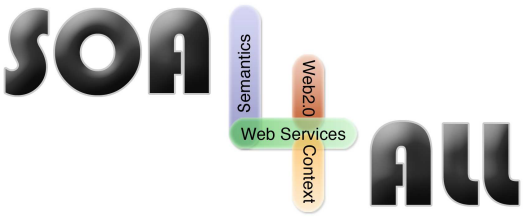

Project Number:	**215219**
Project Acronym:	**SOA4All**
Project Title:	**Service Oriented Architectures for All**
Instrument:	**Integrated Project**
Thematic Priority:	**Information and Communication Technologies**

D2.1.3 Service Provisioning Platform First Prototype

Activity N:	AC1 – Fundamental and Integration Activities	
Work Package:	WP2 – Service Deployment and Use	
Due Date:	M18	
Submission Date:	14/09/2009	
Start Date of Project:	01/03/2008	
Duration of Project:	36 Months	
Organisation Responsible of Deliverable:	The Open University	
Revision:	1.1	
Author(s):	Maria Maleshkova Guillermo Álvaro Rey Alex Simov	OU iSOCO ONTO
Reviewers:	Sven Abels Jacek Kopecky	TIE UIBK

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	21/06/2009	Document structure, table of contents	Maria Maleshkova (OU)
0.2	24/07/2009	Initial version	Maria Maleshkova (OU)
0.3	27/07/2009	Improved version with screenshots and step-by-step guide	Maria Maleshkova (OU)
0.4	04/08/2009	Completed descriptions of the Feedback Framework and the MicroWMSO Editor	Maria Maleshkova (OU), Guillermo Álvaro Rey (iSOCO)
0.5	05/08/2009	First version of Section 2	Maria Maleshkova (OU)
0.6	17/08/2009	Complete initial draft	Maria Maleshkova (OU), Guillermo Álvaro Rey (iSOCO), Alex Simov (ONTO)
0.7	18/08/2009	Refined draft	Maria Maleshkova (OU), Guillermo Álvaro Rey (iSOCO), Alex Simov (ONTO)
1.0	27/08/2009	Addressing reviewers' comments and improving the content	Maria Maleshkova (OU), Guillermo Álvaro Rey (iSOCO), Alex Simov (ONTO)
1.1	31/08/2009	Final version	Maria Maleshkova (OU), Guillermo Álvaro Rey (iSOCO), Alex Simov (ONTO)
1.1	14/09/2009	Final editing	Malena Donato (ATOS)

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

Nowadays, traditional web services are commonly used and have achieved a certain level of acceptance. Still, the wider adoption of web service technologies is hindered by the fact that most web service tasks require the completion of manual activities and as a result, web service-based applications suffer from a lack of automation for key activities such as discovery and composition. A solution to this problem is given by the semantic web service technologies, which enhance services with semantic descriptions that are amenable to automated reasoning, thus paving the way for the application for knowledge-based algorithms to better support the automation of service-related tasks. The SOA4All Provisioning Platform supports the creations 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 Provisioning Platform Prototype focuses on describing the implemented functionalities of the MicroWSMO and WSMO-Lite Editors, which support users in creating semantic descriptions of RESTful and WSDL-based services. In addition, this deliverable presents the first prototype of 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 first 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.

1. Introduction

This deliverable introduces the components and features of the first Provisioning Platform Prototype. It focuses on three main implementation contributions: the MicroWSMO Editor, the WSMO-Lite Editor and the Feedback Framework.

1.1 Purpose and Scope

The purpose of this deliverable is twofold. First, it serves as a documentation of the first Provisioning Platform Prototype, providing information about installation and configuration guidelines as well as a description of the main functionalities of the components. Second, it can be used as 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. 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 first Provisioning Platform Prototype, including a list of the functionalities implemented by each of the prototype components as well as a summary of the planned prototype work for the next Provisioning Platform deliverable; 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 as well as a step-by-step scenario of how to use the editors and the feedback framework; 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;
2. The **Annotations Recommender**, which reduces the manual effort required by users, while annotating services, by automatically suggesting suitable domain ontologies and 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 and tags;
5. The **Import Facilities**, for importing existing semantic services and compositions;
6. The **Process Editor**, which is developed within Task 2.6.

The first Service Provisioning Platform Prototype is based on three main component implementations, focusing on the Simple SWS Editing Framework and the Feedback Management Framework. Figure 1 provides an overview of the architecture of the Provisioning Platform, while the highlighted components are the ones included in the first prototype. The MicroWSMO editor, the WSMO-Lite editor and the Feedback Framework implementations are described in more detail in the following sections, focusing on the provided functionalities and user support.

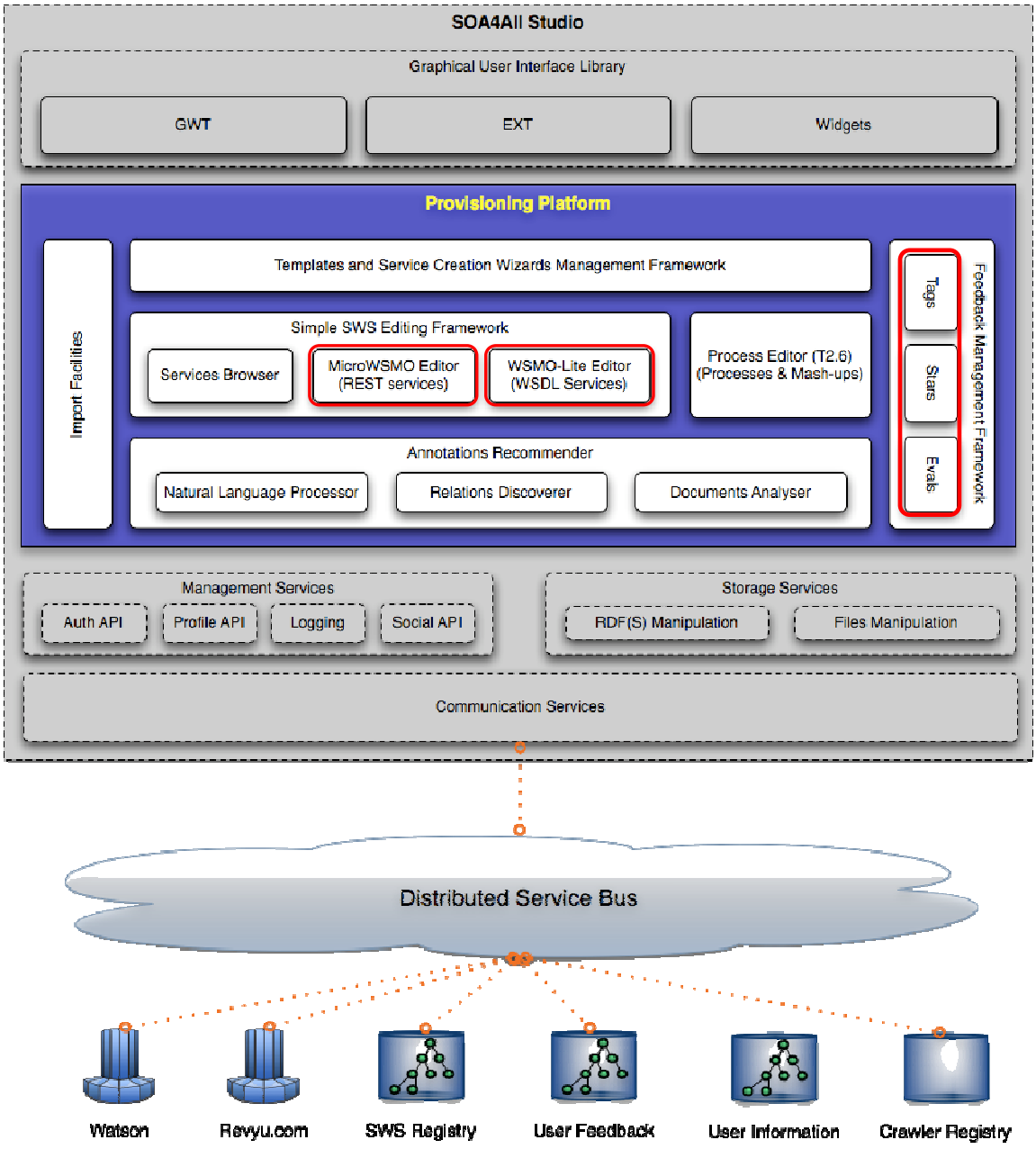

Figure 1: Service Provisioning Platform Architecture

1.2.1 MicroWSMO Editor

Both the MicroWSMO editor and the WSMO-Lite editor are part of the Simple Semantic Web Services Editing Framework. It provides support for browsing existing services, annotating WSDL services via the so-called WSMO-Lite Editor and RESTful services with 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.

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 my the SOA4All dashboard. This first prototype implementation of the Simple SWS Editing Framework focuses on providing functionalities of the MicroWSMO and WSMO-Lite editors, while the Service Browser will be implemented at a later stage.

The MicroWSMO editor, as the name suggests, supports users in creating MicroWSMO semantic annotations of RESTful services. We have decided to call it SWEET: Semantic Web sErvices Editing Tool and this name for the editor is used both in this deliverable and in related publications. The MicroWSMO editor is implemented in two main versions. The first version takes the form of a vertical widget displayed within a web browser. 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. Two current publications about the MicroWSMO editor are given in the Annex.

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.

Each of the MicroWSMO editor implementations addresses different user needs and supports different use cases. The browser-based lightweight version is suitable for users who are browsing for RESTful services and want to directly annotate the currently displayed description. Whenever a user wants to create MicroWSMO annotations of the currently viewed service, he/she can simply click on the bookmarklet[1] of the MicroWSMO editor, initiate it and directly use it. There is no need to start the complete SOA4All dashboard and to load the RESTful service description. Instead, the user can complete all the service annotation tasks directly from the web browser.

On the other hand, the implementation of the MicroWSMO editor, integrated in the SOA4All dashboard, is suitable in cases where the user performs multiple tasks, in addition to RESTful service annotation, and where a set of RESTful services are already loaded in the dashboard, for example through the crawler component. For instance, the user can annotate a multitude of services from the same domain and switch between the MicroWSMO editor and the WSMO-Lite editor, depending on the service type. After the services are annotated, he/she can use them in service compositions or directly execute them. Such more complex use cases are better facilitated by the dashboard-integrated MicroWSMO editor.

[1] A *bookmarklet* is a Web site bookmark that includes processing. Bookmarklets are written with JavaScript code and perform additional useful functions. Just like dragging a URL to the Favorites menu, the bookmarklet can be dragged from any Web site that offers it. When clicked, the JavaScript in the bookmarklet is executed.

Even though, both implementations share the same core functionalities, the dashboard editor has some additional features. It includes a three-based view, which visualizes the annotated service and enables the deleting and editing of annotations. In addition, the insertion of hRESTS tags is made easier for the users by enabling only the tags, which can be used in the current state of the annotation, and disabling all other tags. In this way, the user is intuitively supported in making correct service annotations. The domain ontology functionalities are also improved, by implementing a newer version of Watson's[2] API and using paginated requests, which reduces the waiting times. Finally, the semantic service property annotations can be deleted, which is not possible in the browser-based version of the MicroWSMO editor.

The two MicroWSMO editors differ in implementation technology as well. The browser-based prototype used the ExtJS[3] JavaScript visualization library, JavaScript and Reverse Ajax communication. In contrast, the dashboard editor is aligned with the technology of the other SOA4All components and uses ExtGWT[4], JavaScript and a proxy, which facilitates communication by rewriting the HTML DOM of the website describing the RESTful service. The dashboard is also session based and stores the ontology search results of a user in his/her session.

In summary, we provide two prototype implementations of the MicroWSMO editor, which support users in making semantic annotations of RESTful services in different use cases. The first implementation is a web application displayed directly in the browser. It has all necessary core functionalities but is very lightweight and is suitable for users who want to annotate a RESTful service on-the-fly. The second implementation in completely integrated in the SAO4All dashboard, has additional features and is suitable for users, who want to annotate a multitude of services.

1.2.2 WSMO-Lite Editor

The WSMO-Lite Editor 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. Using the tool, the user is able to create new annotations on existing description and also modify or remove already created annotations. The visual tree-based representation emphasizes the essential parts of the modelling artefacts, while filtering all irrelevant information. Informative tooltip balloons and auxiliary windows provide additional detailed information on demand. The editing process itself uses simple and well-accepted management techniques like *Drag & Drop*, context and drop-down menus to prevent the user from unnecessary technical particularities.

This component intensively makes use of the *Storage Services*, developed in WP2 and described in D.2.4.2. Through their RESTful API, the editor realizes its primary input/output functionalities like retrieval of service descriptions (WSDL) and semantic models (different kinds of ontologies); storage of service annotations (SAWSDL); retrieval of additional resources like lifting and lowering schemas. The editor offers a lightweight repository browser for easier navigation of the storage content.

The editor supports the following set of annotation functionalities:

[2] http://watson.kmi.open.ac.uk

[3] http://extjs.com/

[4] http://extjs.com/products/gxt/

- 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.

