INTEGRATING ONTOLOGIES VISUALIZATION WITH THE EDITION OF PROFILES IN SEMANTIC WEB SERVICES: "ONTOSERVICE"

J. Javier Samper, Vicente R. Tomás, Eduardo Carrillo, Rogerio P. C. do Nascimento, J. José Martínez

Abstract

The present paper describes the main characteristics and components of the tool developed for integrating the definition of profiles for semantic web services. This tool is based on the languages DAML-S and OWL-S and includes the ontology visualization and consistency verification that specify the concepts a web service interacts with. Starting from a service description interpreted by a computer and by the means used for accessing the service, it is possible to find out which software agents use the service. The tool can generate information in different formats, such as the SVG format proposed by the W3C, for representing graphic information based on XML. The tool was developed in Java language and is being used for the visualization of ontologies and for the semantic description of services in traffic information systems that the LISST group (Laboratorio Integrado de Sistemas Inteligentes y Tecnologías de la información de Tráfico) developed at the University of Valencia.

1. Introduction

Content semantic marking and the web service capabilities will make it possible to automate a great variety of reasoning tasks, which are currently developed by human beings through manual codification, which allows a subsequent automation. There are however questions that need to be solved: what does the service do?, how can it be used? which effects will it have? The automation of tasks such as the discovery of the service, interoperability, selection, composition and monitoring of the execution are the basis of Semantic Web Services [6], [12].

The international community has made an effort to look for a more intelligent web – the Semantic Web – and to apply it for the development of web services. For that purpose, ontologies such as DAML-S and OWL-S were proposed, which are based on the semantic-marked languages

1 University of Valencia (Spain) Polígono de la Cañada s/n, Paterna, Valencia, Spain. E-mail: javier.samper@uv.es
2 Autonomous University of Barcelona C/fo. 42, Edificio T1, 08028 Barcelona, Spain. E-mail: viitemas2007@icatel.es
3 University of Valencia (Spain) Polígono de la Cañada s/n, Paterna, Valencia, Spain. E-mail: javier.samper@uv.es
4 University of Minho (Portugal) Department of Electronics and Information Engineering. E-mail: marcosm@di.uminho.pt
5 University of Valencia (Spain) Polígono de la Cañada s/n, Paterna, Valencia, Spain. E-mail: javier.samper@uv.es
DAML+OIL and OWL. Such ontologies have classes and properties that allow specifying the properties and capabilities of web services, no matter how they will be used.

With conventional mechanisms such as a Web Services Description Language (WSDL), the descriptions of the services are not as expressive as the profiles of the services expressed with profiles DAML+OIL / OWL-S (preconditions, postconditions and effects cannot be expressed with WSDL). Also, the Universal Description Discovery and Integration (UDDI) does not support semantic descriptions of the services. Since there is not an explicit semantic, two identical descriptions may be different depending on the context of use. This is clearly a great disadvantage for searching and discovering services, as the client will not know which services are available on a given moment and, also, both providers and clients will have different perspectives and knowledge about the service [8].

2. State of the art

During the research, the main identified problems were the lack of tools for the edition of ontologies that were allowing to generate code in different semantic markup languages (such as DAML+OIL and OWL). The functionality of some of them was composed, such as DUET OntoSource, OntoEdit, SWGOO WebOGE, WebOnto, Protégé and Oracle [2] [9-11], [15-19]. Important research related to some of these tools was developed by Corcho et al. [2].

Most of the OWL-S editor tools start with a WSDL description of the Web service in order to use a WSDL to OWL-S tool which offers a framework of each one of the OWL-S components starting with the information that the WSDL provided. In a later phase, the providers can complete such description, which was automatically generated.

Regarding the tools for the description of semantic web services, the main proposals found are OWL-S Editor and A-Match from Carnegie-Mellon University, OWLS Editor from Semantically Enabled Web-Services from the Department of Computational Sciences and Artificial Intelligence at the University of Maryland [16], OWL-S Editor by SRI International [4] and different tools from the University of Maryland. It can be observed that, although there is help for profile edition, there is not a direct integration with the visualization of ontologies for concept description.

