SEMI-AUTOMATIC WEB SERVICE GENERATION AND CLASSIFICATION

Automated assistance to the web service life cycle

Miguel Angel Corella, José María Fuentes, Pablo Castells
Universidad Autónoma de Madrid, Escuela Politécnica Superior, Campus de Cantoblanco, 28049 Madrid, Spain

Abstract: The convergence of semantic web techniques with web service technologies has enabled the emergence of so-called semantic web services. This new kind of services enacts the automatic manipulation of services by software programs, to perform tasks such as automatic service location, composition, and invocation. In this paper, we propose methods and techniques that enable the semi-automatic generation, deployment, semantic annotation and classification of web services.

Key words: semantic web services; web service generation; web service classification; web service ontologies; reasoning; WSDL; OWL

1. INTRODUCTION

Since the emergence of the semantic web (Berners-Lee et al., 2001), many research efforts have been aiming to use semantics to endow web services with a much higher potential for automation. These efforts have resulted in a new research trend called semantic web services (Terziyan et al., 2003). The basis of this trend is to attach some semantic information to current WSDL-based web service descriptions (Christensen et al., 2001) in order to enable their analysis and manipulation by software programs. This manipulation would be useful to enact powerful capabilities such as automatic location, selection, invocation or composition of web services.

Nevertheless, semantic web services are facing today some problems and limitations that restrain their advancement and evolution. From our point of view there are two main problems to be solved:

- There is no consensus on which language has to be used when describing the semantic information about web services, although there are two main proposals that are gaining momentum at the present time, OWL-S (Martin et al., 2004) and WSMO (Roman et al., 2005).
- There is a lack of automatisms to help web service developers in the creation of services, the annotation of their semantic information or any other task related to the web service life cycle.

The research presented here is set forth to help overcome these problems. More specifically, our work proposes:

- A realistic approach to the semantic information that can be obtained and used from a web service.
- A set of automatisms that will allow web service developers to perform some tasks related to web services. More precisely: creation, deployment, testing, semantic annotation, and classification.

The solutions developed in this work have been implemented and integrated in a web platform called Federica. Besides describing this platform in some detail, the focus of this paper will be to explain our research ideas and discuss the results achieved so far.

The rest of the paper is structured as follows. Section 2 shows all the details of our work towards the semi-automatic generation of web services. Next we explain our approach to the semi-automatic classification of web services. Finally, on Section 4, we provide some conclusions, as well as the next foreseen steps.

2. WEB SERVICE GENERATION

The critical mass of available web services, let alone semantic ones, is still quite limited today. This is an important practical barrier for the advancement of research and innovation in this field, which is difficult to achieve without a sufficient testbed to try and evaluate the innovations. Artificial examples (i.e. built by the innovators themselves) hardly provide an objective basis for measuring the usefulness and performance of new proposals, not to mention the considerable cost implied in building the testbed, just for experimentation purposes.

The semi-automatic generation of web services, from such a widespread commodity as are web applications, can help with this necessity, and is an interesting research problem by itself. Of course, the expected quality of

automatically generated services should not be the same as that of manually defined ones, but we aim at achieving a sufficient quality for the services to be useful for a variety of purposes, where of course, if needed, the generated web services can be completed or refined by a programmer. Moreover, such a facility as we are proposing here can be helpful in the transition from the current World Wide Web of applications to a (Semantic) Web of services.

The idea of the automatic generation of web services from web applications has already been addressed in former research works. Because of its relevance for our research, it is worth citing the work developed by Pham (2004), which already proposes the creation of web services from web applications. In Pham's proposal, web services are used only as gateways to web applications, so that any program can invoke programmatically the functionalities provided by web applications. When the generated web services are called, they translate their input parameters to HTTP parameters, send them to the application server, and wait for the response. Our work takes a step further from this, by recognizing or generating data types, finding associations of types with ontology concepts, if any, and automatically classifying the generated services. Overall, our research aims to push forward the goals undertaken by Pham et al towards the generation of semantically-enhanced web service descriptions.

It is also interesting to cite the early work by Sahuguet et al (1999) in a similar direction. Although this work does not use an explicit notion of web service, since web service standards had not yet seen the light by that time, their approach is very similar to the one proposed by Pham. The main differences can be attributed to the status of the technology at the time of publication – while Pham uses web service languages and tools to build a gateway to web applications, Sahuguet et al. use non-standard descriptions (manually generated) and generate Java applications as a gateway to the functionality.