1.2.3 Feedback Framework

The Feedback Framework is implemented as a server-side module (`SOA4All-dashboard-feedback-service`) inside the SOA4All Studio dashboard, exposing its functionalities to client-side modules within the SOA4All Studio via its API. A client-side module, which is already benefiting from the capabilities of the Feedback Framework is the Service Consumption Platform, which allows users to rate, tag and comment on services, and to retrieve this kind of information provided by previous users about them.

The functionalities that we have implemented permit interacting with the framework in two different directions: Allowing users to provide new feedback information, and allowing them to retrieve it. This information is stored in the Semantic Spaces thanks to the invocation of the Storage Services.

When retrieving feedback information on a particular item, the framework is able to return it related to a particular user, or the generic and aggregated feedback provided by different users. This is the reason there are two overloaded methods for each of the “get” operations, one with the user as parameter, and the other without it.

Following our general explanations on the framework available operations, these are the methods that the Feedback Framework API provides so far:

Ratings:

```
public String rateItem(URI itemId, double rating, URI userId);
public RatingsResponse getRating(URI itemId):
public RatingsResponse getRating(URI itemId, URI userId);
```

Comments:

```
public String commentItem(URI itemId, String comment, URI userId);
public List<Comment> getComments(URI itemId);
public List<Comment> getComments(URI itemId, URI userId);
```

Tags:

```
public String tagItem(URI itemId, String[] tags, URI userId);
public List<TagResponse> getTags(URI itemId);
public List<TagResponse> getTags(URI itemId, URI userId);
```

The class `RatingsResponse` contains information such as the number of ratings and their average value, while `Comment` contains the text of each comment and the user who made it. On the other hand, `TagResponse` provides the text of the tag and its frequency.

Regarding the underlying interaction with the Storage Services: For the “writing” operations, the relevant RDF that follows the Review Schema[5] and Tag Ontology[6] is created in order to invoke the RESTful Storage Operation. For the “reading” operations, a SPARQL query is created to retrieve the necessary information.

1.3 Customisation Done for This Deliverable

The here described first Provisioning Platform Prototype and its components are directly contributing to the functionalities of the final version of the Provisioning Platform. There are no major assumptions or customizations made in this version of the components. There are two minor points, which are part of current work in progress and can be shortly mentioned. First, the layout of the dashboard version of the MicroWSMO editor, given in the screenshots, is not aligned with the overall layout of the dashboard. However, this will be corrected by the time the first Provisioning Platform Prototype is submitted. Second, the MicroWSMO editor is not connected directly to the Storage Services yet. Instead, the user has the option to store the created semantic annotations in a suitable place of his/her choice. This will the corrected for the following version of the editor.

1.4 Roadmap for Future Plans

The next Provisioning Platform Prototype will be available in month 30 and will be the final and complete version of the Provisioning Platform. Still, versions and updates of separate components will be released periodically before this deadline. The prototype will include improved versions of the MicroWSMO and WSMO-Lite editors, in particularly focusing on application stability and usability. However, it will also include a number of newly implemented components. It will contain the Process Editor, which is developed as part of Task 2.6. The Annotations Recommender prototype will assist users in making semantic annotations of web services by automatically suggesting suitable domain ontologies and service property annotations. It will include a component for computing similarity measures between existing SWS and a recommender component, which compares the current service, which the user wants to annotate, to already annotated services. The functionalities of the Annotations Recommender prototype will not be directly visible, since it does not have its own user interface. Instead, they will rather be recognized through new elements of the MicroWSMO and WSMO-Lite Editors.

The next Provisioning Platform Prototype will also include an implementation of the Templates and Service Creation Wizards Framework, containing a set of templates for creating complex composite services and wizards for assisting the user in completing different tasks supported by the Provisioning Platform. In addition, the prototype will include implementation of the Import Facilities, providing functionalities for importing existing knowledge models in order to reduce the overhead for annotating services but also for eventually reducing the mismatches between different knowledge models, by using common ontologies.

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

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

2. Prototype Documentation

This section of the deliverable provides practical information about using the first Provisioning Platform Prototype. It provides installation and configurations instructions as well as guidelines about the functionalities of the different elements of the editors' GUIs. The prototype documentation also includes step-by-step descriptions of how to created RESTful and WSDL-based semantic service descriptions by using the MicroWSMO and the WSMO-Lite editors. This section is concluded by description of the Feedback Framework component functionalities.

2.1 Installation and Configuration

The MicroWSMO editor, the WSMO-Lite editor and the Feedback Framework comply to the installation and configuration requirements of the SOA4All dashboard. These components are part of the Provisioning Platform and are completely integrated in 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.

The latest version of the Dashboard is available and regularly updated at: http://coconut.tie.nl:8080/SOA4All/. However, it will be moved to the official SOA4All website in the process of the development.

The web-browser-based version of the MicroWSMO editor, on the other hand, is not integrated in the dashboard and requires some simple installation steps before it can be used. The current version of the prototype requires the installation of the Firefox[7] browser, version 3.5 or later, with installed and enabled Firebug[8] Firefox add-on, which supports the communication between the website content and the JavaScript-based MicroWSMO editor implementation. The installation of the editor itself is very simple. It is done by dragging the editor link and dropping it in the bookmark panel of Firefox. After this, the editor can be directly started by click on the bookmarklet and no additional installation is required. The current editor prototype is available at http://sweetdemo.kmi.open.ac.uk/.

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 pointing out 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, and the Feedback Framework.

2.2.1 MicroWSMO Editor

As already mentioned, the first prototype of the Provisioning Platform includes two implementations of the MicroWSMO editor, which support users in annotating RESTful services in different use cases. The first implementation presented here is the web-browser-based MicroWSMO editor. Figure 2 visualizes the GUI of the MicroWSMO Editor, which

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

[8] http://getfirebug.com/

takes the form of a vertical widget appearing on top of the currently viewed webpage. It consists of four main panels, including the *hRESTS Tags* panel, the *Service Properties* panel, the *Domain Ontologies* panel and the *Annotations* panel. Each of these panels will be described in detail in the following sections.

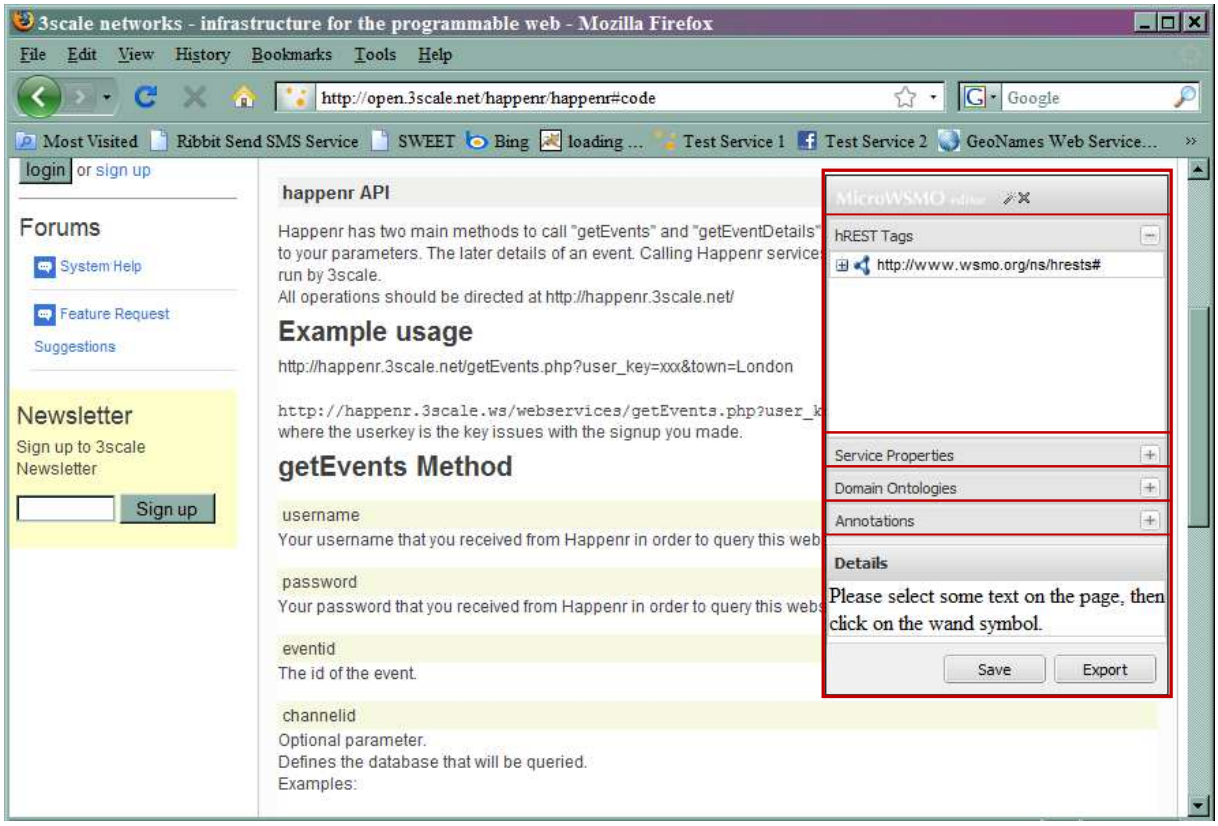

Figure 2: Lightweight MicroWSMO Editor

The GUI has also a *Save* and an *Export* button as well as an icon for searching for domain ontologies. As it can be seen, the implementation is very lightweight and contains only a few elements. Still these elements provide the core functionalities needed in order to support the user in annotating RESTful service descriptions.

The common tasks of the process of annotating RESTful services are:

1. Indentifying service properties by inserting hRESTS tags in the HTML service description.
2. Searching for domain ontologies suitable for annotating the service properties.
3. Annotating service properties with semantic information.
4. Saving (annotated HTML) or exporting (extracted RDF) the annotated RESTful service.

Each of these tasks is described in detail and is visualized by a screenshot of the MicroWSMO editor.

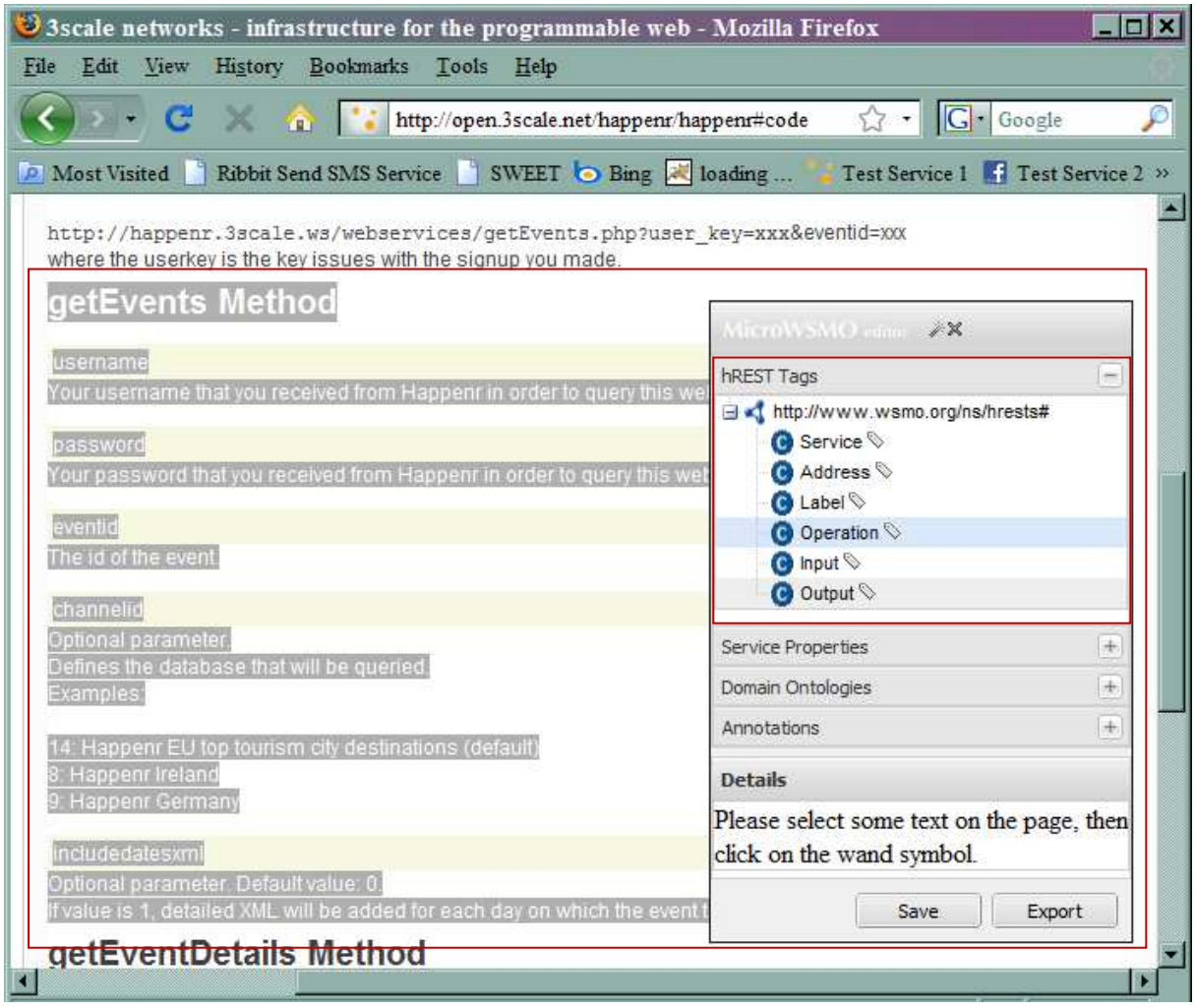

Figure 3: Inserting hRESTS Tags

Indentifying service properties by inserting hRESTS tags

This MicroWSMO editor is designed for use cases when a user is browsing the Web and finds an interesting RESTful service, which he/she wants to semantically annotate. The editor can simply be started by clicking on a bookmarklet, which executes the JavaScript initializing the web application. After the editor is started, the user can begin to insert hRESTS tags for marking the service properties.

