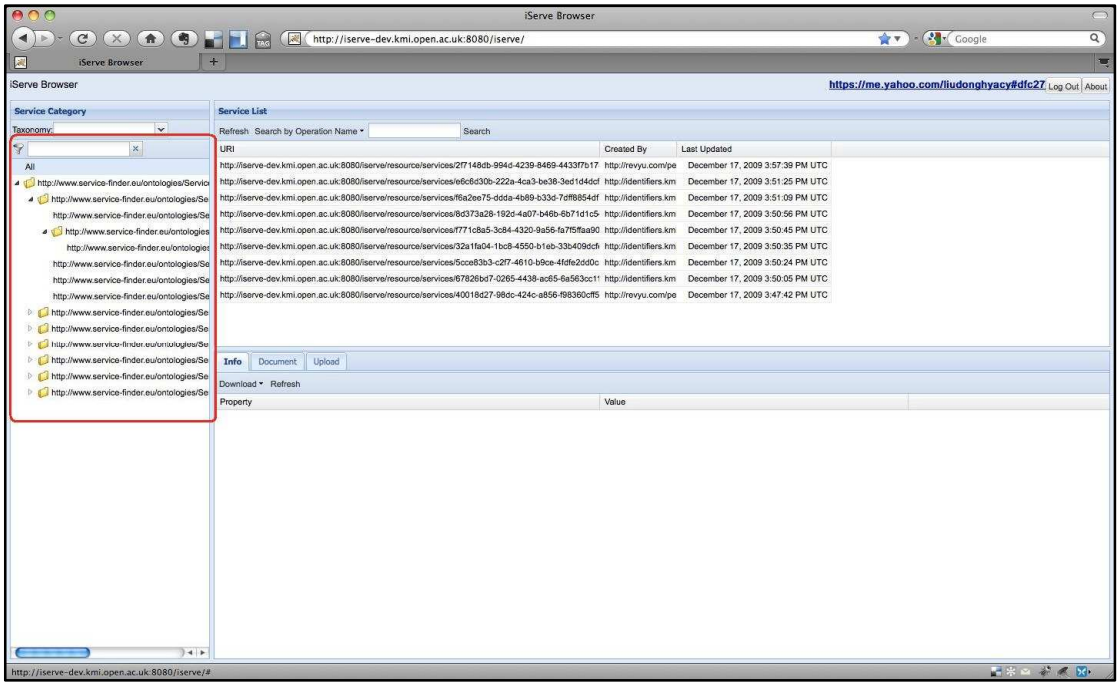

Figure 12: Browsing Services by Category

iServe can dynamically load taxonomies to classify services from different perspectives. For now, a small taxonomy (http://www.service-finder.eu/ontologies/service-categories.rdfs) created by Service Finder (http://www.service-finder.eu) is used as the default taxonomy. Some other taxonomies such as eCl@ss (http://www4.wiwiss.fu-berlin.de/bizer/ecommerce/eClass-4.1.rdf), UNSPSC (*http://www.cs.vu.nl/~mcaklein/unspsc/*) will be added to iServe soon.

