BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **5**, No 2

Sofia • 2005

Semantically Annotating Web Services Using WSMO Technologies*

Ivo Marinchev, Gennady Agre

Institute of Information Technologies, 1113 Sofia

Abstract: *In this paper the differences and relationships between regular web services and semantic web services are discussed. Later on a "bottom-up" approach is introduced for converting existing web services to semantic web services using the emerging WSMO technologies (specifications). We firmly believe that bottom-up approaches are needed to facilitate smooth transition from the existing syntactically defined web services to their future semantically enriched "counterparts". We present also our preliminary tool for semi-automated conversion of WSDL to WSMO services.*

Keywords**:** *Web Services, Semantic Web Services, Web Service Modeling Ontology (WSMO), WSDL, OWL, OWL-S.*

Introduction

Semantic Web technologies describe functional and behavioral aspects of Web services, and their inputs and outputs in terms of ontologies as concepts, their instances and relations among them, with a great portion of the meaning of the data explicit in its ontology. From this point of view, Semantic Web Services and their clients are software agents that produce and consume semantic data.

In order for Semantic Web Services (SWS) to be executed by computer systems and SWS infrastructure (tools) to co-operate with regular web services, a mapping between semantic data and regular web service syntaxes is needed. There are multiple ways (syntaxes) of representing these semantic data on the wire for exchange. The most useful ones are these based on XML syntaxes. This mapping is on two levels – schema mapping and instance mapping.

- Schema mapping is needed in the process of design and composition of Semantic Web Services. It is a process of finding / building correspondence between data types presented in the WSDL file and WSMO ontology concepts that describe the service domain.

* This work has been partially supported by INFRAWEBS - IST FP62003/IST/2.3.2.3 research project No 511723 and by project IIT-010061 "Technologies of the Information Society for Knowledge Processing and Management".

• Instance mapping is needed in the process of execution when the concrete messages are interchanged between the semantic and non-semantic part of the system. Instance mapping is the process of constructing rules for automatic conversion between XML instance documents and ontology instances.

As shown in the WSMO Grounding document [3], there is no direct mapping between an XML vocabulary and WSMO ontology, mostly because relations can be represented in multiple ways in XML, or they can even be implied. XML Schema specifies structure of the documents. At the same time ontology languages provide a formal specification of a shared domain theory. They model semantic relationships in a particular domain. As a result, it is required a human operator to create the mapping. However we think that these problems are more severe on the theoretical level when any possible situations are considered. Some of them are even algorithmically intractable. But in practice the XML Schemas and WSDL files are not so diverse and do not employ all possible syntaxes. Hence the custom (task specific) approaches are feasible.

Basics of XML, XML schema and WSDL

In the paper presented it is required that the reader is familiar with the XML, XML Schema and WSDL specifications. XML is a standard language for describing document types in any domain, facilitating the sharing of data across different systems in the Internet [3]. XML is flexible and extensible allowing users to create their own tags to match their own specific requirements. XML Schema is a W3C Recommendation defining a schema language for XML. XML Schema provides a way to define constraints on the syntax and structure of an XML document [2].

Currently, Web Services are described with the WSDL [1] documents/files. WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. Web services usually communicate with their clients (consumers) using XML messages whose structure is described using XML Schema [2], usually embedded in a Web Services Description Language [1] document.

The INFRAWEBS Grounding Editor uses the information from the Types, Messages and Operations sections (Actually PortType section is also in use but only for enumerating the services). The essential part of the work is done on the information in the Types and Messages sections and in the current version of the Grounding Editor converts the definitions in these sections to their semantic counterparts (according WSMO specifications) in respect to a given set of general domain ontologies.

Basics of WSMO Ontology

WSMO identifies four top-level elements as the main concepts, which have to be described in order to describe Semantic Web services (R o m a n et a l., [8]):

• **Ontologies** provide the terminology used by other WSMO elements to describe the relevant aspects of the domains;

• **Web services** describe the computational entity providing access to services that provide some value in a domain. These descriptions comprise the capabilities,

interfaces and internal working of the Web service. All these aspects of a Web Service are described using the terminology defined by the ontologies.

- **Goals** represent user's wishes, for which fulfillment could be sought by executing a Web service.
- **Mediators** describe elements that overcome interoperability problems between different WSMO elements.

