WSDL RDF Mapping: Developing Ontologies From Standardized XML Languages

Jacek Kopecký

DERI Innsbruck
Technikerstrasse 21a
6020 Innsbruck, Austria
jacek.kopecky@deri.org

Abstract. The W3C is working on a Recommendation for Web Services Description Language, WSDL 2.0, based on XML. The Working Group is chartered to produce a normative mapping of WSDL into RDF, effectively a WSDL ontology. While this work is still ongoing, in this paper we propose guidelines for creating such ontologies coming from XML standards.

1 Introduction

In the World Wide Web Consortium (W3C), the Web Service Description Working Group[1] has been working on version 2.0 of the popular Web service description language WSDL [8]. This XML language allows Web service providers to describe the functional interface of the service, that is, the types and sequences of input and output messages.

In order to add WSDL data to the Semantic Web, the working group was chartered to produce an RDF mapping for the XML-based WSDL documents. This mapping specification has two main parts: a WSDL ontology and a formal mapping from WSDL documents to RDF data that uses that ontology. The author of this paper is the main editor of the WSDL RDF mapping specification [2].

While working on this specification, we have learned several lessons. For example, our initial draft of the WSDL ontology closely followed the formalization of WSDL, the so-called *component model*, and was largely rejected by Semantic Web practicioners[2] for its complexity and opacity. The current version tries to convey the intention of WSDL in what should be a much better RDF form.

In this paper, we describe the WSDL RDF mapping effort and propose a number of guidelines for creating ontologies for standard XML-based vocabularies. In our experience, XML-based standards share an interesting characteristic:

[1] http://www.w3.org/2002/ws/desc/

[2] The initial draft was presented to the Semantic Web Interest Group of W3C at its meeting in Boston, March 2005.

while the XML language itself is usually fairly user-friendly, it is often backed by a strong and detailed formalization. The intricacies of the language are only grasped by experts, yet many non-experts will use the language interoperably only thanks to an intuitive understanding of the syntax.

We can demonstrate this characteristic on XML Schema [1]. In limited forms it is used and even implemented in most systems working with XML data, yet complete and correct support is lacking, and cooperating systems often have conflicting limitations. The W3C has started a working group to document XML Schema Patterns for Databinding[3] in an attempt to specify an interoperable subset of XML Schema that would be supported reliably in heterogeneous systems.

We argue that when mapping an XML language to RDF, attempting to capture all the details of the formalization is counterproductive. Ontology engineering methodologies (cf. [5]) use the notion of *minimal ontological commitment* to limit the level of constraints that are expressed in an ontology, and we argue that ontologies based on XML languages need to aim for even lower explicit ontological commitment.

In the following sections, we first give an overview of WSDL in Section 2, and we describe the draft WSDL ontology in Section 3. In Section 4 we propose our guidelines for working on standards-based ontologies, followed by showing how the proposed guidelines apply to our draft WSDL ontology in Section 5. Finally we discuss related work in Section 6 and we conclude the paper in Section 7.

2 WSDL overview

WSDL 2.0 is a language for describing Web services. In particular, it can describe the structure of the messages the service accepts and produces, simple message exchanges (called operations) and all necessary networking details. On top of this, extensions in WSDL documents can specify that additional features are supported or even required by the service. In effect, WSDL specifies a limited contract that the service adheres to.

The WSDL specification is written in terms of a *component model*, presenting any WSDL document as a set of components with local properties. In the core, WSDL contains a fairly simple set of components: the top-level Description component represents a WSDL document together with all imported and included documents. Every Description may contain Interface, Binding and Service components[4].

An Interface component describes the abstract interface of a Web service—the operations, messages and faults. An operation is represented with an Interface Operation component, which has a property {message exchange pattern}

[3] http://www.w3.org/2002/ws/databinding/

[4] Formally, the Description component contains properties {interfaces}, {bindings} and {services}, each of which is a set of the respective Interface, Binding and Service components. We use the notation {property name} to indicate component properties, as the WSDL specification does the same.

to point to the message exchange pattern (MEP) that this operation follows. An MEP in WSDL prescribes the number and directionality of messages, and the operation populates them with concrete XML elements in its Interface Message Reference components. Each Interface Message Reference represents a single message, refers to the appropriate MEP message and contains an {element declaration} property which points to a schema. Most MEPs also allow fault messages for expected application-level errors. Faults are modeled on the same level as operations, that is, an interface defines a number of faults (Interface Fault components), which are then used within operations (with Interface Fault Reference components).