In contrast to services described in WSDL, which have a clear structure and definition of the operations, inputs and outputs, RESTful service descriptions are commonly given in plain text. Therefore, it is necessary to somehow mark the service properties in the text description. The MicroWSMO specification (D3.4.3) does this by introducing the hRESTS microformat. The user can insert hRESTS tags by selecting the part of the description related to a particular property and clicking on the appropriate tag. Figure 3 visualizes how an operation can be marked. The *getEvents* operation content is selected and is marked by clicking on the *operation* node in the hRESTS tree in the hRESTS Tags panel. As a result the HTML tag class="operation" is inserted in the corresponding HTML element. In this way, the service properties are recognizable and machine processable. Naturally, in order to create a meaningful service structure, the user should start by marking the complete service body, followed by the operations, the inputs and outputs of the operations and the

corresponding address. The user can also mark the names of the service and of the operations as labels, in order to make the resulting MicroWSMO description more user-friendly. After the user builds up the service structure by marking all service properties with hRESTS tags, he/she can continue by searching for suitable domain ontologies for semantically annotating the service.

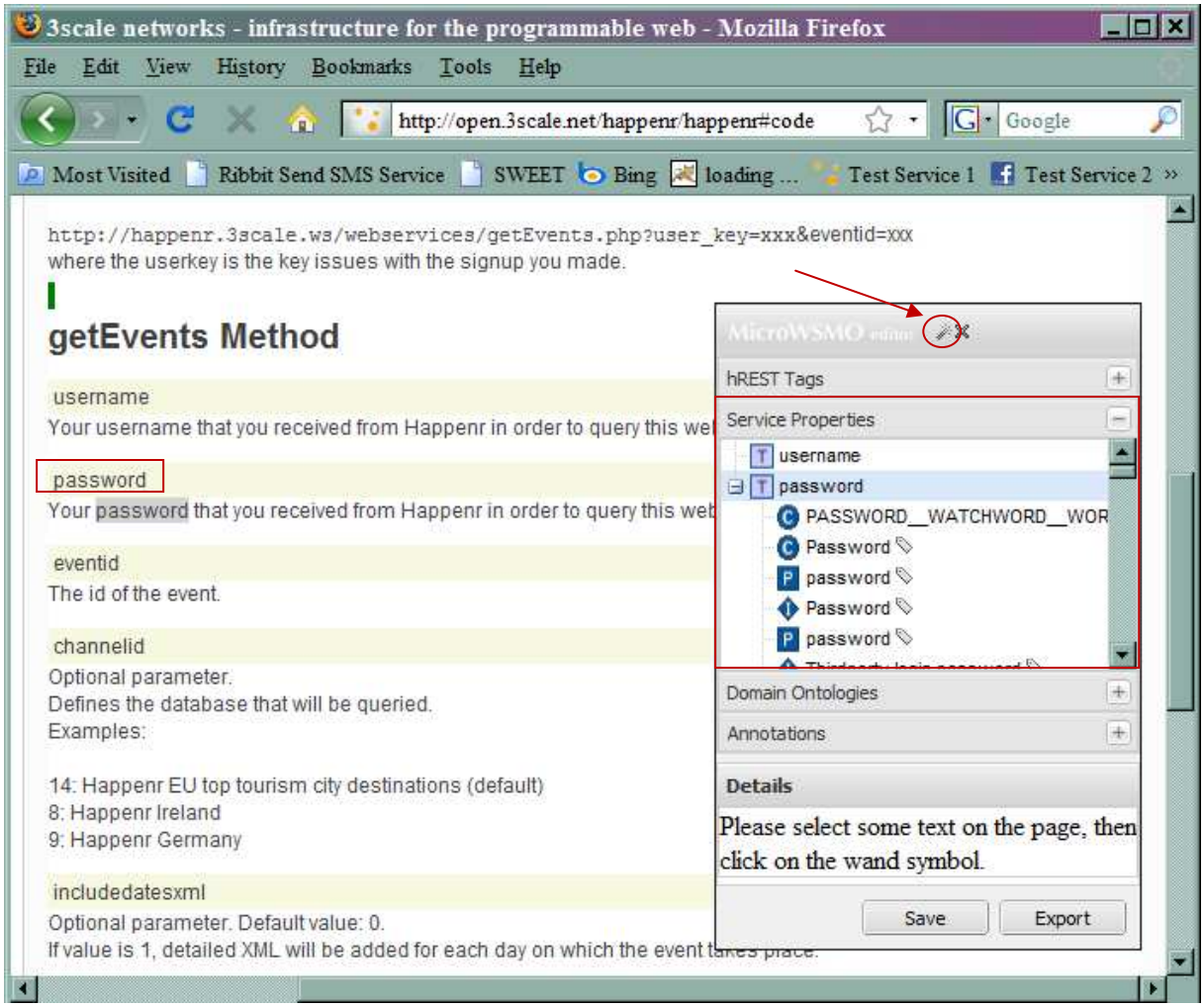

Figure 4: Searching for Domain Ontologies

Searching for suitable domain ontologies

The user can start searching for suitable domain ontologies by selecting a service property, for example the input parameter “password” and clicking on the magic wand symbol. This sends a request to Watson’s API[9] for retrieving matching ontologies. The result is a tree structure, where each of the nodes represents a matching concept, property or instance and the corresponding ontology. For example, as it can be seen in Figure 4, the search based on “password” returned a concept with label “PASSWORD_WATCHWORD_WOR…”, followed by a “Password” concept, followed by a “Password” property, followed by a “Password” instance, etc.

[9] http://watson.kmi.open.ac.uk/WS_and_API.html

However, the user needs some additional information in order to be able to decide if this semantic annotation is suitable for the service property. This additional information is provided by clicking on the matching node, which expands and provides details about the concept and the corresponding ontology. The user can receive more information about each of the ontologies by switching to the *Domain Ontologies* panel (seen in Figure 5), which includes a list of all concepts as well as a list of all the matching properties, which were found for this ontology. For example, the ontology starting with "http://csd.abdn.ac.uk/research/..." is a match for both the password and username input parameters and contains a list of concepts, which can be viewed by clicking on the "All concepts" folder. Based on this information the user can decide to associate a particular service property with semantic information.

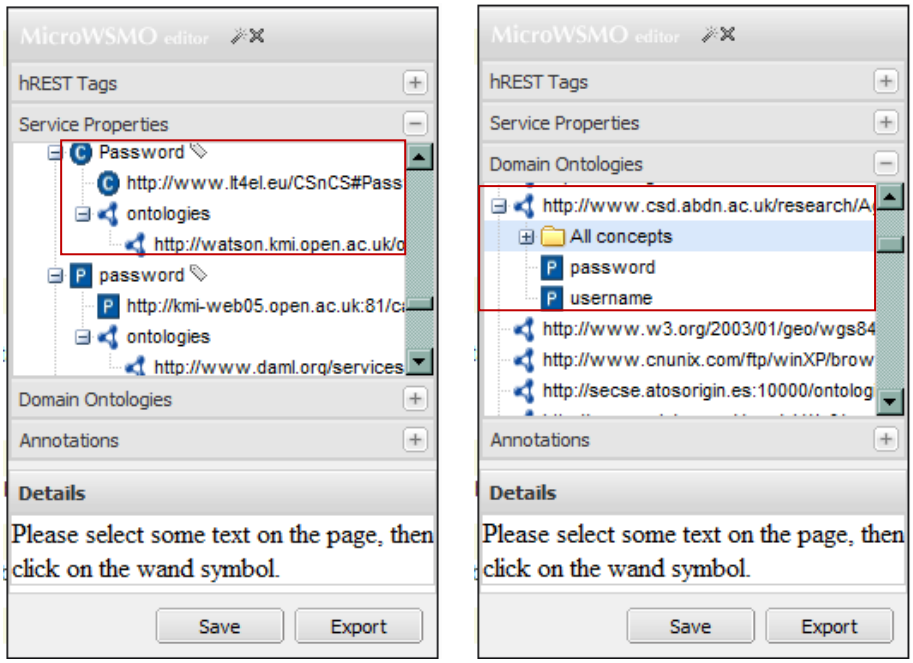

Figure 5: Choosing a Domain Ontology

Annotating service properties

The annotation of service properties can be easily done by selecting the service property and clicking on the corresponding semantic concept. For example, as it can be seen in Figure 6, "password" is automatically highlighted in red and can be associated with the "password" concept by clicking on it. As a result, the rel="model" and href HTML tags are inserted and the modified HTML is the following:

```
<a><h3 id="nid2" background-color="lightgrey" rel="model"
href="http://www.daml.org/services/owl-s/1.1/Concepts.owl#Password">password</h3></a>
```

The user can annotate as many service properties as necessary and use only one or different ontologies. The resulting HTML service annotation is in accordance with the MicroWSMO specification and enables the automatic extraction of RDF MicroWSMO service descriptions from the annotated HTML.

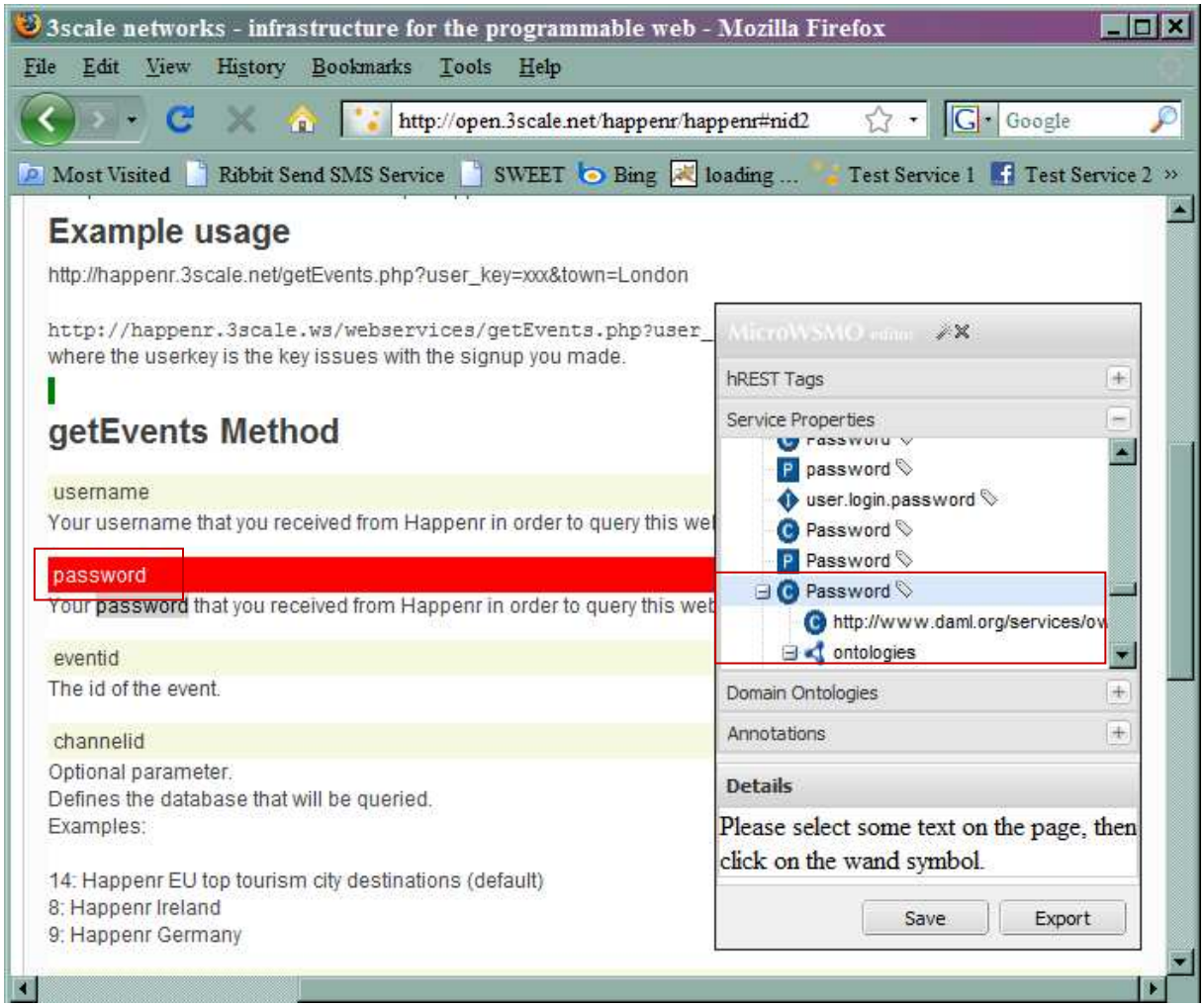

Figure 6: Making Semantic Annotations

Saving or exporting the annotated RESTful service descriptions

After the insertion of hRESTS tags is complete and the required service properties are linked to semantic information, the resulting annotated HTML can be saved. The data can be stored locally by the user or be posted directly in the SOA4All service repository. The annotated HTML can be used for extracting RDF MicroWSMO service descriptions, which can easily be done by clicking on *Export*.