The process flow of our approach to the generation of web services is shown in Figure 1. The input to the automatic generation system is the entry web page of an application. The page is parsed ("HTML parser" box in the figure) into a easier to process in-memory data structure, which is analyzed in order to produce a WSDL description (WSDL Generator module) for the service to be generated, plus some additional semantic descriptions (Semantics Generator module). An implementation of the WSDL service is automatically generated (Service Implementor module) and deployed into a web service support platform (Axis and Tomcat have been used). Once the service is deployed, it can be invoked from any web service client that adheres to the generated WSDL description. One such client is automatically generated for testing purposes. Calls to the generated web service (SOAP requests) are automatically deferred to the original web application (HTTP

request) by the generated service implementation. The result (a web page source code) is returned to the service client (SOAP response).

The steps enumerated above will be explained in more detail in each of the following subsections. More precisely, in section 2.1, we explain how WSDL descriptions are extracted from the web interface of applications. In section 2.2, the linkage of web services with web application functionality, and the deployment of the service, are described. Then, in section 2.3, we show how the execution of the generated web services is managed.

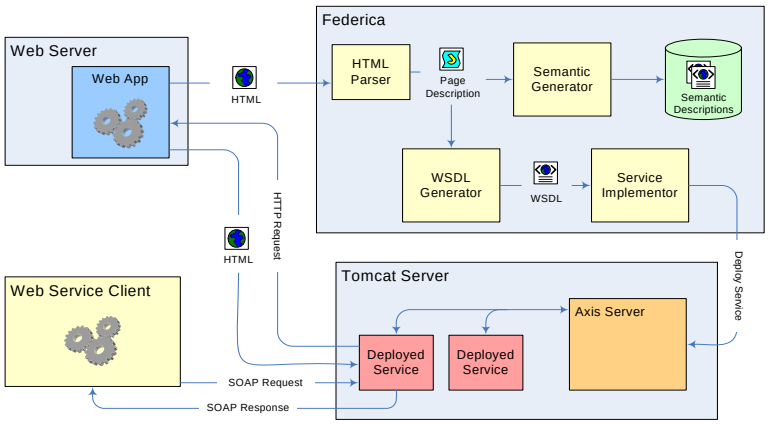

Figure 1. Overview of the web service generation and deployment process in Federica

2.1 Extracting web service descriptions

For the automatic analysis and description of the functionality provided by a web application, the system we propose takes as input a) the URL of the web page which gives access to the application, and b) a name to identify the service to be generated. Our approach for the generation of WSDL descriptions is based on the inspection of the source code of the entry web page to the application, more specifically the HTML forms, as the basic UI construct for web applications. Of course web application UIs are being implemented today using other client-side technologies (JavaScript, Flash, etc.) beyond plain HTML, but as a first approach to the problem, we have circumscribed it to HTML forms.

In addition to the UI elements to interact with the application, the source code of a web page through which a web application is accessed includes all kinds of additional content, just as any other web page does: purely informative contents (titles, instructions, logos, advertisements, etc.),

navigation elements (hyperlinks), plus page style and layout details. Although all these elements surrounding UI controls could be exploited to find cues which help in the analysis of forms, in our current approach we only inspect the UI elements themselves.

From the analysis of the web UI, our system determines the operations of the web service to be generated, their inputs, and the type of these, and this information is retained in a WSDL description. The generation procedure works through the following steps:

1. A service with a unique wsdl:portType is defined for the whole application.
2. The application page source is read from its URL, and the HTML forms that the page contains are extracted using an HTML parser.
3. A wsdl:operation is created for each of the forms found in the page.
4. An output message and an input message are defined for each wsdl:operation.
5. The “input”, “select” and “textarea” HTLM components 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 attributes, a different data type is assigned to each service input, a new XML schema type being defined in some cases. Table 1 shows the correspondence between UI controls and service message part data type.
7. The wsdl:output message will contain only one wsdl:part of xsd:string type which, once the service is invoked, will return the source code for the web page that is returned from the web application form.

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

Table 1. Mapping between HTML components and XML Schema data types

Figure 2 shows an example of the generated WSDL description for the Google home page.