Carnegie-Mellon University has developed different tools, such as the OWL-S editor. The finality, as well as the rest, is the creation of different files that describe the semantic capacity of the services. As the rest, it is a tool oriented to the service provider. It offers graphic visualizations and integrated reasoners, which are capable of verifying the properties of the services and allow the simulation and diagnosis of each service. The reasoner that Carnegie-Mellon University uses is JTP and, unlike our reasoner, it is based on First Order Logic (FOL), not on Description Logic (DL), which allows to load the different ontologies of use and to establish the different classes and properties of the services. The importance of the tool consists in the possibility of specifying composed services and of establishing the service data flows.

A-Match is a web interface of the services matcher RETSINA, which the Intelligent Software Agent group developed at the Carnegie-Mellon University. In the multi-agent system RETSINA, the services matcher maker acts as a link between the agents that request services and those that provide services. With RETSINA, the agents communicate their advertisements and service
requests to the match system automatically through KQML messages. A-Match provides an interface to human users so as to advertise and find agents that have the capabilities needed.

The prototype developed by the University of Maryland combines the description of OWL-S services with current invocations of WSDL descriptions, which allow to execute composed services. The system can generate semantic OWL-S descriptions from WSDL descriptions, which make it possible both service composition and execution. The University of Maryland studies compatibility between outputs and inputs of the services and claims that, in order to take a greater advantage, the outputs and inputs should be concepts rather than class types (strategy used in our tool) in established service ontologies. The University of Maryland further adds the problem related to service composition demanded and the difficulty of establishing a complete automation without the user’s help and, therefore, the automation can only have a partial automation. As we do, they use service hierarchies so as to look for the service and, later, the edition of parameters through concepts. In order to check inconsistency and incompatibility, it uses an OWL reasoner called Pellet [14].

The University of Malta also has a complete editor for describing a Web service in OWL-S, which includes the generation of Profile, Process and Grounding files. The mechanism consists in providing a tool with the WSDL description of the service through an interface called OwsWiz so that the tool is the framework of the different descriptions (Profile, Process, Grounding) and the client only has to provide non-functional parameters, such as the name of the service, the data of the provider or its quality. The tool is formed of three parts depending on its functionality: Creator, validator and visualizer. The validator checks the ontologies through the use of Java API. The Viewer allows the user to visualize and print the descriptions graphically. The heart of the tool is the service composer, who can manage three different types of compositions: Sequence, If-Then-Else and Split.

3. Characteristics of the new tool

The first need found was the lack of a tool that combined the visualization of ontologies with the creation of search profiles. At this time, as an academic experience, a tool based on the integration of capabilities was created, which was used for specifying the profiles of semantic web services with visualization and verification of the concepts consistency over which a given service interacts.

The specification of semantic profiles is addressed at the search of services and, therefore, it is oriented to users rather than to the service provider. Therefore, only the profile file is generated from a template, which implies that the files Process and Grounding are unnecessary. The user can either store this profile on the hard disk or send the profile generated to an agent in charge of searching the Web service, since the tool can be integrated into an agent platform (Figure 1).
Regarding the management of the concept ontologies, the tool implemented includes an ontology viewer, which may be described in the languages DAML or in OWL. An important aspect related to the graphic viewer of ontologies is the need for presenting the information gradually and adapted to the user's needs. Sparse information may be insufficient while excessive information may detract from clarity of the concepts. For that purpose, three layers have been specified, which may help to activate or deactivate the visualization of instances, the dependencies between classes, the properties and their relations with the classes.

Another aspect that distinguishes the tool presented from the existing tools is the possibility of using a cache manager for searching profiles. The use of this profile cache, which stores its information remotely, allows users to access their personal information from any place they get connected. In this way, the process for profile search is accelerated for similar queries, minimizing the network load and the processing in the agent that makes the search.

4. Ontologies visualization

Visualization is carried out through a drawing area that is implemented with BatiK libraries [3]. These libraries allow to generate, visualize and edit SVG graphics with a JAVA program. In order to generate the code in SVG from the code DAML or OWL, a software for managing open source graphics called Graphviz [1] was used. Graphviz generates code in different formats such as SVG, JPG, PNG, among others, starting from a language for graph description, which allows the tool to export the ontology to multiple formats.
Figure 2 shows the steps that the application follows for generating graphics.