An interface is specified on the level of XML messages; the networking details about how the messages are represented on the wire are specified in the next top-level WSDL component—the Binding. The Binding component follows the structure of Interface and uses extensions to specify any protocols and networking parameters.

The last top-level WSDL component — Service — provides a number of endpoints where a service is available. And endpoint specifies an actual address, together with a Binding that indicates how the client should communicate with that address.

Figure 1 shows a WSDL document (adopted from the WSDL Primer [4]) that describes a simple hotel service. This interface provides a single operation for checking the availability of rooms for a given date. Abstractly (i.e., on the Interface level), the operation accepts the dates of check-in and check-out and the required type of room, and it returns the daily rate of available rooms, or zero if nothing is available. In case the room type or input dates are erroneous, a fault can be generated. The Binding level specifies that the data will be transmitted with SOAP over HTTP, plus several other details. Finally the Service level gives one endpoint address, at which the hotel service is available.

WSDL is, by design, a very extensible language, and in fact some parts of the standard are built as predefined extensions (see [3]). There are three distinct kinds of extensibility in WSDL: extension points, Features and Properties, and generic XML-based extensions.

Extension points are those places in WSDL descriptions where a number of options is defined by WSDL, but the list is open; for example, every WSDL operation follows some message exchange pattern, and while WSDL provides a list of eight predefined patterns, WSDL users are free to define new ones, if necessary. Extension points have specific and non-overlapping scopes, for example the mentioned MEPs guide the ordering of messages, *operation styles* generally restrict message schemas, and *binding types* specify the networking details with different underlying protocols.

Features and Properties form a specific extensibility mechanism in WSDL. A Feature is an abstract piece of functionality (e.g. making communication confidential) and a Property is a concrete parameter of that functionality (e.g. the required strength of confidentiality). Every component in WSDL may *offer* or *require* the use of specific Features. For example, many services may be able to

```
<description targetNamespace="http://hotel.example.com/wsdl" ...>
  <types>
    <xs:schema ...>
      <xs:element name="checkAvailability"
                  type="tCheckAvailability"/>
      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element name="checkInDate" type="xs:date"/>
          <xs:element name="checkOutDate" type="xs:date"/>
          <xs:element name="roomType" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="checkAvailabilityResponse"
                  type="xs:double"/>
      <xs:element name="invalidDataError"
                  type="xs:string"/>
    </xs:schema>
  </types>

  <interface name="hotelIface">
    <fault name="invalidDataFault"
           element="ghns:invalidDataError"/>
    <operation name="checkAvailability"
            pattern="http://www.w3.org/2006/01/wsdl/in-out">
        <input element="ghns:checkAvailability"/>
        <output element="ghns:checkAvailabilityResponse"/>
        <outfault ref="tns:invalidDataFault"/>
    </operation>
  </interface>

  <binding name="hotelSOAPBinding"
     interface="tns:hotelIface"
     type="http://www.w3.org/2006/01/wsdl/soap"
     s:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
    <fault ref="tns:invalidDataFault"
           s:code="soap:Sender"/>
    <operation ref="tns:checkAvailability"
      s:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
  </binding>

  <service name="hotelService"
       interface="tns:hotelIface">
     <endpoint name="hotelEndpoint"
           binding="tns:hotelSOAPBinding"
           address="http://hotel.example.com/service"/>
  </service>
</description>
```

Fig. 1. Example hotel service description

use message encryption if the client wishes it (these services will offer communication confidentiality feature), but some will require the client to encrypt all messages to the service. Similarly, every component may restrict the possible values of a Property, either to a single value or to a restricted set of values. In a continuation of the confidentiality example, a service may list the supported encryption algorithms, by constraining the appropriate Property.

Apart from the scoped extension points and the abstract Features and Properties, WSDL is open to XML-based extensibility, i.e. any WSDL element can contain any number of attributes or elements from a foreign (non-WSDL) namespace. Such extensions are not constrained in what they may mean. In general, extensions add properties to the existing WSDL components, so that the processor can use the extended information. As we cannot expect all WSDL processors to know all the extensions they might encounter, extension elements in WSDL are by default *optional*, but they may be marked *mandatory*. Optional extensions can be ignored by a processor that does not recognize them, whereas mandatory extensions must be understood by a processor that needs to process the parent WSDL element. More significantly, mandatory extensions are even allowed to change the meaning of the parent WSDL element.

