SEMIAUTOMATIC WEB SERVICE GENERATION

J. M. Fuentes, M. A. Corella, P. Castells, and M. Rico
Universidad Autónoma de Madrid
Escuela Politécnica Superior, Campus de Cantoblanco, 28049 Madrid

ABSTRACT

The lack of a critical mass of actually deployed web services, semantic or not, is an important hurdle for the advancement and further innovation in web service technologies. In this paper we introduce Federica, a platform for semi-automatic generation and implementation of semantic web services by exploiting existing web applications published in internet. Federica generates semantically enriched WSDL descriptions. Furthermore, the system generates an implementation of the service binding service methods with invocations to the original web application, in such a way methods could be executed immediately.

KEYWORDS

Semantic Web Services, Ontology, Web Application, Automatic Classification.

1. INTRODUCTION

The convergence of semantic web technologies [Berners-Lee 2001] and web services has been identified as a great opportunity for improvement in both fields [Terziyan 2003]. With web services, the semantic web becomes not just a web of information, but a web of functionality. Semantic web technologies, in turn, provide conceptual descriptions of web services, enabling the development of software agents capable of analyzing, manipulating, and reasoning about web services. The SOAP [Gudgin 2003], WSDL [Chirstensen 2001] and UDDI [OASIS 2004] standards have raised high expectations, but their adoption by the software industry is far from being consolidated. More recently, new languages and platforms have emerged for the generation and processing of web services with a more explicit, ontology-based semantics, such as OWL-S [Martin 2004], WSMO [Roman 2005] and SWSL (see http://www.daml.org/services/swsl). In our perception, one important hurdle for the advancement and further innovation in web service technologies is the lack of a critical mass of actually deployed web services, be they semantic or not, available as examples for problem analysis, use case study, and testing purposes in the development of new proposals.

Our research aims at the semi-automatic generation of semantic web services from web applications. This means the generation of service descriptions through the automatic analysis of web application interfaces (forms, input controls, output information), in such a way that the invocation of the generated service results in the execution of the original web application. The motivation of this objective is twofold. On the one hand, the provision of a collection of executable web services based on real examples, at almost no cost, which can serve as a basis for analysis and testing, with a view to address currently open research problems in this field (such as the automatic discovery, composition and invocation of web services). On the other hand, the proposed achievement can help in the transition from the present web, where a large number of web applications are running already, to a web of (semantic) web services.

Generating a functional description from the information implicit in the interface which gives access to an application is an inherently difficult problem which we address on two levels, syntactic and semantic. The syntactic level consists of generating WSDL descriptions. The semantic level, more ambitious, consists of finding and extracting semantic properties from the services.

This paper is structured as follows. Section 2 gives an overview of previous work related to the proposed research. Next we provide a detailed description of our approach for generating WSDL descriptions from web applications, the automatic implementation of the generated descriptions, linking to the original application functionality and the support for the execution and testing of the generated services. Section 4 explains how WSDL descriptions are enriched by adding the description of certain semantic aspects. Section 5 describes the status of the development of our techniques. The paper closes with some final conclusions and an outline of future research possibilities.

2. RELATED WORK

The idea of automatically generating web services from web applications has already been investigated before. For its relevance and its relationship with our proposal, it is worth mentioning the work reported in [Pham 2004] and [Sahuguet 1999].

The main difference between the proposal in [Pham 2004] and ours lies on their ultimate purpose. In our research, we intend not only to create web services which make it possible to access and use web applications, but also to generate semantic information which enables a more powerful description of the generated services and, indirectly, of the web applications they are generated from. The work by Pham, on the other hand, relies on basic state-of-the-art technologies, and does not attempt to improve the expressive power of web service descriptions. The benefit of their approach allows their work to include the use of more aspects related to web services (at present), such as, for example, publication and later search of services within UDDI registers. In our case, use of new semantic descriptions will enable the development of new, more powerful and richer procedures for searching and discovering web services.