In the paragraphs below we provide some short introduction to WSMO elements that we need and use in the current document. For the complete specification of WSMO the reader have to refer to (R o m a n et al., [8]).

The basic blocks of WSMO Ontology are concepts, relations, functions, instances, and axioms.

Concepts are defined by their subsumption hierarchy and their attributes, including range specification. The range of the attributes can be a datatype or another concept. There are two kinds of attribute definitions: constraining definitions, using the keyword ofType and inferring definitions using the keyword impliesType. In the first case, the values of the attribute are constrained to having the mentioned type, while in the latter, the values of the attribute are inferred to have the mentioned type

concept person
id **ofType** xsd:positiveInteger
name **ofType** xsd:string
family **ofType** xsd:string
birthDate **ofType** xsd:date

Relations describe interdependencies between a set of parameters. Optionally, the domain of parameters, which can be a datatype or a certain concept, can be specified using the **impliesType** or **ofType** keywords.

Functions are a special type of relations that have an unary range beside the set of parameters. Although in the conceptual model, they are regarded as entities distinct from relations, in WSML they are modelled as a special type of relations that have one functional parameter.

Instances are embodiments of concepts or relations, being defined either explicitly by specifying concrete values for attributes or parameters or by a link to an instance store.

instance Person1 **memberOf** person
id **hasValue** 2984845
name **hasValue** "John"
family **hasValue** "Dow"
birthDate **hasValue** '1970-06-06'

Axioms are specified as logic expressions and help to formalize domain specific knowledge. Different WSML variants allow different expressivities for the logical expressions (d e B r u i j n et al., [10]). Axioms are declared by using the keyword **axiom** optionally followed by the axiom identifier, by a set of non-functional properties and by the logical expression introduced by the keyword **definedBy**.

axiom myIntegerConstraint
definedBy ?X **ofType** xsd:positiveInteger and ?X>=10000 and ?X <= 99999

Overview of existing approaches

At the moment only one completed similar approach of grounding semantic web services exists. It is presented in [4]. An OWL-S/WSDL grounding is done by creating an instance of the OWL-S Grounding class, which includes all required information regarding the relationships between the relevant OWL-S constructs and the relevant WSDL constructs. Although Grounding instance has all required information, WSDL documents are slightly modified (with the help of extensibility elements) to give some indicators how the WSDL constructs are used to ground OWL-S. For more complex cases OWL-S contains *xsltTransformation* property, which may be used to express more complex mappings between WSDL message parts and atomic process inputs/outputs.

When using OWL-S with WSDL, it is possible to declare OWL classes and properties within the types section. When a types section is used in this way, the extensibility element looks like this:

```
<definitions .... >
  <types>
    <rdf:RDF namespace-declarations ... > .... </rdf:RDF>
  </types>
</definitions>
```

where the "..." within the extensibility element may be replaced by any number of OWL declarations.

Messages consist of one or more logical parts. Each part is associated with a type from some type system using a message-typing attribute. The set of message-typing attributes is extensible. WSDL defines several such message-typing attributes for use with XSD:

- **element** – refers to an XSD element using a QName.
- **type** – refers to an XSD simpleType or complexType using a QName.

Other message-typing attributes may be defined as long as they use a namespace different from that of WSDL. Binding extensibility elements may also use message-typing attributes. For example:

```
<definitions .... >
   <message name="Message1">
     <part name="Part1" owl-s-parameter="ourNS:Param1"/>
     <part name="Part2" owl-s-parameter="ourNS:Param2"/>
  </message>
</definitions>
```

A way is needed to indicate the correspondence between a particular WSDL operation, and an OWL-S AtomicProcess. WSDL offers no extensibility elements for operations. For this purpose, the authors of OWL-S/WSDL grounding propose that WSDL sanction an optional "owl-s-process" attribute for the WSDL operation element, as illustrated in the example below:

```
<operation name="Operation1" owl-s-process="ourNS:AtomicProcess1">
```

It is necessary to mention here that in case we need it, we will adopt the idea of such additional attributes from OWL-S grounding. More exactly, we will introduce the analogous optional attributes "wsmo-parameter" and "wsmo-process". Although this is not a standard approach we need to introduce this "home-made" solution because

it is simple, efficient, easier to understand and non-intrusive. At the same time the current WSMO approach to SWS grounding does not propose any solution so we are free to adopt whatever we think is appropriate.