The second implementation of the MicroWSMO editor is more extensive and is integrated in the SOA4All dashboard. It is based on the ExtGWT technology and has more visualization components than the web-browser-based implementation. Figure 7 illustrates the GUI of the MicroWSMO Editor. It consists of three main panels, including the *Semantic Description* panel, the *Navigator* panel and the *Annotation Editor* panel. It provides the same common functionalities, including: indentifying of service properties by inserting hRESTS tags in the HTML service description; searching for domain ontologies suitable for annotating the service properties; annotating service properties with semantic information and saving or exporting the annotated RESTful service. However, it provides a number of additional features, which make the using of the tool more user-friendly and the operation more stable.

Indentifying service properties by inserting hRESTS tags

The main difference between the web-browser-based editor and this one is that here the service description is visualized within the editor (in the *Navigator* window) instead of in the browser itself. The insertion of hRESTS tags is done in the same way, by selecting the text describing the service property and clicking on the corresponding node in the hRESTS tags tree. However, there are a number of main important additional features that need to be mentioned. First, each of the tag nodes is enabled and disabled depending on whether the user can select it in the current status of the service annotation or not. For example, the Input node is disabled, if there are no marked operations in the service description. In this way, the user is guided through the process of marking service properties and it is ensured that the resulting service structure is correct.

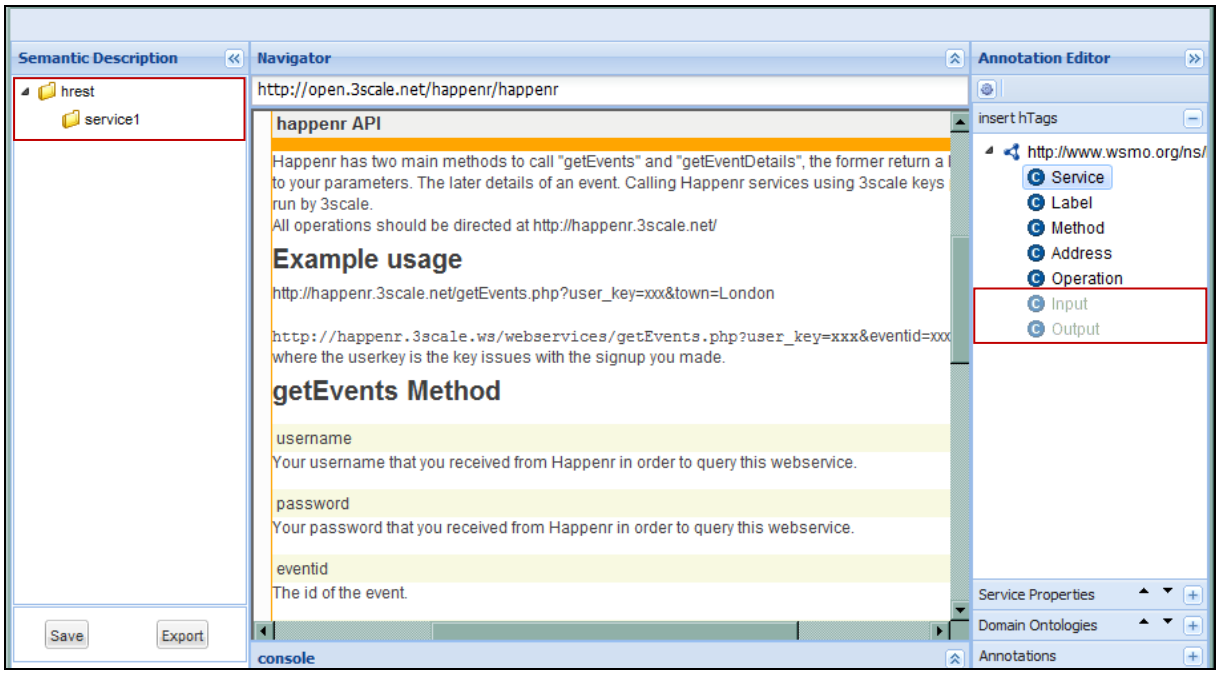

Figure 7: Dashboard MicroWSMO Editor

Second, the inserted hRESTS tags are represented in a tree structure in the *Semantic Description* panel, which tracks the user's annotation actions. The tree shows, which service properties are already marked and if the user recognizes a mistake, he/she can delete the corresponding hRESTS node and remark it.

Searching for suitable domain ontologies

The domain ontologies search is again based on Watson. The user can search for suitable domain ontologies for a given service property by selecting it and clicking on the search icon (Figure 8). The results are displayed in the *Service Properties* panel. This panel has a number of additional useful features. First, the retrieving of ontologies matches is paginated. If the first set of ontology results is insufficient, the user can search for more results by clicking on "view more". Second, the search is session based and the user preserves his/her ontology search while annotating different service descriptions. Third, all search results can be collapsed or expanded, by clicking on the up and down arrows on top of the panel. This makes browsing of the ontology matches easier. Finally, the search is restricted with a timeout so that if there are problems with Watson's API or the response is taking too long, the application will not be blocked.

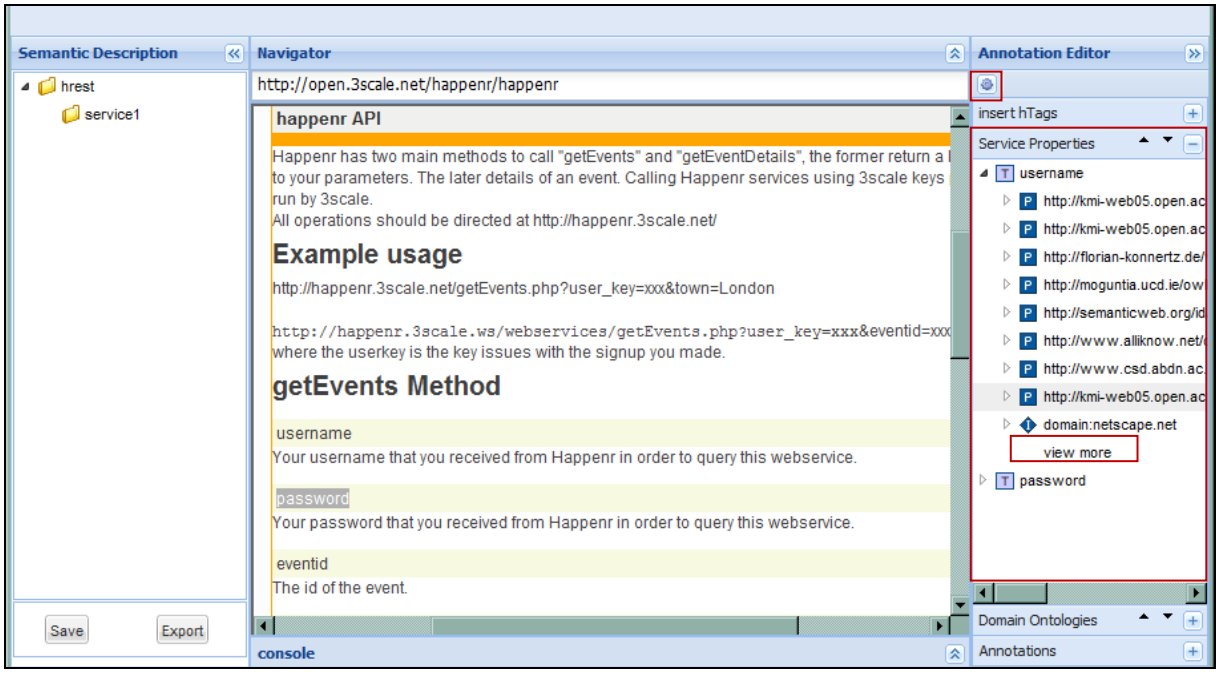

Figure 8: Service Properties Panel

The implementation of the *Service Properties* and *Domain Ontologies* panels supports the user in choosing a suitable ontology for annotating the individual service properties of the complete RESTful service. These supporting functionalities are visualized in Figure 9. The user can view the URI of each of the matching concepts, properties or instances and the corresponding ontology. Additional information is available in the *Domain Ontologies* panel, which shows all service properties that can be annotated with one particular ontology as well as a list of all concepts. The entries of both panels can be expanded or collapsed in order to ease the navigation.

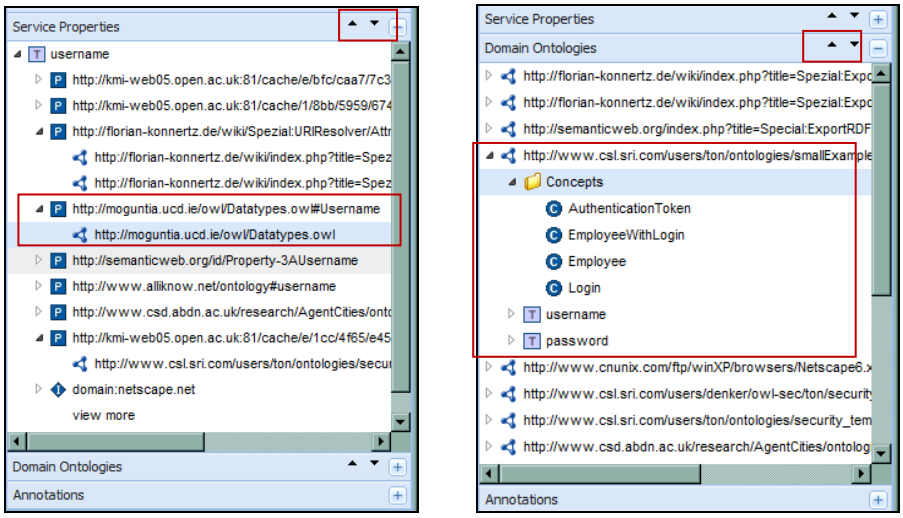

Figure 9: Exploring Domain Ontologies

Annotating service properties

Once the user has decided, which ontology to use for the service property annotation, he/she can do an annotation by selecting the part of the service and clicking on “Semantic

Annotation" in the *Service Properties* context menu. An example annotation is visualized in Figure 10. The result of the annotation is the same as when using the web-browser-based MicroWSMO editor: the model and href tags are inserted in the HTML to mark the association of the particular HTML element with the semantic concept.

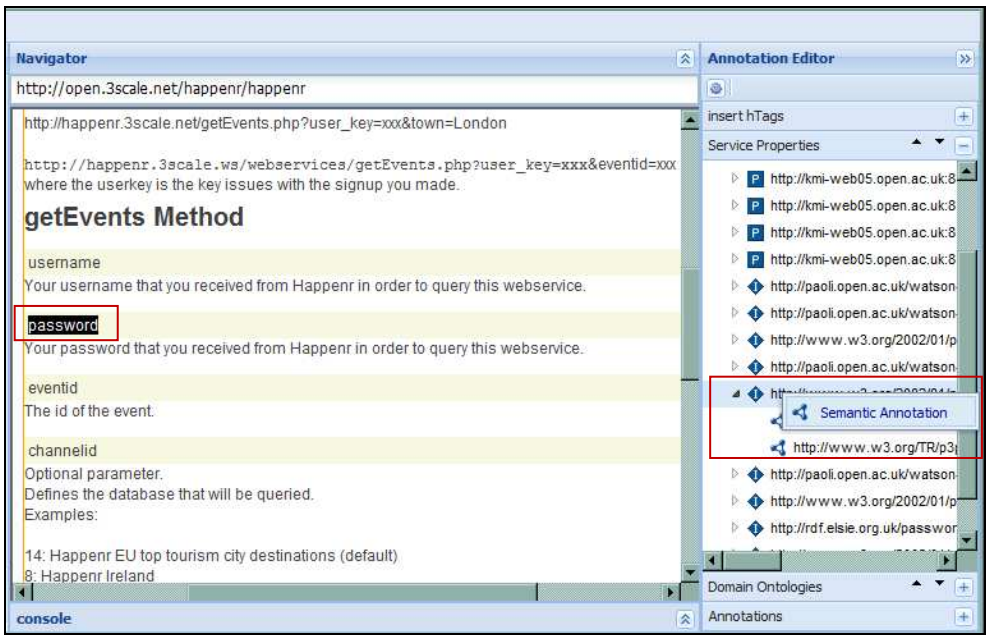

Figure 10: Inserting Semantic Annotations

A summary of the already made annotations it given in the *Annotations* panel. An additional functionality of this panel is that annotations can be removed by choosing "Delete" from the context menu. In this way, the user can remove incorrect annotations and substitute them with new ones without having to reload the tool and start the annotation process from the very beginning.