WSDL is specified as an extensible component model as opposed to an XML grammar in order to abstract away, among others, many ambiguity issues of XML serialization and document includes and imports. Apart from the actual specification prose, the component model of WSDL is additionally defined using the Z notation [7], which allows consistency and coverage verification. In particular, the Z notation was used to help ensure that the test suite covers all important WSDL component model constraints, and it can be used for validation of WSDL documents.

The Z notation is also useful to us when we work on the WSDL ontology; it serves as a concise listing of all the components, their properties and all the component model constraints from WSDL.

3 WSDL ontology

This section describes the WSDL ontology defined in the WSDL RDF mapping specification [2], highlighting the main differences between the component model and the ontology structure. The ontology is defined in OWL [6], but due to the layering of the Semantic Web languages, we expect that WSDL documents mapped into this ontology will be useful even to generic RDF processors that do not support OWL inference.

The core components of WSDL are mapped to corresponding classes with matching names, that is, the ontology defines classes `Description`, `Interface`, `Binding` and `Service` etc. Instances of these classes are connected using properties whose names are the same as those of their range classes, only beginning with lower case, so for instance the property `interface` links a concrete instance of `Description` with the concrete instance of `Interface` that it contains.

```
@prefix rwsdl: <http://www.w3.org/2005/10/wsdl-rdf#> .
@prefix rsoap: <http://www.w3.org/2005/08/wsdl/soap#> .

<http://hotel.example.com/wsdl#wsdl.description()>
   a rwsdl:Description ;
   rwsdl:interface <http://hotel.example.com/wsdl#wsdl.interface(hotelIface)> ;
   rwsdl:binding <http://hotel.example.com/wsdl#wsdl.binding(hotelSOAPBinding)> ;
   rwsdl:service <http://hotel.example.com/wsdl#wsdl.service(hotelService)> .

<http://hotel.example.com/wsdl#wsdl.interface(hotelIface)>
   a rwsdl:Interface ;
   rwsdl:interfaceFault
      <http://hotel.example.com/wsdl#wsdl.interfaceFault(hotelIface/invalidDataFault)> ;
   rwsdl:interfaceOperation
      <http://hotel.example.com/wsdl#wsdl.interfaceOperation(hotelIface/checkAvailability)> .
<http://hotel.example.com/wsdl#wsdl.interfaceFault(hotelIface/invalidDataFault)>
   a rwsdl:InterfaceFault .
<http://hotel.example.com/wsdl#wsdl.interfaceOperation(hotelIface/checkAvailability)>
   a rwsdl:InterfaceOperation ;
   rwsdl:messageExchangePattern <http://www.w3.org/2006/01/wsdl/in-out> ;
   rwsdl:interfaceMessageReference
      <http://hotel.example.com/wsdl#wsdl.interfaceMessageReference
                                          (hotelIface/checkAvailability/In)> .

<http://hotel.example.com/wsdl#wsdl.binding(hotelSOAPBinding)>
   a rwsdl:Binding ;
   a <http://www.w3.org/2006/01/wsdl/soap>;
   rwsdl:interface <http://hotel.example.com/wsdl#wsdl.interface(hotelIface)> ;
   rsoap:protocol <http://www.w3.org/2003/05/soap/bindings/HTTP> .

<http://hotel.example.com/wsdl#wsdl.service(hotelService)>
   a rwsdl:Service ;
   rwsdl:interface <http://hotel.example.com/wsdl#wsdl.interface(hotelIface)> ;
   rwsdl:endpoint <http://hotel.example.com/wsdl#wsdl.endpoint(hotelService/hotelEndpoint)> .
<http://hotel.example.com/wsdl#wsdl.endpoint(hotelService/hotelEndpoint)>
   a rwsdl:Endpoint ;
   rwsdl:binding <http://hotel.example.com/wsdl#wsdl.binding(hotelSOAPBinding)>;
   rwsdl:address <http://hotel.example.com/service> .
```

Fig. 2. An excerpt of the RDF form of the example WSDL from Figure 1

In the component model of WSDL, components are usually identified by their type, their name, potentially the names of the components in the owner hierarchy, and with the namespace of the defining WSDL file. The example WSDL document in Figure 1 contains the Interface component named `hotelIface` in the namespace `http://hotel. example.com/wsdl`. For identifying the components with URIs, as is required by RDF, the WSDL specification provides a set of URI references formed from the namespace, component type and all the relevant names. The example interface is identified as `http://hotel.example.com/wsdl #wsdl.interface(hotelIface)`. For more information on WSDL component URIs see appendices A.2 and C in [8]. It is important to note here that the ontology only uses these composite component identifiers—the names and namespaces are not modeled separately, but they can be reconstructed from the URI references, if necessary.