The latest working draft on WSMO grounding [4] enumerates the following three possible approaches:

1. Create mappings at the conceptual (WSMO) level involving creating WSMO ontology for the XML Schema used in the WSDL.
2. Use XSLT to create direct mapping between XML and the XML syntax of the WSMO ontology.
3. Use a direct mapping between the source XML data and the target WSMO ontology, using a mapping language specifically developed for this purpose.

The second and third approaches are not feasible since the mappings would take place only on the syntactic level and there would be no possibility to use reasoning to provide a more sophisticated mapping in the second one. Another language invented specifically for the transformation is also needed in the third case.

The first approach requires three distinct activities to ground the data part of WSMO service descriptions to the XML Schema used in WSDL:

- Define a mapping between the XML Schema Conceptual Model (XML Schema) to the WSMO Ontology metamodel.
- Create an executable description of the mappings in the first point to enable the automatic creation of ad-hoc WSMO ontologies from specific XML Schema.
- Create the bi-directional mappings rules to be used for the transformation between XML instances and WSMO instances. These mapping rules should be created at the same time as the generation of the ad-hoc WSMO ontology from an XML Schema. The creation of these mapping rules should be automatic, as they should be completely derived from the actions described in the first two bullet points.

In the INFRAWEBS Grounding Editor we follow the above ideas although there are some differences that stem from the specific problems that appear in the practical application of this approach or when some missing parts of the specification have to be completed.

The approach of service grounding in INFRAWEBS

The INFRAWEBS Designer is a tool aiming at converting an existing WSDL-based service to a semantic WSMO-based one. So to ground such a semantic service we need to perform only XML Schema to WSMO ontology mapping. As the WSDL files contain descriptions of services implemented in some of the modern programming languages, the schemas found in the WSDL files actually represent the data types used by the services described by set of SimpleType and ComplexType declarations.

Since in most cases concrete services do not use the data types that correspond exactly to ontologies concepts, as a result of the mapping process a new service specific ontology will be built.

The INFRAWEBS “bottom-up” approach to service grounding can be summarized as follows:

1. Lifting XML Schema to “ad hoc” WSML ontology. Simple and complex data-types found in the WSDL file are converted to a service-specific (“ad hoc”) WSML

ontology. This step is basically a conversion of the data-types hierarchy to concept hierarchy.

2. Mediation between the constructed ad-hoc WSML ontology and given general-purpose WSML ontologies. At this step the user mediates the concepts of the ad hoc ontology to the closest possible concepts of the available general-purpose ontologies. In its turn this second step is implemented according to the following algorithm:

- Map to existing ontology concepts wherever it is possible.
- In case the exact matching could not be found, use the closest "counterpart" concept from some of the domain ontologies and refine it (create a sub-concept of it) by the addition of missing attributes and/or axioms.

The step of finding the best ontology "counterpart" can not be done completely automatic. There are some approaches for automatic detection of the closest ontology concepts but most of them attain limited success (accuracy of 20% – 30%) or are applied in very limited domains (P a t i l et al., [6]).

That is why we decided to use semi-automatic approach: the user is the one that makes explicit mapping using the tools for reducing the set of possible candidates for mapping. Such "candidates" are automatically proposed based on similarity between the data structures to be compared as well as on performing type checks. To make the work easier for the user the tool follows "bottom-up" mapping approach – from simple to complex data types mapping.

As a result our tool allows semi-automatic conversion process, in which the user evaluates similarity between pair of concepts based on their meanings and the tool assists him by presenting on the screen only relevant information and performing type checks.

Lifting XML Schema to ad hoc WSML ontology

In this section we represent our approach to converting the different type of structures (types) contained in XML schemas that is a part of WSDL files to the service-specific (ad hoc) WSML ontology concepts. The initial version of our software tool uses conversion (lifting) rules defined by us because at the corresponding moment (June/ July 2005) the lifting problem was not addressed in the WSMO grounding specifications [3]. The corresponding rules was implemented in Java programming language and was intended as a temporary solution until the specification catches up with us. The latest version of WSMO grounding [3] has introduced very good lifting specification. This specification is based on mapping rules that are in fact declarative form of syntactical transformation needed in order to convert XML Schema to WSMO Ontology. The only drawback in their approach that we see is that they are focused on WSML version 2.0 that is still in working draft stage and all available web services and the corresponding tools work with WSDL version 1.1. At the same time in our project we must take into account exiting web services infrastructure and that's why we decided to modify the rules so that they can be used with the WSDL version 1.1 files. Fortunately this task does not appear to be very hard as the differences between the two versions of the WSDL files are mainly syntactic. As additional flexibility the new version of our tool uses XSLT transformation rules to implement the lifting process so that we can easily support different version of WSDL and/or WSMO syntaxes as they evolve over the time.