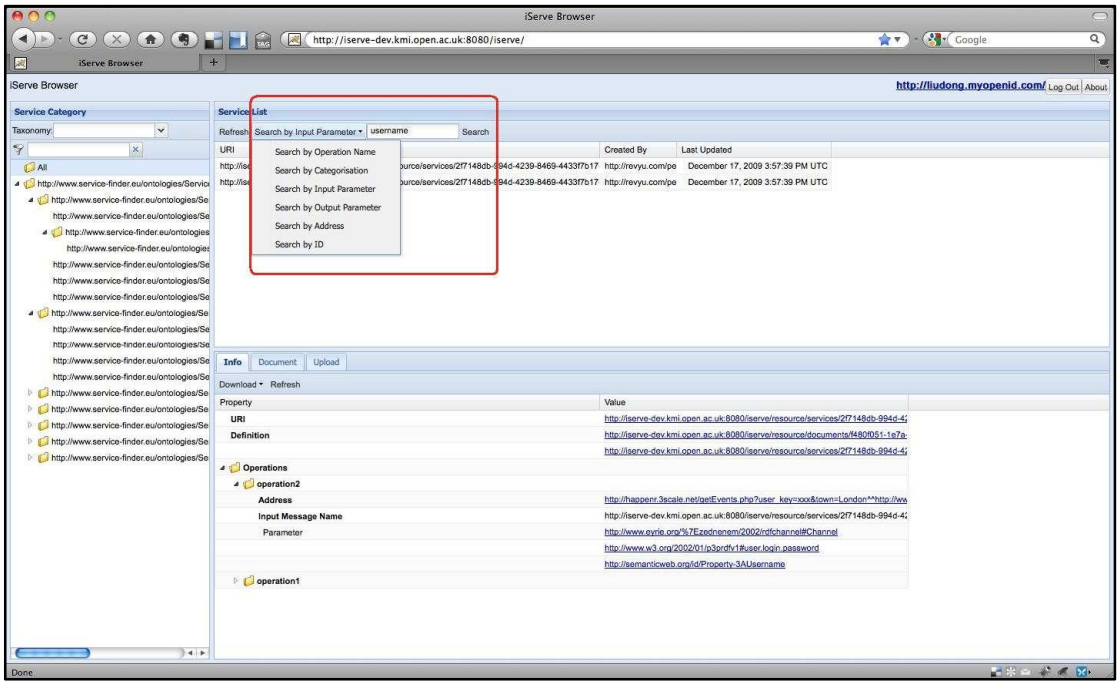

Figure 13: Keyword-based Search

iServe also enables the searching of services by keyword (Figure 13). Keyword-based search for services can be done by name or address of operation, categorization, input/output parameter and service ID. The results will be shown as a list underneath the searching toolbar.

After a particular service of interest is found, more details about it will be displayed in the tab named *Info* (Figure 14). This information panel includes the properties of the selected service as well as its operations, such as model references, parameters, addresses, etc. The document that defines the service will be shown in the *Document* tab.