Even though web services technologies had not yet seen the light at the time Sahuguet's work was published [Sahuguet 1999], it is interesting to find already some of our concerns in this work. In this work, the description of web application functionalities is manually written. Instead of web services, their system generates wrappers consisting of Java applications, thereby generating a Java API for the invocation of web application functionalities. In our proposal, the descriptions themselves are also automatically generated, and so are the wrappers, which in our case consist of standard web services..

3. GENERATION OF BASIC DESCRIPTIONS

The next subsections explain our proposed web service generation procedure. Section 3.1 deals with the construction of WSDL service descriptions by inspecting the HTML front-end of web applications. Section 3.2 describes the automatic implementation of functionality for such descriptions. Finally, Section 3.3 shows how means are provided for executing the web services thus created.

3.1 Automatic Form Analysis and Extraction of Descriptions

For the automatic analysis and description of the functionality provided by a web application, the system we propose takes as input the URL of the web page giving access to the application. In addition, a service name is required as input, which will be used to identify the service to be created. The HTML source that can be retrieved from the URL of a web application usually includes:

- Purely informative elements (titles, instructions, logos, advertising, etc), navigation elements (e.g. HTML links), plus presentation settings, such as page design, layout, and style properties.
- UI controls to interact with (enter input and invoke) the application functionality.

In our current proposal, purely informative content in the web page is disregarded, and only the interface elements of the web application are analyzed. On the WWW, the most widespread construct for interacting with the user are HTML forms. Therefore, as a first step, we have restricted our techniques to the analysis of such forms, although we foresee the extension of our techniques in the future to other client-side interaction support technologies, such as JavaScript, by adding the suitable modules to allow the analysis of the syntax extensions (e.g. using JavaScript parsers).

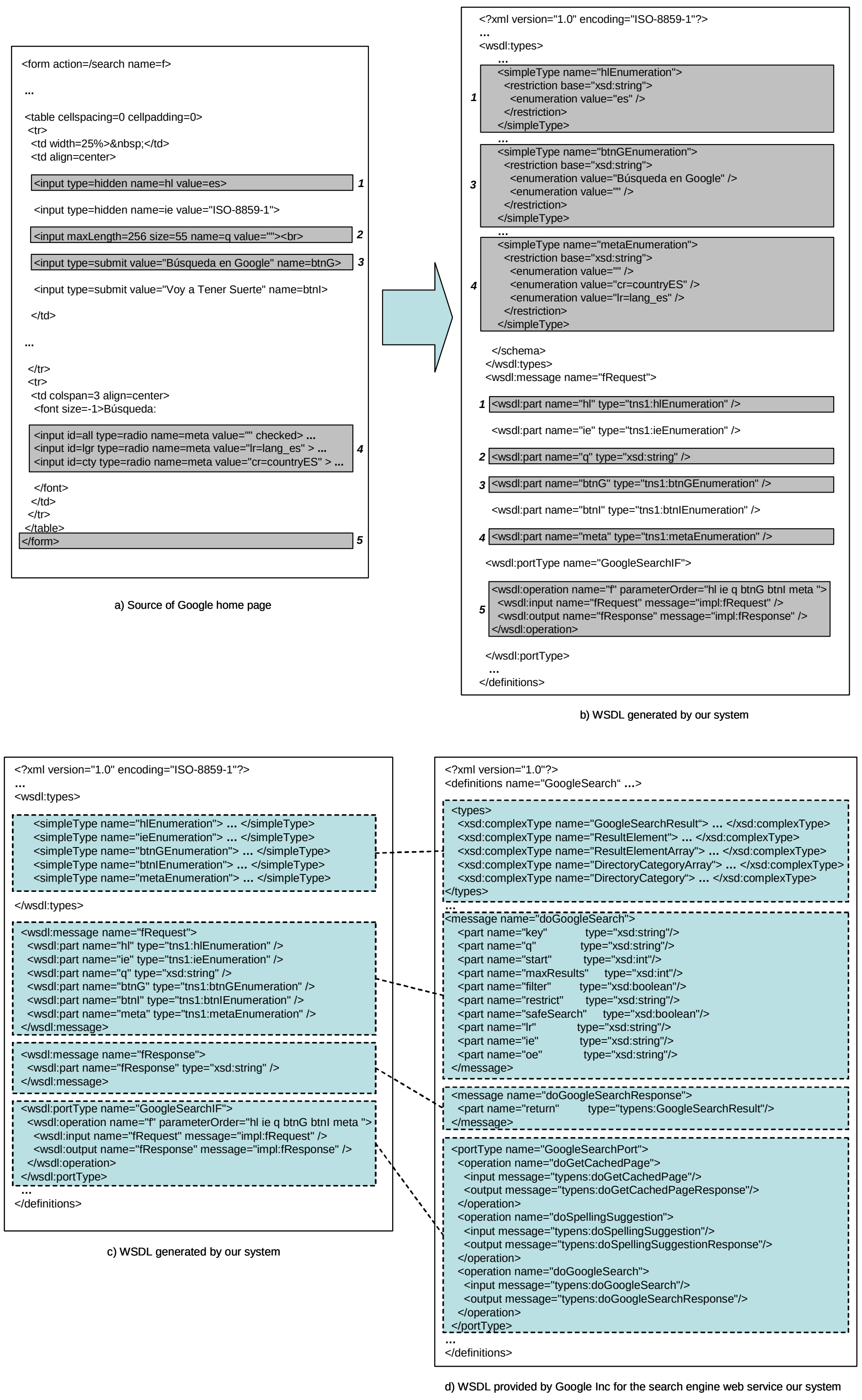
The process by which the automatic generation of WSDL descriptions is achieved consists of the following steps:

1. A service with a unique `wsdl:portType` is defined for the whole application.
2. The source code of the HTML forms retrieved from the application URL are extracted by means of an HTML parser.
3. A `wsdl:operation` is created for each of the forms found in the web page of the application.
4. An out-message and an in-message are defined for each `wsdl:operation`.
5. The input, select and textarea HTLM controls are identified as inputs for the service (that is to say, as `wsdl:part` elements in the in `wsdl:message`) for each of the forms.
6. Depending on the HTML control type and features, a different data type is assigned to each service input, a new XML schema type being defined where needed Table 1 shows the type correspondence for each type of HTML form input control.
7. The `wsdl:output` message of each operation will contain only one `wsdl:part` of an `xsd:string` type which, once the service is invoked, will include the source code for the web page returned from the invocations to the application through each of its forms.

Table1. Mapping between HTML components and XML Schema data types

HTML control	XML Schema data type
text	xsd:string
password	
textarea	
submit	xsd:string, enumeration
radio	
checkbox	
hidden	
select (simple selection)	
inputs (same name)	
inputs (with value and readOnly attributes)	
select (multiple selection)	xsd:string array, enumeration

Figure 1 shows an example of the generation of a WSDL description for Google's entry web page . The example allows the comparison of the description generated by our platform and that supplied by a provider, Google Inc., of both the web application and the search engine web service, It can be seen that our techniques achieve an acceptable approximation. The main differences, in this particular example, are mainly two: the names for the service method parameters, and the inclusion of an additional functionality in Google's web service that is not present in their web application forms (such as the "did you mean…?" functionality).



```
<form action=/search name=f>

 ...

 <table cellspacing=0 cellpadding=0>
  <tr>
   <td width=25%>&nbsp;</td>
   <td align=center>

    <input type=hidden name=hl value=es>                                      1

    <input type=hidden name=ie value="ISO-8859-1">

    <input maxLength=256 size=55 name=q value=""><br>                         2

    <input type=submit value="Búsqueda en Google" name=btnG>                  3

    <input type=submit value="Voy a Tener Suerte" name=btnI>

   </td>

 ...

  </tr>
  <tr>
   <td colspan=3 align=center>
    <font size=-1>Búsqueda:

    <input id=all type=radio name=meta value="" checked> ...
    <input id=lgr type=radio name=meta value="lr=lang_es" > ...              4
    <input id=cty type=radio name=meta value="cr=countryES" > ...

    </font>
   </td>
  </tr>
 </table>
</form>                                                                       5
```