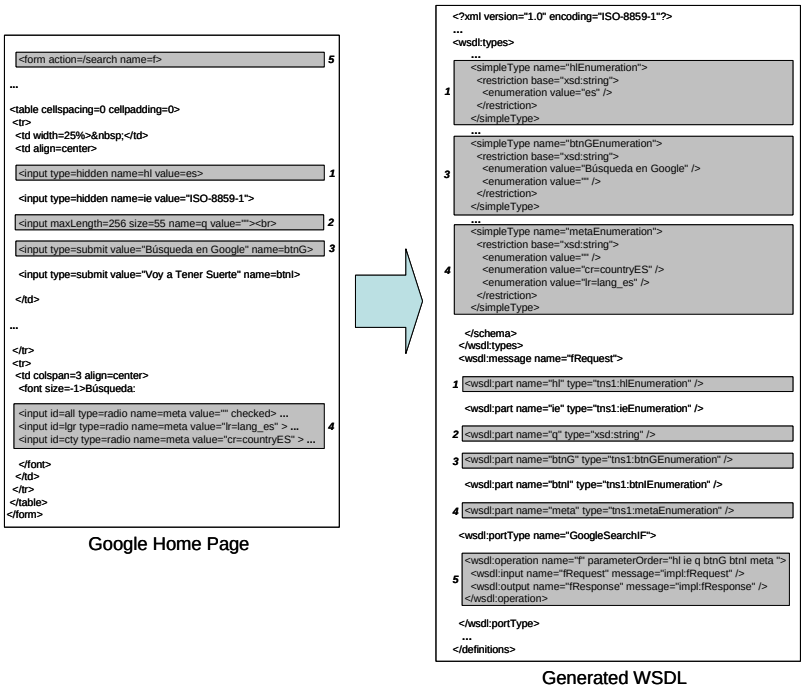

Figure 2. Generated WSDL for Google home page

2.2 Linking web services to web applications

Once an WSDL is generated, our system provides an automatic implementation of the service, consisting of an application wrapper from which calls to the service are transferred as requests to the initial web application. To achieve this, a link engine has been implemented using Axis for sending and receiving SOAP messages (Gudgin et al., 2003) in the communication with web services.

Using the Axis WSDL2Java tool, a set of empty Java classes is generated from a WSDL description, which once implemented will define the desired web service functionality. Service deployment is also done with Axis, the engine of which processes the SOAP messages received by the service.

The result of this process is a fully functional web service, whose operations carry out the same tasks as those the web application provides.

2.3 Execution of generated web services

In order to try the generated services, our environment provides the automatic generation of a client and a web interface to invoke the generated services. This facility is an invaluable tool for development and testing. Furthermore, we plan to integrate this capability with a sophisticated editor with which developers can refine the service response description, and make it more precise, e.g. as an XML Schema data type, rather than a plain web page fragment source. Furthermore, this option opens further possibilities to augment descriptions with richer semantics, in the spirit of the Semantic Web Services vision. These would also be useful, in particular, for the automatic classification of the service, which is addressed in the next section. We have started the development of this edition tools, but, at the present time they are not yet integrated with the rest of the platform.

3. WEB SERVICE CLASSIFICATION

With the techniques described in previous sections, we have the capacity to generate functional web services almost automatically and have them deployed (and stored) in a repository. Nevertheless, those services are stored in a chaotic way, without any type of structure. This lack of structure in the service storage can be seen as a problem because tasks such as, for instance, searching for a concrete service, gain much complexity. In addition, if we analyze the current situation of web services we see that classification is already an important concern by itself.

Nowadays, UDDI (OASIS UDDI, 2004) is the most accepted and used protocol for publishing, searching, and finding services over the web. These actions are usually performed using UDDI registries, which can be seen as service repositories easily accessed through a URL. In these registries, the published services are classified using some kind of taxonomy (i.e. UNSPSC, NAICS, etc.). Nevertheless, this classification is performed manually by the human publisher of the service. Due to the huge quantity of service classes in taxonomies as the ones mentioned above, the classification process could be very complex and tedious.

What we are proposing here is to use a software agent to help service publishers in the classification task. In order to meet this goal, in the context and spirit of the generation process described in previous sections, our basic starting point for any automated processing of the generated services are WSDL descriptions. Nevertheless, the information contained in a WSDL description is not sufficient to perform any type of classification. So, we need more information about services, their parameters, their operations, etc.