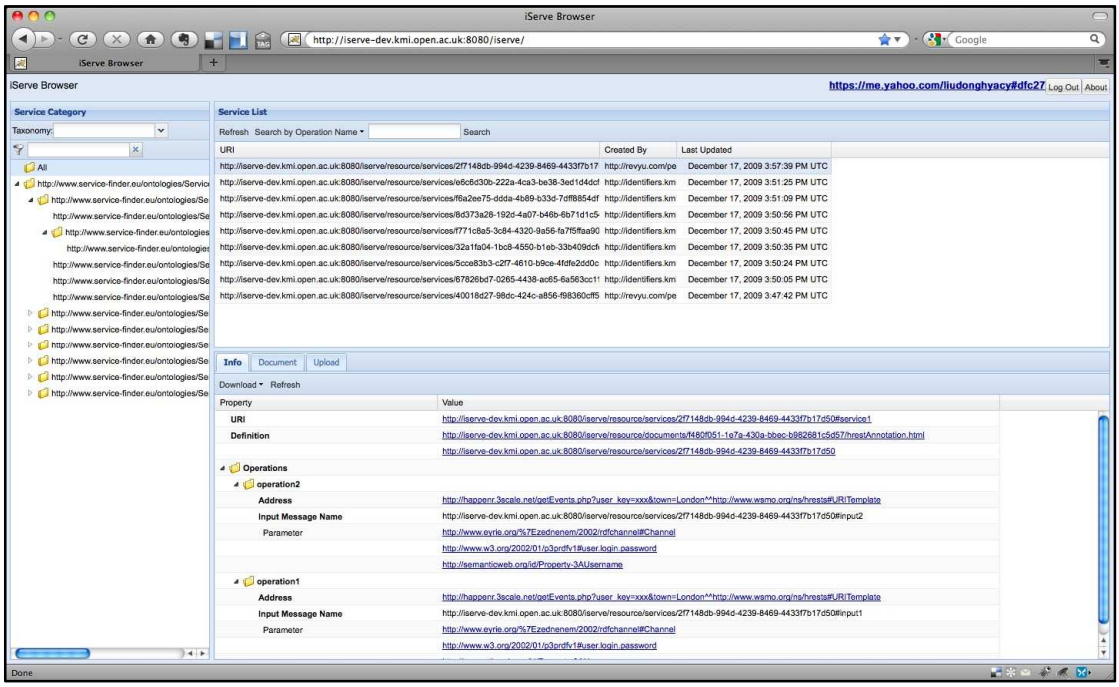

Figure 14: Viewing Service Details

iServe also enables the adding and removing of services through the Service Browser user interface. By switching to the *Upload* tab, the user can create new services by uploading the service descriptions, for which there are three different means: giving an accessible URI for retrieving the description file, directly uploading it, or manually typing the content into the text box. iServe currently can take as input annotated service descriptions as SAWSDL, WSMO-Lite annotations, MicroWSMO annotations, and OWL-S.

In order to remove services, the user needs to choose some entries from the service list, right click on them, and then select the item named *Remove service* in the pop-up menu. Please note that only the creator (a.k.a. the owner) of the services has the privilege to remove them. iServe checks the ownership of the service before the removal of services.

The Service Browser enables viewing service information in HTML as well as in RDF. Depending on the format that the user prefers to access and browse iServe, the URIs of services and documents are listed as follows:

- URI of service list: http://iserve-dev.kmi.open.ac.uk:8080/iserve/resource/services
- URI of document list: http://iserve-dev.kmi.open.ac.uk:8080/iserve/resource/documents

Links to services or documents can be respectively found from the two lists. Screenshots of browsing descriptions of services and content of documents are shown in the figures below (Figure 15 and Figure 16).