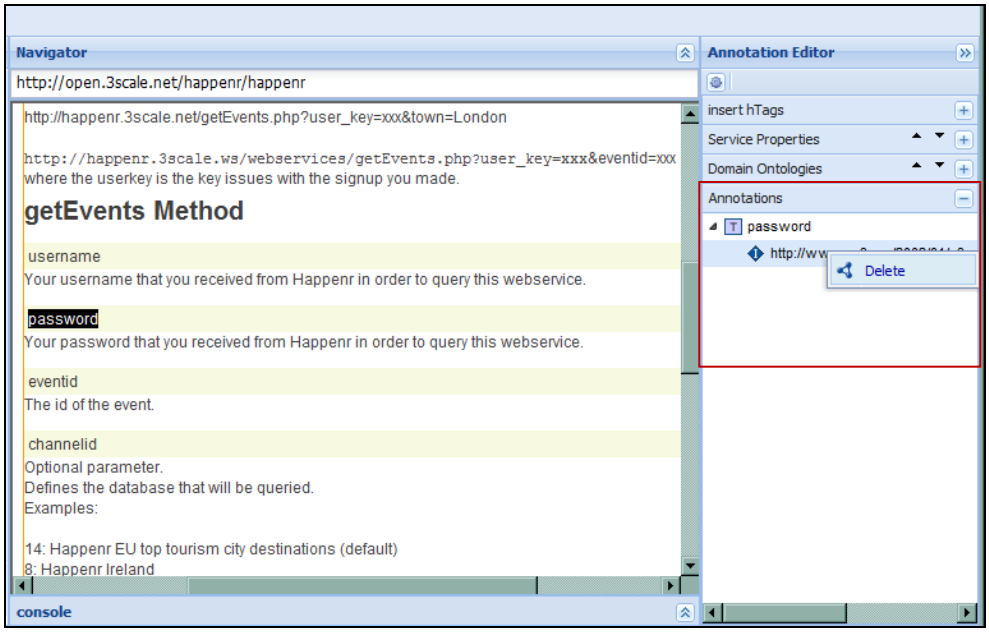

Figure 11: Deleting Semantic Annotations

Saving or exporting the annotated RESTful service descriptions

Finally, when the user is finished with annotating the HTML RESTful service description with hRESTS and MicroWSMO tags, the resulting annotated service can be saved or exported by clicking on the “Save” or “Export” buttons. Figure 12 shows that when the user clicks on “Save”, a pop-up window is opened, which allows him/her to choose a destination for saving the annotated HTML. The export provides a RDF transformation of the annotated HTML. The result is the MicroWSMO description of the semantically described RESTful service. In future implementations of the editor, SWS can directly be published in the SAO4All service repository, where they can be reused and discovered by other users.

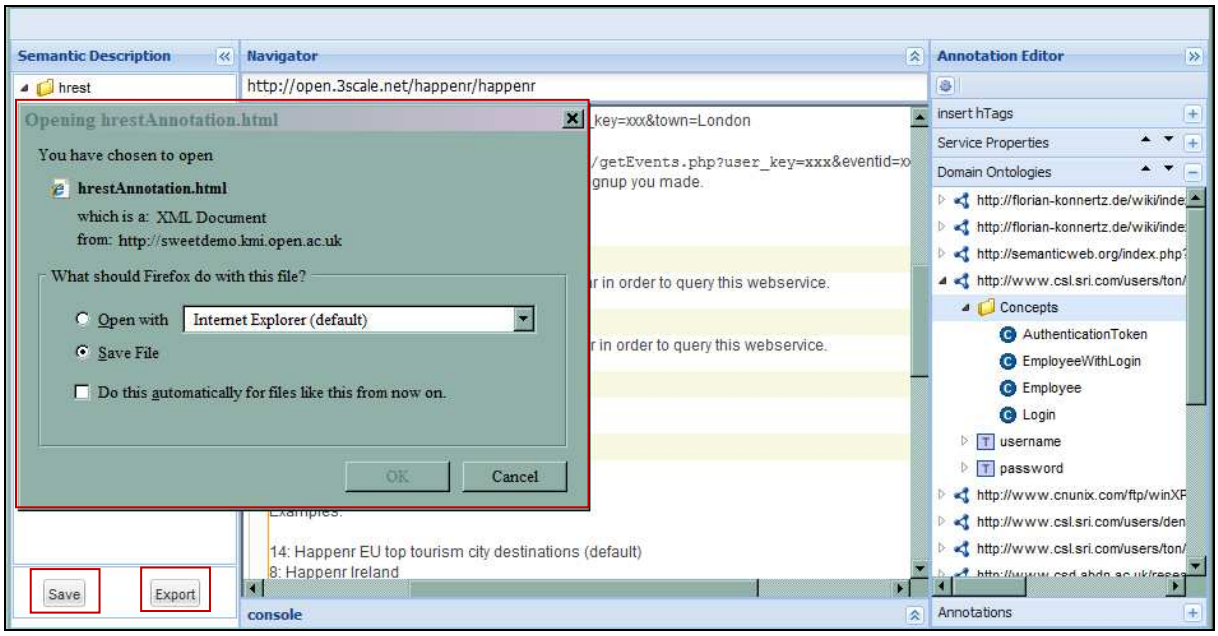

Figure 12: Saving Annotated HTML

Further functionalities of the dashboard MicroWSMO Editor

It is important to point out that the editor provides options for customizing the way service descriptions are viewed. First, if the *Navigator* panel displays services, which already contain MicroWSMO elements, these elements will be recognized and automatically highlighted so that the user can manipulate them and integrate them in his/her own annotation of the service. Second, the way the service properties and semantic information is highlighted can be modified very easily by simply substituting the current CSS file with a new one, which uses different text font and colours.

2.2.2 WSMO-Lite Editor

The WSMO-Lite editor is implemented as a component of the dashboard of the SOA4All Studio. It can be activated either from the main menu of the dashboard (Figure 13) under *WSMO-Lite* category, or from the *Plugin overview* page (Figure 14).

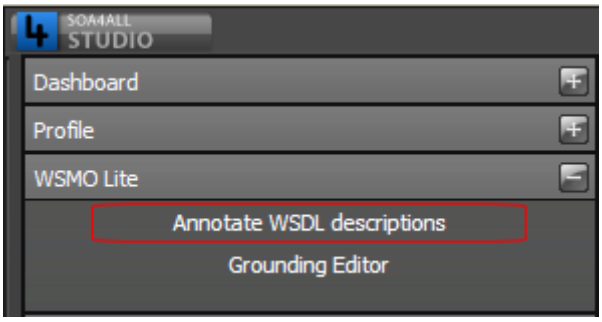

Figure 13: WSMO-Lite Editor Activation from the Dashboard Menu

Figure 14: WSMO-Lite Editor Activation from the Plug-in Overview Page

Having selected any of the two activation methods, the browser window is occupied by the WSMO-Lite Editor main window (Figure 15).

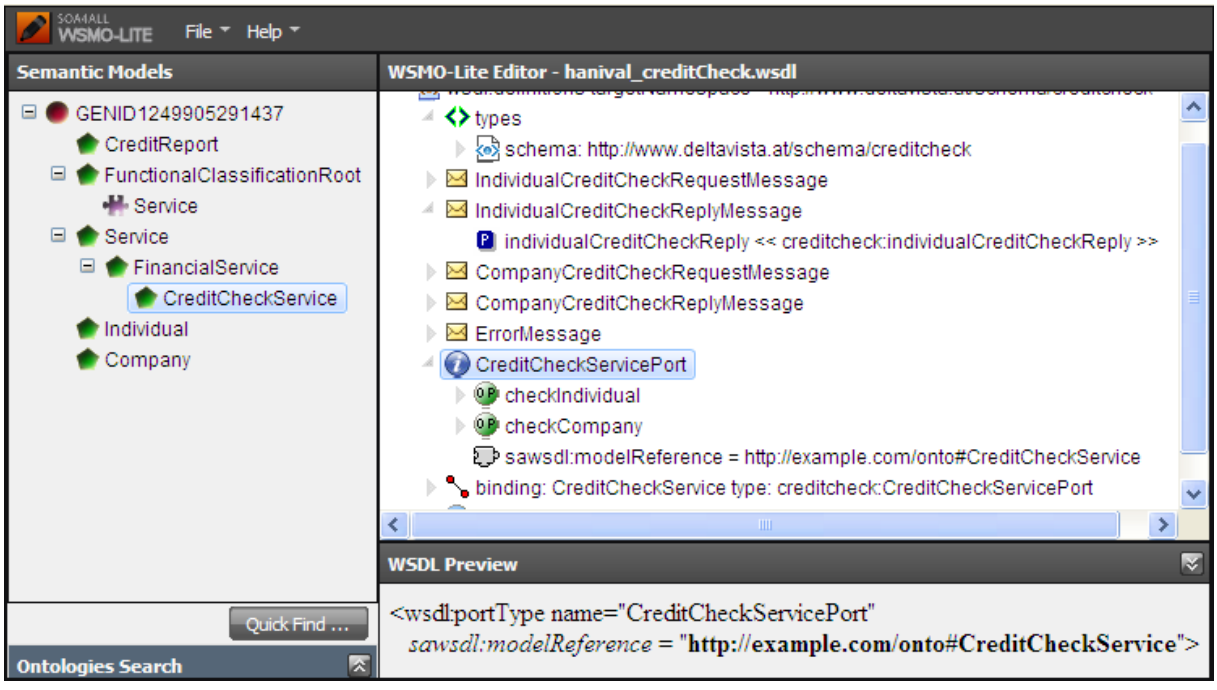

Figure 15: WSMO-Lite Editor

The editor area consists of several major components:

Semantic Models view

This panel contains tree representations of ontological data, which are relevant for the annotation process. The user controls what kinds of ontologies are loaded in this view. Regardless of the representation format, the ontologies are visualized in a uniform hierarchical style, representing class/sub-class, member-of and part-of relations. The nodes in the representation are described in more details (types, URIs, etc.) by tooltip messages when the mouse pointer is moved over them. Nodes corresponding to ontological classes and individuals are reference targets for service annotations. They can be dragged to certain WSDL elements in the service representation in the editor area and thus establishing links between the service descriptions and the semantic models. To facilitate the navigation in the ontological data (which in general can be large and complex), a ***Quick Find …*** functionality is supported, which helps the user in locating certain concepts while typing fragments from the names/URIs.

Main editor window

This is the place where the annotation process takes place. The component replicates to certain extent the WSDL structure of a service description in a tree-like form, containing self-explanatory nodes with appropriate image icons and text labels. In contrast to the XML representation of WSDL, which is intended mainly for machine processing, the visual component reveals only the essentials of the definitions, which are interesting for the human user. If necessary, additional information can be found in tooltip balloon messages on certain, selection sensitive elements.

The annotation of the service elements with references to semantic model entities is done by dragging the target entity from the *Semantic Models* view and releasing it on the selected for annotation schema element. As a result, a new visual node, labelled with *sawsdl:modelReference* prefix, appears as a child of the annotated node, denoting the annotation itself. This or any other annotation can be deleted (if necessary) with context menu action *Delete*.

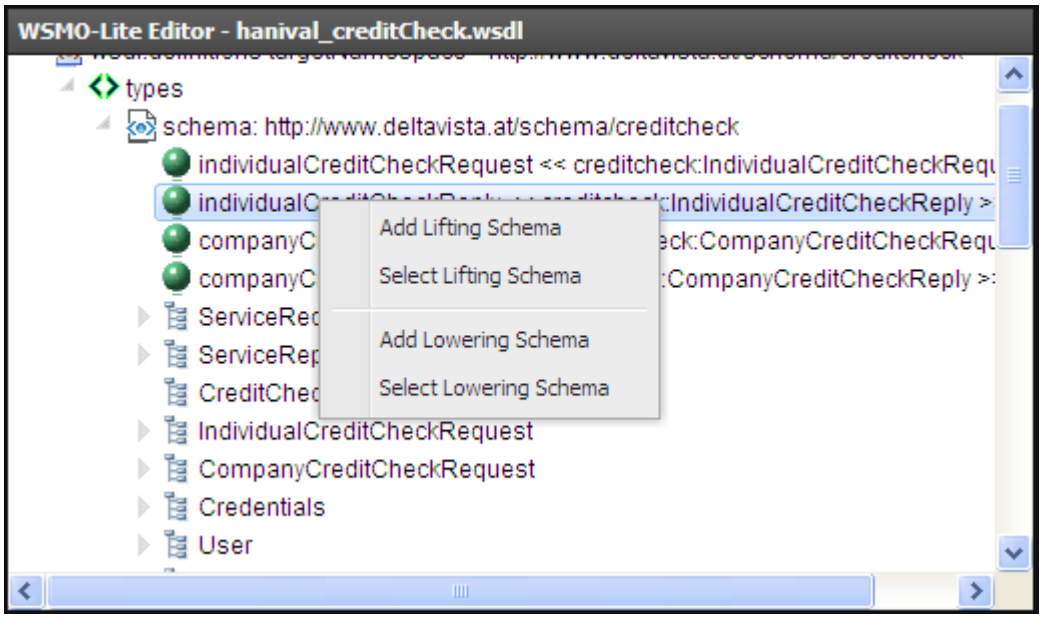

Figure 16: Lifting and Lowering Annotation

Certain service elements, more precisely XML schema elements, might have structural transformation annotations, which define the mapping between xml structures and ontological data. These mapping specifications play essential role when the service functionality is consumed, first by transforming conceptual information into xml message elements (lowering) and afterwards transforming the result messages into conceptual level responses (lifting). The annotations with such structural mappings references can be accomplished by a set of context menu actions on selected schema element (Figure 16).