a) Source of Google home page

```
<?xml version="1.0" encoding="ISO-8859-1"?>
...
<wsdl:types>
   ...
   <simpleType name="hlEnumeration">
     <restriction base="xsd:string">
1      <enumeration value="es" />
     </restriction>
   </simpleType>
   ...
   <simpleType name="btnGEnumeration">
     <restriction base="xsd:string">
3      <enumeration value="Búsqueda en Google" />
       <enumeration value="" />
     </restriction>
   </simpleType>
   ...
   <simpleType name="metaEnumeration">
     <restriction base="xsd:string">
       <enumeration value="" />
4      <enumeration value="cr=countryES" />
       <enumeration value="lr=lang_es" />
     </restriction>
   </simpleType>

  </schema>
 </wsdl:types>
 <wsdl:message name="fRequest">

1  <wsdl:part name="hl" type="tns1:hlEnumeration" />

   <wsdl:part name="ie" type="tns1:ieEnumeration" />

2  <wsdl:part name="q" type="xsd:string" />

3  <wsdl:part name="btnG" type="tns1:btnGEnumeration" />

   <wsdl:part name="btnI" type="tns1:btnIEnumeration" />

4  <wsdl:part name="meta" type="tns1:metaEnumeration" />

 <wsdl:portType name="GoogleSearchIF">

   <wsdl:operation name="f" parameterOrder="hl ie q btnG btnI meta ">
5    <wsdl:input name="fRequest" message="impl:fRequest" />
     <wsdl:output name="fResponse" message="impl:fResponse" />
   </wsdl:operation>

 </wsdl:portType>
  ...
</definitions>
```

b) WSDL generated by our system

```
<?xml version="1.0" encoding="ISO-8859-1"?>
...
<wsdl:types>

    <simpleType name="hlEnumeration"> ... </simpleType>
    <simpleType name="ieEnumeration"> ... </simpleType>
    <simpleType name="btnGEnumeration"> ... </simpleType>
    <simpleType name="btnIEnumeration"> ... </simpleType>
    <simpleType name="metaEnumeration"> ... </simpleType>

</wsdl:types>

 <wsdl:message name="fRequest">
   <wsdl:part name="hl" type="tns1:hlEnumeration" />
   <wsdl:part name="ie" type="tns1:ieEnumeration" />
   <wsdl:part name="q" type="xsd:string" />
   <wsdl:part name="btnG" type="tns1:btnGEnumeration" />
   <wsdl:part name="btnI" type="tns1:btnIEnumeration" />
   <wsdl:part name="meta" type="tns1:metaEnumeration" />
 </wsdl:message>

 <wsdl:message name="fResponse">
   <wsdl:part name="fResponse" type="xsd:string" />
 </wsdl:message>

 <wsdl:portType name="GoogleSearchIF">
   <wsdl:operation name="f" parameterOrder="hl ie q btnG btnI meta ">
     <wsdl:input name="fRequest" message="impl:fRequest" />
     <wsdl:output name="fResponse" message="impl:fResponse" />
   </wsdl:operation>
 </wsdl:portType>
  ...
</definitions>
```

c) WSDL generated by our system

```
<?xml version="1.0"?>
<definitions name="GoogleSearch“ ...>

 <types>
   <xsd:complexType name="GoogleSearchResult“> ... </xsd:complexType>
   <xsd:complexType name="ResultElement"> ... </xsd:complexType>
   <xsd:complexType name="ResultElementArray"> ... </xsd:complexType>
   <xsd:complexType name="DirectoryCategoryArray"> ... </xsd:complexType>
   <xsd:complexType name="DirectoryCategory“> ... </xsd:complexType>
</types>
...
<message name="doGoogleSearch">
   <part name="key"          type="xsd:string"/>
   <part name="q"            type="xsd:string"/>
   <part name="start"        type="xsd:int"/>
   <part name="maxResults"   type="xsd:int"/>
   <part name="filter"       type="xsd:boolean"/>
   <part name="restrict"     type="xsd:string"/>
   <part name="safeSearch"   type="xsd:boolean"/>
   <part name="lr"           type="xsd:string"/>
   <part name="ie"           type="xsd:string"/>
   <part name="oe"           type="xsd:string"/>
 </message>

 <message name="doGoogleSearchResponse">
   <part name="return"       type="typens:GoogleSearchResult"/>
 </message>

 <portType name="GoogleSearchPort">
   <operation name="doGetCachedPage">
     <input message="typens:doGetCachedPage"/>
     <output message="typens:doGetCachedPageResponse"/>
   </operation>
   <operation name="doSpellingSuggestion">
     <input message="typens:doSpellingSuggestion"/>
     <output message="typens:doSpellingSuggestionResponse"/>
   </operation>
   <operation name="doGoogleSearch">
     <input message="typens:doGoogleSearch"/>
     <output message="typens:doGoogleSearchResponse"/>
   </operation>
 </portType>
...
</definitions>
```