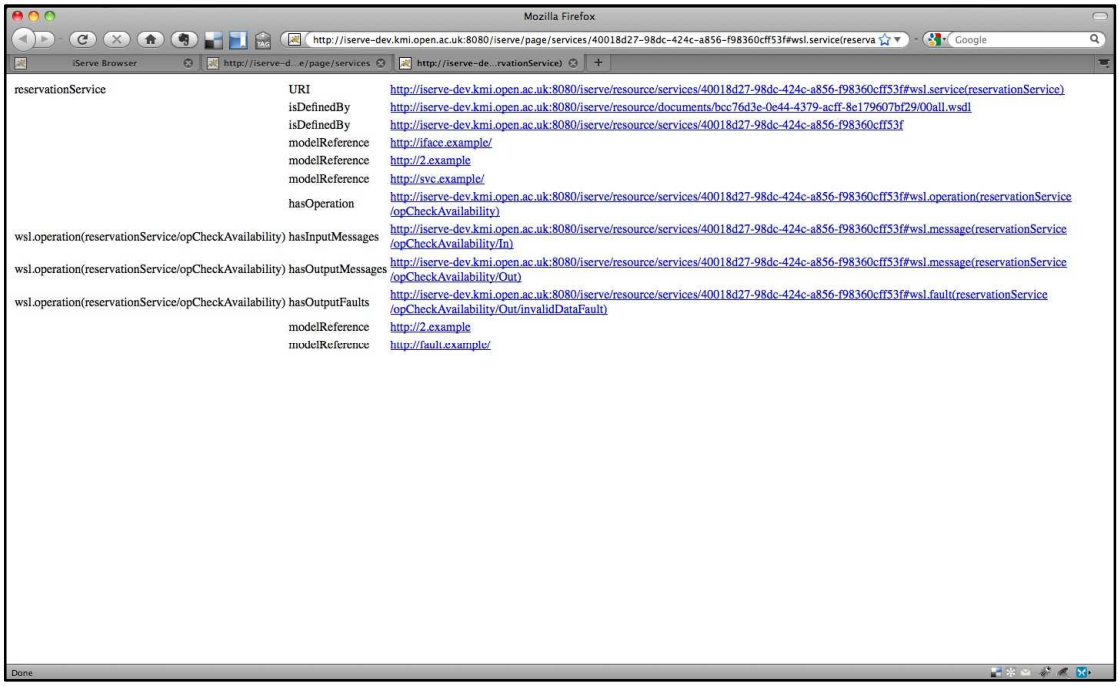

Figure 15: Detailed Service Description

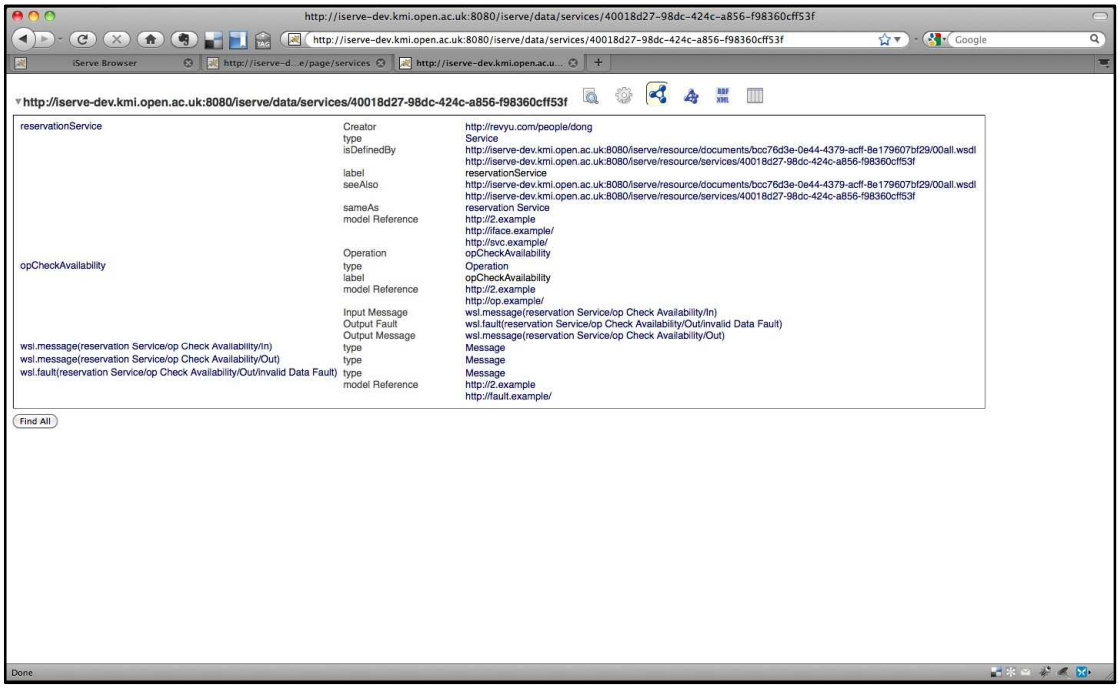

Figure 16: Browsing Service Descriptions

2.2.3 Feedback Framework

The online version of the Linked User Feedback service prototype can be accessed through URLs located at `http://soa4all.isoco.net/luf/*`. Next, we describe the four main interactions with the platform, namely i) via the REST interface, ii) as Linked Data, iii) through the SPARQL endpoint, and iv) via a SPARQL Web interface. It is worth noting that the LUF

service is not intended to be accessed directly by end-users, but rather indirectly through the platforms that interact with the API (currently, the iServe browser and SPICES).

2.2.3.1 LUF in a REST interface

The REST API of LUF permits retrieving information about each of the stored ratings, comments and taggings through different GET operations. It also permits storing new ratings, comments and taggings through different POST operations. Additionally, there is a search operation to retrieve the feedback information on a particular item and/or performed by a particular user.

Operation 1: Retrieve a rating (method: GET; Requires Auth: False)

URL: `http://soa4all.isoco.net/luf/api/ratings/{id}`

Parameters:

- `{id}`: The identifier of the particular rating.