In conclusion, we need a higher level description of the service including some semantic information that enables a software agent to help publishers in the classification task.

In addition, as we are trying to classify web services in a taxonomy, we will need, of course, to define that taxonomy. For this purpose, our system can use any available taxonomy, such as the standards currently recommended by UDDI registries, e.g. UNSPSC or NAICS.

Now starting from a taxonomy and a means to represent the semantic information of a web service, the classification process is based on comparing the new generated web services with some services previously classified in the taxonomy. From this comparison, we obtain a similarity measure representing the probability that a service should be placed under a certain classification category in the taxonomy. The definition of similarity measures between taxonomy concepts has been already addressed, for example, by Bernstein et al. (2005).

All the details about each issue taking place in the classification process can be found in the following subsections. More precisely, in section 3.1, all the information about the representation of web service semantics can be found. Next, in section 3.2, we will explain some details related to the taxonomy we are going to use and the previously classified services we will use for comparisons. Finally, in section 3.3, we will show the complete process we will apply to obtain the similarity measure between services and taxonomy classes.

3.1 Semantic annotation of web services

Our first step towards the achievement of the classification mechanism is to obtain, define and represent all the extra needed semantic information for a service. As it has been already said, there are mainly, at the present time, two initiatives on semantic description languages for web services: OWL-S and WSMO. Both of these languages provide a highly generic way to describe semantic information about a web service. Because of this, they are considerably complex and difficult to use, even for very simple services. Therefore, we have decided to define a simpler ontology that fits better with our purposes. We do not discard using one of these initiatives in the future, as we gain more generality in our own service descriptions.

We have chosen OWL (McGuiness et al., 2004), widely accepted in the semantic web community, as the ontology-definition language for our service ontology. With this language, we have developed a simple ontology containing all the concepts (classes) and relations (properties) that are described in WSDL service descriptions. The ontology is shown in Figure 3. In the current status of our research, the ontology does not include much

semantics about the different service concepts, but we expect to extend it with new information as soon as it can be automatically or semi-automatically retrieved.

Using the aforementioned ontology we have an OWL representation of the service in terms of the information described in WSDL. The "extra semantics" in this description with respect to plain WSDL is held:

- In the "serviceType" property of the "WebService" class. This allows linking the description of the service to a type of service defined in other ontologies (or taxonomies), i.e. the one explained on the section 3.2.
- In the "parameterType" property of the "Parameter" class. This allows linking the description of a data type to a type defined in other data type ontologies.

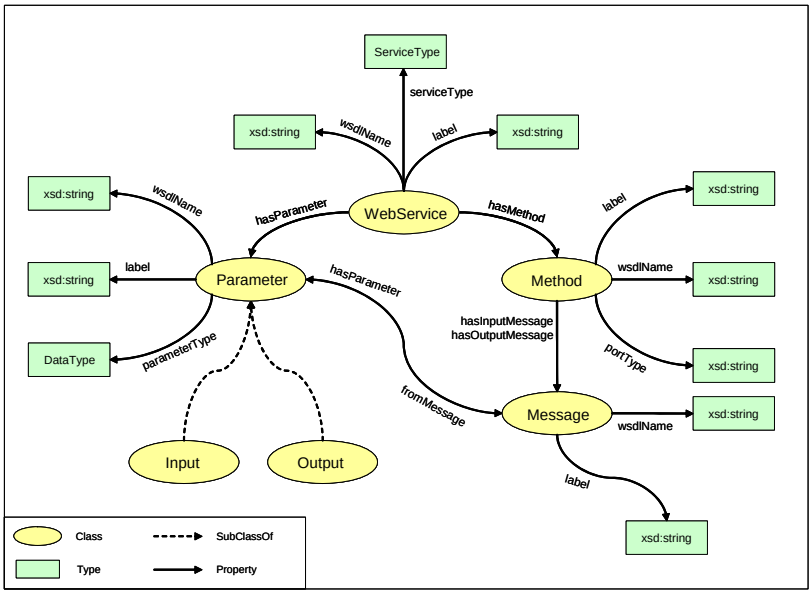

Figure 2. Web service ontology

With this ontology we add to WSDL the possibility of representing the classification of a web service and a higher-level description of data types. Regarding the latter, and in order to enrich service parameter descriptions, we have defined a data type ontology to which parameter data types will be mapped. This mapping means that the range of the "parameterType" property of the web service ontology described above is conformed by the classes of the data type ontology, the main elements of which are shown in Figure 4.