The user can specify directly URLs of mapping schemas (*Add* actions) or can select schemas from the *Storage Services* repository (D.2.4.2). Similar to the model reference annotations, the structural transformation annotations appear as child nodes of the annotated element, marked by *sawsdl:loweringSchemaMapping* and *sawsdl:liftingSchemaMapping* labels and corresponding image icons. Multiple annotations for service elements are allowed for both model references and transformation annotations.

WSDL Preview

This window provides a short overview of the service element selected in the editor, emphasizing on the SAWSDL annotation. This way the user is able to observe the annotation both in graphical and textual formats in parallel.

Main Menu

The main menu component (Figure 17) provides the basic input/output related functionalities of the editor.

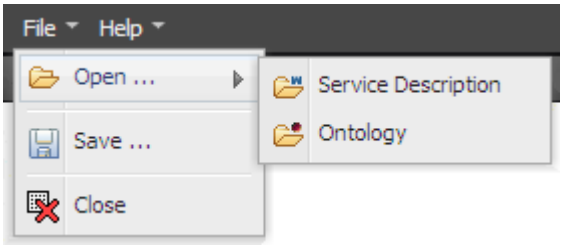

Figure 17: WSMO-Lite Editor Main Menu

- **Open … > Service Description**

  Opens a WSDL service description in the editor for annotation. The description might contain already SAWSDL annotations, which are detected and visualized correspondingly. The selection of a service description from the repository is supported by a simple *Repository Browser* component.

- **Open … > Ontology**

  Opens an ontology definition in the *Semantic* Models view. This conceptual information (classes, individuals, etc.) serves for annotating service elements. The selection of ontologies from the repository is supported by the *Repository Browser* component.

- **Save …**

  Saves the service description (opened in the editor) and any annotations made on it. The annotated service is stored in a repository supported by the *Repository Browser* component. On successful completion, the user is provided a direct link (URL) to the stored description, which can be shared or opened directly in a new web browser window.

- **Close**

  Closes the service description currently being opened in the editor.

Repository Browser

This component (Figure 18) enables browsing any repository, based on the *Storage Services* API developed in WP2 (D. 2.4.2) directly from the WSMO-Lite editor. Its primary role is facilitating the storage and retrieval of service and ontological descriptions.

The visual layout consists of two sections:

- **Storage Service URL**

  The deployment location of the *Storage Services*. Having provided a correct URL, the user can retrieve the content information of the repository by using the **List** button.

- **Repository Content View**

  Provides information about the content of the currently selected storage service instance. Apart from browsing the repository structure, the user can also retrieve a quick textual preview of any selected document without opening it in the editor. This is done by using context menu action **Preview** on any selected document.

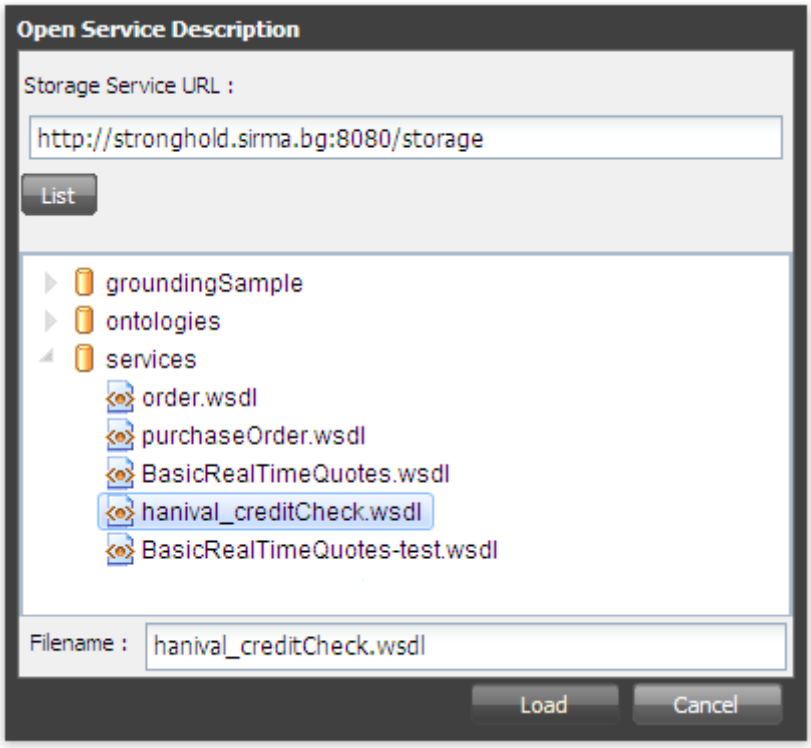

Figure 18: Repository Browser

A typical use case scenario for the WSMO-Lite editor component is:

1. Locating and opening a WSDL service description (which might already contain any annotations).
2. Locating and loading semantic models (ontologies) appropriate for annotation (if they are not already loaded).

3. Adding, deleting or simply browsing service annotations.
4. Saving the annotated service description in a repository.
5. Closing the service description (done automatically if a new one is loaded).

2.2.3 Feedback Framework

The Feedback Framework is a server-side module inside the SOA4All Studio. As such, it is not intended to be used by end-users directly, but rather through the interaction with client-side modules that make use of the framework.

Suitably enough, the Feedback Framework is already connected to the Service Consumption Platform client-side module, which leverages the functionalities of this framework. Thus, the best way of "using" the framework is by connecting to the Consumption Platform (available through the main SOA4All Studio dashboard area) and interacting with it.

Inside the Consumption Platform, one can open a service, selecting it from the different options in the left panel. General feedback information about the service is provided under the "Details" tab, so the aggregated rating is displayed as well as the tags used by different users to describe the service and the comments made on it. The Consumption Platform module retrieves this information by invoking the three different "get" methods of the Feedback Framework without specifying a particular user.

If the user is logged into the platform, he will be able to see the rating he has assigned to the service, the tags he has associated to it, and the comments he has made. The Consumption Platform retrieves this information by invoking the three different "get" methods of the Feedback Framework specifying the logged user.

Additionally, if the user is logged in, he/she can update his/her rating on the service, as well as providing a new tag or set of tags, or a new comment. The Consumption Platform is able to store new feedback information by calling the three different "write" methods of the Feedback Framework.

Finally, even though it is out of scope of the Feedback Framework itself, it is worth mentioning that the Consumption Platform displays some of the feedback information by making use of the widgets made available by the SOA4All Studio UI Components. (Additionally, any other module that integrates with the Feedback Framework will also be able to do the same.) We are particularly referring to the Rating Widget (which displays the 1-to-5 ratings with stars) and the Tag Cloud widget (which shows tags in different sizes and colours, depending on their "weight") described in more detail in D2.4.2.

2.3 Additional Documentation

In addition to this deliverable, which described the main components of the first Provisioning Platform Prototype, we created a website, which provides additional information on the MicroWSMO editors. 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.

The WSMO-Lite editor is delivered with a flash demo movie representing a complete end-to-end scenario, focusing on the different steps of the annotation process.

3. Conclusions

The deliverable describes the components and the functionalities of the first 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 the MicroWSMO and WSMO-Lite Editors, which support users in creating semantic descriptions of RESTful and WSDL-based services. In addition, this deliverable presents the first prototype of the Feedback Framework, which enables users to rate and recommend services based on their experience.

Annex A.

This annex contacting two publications describing research done within the scope of the SOA4All Provisioning platform:

1. Maria Maleshkova, Jacek Kopecky, Carlos Pedrinaci: Adapting SAWSDL for Semantic Annotations of RESTful Services. To appear in Proceedings of Beyond SAWSDL Workshop, OnTheMove Federated Conferences & Workshops 2009.
2. Maleshkova, M., Gridinoc, L., Pedrinaci, C., and Domingue, J. (2009) Supporting the Semi-Automatic Acquisition of Semantic RESTful Service Descriptions, Poster at ESWC 2009

Adapting SAWSDL for Semantic Annotations of RESTful Services

Maria Maleshkova[1], Jacek Kopeck y[2], Carlos Pedrinaci[1]
[1] Knowledge Media Institute (KMi)
The Open University, Milton Keynes, United Kingdom
m.maleshkova@open.ac.uk, c.pedrinaci@open.ac.uk
[2] STI Innsbruck, Innsbruck, Austria
jacek.kopecky@sti2.at

Abstract. RESTful services are increasingly been adopted as a suit-able lightweight solution for creating service-based applications on the Web. However, most often these services lack any machine-processable description and therefore a significant human labour has to be devoted to locating existing services, understanding their documentation, and implementing software that uses them. In order to increase the automation of these tasks, we present an integrated lightweight approach for the creation of semantic RESTful service descriptions. Our work is based on hRESTS, a microformat for including machine-readable descriptions of RESTful service within existing HTML service documentation. We complement hRESTS by the MicroWSMO microformat, which uses SAWSDL-like hooks to add semantic annotations. Finally, we present SWEET – Semantic Web sErvices Editing Tool - which effectively supports users in creating semantic descriptions of RESTful services based on the aforementioned technologies.

1 Introduction

Currently, there is an increased use and popularity of RESTful services [1], which offer a more lightweight alternative to the SOAP- and WSDL-based approach. As a result, more and more Web applications and APIs follow the Representational State Transfer [2] (REST) architecture principles and expose functionalities in the form of RESTful Web services. This trend is supported by the Web 2.0 wave, which drives the creation of user-centered Web applications for communication, information sharing and collaboration. Popular Web 2.0 applications by Yahoo, Google and Facebook offer easy-to-use, resource-oriented APIs, which not only provide simple access to diverse resources but also enable combining heterogeneous data coming from diverse services, in order to create data-oriented service compositions called mashups. Even though RESTful services are already widely accepted, their potential is restricted by the current low level of automation due to the lack of machine-readable descriptions and the limited applicability of the Semantic Web Services automation technologies.

Web 2.0 principles contributed significantly to the uptake of RESTful ser- vices. As a result, the value of Web 2.0 applications is not only for the direct user, who can receive customized information, but also in exposing functionality through public REST-based APIs. However, the fact that these APIs were inspired by user-centered applications has resulted in the creation of user-oriented descriptions. Even though, the APIs are meant for machine consumption, the descriptions themselves are plain unstructured HTML documents.

The lack of machine-readable descriptions is only one of the challenges, which have to be addressed in order to provide a certain level of automation for RESTful service. The fact that the majority of existing RESTful service descriptions have no semantic annotations, also has to be taken into consideration. Semantic Web Services (SWS) are proposed as means for automating many common tasks involved in using Web services. Discovery, negotiation, composition and invocation can have a higher level

of automation, when Web service are supplemented with semantic descriptions of their properties. Similarly to traditional SWS, which improve the automation of WSDL-based solutions, the adding of annotations to RESTful services can bring further automation to the process of creating mashups, in addition to the discovery and invocation tasks.

In this paper we present an integrated lightweight approach for the creation of semantic annotations of RESTful service by adapting SAWSDL [3]. For the creation of machine-readable RESTful service descriptions we use the hRESTS (HTML for RESTful Services) microformat [4]. Microformats [5] offer means for annotating human-oriented Web pages in order to make key information machine-readable, while hRESTS, in particular, enables the creation of machine- processable Web API descriptions based on available HTML documentation.

hRESTS is complemented by the MicroWSMO microformat [6], which enables using SAWSDL-like annotations to RESTful services. MicroWSMO introduces additional HTML classes, in order to enable the specification of a model reference, in addition to lifting and lowering relations for data grounding. Moreover, concrete semantics can be added by applying WSMO-Lite [7] service semantics, which enable the integration of RESTful services with WSDL-based ones. In this way, discovery and composition approaches no longer need to differentiate between WSDL and RESTful services, but rather simply be based on the integrated WSMO-Lite service semantics.

The creation of semantic RESTful services, including both hRESTS tagging and semantic annotation, is supported by SWEET: Semantic Web sErvices Editing Tool[10]3. SWEET assists users in injecting hRESTS and MicroWSMO within RESTful HTML service descriptions, thus enabling the effective creation of se- mantic descriptions. It hides formalism complexities from the user and assists him/her in adding service metadata.

The remainder of this paper is structured as follows: Section 2, provides an analysis of common ways of describing RESTful services, while Section 3 uses this analysis to deduce a lightweight RESTful service model. Our approach for creating machine-readable service descriptions, including the hRESTS microformat and the provided tool support, is described in Section 4. In Section 5, we present our adaptation of SWASDL for RESTful services, by describing the properties of MicroWSMO and the tool support for semantic annotations offered by SWEET. Section 6 presents an overview of related formalisms and approaches. Finally, 7 provides some detail on future work and a conclusion.

2 Common RESTful Service Descriptions

In order to be able to find services and interact with them, RESTful services, and services in general, need to be described in some way. While Web applications and Web APIs contain HTML documentation, which is understandable for human users, it needs to be extended in order to become machine-readable and -processable as well. WSDL [8] is an established standard for Web service descriptions, however, it has not found wide adoption for RESTful services and only a few such services have WSDL descriptions. Similarly, WADL [9] does not seem to be gaining acceptance among API provides and instead Web applications and APIs are usually described in textual documentation. However, in order to support the automation of RESTful services, certain key aspects of the descriptions have to be made machine-readable.