The responses contain the following information:

```
<response>
  <item>{service uri}</item>
  <rating>{rating number}</rating>s
  <minRating>{min value}</minRating>
  <maxRating>{max value}</maxRating>
  <reviewer>{reviewer}</reviewer>
  <createdOn>{creation date}</createdOn>
</response>
```

Operation 2: Create a rating (Method: POST; Requires Auth: True, OAuth)

URL: `http://soa4all.isoco.net/luf/api/ratings`

POST body parameters:

- `{itemId}`: The URI of the item being rated.
- `{userId}`: The URI of the reviewer.
- `{rating}`: A numeric value.
- `{min value}`: The minimum value in the scale (optional, defaults to 1).
- `{max value}`: The maximum value in the scale (optional, defaults to 5).

The responses contain the following information:

```
<response>
  <ratingUri>{rating uri}</ratingUri>
</response>
```

Operation 3: Retrieve a comment (Method: GET; Requires Auth: False)

URL: `http://soa4all.isoco.net/luf/api/comments/{id}`

Parameters:

- `{id}`: The identifier of the particular comment.

The responses contain the following information:

```
<response>
  <item>{service uri}</item>
  <comment>{comment text}</comment>
  <reviewer>{reviewer}</reviewer>
  <createdOn>{creation date}</createdOn>
</response>
```

Operation 4: Create a comment (Method: POST; Requires Auth: True, OAuth)

URL: `http://soa4all.isoco.net/luf/api/comments`

POST body parameters:

- `{itemId}`: The URI of the item being rated.
- `{userId}`: The URI of the reviewer.
- `{comment}`: Text.

The responses contain the following information:

```
<response>
  <commentUri>{comment uri}</commentUri>
</response>
```

Operation 5: Retrieve a tagging (Method: GET; Requires Auth: False)

URL: `http://soa4all.isoco.net/luf/api/taggings/{id}`

Parameters:

- `{id}`: The identifier of the particular tagging.

The responses contain the following information:

```
<response>
  <item>{service uri}</item>
  <comment>{comment text}</comment>
  <reviewer>{reviewer}</reviewer>
  <createdOn>{creation date}</createdOn>
</response>
```

Operation 6: Create a tagging (Method: POST; Requires Auth: True, OAuth)

URL: `http://soa4all.isoco.net/luf/api/taggings`

POST body parameters:

- `{itemId}`: The URI of the item being rated.
- `{userId}`: The URI of the reviewer.
- `{tags}`: A set of comma-separated tags.

The responses contain the following information:

```
<response>
  <taggingUri>{tagging uri}</taggingUri>
</response>
```

Operation 7: Search (Method: GET; Requires Auth: False)

URL:`http://soa4all.isoco.net/luf/api/search?itemId={itemId}&userId={userId}`

Parameters:

- `{itemId}`: The URI of a particular item (optional).
- `{userId}`: The URI of a particular user (optional).

The responses contain the following information:

```
<response>
  <ratings>
    <ratingUri>{rating id}</ratingUri>
    <ratingUri>{rating id}</ratingUri>
    ...
    <ratingsAverage>{ratings average}</ratingsAverage>
    <ratingsNumber>{number of ratings}</ratingsNumber>
  </ratings>
  <comments>
    <commentUri>{comment id}</commentUri>
    <commentUri>{comment id}</commentUri>
    ...
    <commentsNumber>{number of comments}</commentsNumber>
  </comments>
  <taggings>
```

```
        <taggingUri>{tagging id}</taggingUri>
        <taggingUri>{tagging id}</taggingUri>
        ...
        <taggingsNumber>{number of taggings}</taggingsNumber>
        <tagAggregate>
          <tag>{tag name}</tag>
          <times>{number of ocurrences}</times>
        </tagAggregate>
        <tagAggregate>
          <tag>{tag name}</tag>
          <times>{number of ocurrences}</times>
        </tagAggregate>
        ...
      </taggings>
    </response>
```

2.2.3.2 LUF as Linked Data

LUF exposes the feedback produced by users (ratings, comments, taggings) as Linked Data at the following locations:

- `http://soa4all.isoco.net/luf/ratings/{id}`
- `http://soa4all.isoco.net/luf/comments/{id}`
- `http://soa4all.isoco.net/luf/taggings/{id}`

, where `{id}` is the particular identifier of each piece of feedback information. Content-negotiation is handled so the information can be rendered as HTML, RDF, etc.