As can be seen, the ontology defines three kinds of data types:

- Simple data types: This is an ontological representation of XML Schema data types. The subclasses represent the different kind of data types that can be used in XML Schema (although the classes represented each data type have not been shown in the figure).
- Complex data types: This is an ontological representation of XML Schema complex types, i.e. structures, which could appear in a WSDL. Obviously, as we do not know, a priori, which structures will be in the analyzed WSDL descriptions, this class is initially empty.
- Enumerated data types: Despite these data types could be seen as restricted simple types, we have decided to place them under a separate class. This decision is based on the fact that, due to the basis of the generated web services (this is, web applications), the appearance of enumerated data types is very common (i.e. every combo box of an HTML form).

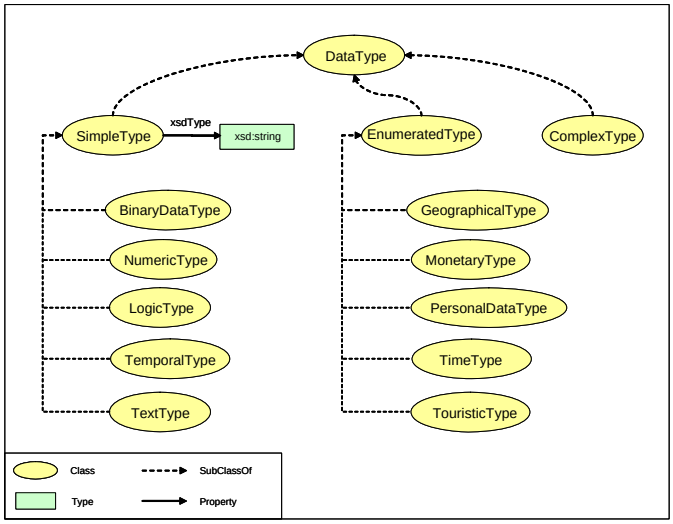

Figure 3. Data type ontology (main classes)

Thus, with this ontology we have an OWL representation of the different data types used by a service. These representations are very useful when trying to classify a service, as they can be used as source for a data type similarity comparison between two services.

Then, with the union of both ontologies we have all the service information needed represented in an ontological way and the possibility of using software agents (i.e. based on OWL reasoners) to perform the classification of the service.

3.2 Classification procedure

Given a set of classified WSDL descriptions under some classification taxonomy, and a newly generated web service description, based on the service ontology and techniques described in previous sections, in order to deploy the new service, and make it available in a service registry, an appropriate classification category should be assigned to the new service, e.g. for easy retrieval.

As it has been already said, we use the classification standards used by UDDI registries (such as UNSPSC or NAICS) as the taxonomy where our web services will be classified, but our techniques are independent from the standard used. To facilitate our development, and in anticipation of future semantic-web-oriented extensions of our work, we are using an OWL representation of the taxonomy. This has the advantage that the same tools, e.g. OWL reasoners, can be used to process service descriptions and classification hierarchies in a uniform way (e.g. for the computation of similarity measures between services and categories, reasoning about class hierarchies, etc.).

We have developed a heuristic for automated service classification, based on the comparison of an unclassified service with available classified services, whereby a measure of the likelihood that the service can be correctly classified under some category is computed. This problem is related but different from the ones addressed by other service comparison techniques found in the literature (Bernstein et al., 2005; Sirin et al., 2004; Bansal and Vidal, 2003; Field and Hoffner, 2003). In particular, in our approach, the fact that two services are found similar does not mean they are interchangeable for e.g. automatic service selection, composition, or invocation. Instead, the similarity measures provide a ranking of candidate classifications for a new service, which can be of great help for a human service administrator that is facing classification taxonomies with thousands of categories.