Mediation between ad-hoc WSMO ontology and general-purpose WSML ontologies

The next step is mediation between ad hoc WSMO ontology and general-purpose ontologies that represent the problem domain. The result of the mediation process is the creation of the so-called “mapping ontology” that consists of service specific refinement of the concepts of the general ontologies. Actually the mapping ontology is an extension of the ad hoc ontology and is created from it by the procedure described below:

For any concept found in the ad hoc ontology:

1. If that concept corresponds exactly to the concept of a certain general ontology then it is mapped to this general concept. Mapping here means that in the mapping ontology a new concept is created that is exact copy of the general concept (has the same name and the same attributes and axioms).

2. If exact correspondence is not available then the closest concepts from the upper ontologies are selected and the mapping is done to one of them. The closest concepts are all of the concepts that has less or equal number of attributes that correspond to the subset of the attributes of the “ad hoc” concept. When the “ad hoc” concept is mapped to a closest general concept a new sub-concept of the general concept is created in the mapping ontology that has the same name as the original concept and all of the attributes from the “ad-hoc” concept that are missing in the general concept are added to it (to the mapping concept).

3. In the worst case when there is no appropriate “closest concept” completely new (is not a sub-concept of existing general concept) concept is created in the mapping ontology that corresponds exactly to the one in the “ad-hoc” ontology. In fact this step is analogous to the previous one with selected “the closest concept” Thing (the most general concept of the WSMO ontologies).

As we mentioned above the result of the mediation process is a small custom ontology (mapping ontology) that is service specific but at the same time is closely related to the existing domain ontologies. We think that the creation of this service specific ontology can not be avoided, as one can not model precisely a certain web service using only the concepts from the general ontologies. For example let we consider the credit card processing service. The general concept creditCardProcessor will have an attribute creditCardNumber that can be for example of type xsd:string. But our concrete service can be restricted to process only VISA and MasterCard cards. Hence in our service specific ontology (mapping ontology) we have to create the new concept ourCreditCardProcessor that is a sub-concept of the general credit-CardProcessor concept that have additional axiom restricting the creditCardNumber attribute to accept the numbers of the VISA and MC cards only.

Grounding editor architecture and interface

From a theoretic point of view our grounding editor implements the two step “lifting-mediation” process described in the previous sections. However, it is not efficient in practice the grounding process to be implemented in a straight-forward manner using the above receipt. There are several reasons (criteria) that have to be considered:

1. The end user is not required to be familiar neither with the grounding specifications nor with the specific terms as “lifting”, “mediation”, “refinement”, etc.

2. Lifting (converting) the XML Schema to “ad hoc” ontology is not feasible to be performed in a bulk as the user is sometimes required to check some of the conversions or to add axioms manually.

3. Often not all of the XML Schema types that are available in the WSDL file are needed for the semantic service. Usually the user will map only some of them that are required for the corresponding semantic service. It is often the case that WSDL files contain not used or legacy constructs that are kept in them only for backward compatibility with previous versions of the same service.

4. As the user assistance is mandatory on the mediation step the information on the screen have to be presented in a logical consequence as the user have to guess the semantic meaning of the message parts based only on the operation name, message name, type name, surrounding context, and optionally end user documentation of the web service.

According to the above requirements in our implementation of the grounding editor we do not follow blindly the two step process but inter-mix the two steps in one interface operation. This way from the end user point of view it looks like the mapping is done directly between the WSDL structures (messages and types) and the concepts of the general ontologies.

In order to simplify additionally the process of types mapping and to make it straightforward we decided to enforce the bottom-up approach to type-concept mapping. In this context bottom-up mapping means that the user can not map a certain type until all of its elements are mapped. In the terms of the first example (personType) of the previous section it means that the user will have to map the name and family elements (actually the full names personType.name and personType.family) to a certain concepts (for example humanName) and then it will be allowed to map the complex type personType.