2.2.3.3 LUF through a SPARQL endpoint

Additionally, it is possible to query the RDF repository via its SPARQL endpoint, located at `http://soa4all.isoco.net/luf/sparql`.

2.2.3.4 LUF by SNORQL

Finally, a Web interface for performing queries over the repository has been enabled at `http://soa4all.isoco.net/luf/snorql`.

2.2.4 Annotations Recommender

The Annotations Recommender does not have its own user interface, but we use the visualization tool provided by Weka[25] in order to show the different steps involved in the classification process. In this section we explain in detail how service classification is done and what input information is included.

The service classification task requires some pre-processing work. Currently, all the steps preceding the actual classification are implemented in a standalone Java package that is able to crawl the web and output data in a format that can be used by the classifier. The tokenization is done by Lucene[26] and the pages are processed and sanitized using HtmlCleaner[27].

The classifier runs as a Weka experiment: the tokenizer outputs its data in a file that can be loaded by Weka to train and validate the classifier. The advantage of using Weka is that different classifiers and different options can be tested and compared before embedding the

[25] http://www.cs.waikato.ac.nz/ml/weka/

[26] http://lucene.apache.org/java/docs/index.html

[27] http://htmlcleaner.sourceforge.net

best one in a standalone component. The experiment setup in Figure 17 shows how the classification is done:

- The data is loaded from the file generated by the tokenizer component. The data comes from the API directory of ProgrammableWeb.com, where more than 2000 Web APIs are currently referenced.
- The input data is pre-processed by Weka: the tokens are transformed to a "word vector" representation and the category of each Web API is declared to Weka.
- The classifier is setup to do cross-validation of the classifier: the data set is split into ten folds and each fold is used to validate a classifier trained with the nine other folds. The advantage of cross-validation is that the risk of overfitting is much lower.
- The training and validation sets are sent to a classifier. Different classifiers can be swapped and the results of each run are stored.

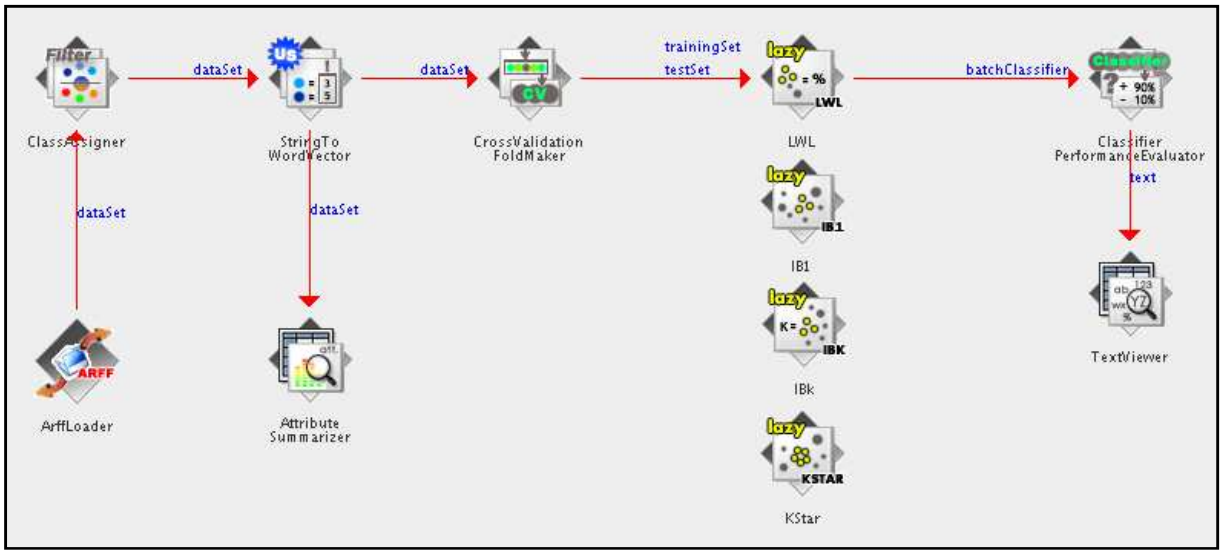

Figure 17: Classification Workflow

The training is done by collecting already categorized data from the Open Directory Project[28] and ProgrammableWeb[29]. The component is able to collect a list of links from ProgrammableWeb's API or an Open Directory Project data dump. Once the links are collected, they are processed by the downloader, sanitizer and parser components and the resulting text is serialized into an ARFF[30] file. The ARFF file is then loaded and processed by Weka as training data for the classifier.