Our heuristic works as follows. Let $\mathcal{S}$ be the set of all web services, and let $\mathcal{C}$ be a classification taxonomy. Since $\mathcal{C}$ is used to classify the services in $\mathcal{S}$, we may define $\mathcal{C}$ as a subset of the parts of $\mathcal{S}$, $\mathcal{P}(\mathcal{S})$, i.e. each $c \in \mathcal{C}$ is a labelled set of services $c \subset \mathcal{S}$ (note that it is possible that $c = \emptyset$). Given a new service $s \in \mathcal{S}$, we want to find the category $c \in \mathcal{C}$ that maximizes its similarity with s. We measure the similarity between s and c is by the following formula:

$$\mathrm{sim}(s,c) = \sum_{c' \subset c} (-1)^{|c'|+1} \prod_{s' \in c'} \mathrm{sim}(s,s')$$

whereby the similarity between a service and a category is computed in terms of the similarity between the service and the services classified under that category. We have chosen the above formula because it meets the following desired properties:

- $\text{sim}(s,c) \in [0,1]$ provided that $\text{sim}(s,s') \in [0,1]\ \forall s' \in c$
- $\text{sim}(s,c) \geq \text{sim}(s,s')\ \forall s' \in c$ (in particular this means that $\text{sim}(s,c) = 1$ if $\text{sim}(s,s') = 1$ for some s').
- $\text{sim}(s,c)$ increases monotonically with respect to $\text{sim}(s,s')\ \forall s' \in c$,
- Since $\forall s' \in c$, $\text{sim}(s,c) = \text{sim}(s,s') + (1 - \text{sim}(s,s'))\ \text{sim}(s, c - \{s\})$, $\text{sim}(s,c)$ can be computed efficiently (i.e. with linear $\Theta(|c|)$ complexity).

Now, the similarity between two services is measured in terms of the similarity of their operations and parameters. If we denote by P_s the set of the parameters of service s, and by OP_s the set of its operations, the similarity of two services is defined as:

$$\text{sim}(s, s') = f(\text{sim}(P_s, P_{s'}), \text{sim}(OP_s, OP_{s'}))$$

Developing a measure for comparing service operations is still work in progress in our research as of this writing, so in the meantime we are working with $f(x, y) = x$.

The similarity between two parameter sets P and P' is computed as the average of the best possible pairwise similarities obtained by an optimal pairing of the elements from the two sets. This can be formalized as follows. We define $\text{top}(P,P')$ as the pair $(p,p') \in P \times P'$ that maximizes $\text{sim}(p,p')$. Then let $P_1 = P$, $P_1' = P'$, $P_k = P_{k-1} - \{p_{k-1}\}$ and $P_k' = P'_{k-1} - \{p'_{k-1}\}$, where $(p_{k-1},p'_{k-1}) = \text{top}(P_{k-1},P'_{k-1})$. With these definitions, the similarity between two parameter sets is given by:

$$\text{sim}(P, P') = \frac{\sum_{i=1}^{\min(|P|,|P'|)} \text{sim}\left(\text{top}\left(P_i, P_i'\right)\right)}{\max\left(|P|, |P'|\right)}$$

In our current model, the similarity between two parameters is defined as the similarity between their respective types. Let $\mathcal{T}$ denote the set of all types (i.e. the set of domain ontology classes). We define the similarity between two types $t \in \mathcal{T}$, $t' \in \mathcal{T}$ as:

$$\text{sim}(t,t') = \begin{cases} 1 \text{ if } t = t' \\ \dfrac{|t \cap t'|}{|t \cup t'|} \text{ if } t \text{ and } t' \text{ are subclasses of } \texttt{EnumeratedDataType} \\ 0.5^{h(t,t')} \text{ if } t \text{ subtype of } t' \\ 0.5^{h(t',t)} \text{ if } t' \text{ subtype of } t \\ 0 \text{ otherwise} \end{cases}$$

where $h(t, t')$ denotes the distance between t and t' in the type hierarchy (e.g. the distance between a type and its immediate supertype is 1).

With this process, we can have finally a vector of similarity measures between a new analyzed service and all the taxonomy classes. Then it is time for service publishers to decide on which class (from a ranked list) the service best fits.

4. CONCLUSIONS AND FURTHER WORK

Our work shows that automatic web service generation is feasible. In fact, we have implemented all the ideas shown in this document and have developed a first version of the Federica platform (accessible at http://rhadamanthis.ii.uam.es:8080/federica). In that platform it can be seen that tasks such as generation and deployment of web services are done automatically. It also includes the tools needed for web service execution and testing. Nevertheless, the edition tools that have been discussed in section 2 are not yet integrated with the rest of the platform.