d) WSDL provided by Google Inc for the search engine web service our system

Figure 1. An example of the generation of a WSDL description from Google's home page, and comparison of the automatically generated WSDL with the manually defined WSDL provided by Google Inc. The code excerpts b) and c) correspond to slightly different views of the WSDL description output by our system, where different fragments are highlighted

3.2 Linking the Generated Services to the Original Applications

Once a WSDL description has been created, our system generates an implementation of the web service. The implementation consists of an application wrapper that defers service requests to application calls. In order to achieve this, a link engine has been implemented using Axis for sending and receiving SOAP messages to communicate with the service. Using the Axis WSDL2Java tool, from the WSDL descriptions, a set of empty Java classes is generated which, once implemented, will provide the required functionality. Finally, the service deployment is done in the Axis platform, the engine of which will process the SOAP messages from the service clients at runtime. The result of this process, illustrated in Figure 2, is a totally functional web service that provides the same functionalities as the original web application, but in service form.

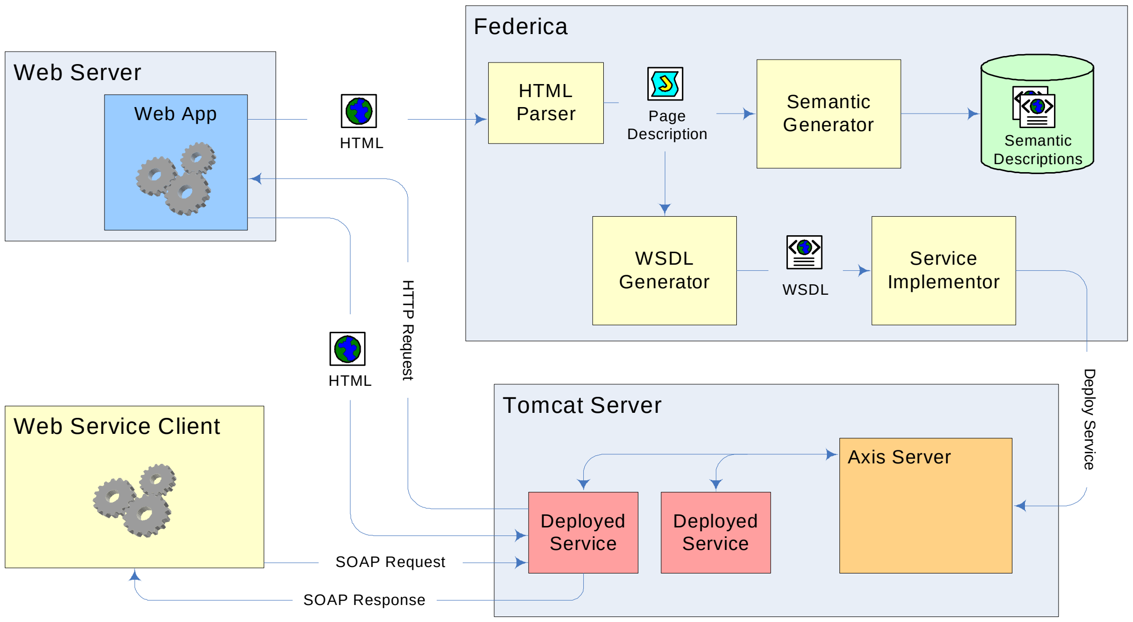


Figure 2. Overview of the service generation process and components

3.3 Execution of the Generated Services

With the mechanism described above, a large number of web applications can be made readily available as standards-compliant web services, gaining all the benefits and the potential for interoperability of web service technologies.