Listing 1. Example HTML Service Description

```
1 <h1>Send SMS Service</h1>
2 <p>This is a Short Message Service (SMS).<p>
3 <b>Example usage</b> http://my.test.serv.com/SendSMS.php?recipient=tel:
4 447712345678&message=messagetext&sender=User&title=TheMessage</br>
5 <h2>SendSMS Method</h2>
6 <b>recipient</b><p>List of recipent phone numbers in the format "tel:"
7 followed by an international phone number</p><br/>
8 <b>message</b><p>Content of the message.</p><br/>
9 <h2>The result is a sent SMS.</h2>
```

Our approach suggests the hRESTS microformat, which can be used to tag service properties and produce a machine-readable service description on top of existing HTML documentation. In order to identify which elements of the service description need to be marked with hRESTS tags, we analyze

[10] http://sweet.kmi.open.ac.uk/; http://sweetdemo.kmi.open.ac.uk/

existing RESTful service descriptions and derive a lightweight RESTful service model, which consists of service properties required for the completion of the discovery, composition and invocations tasks.

Common RESTful service descriptions are usually given in a Web page, which contains a list of the available operations, their URIs and parameters, expected output, error message and an example. The description includes all details necessary for a user to execute the service or use it in a mashup. Based on an existing repository for Web APIs[4], which contains more than 1380 APIs, manually collected over time, we have identified three basic types of descriptions.

Listing 1 shows an example of the first type of HTML descriptions of RESTful services. These descriptions contain only one or a number of operations, which are described with their corresponding parameters and URIs, within the same Web page. However, a lot of Web 2.0 applications contain a plenitude of operations. In this case, the second type of descriptions, includes one main page for [4] programmableweb.com the service and a number of linked pages, each of which describes one operation. This results in the requirement that the microformat used to annotate the service descriptions should include not only operations, URIs, HTTP methods, inputs and outputs but should also enable the linking of multiple operations from different Web sites to one \root" service description.

Listing 2. Example Resource-Oriented Description

activity blogs auth

- Flickr . activity .userComments - Flickr. blogs . getList - Flickr.auth.checkToken
- Flickr. activity .userPhotos - Flickr. blogs .postPhoto - Flickr.auth.getFrob

Finally, the last type of RESTful service descriptions are the resource-oriented ones. Listing 2 shows parts of the flickr[11] API documentation, where operations are not simply listed but they are rather grouped, based on the resources which they manipulate. In the example, there are three resources (activity, blogs and auth), each of which has a set of operations. Similarly, in this case the requirements for the microformat include that separate operation descriptions can be linked to a particular resource and one RESTful service. Based on these three types of RESTful service descriptions, we derive a lightweight RESTful service model that can effectively support the creation of machine-readable service descriptions.

3 Lightweight RESTful Service Model

The service examples given in Section 2 serve as the basis for identifying key service properties present in the textual documentation. These properties are formalized in a model, which specifies the service properties used for creating machine-readable RESTful service descriptions by marking HTML with hRESTS microformat tags.

Generally, a RESTful service description consist of several *operations*, each of which is performed over a *HTTP method* and has a particular *address* (URI or a parameterized URI). Operations have *inputs* and *outputs* with corresponding data formats. In addition, outputs of one operation may link to other operations, creating a resource based \choreography". Also, a number of operations can have distributed descriptions but belong to the same service, or can have different outputs but a common set of input parameters.

In summary, the elements, which have to be identified in an unstructured HTML service description are the service body, the operations, the input and output, the address and the HTTP method. As it can be seen, this list is very similar to the one of the WSDL structure and provides the basis for a machine readable description, which can be extended and annotated with additional in- formation such as semantic descriptions and nonfunctional properties.

4 Machine-readable Descriptions of RESTful Services

In order to support the creation of machine-readable descriptions, we use the hRESTS microformat. Microformats facilitate the translation of the HTML tag structure into objects and properties. As a result, the visualization of the HTML description remains unchanged, while the microformat uses *class* and *rel* XHTML attributes to mark key service properties. In this way, the hRESTS microformat enables the creation of machine-readable RESTful service descriptions, on top of existing HTML documentation.

[11] www.flickr.com/services/api/

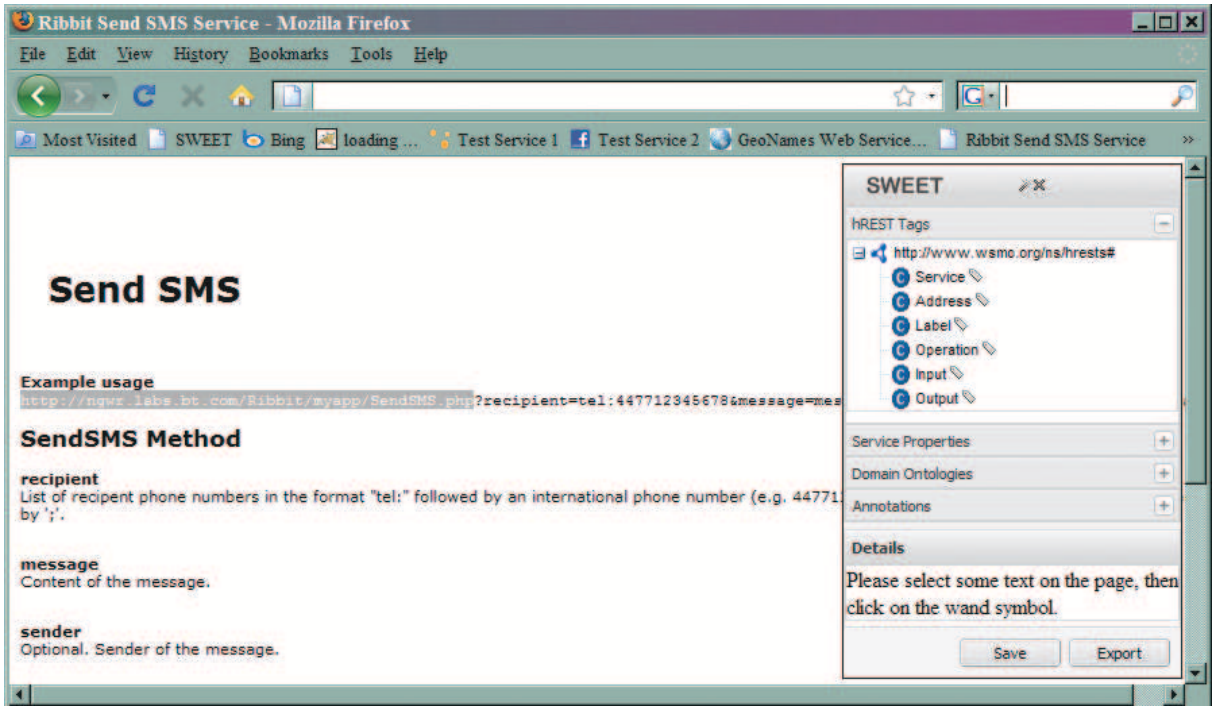

Figure 1:SWEET: hRESTS Annotation

hRESTS consists of a number of classes based directly on the properties identified in the previous section. However, even if the hRESTS microformat enables the creation of machine-readable RESTful service descriptions, the manual annotation of an HTML service description with hRESTS is a time-consuming and complex task. In order to support users in adopting hRESTS, we have developed SWEET. SWEET is a JavaScript Web application, which requires no installation and has the form of a vertical widget, which appears on top of the currently browsed Web page. This tool overcomes a number of difficulties associated with the annotation of Web applications and API descriptions, including the fact that the HTML documentation is viewed through a browser and that the user who wants to annotate the service descriptions, usually does not have access to manipulate or change the HTML.

Therefore, we provide a browser-based annotation solution, which can be started on top of the currently viewed RESTful service description. Figure 1 shows a screenshot of SWEET. hRESTS tags can simply be inserted by selecting the relevant part of the HTML and clicking on the corresponding class node in the hRESTS panel of SWEET. In this way, the hRESTS annotation process is less error-prone, less time-consuming and much simpler for the user. The result is hRESTS annotated HTML, which can easily be converted into RDF (\Export" button), by using an implemented XSLT stylesheet.

Listing 3 shows the previous service description example, annotated with hRESTS by using SWEET. It visualizes the usage of the microformat annotations, as well as the structure restrictions, which exist for the different classes. The complete Web service or API description is marked by the service class. The service may have a label, which can be used to mark the human-readable name of the service. A machine readable description can be created, even if there is no service class inserted. It is sufficient that the HTML description contains at least one operation, which is marked with the operation class. The operation description itself includes the address where it can be executed and the HTTP method. Input and output details as well as a label can also be part of the operation.

Listing 3. Example hRESTS Service Description

```
1 <div class="service" id="svc">
2 <h1>Send <span class="label">SMS Service</span></h1>
3 <p>This is a Short Message Service (SMS).<p>
4 <b>Example usage</b>
5 <span class="address">http://my.test.serv.com/SendSMS.php?recipient=
6 tel :447712345678&message=messagetext&sender=User&title=TheMessage</span>
7 <div class="operation" id="op1">
8 <h2><code class="label">SendSMS Method</code></h2>
```

```
9 <span class="input">
10 <b>recipient</b><p>List of recipent phone numbers in the format "tel:"
11 followed by an international phone number</p><br/>
12 <b>message</b><p>Content of the message.</p></span><br/>
13 <h2><span class="output">The result is a sent SMS.
14 </span></h2></div></div>
```

The hRESTS address class is used to specify the URI of the operation. Similarly, the method class specifies the HTTP method used for the operation. The final two elements used are input and output. They are used on block markup and indicate the operation's input and output. These two elements serve as the basis for extensions given by microformats, which provide additional semantic information or details about the particular data type and schema information. Finally, user-oriented labels can be attached to the service, operation, input or output classes.

The current version of SWEET directly supports the annotation only of the ¯rst, most common type of RESTful service descriptions, where all details about a service are provided within one webpage. However, the two other types of descriptions can easily by annotated as well, by making some minor manual modifications. First, one API description can have one main page and a number of separate pages, which describe each of the operations. In order not to lose the link between the separate parts of the description, the service page is modified to include rel="section" after each of the links pointing to the operation Web pages and each of the operations is modified to include rel="start", which points to the main service description. As a result the start and section relations link the pages together. The second requirement, that a number of operations can be grouped based on the resource, to which they apply, is implicitly solved. All of the grouped operations have the same subset of input parameters, which can additionally be emphasized by adding semantic annotations as described in Section 5.

5 Semantic Descriptions of RESTful Services

hRESTS marks the key properties of the RESTful service and provides a machine- readable description based on the available HTML documentation. The result can effectively be used as the basis for adding extensions for supplementary in- formation and annotations. In addition, it enables the adapting of SAWSDL [3] properties for adding semantic annotations to RESTful services because the hRESTS view of services is analogous to the WSDL one. Even though, hRESTS already provides a certain level of automation by enabling the creation of machine-readable descriptions, a higher level of automation of the discovery, composition, ranking, invocation and mediation service tasks can be archived by extending service descriptions with semantic annotations of their properties. As a result, Semantic RESTful Services (SRS) can be developed following and adapting approaches from the Semantic Web Services (SWS).

SAWSDL specifies how to annotate service descriptions, provided in WSDL, with semantic information by defining three XML attributes with equivalent RDF properties. The modelReference points to URIs identifying appropriate semantic concepts, while liftingSchemaMapping and loweringSchemaMapping associate messages with appropriate transformations between the level of technical descriptions (XML) and the level of semantic knowledge (RDF).We adopt these SAWSDL properties as extensions to hRESTS as part of the here described MicroWSMO microformat. Therefore, MicroWSMO represents the SAWSDL layer for RESTful services, based on top of hRESTS, instead of WSDL. Consequently, MicroWSMO has three main elements, which represent links to URIs of semantic concepts and data transformations. model indicates that the URI is a link to a model reference, while lifting and lowering point to links for lifting and lowering transformations.

Since the WSMO-Lite [7] ontology is used for describing the content of SAWSDL annotations in WSDL, we also adapt it for MicroWSMO. WSMO- Lite specifies four aspects of service semantics including *information model*, *functional semantics*, *behavioral semantics* and *nonfunctional descriptions*, instances of which are linked to the MicroWSMO annotations. It is important to point out that since both MicroWSMO and SAWSDL can apply WSMO-Lite service semantics, RESTful services can be integrated with WSDL-based ones. Tasks such as discovery, composition and mediation can be performed based on WSMO-Lite, completely independently from the underlying Web service technology (WSDL/ SOAP or REST/HTTP).

The task of associating semantic content with RESTful service properties is even more time- and effort-demanding than the insertion of hRESTS tags. Therefore, SWEET supports users in searching for suitable domain ontologies and in making semantic annotations in MicroWSMO. Whenever, a user wants to add semantics to a particular service property, for example, an input parameter, he/she has to select it

and click on the \magic wand" symbol, which send a request to Watson [10]. Waston is a search engine, which retrieves relevant ontologies based on keyword search.