Our work on semi-automatic classification has some commonalities with the research on web services matchmaking (Sirin et al., 2004; Bansal and Vidal, 2003; Field and Hoffner, 2003). The approaches in those proposals are based on the matching between IOPEs (input, output, precondition and effects). As available datasets providing such descriptions are scarce today, at present we restrict our matching methods to service inputs and outputs. Of course, as both semantic web services technology and our research evolve, we could extend and improve our classification techniques by exploiting these new semantic features. We expect this to be an important direction of progress for our work, since the richer the description of services, the more advanced automation techniques can be devised.

Meanwhile, the goal of this paper is to show it is possible to develop automated assistance capabilities for service generation and classification, using only syntactic descriptions (WSDL), service taxonomies, and data type ontologies (in OWL), which are already in use in the current state of semantic web and web service technology development and adoption, or

which can be created or completed with little effort. Some current gaps, that are requiring an effort from our part in compensation, include the construction of a large-enough repository of manually classified WSDL service descriptions (say, by the hundreds), to serve as testbed for our techniques, and some current limitations of available semantic web tools, such as the OWL reasoners. It is to be expected that our workarounds can be removed as these tools keep reaching maturity.

Rather than aiming at, or starting from, a maximum semantic web service representation expressiveness, as in OWL-S and WSMO, we have approached our research objectives in a bottom-up approach, starting from consolidated, ready-to-use technology, such as WSDL and OWL, already being adopted by industry as of today. From this standpoint, our long-term goals are the same as those of OWL-S and WSMO, namely to bridge the gap between current web service technology, and the vision of Semantic Web Services.

Our next steps in this direction include: growing the collection of services generated by our platform, stored in our repository; extend the semantic information represented in our service ontology, that can be obtained in an automated way; complete the development of the edition/administration tools to customize and improve the quality of the generated services; further testing, evaluation, and tuning of the automatic web service classification techniques.

5. ACKNOWLEDGEMENTS

We would like to thank Ruben Lara for his thorough revision, comments, and suggestions on the early drafts of this paper, which greatly helped to improve our work.

REFERENCES

Bansal, S., Vidal, J. M., 2003, Matchmaking of web services based on the DAML-S service model, *Autonomous Agents & Multiagent Systems 2003* (AAMAS 2003), pp. 926 – 927.

Berners-Lee, T., Hendler, J., Lassila, O., 2001, The semantic web, In *Scientific America*.

Bernstein, A., Kaufmann E., Bürku, C., Klein, M., 2005, How similar is it? Towards personalized similarity measures in ontologies, *Wirstchaftsinformatic 2005*, pp. 1347 – 1366.

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., 2001, Web Service Description Language (WSDL), v1.1; http://www.w3.org/TR/wsdl

Field S., Hoffner., Y., 2003, Web services and matchmaking, *Int. J. Networking and Virtual Organisations*, Vol. 2, pp. 16 – 32.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J-J., Nielsen, H. F., 2003, SOAP Version 1.2 - Part 1: Messaging framework. http://www.w3.org/TR/soap/

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Scycara, K., 2004, OWL-S: Semantic markup for web services, v1.1; http://www.daml.org/services/owl-s/

McGuiness, D. L., van Harmelen, F., 2004, OWL: Web Ontology Language overview, 2004. http://www.w3.org/TR/owl-features/

OASIS UDDI, 2004, UDDI: The UDDI technical white paper; http://www.uddi.org/

Pham, H-H., 2004, B2B with Toshiba Web Service Gateway, In Proc. *Recherche Informatique Vietnam & Francophonie 2004* (RIVF 2004), pp. 137 – 142.

Roman, D., Lausen, H., Keller, U., de Brujin, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Kifer, M., König-Ries, B., Kopecky, J., Lara, R., Oren, E., Polleres, A., Scicluna, J., Stollberg, M., 2005, Web Service Modeling Ontology (WSMO); http://www.wsmo.org/

Sahuguet, A., Azavant, F., 1999, WysiWyg web wrapper factory (W4F)". Unpublished, 1999.

Sisrin, E., Parsia, B, Hendler, J., 2004, Filtering and selecting semantic web services with interactive composition techniques, *IEEE Intelligent Systems*, 19(4): 42 – 49.

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