In order to validate the generated services, and as a tool to test and improve our system, our environment supports the automatic creation of a client to invoke the generated service. For this purpose, the Jacinta system [Rico 2004] has been integrated. Jacinta is a software agent specialized in interacting with human users on the basis of traditional web services (based solely on WSDL), enhanced with custom semantic extensions created by a service administrator. Using Jacinta, service administrators can define and associate an interaction ontology to WSDL elements, so that the web service gets semantically marked up with a description of the way its functionality should be presented to the end user. Using this information, plus the WSDL description, Jacinta automatically generates a web interface for the service at runtime. In our system, we do not exploit the full capabilities of Jacinta to specify the service UI to be generated, which requires the definition of a UI model. Instead, we rely on a default UI generated by Jacinta in the absence of a user-defined UI model. A designer, however, can define the semantics of presentation if she wishes, to improve the generated interface.

Jacinta provides further facilities that are not exploited either in our system, such as interactive support for searching web services. Jacinta's modular design allows the activation or deactivation of its different modules, in such a way that it can, for instance, be used only for treating the results returned by the services, as in our case, or the case when Jacinta acts as an interface for WSMX [Oren 2004], GODO [Gómez 2004] or others [Gómez 2005]. When Jacinta interacts with our platform, it takes advantage of semantic information added by our system to WSDL descriptions (see Section 4), or by service administrators.

Another envisioned goal of our invocation environment is a better characterization of the response of the generated services. At present, as described in 3.1, the service output is characterized as a string containing the source of the web page returned by the web application in response to a request. Our system would gain a considerable value if it were able to extract and characterize the response semantics, and the data within the returned web page, in a more precise way. To achieve this in a fully automated way is, in general, a very difficult problem. It would be more feasible, however, with the help of a user or service administrator. The idea would be to provide an interactive environment where the user can point at relevant fragments within the page, to help the system identify and describe the essential response data. This is a future research direction of our work.

4. SEMANTIC SERVICE DESCRIPTIONS

As stated in the introduction, our goals include the enrichment of service descriptions with higher-level information beyond the expressiveness of a language like WSDL, which is essentially limited to the description of syntactic aspects.

The range of semantic properties that one could consider in the description of a web service is virtually infinite. The definition of a general semantic model for this purpose is still, in our view, an open problem. Consequently we have restricted the semantics to be generated to a limited set of meaningful features. The set considered so far includes parameter descriptions and service classification. Regarding the former, our semantic model comprises:

- Mapping parameter types to ontology classes. For the time being this feature is applied to enumerated types such as countries, airports or currency types.
- Conditions that input service parameter values must fulfil, such as range restrictions, or numerical precision, obtained by monitoring user-service interaction.
- Assignment of labels to service methods and parameters, to enhance human readability, and enable the future extension of our framework with intelligent service analysis capabilities based on automatic text-based processing.
- HTLM input type descriptions for the generation of web UIs to web services, for user-service interaction at runtime.

Regarding the automatic classification of web the generated services, we have developed an algorithm that is able to propose a candidate classification (actually, a ranked list of most likely candidates) for a new service, based on a collection of classified WSDL descriptions. Our notion of service classification is the same as that of UDDI registries, where services are assigned a category from a standard taxonomy, such as UNSPSC or NAICS. Our system deduces the classification of a new service by estimating a measure of the similarity of its WSDL description with that of the classified services.

More specifically, the similarity between a WSDL description s and a category C is computed as:

$$sim(s,C) = \max_{t \in C} sim(s,t)$$

in other words, as the highest similarity between s and the services classified under C. As future work, we shall experiment with other functions, such as the mean or more sophisticated ones, instead of the maximum. The similarity between two service descriptions s and t are estimated by heuristics methods, which analyze the coincidence between parameter types, giving higher relevance to the more specific or rare types (for instance, a coincidence in the type "currency" is considered more significant than a coincidence in a string type). Even though such similarity measurements are highly approximate, in practice they achieve a reasonable success rate and in the long run lead to a potential classification ranking the user can pick from so as to choose the right one.