Internally the tool starts with generation of unique type names based on the names found in the WSDL file. All types that are “global” in the WSDL file have unique names. Here we refer to the names of the XML Schema elements that are “local”. For example elements of the complex types or complex types that are defined inside another complex type. Two elements that are part of two different complex types can have the same name but this fact does not guarantee that they have the same meaning (semantics). In order to distinguish them we construct unique names of the elements by creating the “full names” of the elements – the names that are prefixed with the names of their corresponding enclosing types. For example the full name of the element named “page” (Fig. 3 below) in the complex type SellerProfileRequest is SellerProfileRequest.page and the full name of the element named “page” in the complex type SellerRequest is SellerRequest.page.

In fact these full names are used internally by the tool and probably will never be exposed to the end user, as they are not convenient for him. The context of the elements in presented to the end user with the help of tree structures. At the same time these names are needed for the implementation of instance mapping during run-time. At this stage of development it is not completely clear how to handle the above problem in the best way, so it will be solved when we start integrating the different components of the INFRAWEBS framework and the WSMO execution environment (WSMX).

Fig. 1 depicts the general architecture of the INFRAWEBS Grounding Editor as a components diagram.

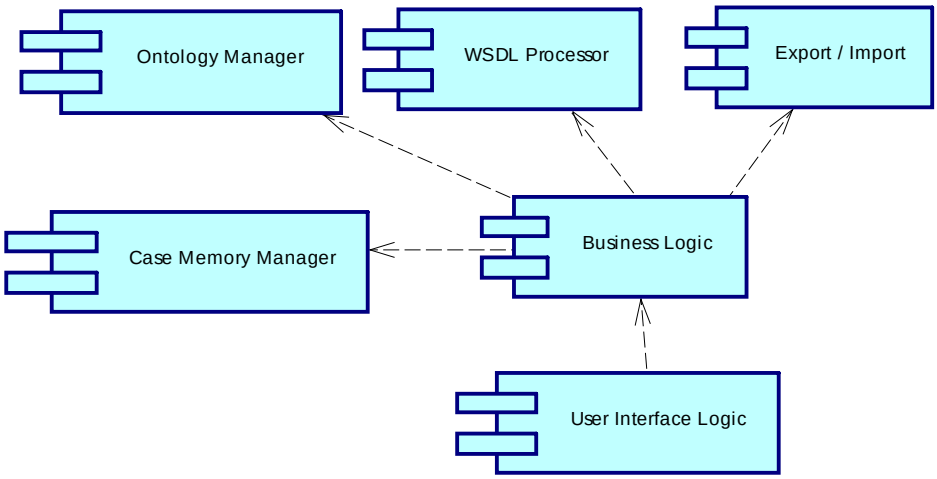

Fig. 1. Component diagram of INFRAWEBS Grounding Editor

Our tool follows “bottom-up” approach in type-concept mapping (from simpler types to more complex ones). The user is not allowed to map a certain type if any of it elements (and sub-types) is not already mapped/annotated. So the actual work process is as follows: The user selects the type that she/he wants to map/annotate. If there is any element (sub-types, and elements of sub-types at any level) that is not annotated the user is warned about it and is sent to this sub-type to map it. Otherwise it allows the annotation (see the next step). This approach not only simplifies and unifies the annotation process from the user’s point of view but, more importantly, it allows any mapping-time checks to be performed automatically and helps the user to avoid the mapping incompatible types. In fact in the complex real-world ontologies without such computer aided mapping it is almost impossible for the user to notice many of the potential problems beforehand as a certain concepts of the ontology may inherit many different attributes and restrictions from its direct and indirect super-concepts.

Fig. 2 is a screenshot of the graphical user interface of the current version of the INFRAWEBS Grounding Editor. We have to explicitly mention here that this is not the final interface but just some sort of mock-up. In the later versions the grounding editor will be integrated with the axiom editor as they share many common components as ontology manager, case memory manager and probably others.