![Diagram showing steps for graphic generation](image)

Ontologies are loaded through two reading modes. Local files can be loaded by selecting them directly on the hard disk, although it is also possible to read a file remotely from an URL.

The origin OWL or DAML file can be translated with an interpreter in DOT code. In the translation process, the unnecessary information that the user does not want to visualize (layers) is eliminated, at the same time that further information on the graphic elements is generated. Graphviz uses the file in DOT format so as to create the code in SVG format, which is the one loaded in the SVG drawing area of BATIK and to export the file to other graphic formats.

Since the objective of the tool is to try to visualize the maximum number of possible aspects of an ontology, it is necessary to include an information filtering system so that the representation is clearer. For that purpose, three layers have been specified:

- The first layer, labelled as "Instances", allows to activate or deactivate the possible instances or individuals of the ontology. In the current version of the tool, it is necessary that the instances and the classes be specified in the same file.
- The layer "Class Dependences" visualizes the relations between classes, which include the DAML tags sameClassAs, sameOrEquivalentClass, equivalentClass, equivalentTo, sameAs, disjointWith, complementOf, intersectionOf, disjointUnionOf, UnionOf as well as their equivalent tags in OWL.
- The layer "Object Properties" shows the different properties defined and their relations with the classes. In this way, the visualizer allows to activate or deactivate different elements of the ontology so it is possible to eliminate elements that are not of interest, which helps to give an overview as clear as possible. When all the layers are deactivated, the class hierarchy is shown, which is expressed with the label subclassOf.

Moreover, new functionalities are added to the interface. These functionalities are located in four buttons on the bottom of the interface (see Figure 2): The first button is called "Ontology info", which opens a dialogue that shows general information associated to the ontology. Such information includes the total number of classes loaded, the properties loaded, the information contained in the loaded "Ontology", which includes the version, comments associated and listing of imports used, listing of namespaces and classes belonging to each namespace.
The second button labelled as "Reload" updates the ontology visualized, which serves the purpose of using the possible changes both at the source code and at the visualization. For example, activating or deactivating layers or changing the size of the graphic.

The so-called "Export" button allows to store the visualized graphic in a given output format. The listing of available formats is GIF, JPG, PNG, SVG, PostScript, PostScript with annotations PDF, CMAP, PICT and VRML.

Finally, the check button is used to search inconsistency and incoherence. For that purpose, the application makes use of RACER [7], an inference engine and a descriptive logic reasoner that has been incorporated in the interface. Racer works as an application client-server. The client sends a query through HTTP to the server, which sends back the queries.

When clicking the check button, three types of queries are sent to the server:
- (data-read-file "file") or (owl-read-file "file"), which is in charge of loading the file to be examined.
- (classify-bool), which checks that all the concepts have been met.
- (oboe-consistency), which checks that all the instances are consistent.

If everything is correct, the following messages show up:
- "Ok - Concepts are satisfiable"
- "Ok - Individuals are consistent"

On the other hand, the drawing area has associated some mouse events that provide additional information about the ontology.

The classes that have comments in the file OWL or DAML, using the label "<comment>" have associated events of the onMouseOver type. When passing over the mouse pointer, the statements show up in an information chart below.

All classes have an event of the type onMouseOver, which displays the Meandclustering over the class and shows additional information about the class that should not be seen in the drawing, mainly per clarity purposes. The user can find out about the classes that have additional information, since they are those that have an octagonal form. If not, a description of the error found is given.
5. Search for Web Services

Another functionality of the tool implemented is the search and call to Web services starting from
the generation of a search profile, where parameters may be configured, such as the name of the
service, keywords for the search, inputs, and outputs of the service being searched, geographical
localization area of the service, parameters related to the quality or the name of the service
provider. In the service search process, one of the hardest is the "match machine" process, which
consists in locating services by mapping the profile generated with the profiles provided by the
service providers.

The feature that deserves further mention is the possibility of introducing new search parameters in
a text mode or filling them automatically by selecting the concepts in the ontology visualized in
the graphic interface.