In order to represent semantic information as that described above, extensions to the expressiveness of WSDL are needed. OWL-S and WSMO provide such extensions, but the generality of these languages tends to make the representation of relatively simple information an extremely complex task. Because of this, we have decided to define our own ontology, one which extends and complements the expressivity of WSDL, but keeps the representation of semantic information simpler than OWL-S and WSMO, and avoids the computational problems of these languages. Nevertheless, we do not discard the future (perhaps partial) adoption of such powerful languages. At the same time as these languages gain maturity, we foresee a growth and increase of generality of our own ontology, so that at some point we might adapt our representation to converge with the OWL-S or WSMO approach. Figure 3 shows our current service ontology.

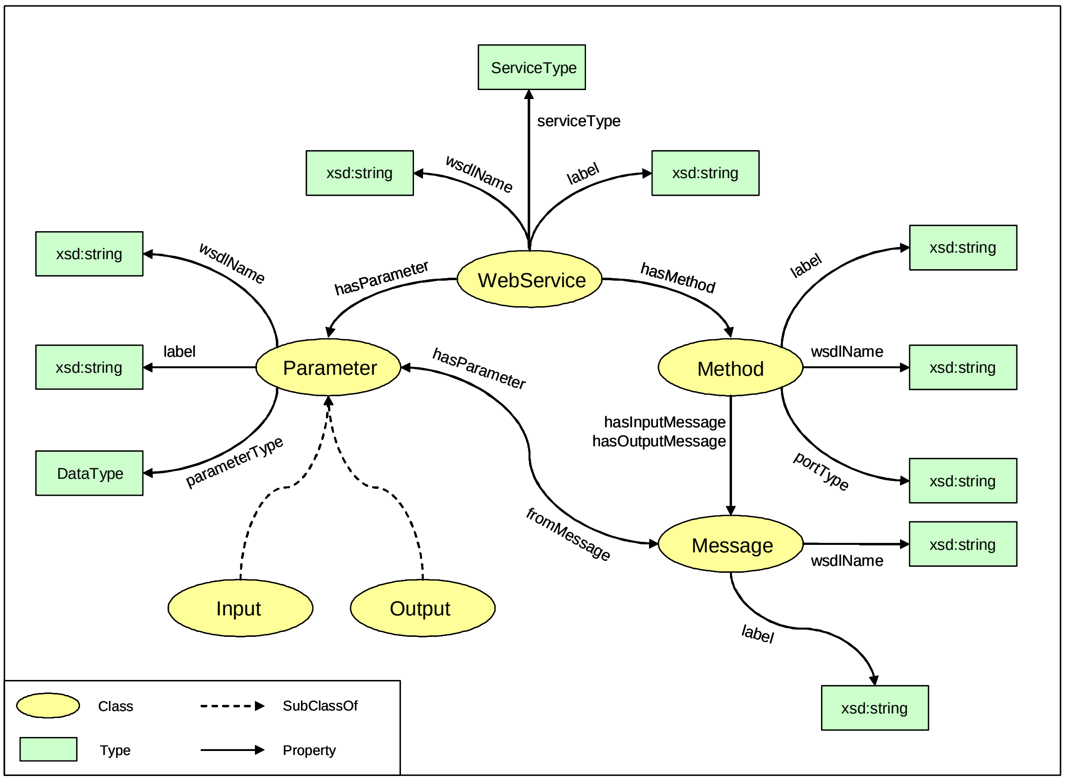


Figure 3. Web service ontology

5. CONCLUSIONS

We have described a platform that exploits the wealth of functionality available today on the WWW in the form of web applications, for the semi-automatic provision of large-scale collections of web services, including support for the execution of the services. This platform can help build a considerable mass of ready-to-use web services with little effort, thus serving to feed other research initiatives on web service technologies with low-cost testbeds.

There is a big gap between the web service technologies that are currently achieving adoption in industry (WSDL / SOAP / UDDI) and the semantic web services vision. Our proposal takes a realistic, incremental approach to bridge this gap, in contrast to more ambitious ones like the OWL-S and WSMO initiatives. The semantic reach of our techniques shall grow as the amount of semantic information we are able to extract increases. Our bottom-up strategy is thus complementary to the top-down approach taken by OWL-S and WSMO, sharing the final purpose of overcoming the distance between available technologies and the envisioned semantic web of services.