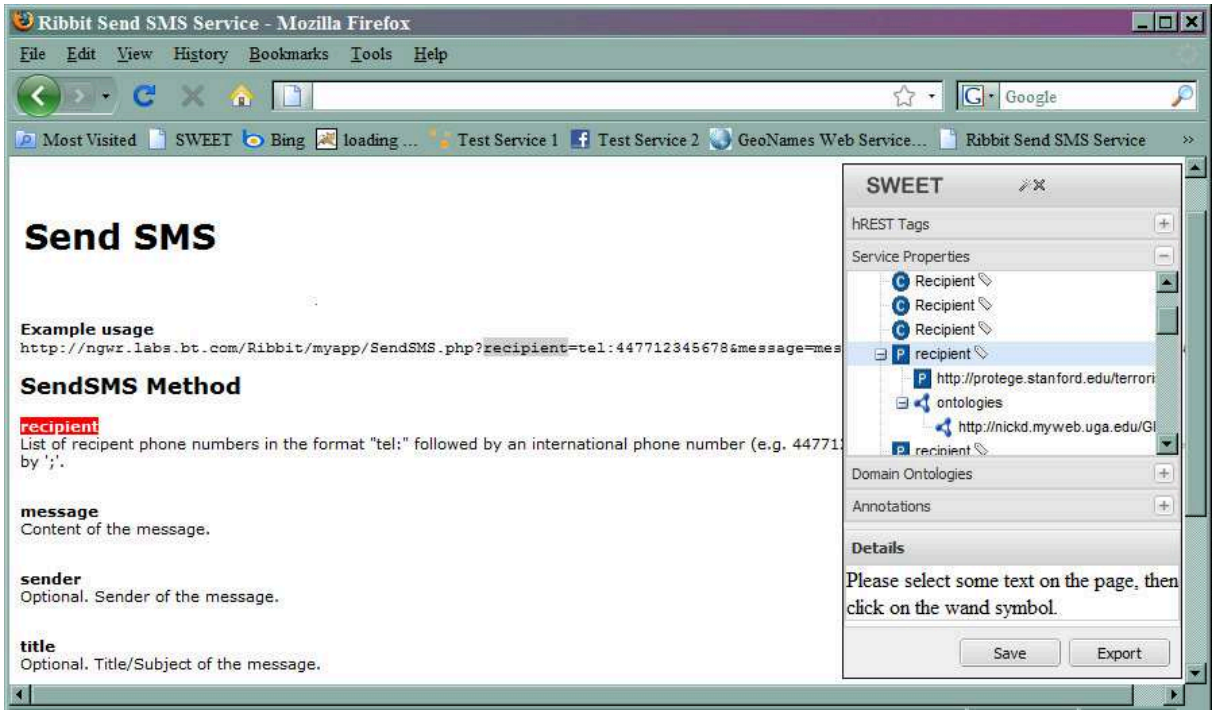

Figure 2. SWEET: Semantic Annotation

The results are presented in the service properties panel visualized in Figure 2. The searched for property is the root of the tree, populated with nodes that represent the found matches. In the example, *recipient* was found to be a property (\P") in an ontology located at http://protege.stanford.edu. If the user needs additional information in order to decide whether the particular semantic annotation is suitable or not, he/she can switch to the domain ontologies panel, which provides a list of all concepts and the corresponding property matches. The user can make a semantic annotation by simply selecting the property instance and clicking on one of the semantic matches in the service properties panel. The result is a MicroWSMO annotated HTML description, which can be saved in a repository (button \Save") or be converted into RDF (button \Export").

Listing 4 shows our example service description annotated with MicroWSMO using SWEET. Line 4 uses the model relation to indicate that the service sends SMS, while line 12 associates the input parameter *recipient* with the class *Recipient*. The lowering schema for the recipient is also provided in line 13.

Listing 4. Example MicroWSMO Service Description

```
1 <div class="service" id="svc">
2 <h1>Send <span class="label">SMS Service</span></h1>
3 <p>This is a
4 <a rel="model" href="http://example.com/telecommunications/sendSMS">
5 Short Message Service (SMS).</a><p>
6 <b>Example usage</b>
7 <span class="address">http://my.test.serv.com/SendSMS.php?recipient=
8 tel :447712345678&message=messagetext&sender=User&title=TheMessage</span>
9 <div class="operation" id="op1">
10 <h2><code class="label">SendSMS Method</code></h2>
11 <span class="input">
12 <b><a rel="model" href="http://example.com/data/onto.owl#Recipient">
13 recipient </a>(<a rel="lowering" href="http://example.com/data/sms.xsparql">
14 lowering</a>)</b><p>List of recipient phone numbers in the format "tel:"
15 followed by an international phone number</p><br/></div></div>
```

6 Related Work

hRESTS is not the only alternative that can be used for the creation of machine- readable descriptions of RESTful services. WADL (Web Application Description Language) [9] and even WSDL 2.0 [8] can be used as description formats. They provide well-structured and detailed forms of descriptions. However, probably due to the user-centered context of Web 2.0 and of the resulting API descriptions, WADL and WSDL seem to add complexity and still the majority of the API descriptions are provided in unstructured text. We use hRESTS, which is relatively simple, easy to use, can be applied directly on the existing HTML descriptions, supports the extraction of RDF and can provide a basis, for the future adoption of dedicated formats such as WADL.

Another description approach if offered by RDFa [11]. RDFa can be effectively used for embedding RDF data in HTML. However, following the simplicity and lightweight principles perused with hRESTS, it needs to be investigated to what extent and in which use cases RDFa can be used for hRESTS. A parallel approach to RDFa would be the use of GRDDL [12] on top of hRESTS. GRDDL is a mechanism for extracting RDF information from Web pages and is particularly suitable for processing microformats.

In the area of tools supporting the semantic annotation of services, ASSAM [13] enables the annotations of services with WSDL-based descriptions. It provides user interface tools as well as some automatic recommendation components, however, it can only be used on WSDL-based descriptions and does not support RESTful services.

7 Conclusion and Future Work

Current RESTful services can be found, interpreted and invoked not without the extensive user involvement and a multitude of manual tasks. This situation can be alleviated through the creation of machine-readable descriptions. Based on such descriptions, crawlers and search engines can better find services, and developers can better use them. Moreover, extended with semantic annotations, RESTful services can even be discovered, composed and invoked automatically, following the principles of the SWS.

In this paper, we have built on a lightweight RESTful service model, based on the hRESTS microformat that enables the tagging of key service properties and therefore supports the creation of machine-readable service descriptions. We complemented hRESTS by the MicroWSMO microformat, which adapts SAWSDL for the semantic annotation of RESTful services. Finally, we have shown the tool SWEET, which effectively supports users in creating semantic descriptions of RESTful services based on hRESTS and MicroWSMO.

Future work will include the development of additional functionalities of SWEET, which will better support users in the creation of semantic RESTful annotations. Better visualization components, such as structure and properties highlighting are planned. In addition, some work will be devoted to the automatic recognition of service properties such as operations and input parameters, so that the amount of manual work required by the user can be reduced.

The work presented here is partially supported by EU FP7 project SOA4All.

References

1. L. Richardson, S. Ruby: RESTful Web Services. O'Reilly Media, May 2007.
2. R. T. Fielding: Architectural styles and the design of network-based software architectures. PhD thesis, University of California, 2000.
3. J. Kopeck y, T. Vitvar, C. Bournez, J. Farrel. SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE Internet Computing, 11(6):60-67, 2007.
4. J. Kopeck y , K. Gomadam, T.Vitvar: hRESTS: an HTML Microformat for Describing RESTful Web Services. Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence (WI-08), 2008.
5. R. Khare, T. Celik: Microformats: a pragmatic path to the semantic web (Poster). Proceedings of the 15th international conference on World Wide Web, 2006.
6. J. Kopeck y, T. Vitvar, D. Fensel, K. Gomadam: hRESTS & MicroWSMO. Technical report, available at http://cms-wg.sti2.org/TR/d12/, 2009.
7. T. Vitvar, J. Kopeck y, J. Viskova, D. Fensel. WSMO-Lite Annotations for Web Services. In the Semantic Web: Research and Applications, ESWC 2008.
8. Web Services Description Language (WSDL) Version 2.0. Recommendation, W3C, June 2007. Available at http://www.w3.org/TR/wsdl20/.
9. M. J. Hadley: Web Application Description Language (WADL). Technical report, Sun Microsystems, November 2006. Available at https://wadl.dev.java.net.

10. Watson - The Semantic Web Gateway: Ontology Editor Plugins. http://watson.kmi.open.ac.uk. Online November 2008.
11. RDFa in XHTML: Syntax and Processing. Proposed Recommendation, W3C, September 2008. Available at http://www.w3.org/TR/rdfa-syntax/.
12. Clarke, F., Ekeland, I.: Gleaning Resource Descriptions from Dialects of Languages.Recommendation, W3C, September 2007. http://www.w3.org/TR/grddl/.
13. A. Hess, E., Johnston, N., Kushmerick: ASSAM: A tool for semi-automatically annotating semantic web services. In Proceedings of International Semantic Web Conference, 2004.

Supporting the Semi-Automatic Acquisition of Semantic RESTful Service Descriptions

Maria Maleshkova, Laurian Gridinoc, Carlos Pedrinaci, John Domingue
Knowledge Media Institute (KMi)
The Open University, Milton Keynes, United Kingdom
fm.maleshkova, l.gridinoc, c.pedrinaci, j.b.domingueg@open.ac.uk

Abstract. This paper presents SWEET: Semantic Web sErvices Editing Tool, the first tool developed for the semi-automatic acquisition of semantic RESTful service descriptions, aiming to support a higher level of automation of common RESTful service tasks, such as discovery and composition.

1 Introduction

The increasing importance and use of Web services have resulted in researchers proposing Semantic Web Services (SWS) as means for automating many common tasks involved in using Web services. Discovery, negotiation, composition and invocation can have a higher level of automation, when Web service are supplemented with semantic descriptions of their properties. On the other hand, the growing popularity and use of Web 2.0 technologies has led to the increased adoption of the RESTful paradigm. Not only are RESTful services more lightweight than WSDL-based services because they follow no strict description format. They also enable combining heterogeneous data coming from diverse services, in order to create data-oriented service compositions called mashups. Similarly to traditional SWS, which improve the automation of WSDL-based solutions, the adding of annotations to RESTful services can bring further automation to the process of creating mashups, in addition to the discovery and invocation tasks.

Current approaches in the area of semantic RESTful services (SRS) include the development of formalisms for the semantic annotation. MicroWSMO [3] is one such formalism, which relies on the hRESTS [1] microformat for describing the main aspects of a service such as operations, inputs and outputs, and uses hooks for relating semantic information. SA-REST [2], on the other hand, uses the grounding principles of SAWSDL and RDFa for marking service properties. Even though, there is some research done targeted at supporting the use of SRS, for example in the form of mashups [2], there are no contributions aiming to support the acquisition of semantic descriptions of RESTful services.

SWEET: Semantic Web sErvices Editing Tool is the first tool developed to assists the semi-automatic acquisition of SRS. Its goal is to hide formalism complexities from the user and to assist him/her in adding metadata by recommending suitable annotations. As a result, SWEET contributes directly to improving the automation of the discovery and composition of RESTful services.

2 Supporting the Annotation of RESTful Services

SWEET consists of three main components, including the visualization component, the data preprocessing component and the annotations recommender. The annotations recommender assists the user in annotating a service by suggesting suitable annotations for the service as a whole (domain ontology recommendation) and for its individual properties. This component is based on a hybrid recommendation approach combining content based recommendation, implemented by computing similarity measures, between the description of the new service to be annotated and previously annotated services, and

ontology-based recommendation. The data preprocessing component implements functionalities for data preparation for the visualization component, cashing mechanisms and simple rule-based analysis. Finally, the main task of the visualization component is to implement the user interface, which enables users to identify service properties based on the MicroWSMO ontology, to choose an appropriate domain ontology based on the annotations recommender suggestions, and to annotate service properties by making model reference to the domain ontology. In addition, the visualization component assist the user in recognizing the service structure and properties by graphically highlighting the service address, operations, inputs and outputs. This component is inspired by PowerMagpie [4], taking the form of a vertical widget as an extension of a classical web browser. In this way, the user can view the RESTful service description in a browser, start SWEET and make annotations. The result is a MicroWSMO service description, which can be shared and reused by other users. While all three components have complete specifications, including design models and computational methodologies, only the visualization component is fully implemented and the other two are still under development.

SWEET is a major contribution towards providing semantics for RESTful services, since it reduces the amount of necessary manual work by making an- notation suggestions and hides formalism complexity behind an effective and easy-to-use user interface. The development of SWEET is based upon work partially supported by the EU funding under the project SOA4All (FP7 - 215219).

References

1. Kopecky, J., Gomadam, K., Vitvar, T.: hRESTS: an HTML Microformat for Describing RESTful Web Services. In Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence (WI-08), 2008.
2. Sheth, A. P., Gomadam, K., Lathem, J.: SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. In IEEE Internet Computing, 11(6):9194, 2007.
3. Kopecky, J., Vitvar, T.: MicroWSMO. WSMO Working Draft. http://www.wsmo.org/TR/d38/v0.1/20080219/, February 2008.
4. Dzbor, M., Motta, E., Domingue, J.: Magpie: Experiences in supporting Semantic Web browsing. Web Semantics: Science, Services and Agents on the WWW, 2007