On the left side the structure of a real web service is presented. In fact this is an old version of the Amazon Web Service. Top left tree depicts the operations found in this WSDL file and their input, output, and fault messages. Below it is the tree that depicts the structure of the selected message. On the right side is the so-called “ontology pane”. This is the graphical user interface of the ontology manager component. It will be replaced (shared) with the one presented in the current version of the axiom editor as it has a lot of functionality that is already implemented. The tree in the middle represents a version of the ontology view that show only the “closest possible concepts” (in the terms discussed in the mediation section) that correspond (are compatible) to

the data-type (build in, simple, or complex) of the WSDL file selected in the left middle tree. To help the user in the process of type – sub-concept mapping our tool selects (filters) only the concepts from the opened ontologies that contain less or equal number of attributes to the number of the elements of the selected XML Schema type. Fig. 3 shows the original ontology (on the right) and the filtered one (on the left) when the user selects a type that has no sub-elements (simple type). In this case all concepts that have attributes are skipped. When the appropriate concept is selected the user clicks the annotate button (we will probably rename it in the next versions) and the mediation step is performed internally? i.e. the tool creates a new sub-concept to the selected concept and if the schema type has more elements that the concept attributes it creates the new attributes that correspond to these extra elements. As the attributes are global in respect to WSMO ontologies we use internally the unique names discussed above. As a logical consequence to the above, in the future versions of our tool all restrictions found in the XML Schema will be converted to axioms (WSMO axioms). At the moment the axiom language is still not specified formally in the WSMO working drafts. That is why we decided to postpone the implementation of this step for now. The next version will have on this place a text editor or directly will be integrated to the axiom editor that will allow the user to manually convert the XML Schema restrictions to WSMO axioms.

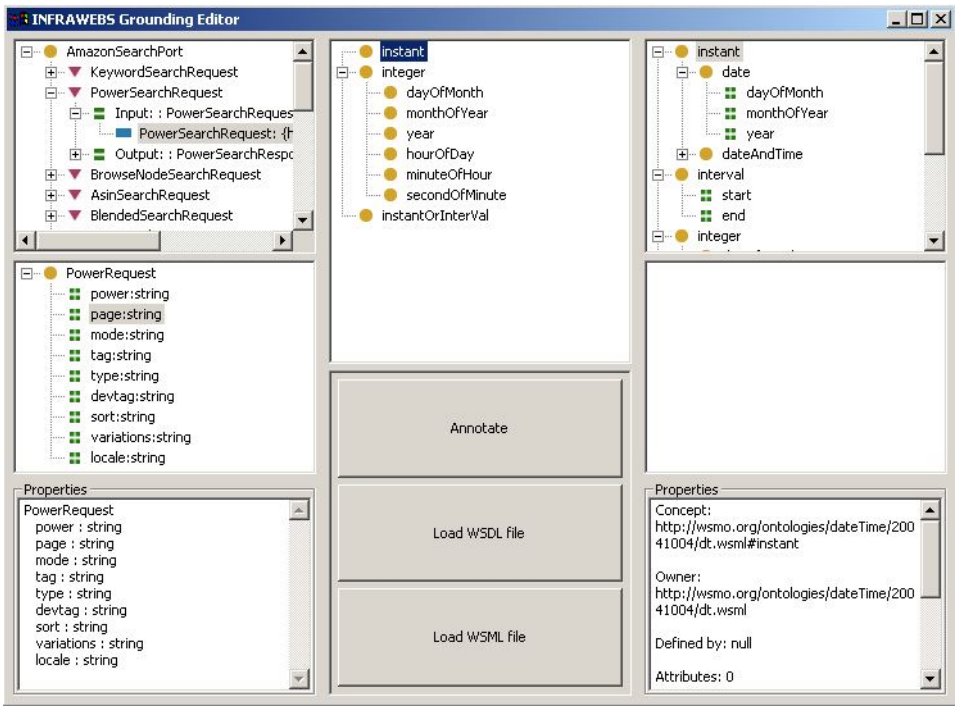

Fig. 2. INFRAWEBS Grounding Editor tool – WSDL file on the left, WSML ontology on the right