The advantage of having such a setup is twofold. First, the training of the classifier can be done offline and independently of the actual annotation process. Second, by using the data for each run, we can compare the different classifiers and the influence of their parameters, and choose the best performing one for the end results. The Service Classifier is available through a simple API, which has the service URL as input and a list of categories as output.

[28] http://www.dmoz.org/about.html

[29] http://www.programmableweb.com

[30] http://weka.wikispaces.com/Creating+an+ARFF+file

2.3 Additional Documentation

This deliverable captures the main functionalities of the Provisioning Platform Prototype, however, there is a plenitude of further documentation for each of the components. We created a website, which provides additional information on SWEET. It is available at http://sweet.kmi.open.ac.uk and contains a short introduction, as well as descriptions of the prototypes and documentation how to use them. SOUR is delivered with a flash demo movie representing a complete end-to-end scenario, focusing on the different steps of the annotation process. LUF is documented in its own web page as well (http://soa4all.isoco.net/luf/about), providing further details about the functionalities and explaining its usage.

iServe has its own pages too (http://iserve.kmi.open.ac.uk) and a wiki (http://iserve.kmi.open.ac.uk/wiki/index.php/Home), explaining what the purpose of the tool is, what are the main functionalities and how it can be used. All of the components are described in the SOA4All main wiki as well (http://soa4all.sti2.at/index.php/Main_Page). In addition, since parts of the Provisioning Platform have been used in two hands-on sessions, there is also a complete use-case scenario available at: http://iserve.kmi.open.ac.uk/wiki/index.php/Hands-on_Session.

3. Conclusions

This deliverable describes the components and the functionalities of the second Provisioning Platform Prototype. Through the SOA4All Provisioning Platform semantic web service technologies are more widely adopted, especially by supporting the creation of semantic web service descriptions by using both direct user input and automated information processing, based on prior user-provided input. In particular, this is enabled through SWEET and SOUR, which support users in creating semantic descriptions of RESTful and WSDL-based services. The annotation process is aided through the Annotations Recommender and the resulting semantic service descriptions can be directly published to iServe, where they can be browsed, searched and retrieved. This deliverable also presents the second prototype of the Feedback Framework, which enables users to rate and recommend services based on their experience.

Annex A.

This annex contains three publications and two submissions describing research done within the scope of the SOA4All Provisioning Platform:

1. Maleshkova, M., Pedrinaci, C., and Domingue, J. (2010) Semantic Annotation of Web APIs with SWEET, Workshop: 6th Workshop on Scripting and Development for the Semantic Web at Extended Semantic Web Conference, Heraklion, Greece

Available at:
http://www.semanticscripting.org/SFSW2010/papers/sfsw2010_submission_3.pdf

Abstract. Recently technology developments in the area of services on the Web are marked by the proliferation of Web applications and APIs. The development and evolution of applications based on Web APIs is, however, hampered by the lack of automation that can be achieved with current technologies. In this paper we present SWEET (Semantic Web sErvices Editing Tool) lightweight Web application for creating semantic descriptions of Web APIs. SWEET directly supports the creation of mashups by enabling the semantic annotation of Web APIs, thus contributing to the automation of the discovery, composition and invocation service tasks. Furthermore, it enables the development of composite SWS based applications on top of Linked Data.