With regards to the current status of development of our system as of this writing, we are extensively testing our platform, and extending the collection of web applications to experiment with. The work described in 3.1 and 3.2 on the automatic generation of functional web services from web applications has been fully developed in a first prototype. The prototype, named Federica, can be tried at http://rhadamanthis.ii.uam.es:8080/federica. This prototype is currently under trial and improvement. The incredible diversity in the ways and styles by which web applications and web pages are encoded constitutes a real challenge that requires a lengthy testing and adjustment phase, in order to achieve a high enough success rate by a system which, as ours, is based on source code analysis. Of especial value for our tests are the web applications for which, either directly or by similarity, a freely available web service description exists, because it provides an objective assessment criteria for our results by comparison. Our preliminary tests with search portals such as Google, Yahoo, and YASE (see http://rhadamanthis.ii.uam.es:8080/YASE), currency exchange services such as XE Currency Converter, Oanda, and X-Rates.com have shown satisfactory results.

The integration with Jacinta, described in 3.3, is still work in progress. Meanwhile, in order to execute and test our generated services as described in 3.3, a temporary, simpler mechanism has been devised which, although less general, still allows the generation of functional clients with a simple web interface. The work described in Section 4 is also largely under development.

Our most immediate future work includes the identification and modeling of new semantic aspects that can be subject to automatic description, and the extension of our system to acquire this information semi-automatically. Another foreseen improvement of our current work is the characterization of service responses, which currently consists of a string containing the answer page of the original application. For this purpose, we are complementing the environment with service administration tools to allow advanced users to edit the generated service descriptions. One of such tools will allow the interactive definition and refinement of output service parameters. This extension of the environment will be based on the manual selection of values for the service outputs within the HTML response. By analyzing these selections, and their surrounding context, the system must identify the data type of these values, using XML Schema complex types if needed, and define value access expressions from the web page.

REFERENCES

Berners-Lee, T. et al, 2001. The Semantic Web. *In Scientific America*, Vol. 284, No. 5, pp 34-43.

Christensen, E. et al, 2001. Web Service Description Language (WSDL), v1.1. *In http://www.w3.org/TR/wsdl*.

Gómez, J.M. et al, 2004. GODO: Goal Driven Orchestration for semantic web services. *In Proceedings of Workshop on WSMO Implementations (WIW)*. Frankfurt, Germany.

Gómez, J.M. et al, 2005. The Semantic Web Service Tetrahedron: Achieving integration with Semantic Web Services. *Accepted for publication in 5th International Conference on Web Engineering (ICWE2005)*. Sydney, Australia.

Gudgin, M. et al, 2003. SOAP Version 1.2 – Part 1: Messaging framework. *In http://www.w3.org/TR/soap/.*

Martin, D. et al, 2004. OWL-S: Semantic markup for web services, v1.1. *In http://www.daml.org/services/owl-s/.*

OASIS 2004. UDDI: The UDDI technical white paper. *In http://www.uddi.org*.

Oren, E. et al, 2004. Overview and Scope of WSMX. *In http://www.wsmo.org/TR/d13/d13.0/v0.2/*.

Pham, H-H., 2004. B2B with Toshiba Web Service Gateway. *In Proceedings of Recherche Informatique Vietnam & Francophonie*, pp. 137 – 142.

Rico, M. and Castells, P., 2004. Jacinta: a mediator agent for human – agent interaction in semantic web services. *In Proceedings of International Semantic Web Conference*, Selected Posters, Hiroshima, Japan.

Roman, D. et al, 2005. Web Service Modeling Ontology (WSMO). *In http://www.wsmo.org/*.

Sahuguet, A. and Azavant, F., 1999. WysiWyg Web Wrapper Factory (W4F). *Unpublished*.

Terziyan, V. Y. and Kononenko, O., 2003. Semantic web enabled web services: State-of-the-art and industrial challenges. *In Proceedings International Conference on Web Services 2003 (ICWS 2003),* pp. 183 – 197.