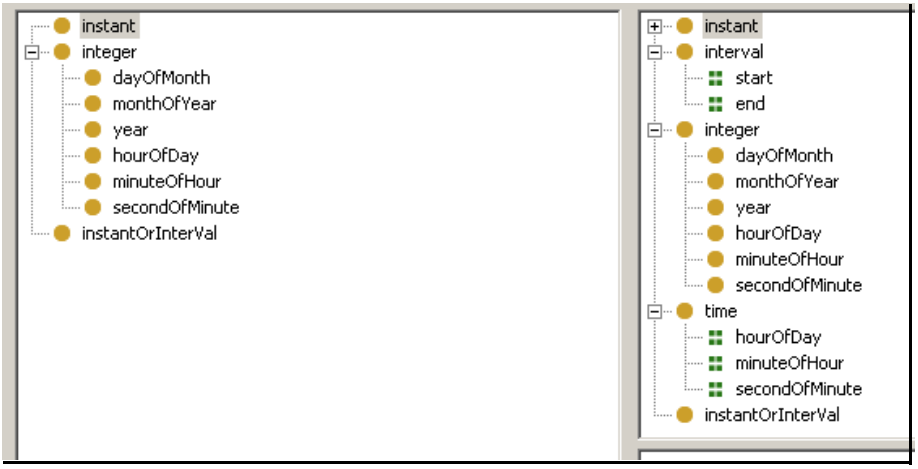

Fig. 3. An ontology (on the right) and the filtered one (on the left) when the user selects a type that has no sub-elements (simple type)

When the WSDL document messages and types are mapped (annotated) to the semantic concepts the tool creates internally a table of correspondences between the variable names and their corresponding syntactic type (XML Schema) and semantic type (WSMO concept). This step is needed as the semantic types will be used during discovery and composition (and probably by the semantic executors) but the syntactic types are used at run-time. We intended to follow the approach proposed in the recent versions of the WSMO grounding [3] that is based on the introduction of new non-functional properties. If it appears that the WSDL files have to be annotated as well and this is not specified in the WSMO working drafts at the time the need arises we can use the approach similar to the one employed by the OWL-S grounding [4]. It is based on the extensibility elements. The extensibility elements of WSDL are used for a straightforward means of combining WSDL with any other XML-based language. For example reusing the OWL-S approach the correspondence between the syntactic and semantic names of the message parts and operation names can be done as follows:

<part name="BookName" wsmo-wsdl:wsmo-variable="serviceOntology:BookName"/>
<operation name="BuyBook" wsmo-wsdl:wsmo-service="serviceOntology:CongoBuy">

Conclusions

The work presented in this paper is work in progress. It is not completed but outlines the general directions of solving the problem of making semantic and syntactic web service worlds to coexist and leverage each other's strengths. Given the incompleteness of the WSMO specifications and the introduction of some very complicated approaches in it we developed custom approach for converting existing web services to semantic ones. Our solution is simple enough to be used by the people with only superficial knowledge of semantic technologies and at the same time it is general enough to be employed in most of the practical situation.

References

1. WSDL 1.1 specification
   http://www.w3.org/TR/2001/NOTE-wsdl-20010315/
2. H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, Eds. XML Schema part 1: Structures, W3C Recommendation, 2001.
   http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
3. Working Draft D24.2v0.1.
   http://wsmo.org/TR/d24/d24.2/v0.1/
4. Martin, D., M. Burstein, O. Lassila, M. Paolucci, T. Payne, S. McIlraith. Describing Web Services using OWL-S and WSDL.
   http://www.daml.org/services/owl-s/1.1/owl-s-wsdl.html
5. Web Service Modeling Ontology (WSMO).
   http://www.wsmo.org
6. Patil, A., S. Oundhakar, A. Sheth, K. Verma. METEOR-S web service annotation framework. – In: Proc. of the 13th conference on World Wide Web, July 2004.
7. Nern, H.- Joachim, G. Agre, T. Atanassova, J. Saarela. System framework for generating open development platforms for web-service applications using semantic web technologies, distributed decision support units and multi-agent systems – In: INFRAWEBS II. - Trans. on Information Science and Applications, Issue 1, Vol. **1**, July 2004, 286-291.
8. Roman, D., H. Lausen, U. Keller, J. de Bruijn, Ch. Bussler, J. Domingue, D. Fensel, M. Kifer, J. Kopecky, R. Lara, E. Oren, A. Polleres, M. Stollberg. D2v1.1. Web Service Modeling Ontology (WSMO) – WSMO Final Draft, 10 February 2005.
   http://www.wsmo.org/TR/d2/v1.1/
9. Roman, D., J. Sciclunna, C. Feier, M. Stollberg, D. Fensel. D14v0.1. Ontology-based Choreography and Orchestration of WSMO Services, WSMO Final Draft 1 March 2005.
   http://www.wsmo.org/TR/d14/v0.1/
10. De Bruijn, D., H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, D. Fensel. D16.1: The Web Service Modeling Language WSML. WSML final draft, DERI, March 2005.
    http://www.wsmo.org/TR/d16/d16.1/v0.2/