2. Pedrinaci, C., Liu, D., Maleshkova, M., Lambert, D., Kopecky, J., and Domingue, J. (2010) iServe: a Linked Services Publishing Platform, Workshop: Ontology Repositories and Editors for the Semantic Web at 7th Extended Semantic Web Conference

Available at: http://kmi.open.ac.uk/people/carlos/publications/iserve-ORES2010.pdf

Abstract. Despite the potential of service-orientation and the efforts devoted so far, we are still to witness a significant uptake of service technologies outside of enterprise environments. A core reason for this limited uptake is the lack of appropriate publishing platforms able to deal with the existing heterogeneity in the service technologies landscape and able to provide expressive yet simple and efficient discovery mechanisms. In this paper we describe iServe, a novel and open platform for publishing services which aims to better support their discovery and use. It exposes service descriptions as linked data expressed in terms of a simple vocabulary for describing services of different kinds with annotations in diverse formalisms. In addition to describing iServe, this paper also highlights the set of principles behind iServe, which we believe are essential for other generic repositories of semantic information notably ontology repositories.

3. Pedrinaci, C., Lambert, D., Maleshkova, M., Liu, D., Domingue, J., and Krummenacher, R. (2010) Adaptive Service Binding with Lightweight Semantic Web Services Adaptive Service Binding with Lightweight Semantic Web Services, in eds. Schahram Dustdar and Fei Li, Service Engineering: European Research Results, Springer

Abstract Adaptive service selection, also referred to as late-binding, is acknowledged to provide a certain number of advantages to optimize the service provisioning process or to cater for advanced service brokering. SemanticWeb Services, that is services that have been enriched with semantic annotations is a typical technical approach to providing late-binding facilities. However, Semantic Web Services approaches such as WSMO and

OWL-S have not been popular with service providers. It has been hypothesized that a major cause of this is the quantity and complexity of logical statements required to describe a service in such heavyweight frameworks. This fact has indeed affected the acquisition of semantic annotations for services as well as the potential scalability for Semantic Web Services infrastructure. In this paper, we show how a lighter weight approach based on existing Web standards such as RDF and SPARQL can support the definition of service descriptions, service templates, and enable quite advanced service matchmaking in a scalable manner.

4. Pedrinaci, C., Liu, D., Kopecky, J., Maleshkova, M., and Domingue, J.: iServe: Using Lightweight Semantics for Unified Service Publication and Discovery, Submitted to Semantic Web In Use track of ISWC 2010.

Abstract. Semantic Web Services research has produced several conceptual models for enriching Web service descriptions and a plethora of discovery algorithms that exploit these semantic annotations for supporting discovery. Despite the efforts, however, we are still to witness a significant uptake of Semantic Web Services on the Web due in part to the complexity of the technologies and a considerable fragmentation between the conceptual frameworks. In this paper we describe iServe, a public registry that unifies service publication and discovery on the Web through the use of lightweight semantics. iServe builds upon lessons learnt from the Semantic Web to provide the first system that supports the publication and advanced discovery of different kinds of services (including “RESTful" Web APIs) described using previously incompatible formalisms. The registry contains all the existing Semantic Web Service test collections and additional annotations of real services, on which our evaluation illustrates the efficient semantic service discovery over Web services and Web APIs across different formalisms.

5. Maleshkova, M., Pedrinaci, C., Domingue, J., Alvaro, G., Martinez, I.: Using Semantics for Automating the Authentication of Web APIs Submitted to main track of ISWC 2010.

Abstract. Recent technology developments in the area of services on the Web are marked by the proliferation of Web applications and APIs. The implementation and evolution of applications based on Web APIs is, however, hampered by the lack of automation that can be achieved with current technologies. Research on semantic Web services is therefore trying to adapt the principles and technologies that were devised for traditional Web services, to deal with this new kind of services. In this paper we show that currently more than 80\% of the Web APIs require some form of authentication. Therefore authentication plays a major role for Web API invocation and should not be neglected in the context of mashups and composite data applications. We present a thorough analysis carried out over a body of publicly available APIs that determines the most commonly used authentication approaches. In the light of these results, we propose an ontology for the semantic annotation of Web API authentication information and demonstrate how it can be used to create semantic Web API descriptions. We evaluate the applicability of our approach by providing a prototypical implementation, which uses authentication annotations as the basis for automated service invocation.