In the left side of the tool there is an interface with a tree structure (see Figure 3), which show three
types of hierarchical classifications of concepts: NAICS, UNSPSC and TRAFFIC. Going through
the tree structure in any one of the three classifications, it is possible to get to a leaf node of a tree.
Each leaf node has a link to the concept ontology. The interface shows information regarding the file
access to its concept ontology. The interface shows information regarding the file access through a
code codification. If the leaf node has the green colour means that the URL can be accessed and
has information about the concept ontology. If it is however in red colour, the concept ontology
cannot be accessed and, therefore, it should be loaded manually, for example, looking for it in
another URL.
The field "Keywords" serves the purpose of indicating which keywords should be included in the search. All the content of the field should be included within the label <proRef:profileTextDescription> of the profile. The field "Service Name" indicates the name of the service, in case that the user recognizes it. The content of the file is inserted inside a label <proRef:serviceName>.

The fields "inputs" and "outputs" can be filled out through the graphic interface. For that purpose, there are two charts that indicate that the corresponding field is to be filled. In this way, when clicking on a class in the ontology Viewer, it is possible to make a distinction between information for reading about a class or information for selection, such as input or output.

After clicking on a class that represents a given concept, the name of the class is copied into the field selected, including its namespaces. The fields that are filled in the profile are <proRef:profileParameterName> with the name of the class selected and <proRef:profileRestrictTo refResource="X">, where X is the name of the namespaces.

The field Geographic indicates the geographical scope of the service. For that purpose, it is possible to select the field by clicking on the label "Geographic".

The button "Search" has two functions:

1) It searches in the cache memory whether or not there is a similar profile to the current one. If so, the user informs that a similar search was made and some services were found. The user may use anyone of the services previously found in such a way that he does not have to make a similar search. The norm for comparing the profiles in memory with those stored is to compare inputs and outputs. If all of them coincide, it is then considered that the service previously found is of interest for the user and should therefore be shown.

2) It sends the profile generated to the agent in charge of searching the service. After sending a search profile, the agent sends back information in a SOAP message regarding which entry parameters should be known so as to execute the Web service that receives the information. When the application receives the information, it should show a form which was created dynamically and included the fields that should be filled in. For that purpose, the form was generated dynamically with HTML code shown in a window with canvas for the visualization of the HTML language.

6. Optimization using caches

In a service search process, one of the hardest parts is the process of "match machine", which consists in locating the services by mapping the profile generated with the profiles given by the service providers. The use of a cache that stores the latest queries sends back the services sought quickly without a search process, increasing the system speed since the users will probably search a reduced number of service classes. A storage architecture has been designed which allows a user to get connected with the profile server from any place and to load the latest searches made.

When a user loads the tool, he can check in the server by opening a new account. He inserts the name of the user and a password and then he presses the button "create new account". If he has an account already, he should get in the application as a registered user. The options selected is then "Login". The application accesses the service profile and loads the profile that was sent back.
by the server. From that moment on, the profile search is made in the local cache but, if it is not found, the request is made to the agent in charge of the “match machine”. Taking the option of anonymous user, the function of profile cache is unauthorized and, therefore, it cannot use its functionalities. The agent in charge of the search profile manages any query made.

The implementation of the profile server was made with J2EE technology. Three servlets were implemented for these basic needs: creating new user accounts and loading and storing profiles remotely. The exchange of information between clients and server is made through the SOAP protocol.

OntoServices was used in the edition process of semantic web services for a multi-agent system prototype for incidents traffic information.

7. Conclusions

The present article presents a summary of the state of the art of those tools that allow a description of semantic web services. From the analysis made, a tool was designed and implemented so as to combine the visualization of concept ontologies with the creation of semantic web services profiles. The use of a cache that stores the latest queries (user profiles), increasing the system speed since the users will probably search a reduced number of types of services and so the searching will not be always necessary. The result that the tool generated was used with the multiagent system for listings semantic web services developed in the research group at the University of Valencia. Nowadays, the research group is testing the prototype in real systems belong to Spanish traffic administrations for a future integration with additional issues.

6. Acknowledgments

We would like to thank to Spanish Ministry of Science and Technology. This research has been supported by the CICYT project with reference TRA2004-06276 MODAL: “Development of a system using a conceptual infrastructure based on ontologies, to exchange good information between trucks and external entities”.

7. References


[18] WebODE Available on the Internet at http://doc.is.in.tum.de/webODE/