Figure 2 shows the N3 form of an excerpt of the RDF form of the core components from Figure 1. A graphical representation of the same triples is in

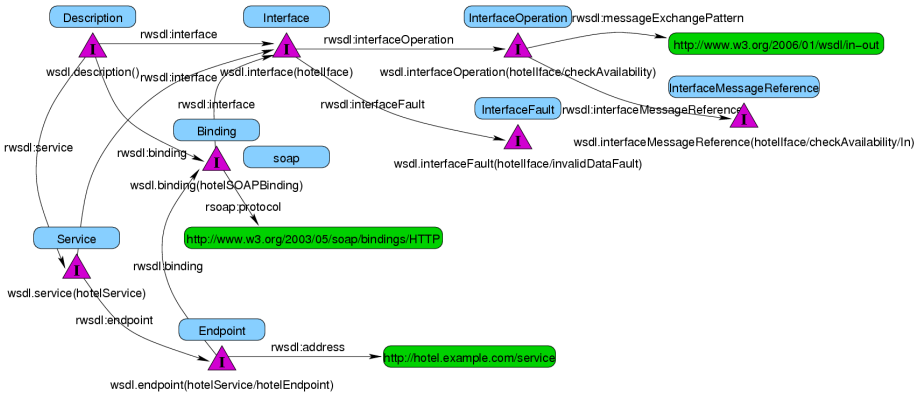

Fig. 3. Graph of the RDF triples from Figure 2

Figure 3. For conciseness and readability we only show a limited part of the RDF data.

Because major pieces of WSDL are specified as standard extensions, the RDF mapping also defines how these predefined extensions are represented in RDF. For example, the RDF ontology contains a class of *operation styles* along with the property `operationStyle` for pointing to particular instances of the class, like the three styles defined by the standard: RPC, IRI and Multipart.

Both standard WSDL bindings (SOAP and HTTP) are also extensions that define additional properties for WSDL components, and these properties are translated into RDF in a straightforward manner.[5] The two RDF figures show how the extensibility attribute `s:protocol` (specifying the underlying protocol that carries the SOAP messages) is mapped into a simple RDF property `protocol`.

Feature components, which serve in WSDL to point to actual features, are skipped in the RDF form; instead the required features are indicated using the property `requiresFeature` and optional ones are indicated using the property `offersFeature`, pointing directly to the feature URI. Here the RDF mapping differs from the component model, as in the component model a Feature component can contain documentation or extensions which apply to the particular instance of use of the feature, not to the feature itself—with our mapping the documentation or extensions are lost. *Property* components, however, are modeled as instances of the class `PropertyValue`, because in the component model, each Property component actually specifies a constraint on the values of the property in the scope of a particular service, binding or interface.

[5] Unfortunately the word *property* is overloaded here: the component-model properties from WSDL are represented as RDF properties; and soon the Property components come into play as well.

```
<wsdl:binding name="hotelSOAPBinding" ...>
  <ns:ext xmlns:ns='http://example.com/'/>
  ...
</wsdl:binding>
```

```
@prefix rwsdl: <http://www.w3.org/2005/10/wsdl-rdf#> .
<http://hotel.example.com/wsdl#wsdl.binding(hotelSOAPBinding)>
    a rwsdl:Binding ;
    rwsdl:extensionElement
        "<ns:ext xmlns:ns='http://example.com/'/>" .
```

Fig. 4. An example WSDL binding with an unknown optional extension, its RDF form in N3

As we already mentioned, WSDL allows arbitrary XML element and attribute extensions. All such extensions should also define how they are represented in RDF, as the HTTP and SOAP bindings do, for example. As WSDL explicitly allows unknown extensions, we can also expect that WSDL documents with unknown extensions may be mapped into RDF. For this case, the mapping has a fallback mechanism for dealing with unknown extensions. Optional unknown extensions are represented in RDF using an XML literal (for extension elements) or an instance with a QName and a literal value (for extension attributes), and the parent components point to these extensions using the properties `extensionElement` and `extensionAttribute`. An example with an unknown extension element is shown in Figure 4. Mandatory extensions, however, may change the meaning of their parent components, therefore the RDF mapping explicitly skips any components that contain mandatory unknown extensions, as the actual meaning of the component must be treated as unknown as well.

To summarize, the RDF mapping is mostly straightforward, mapping components to class instances and component properties to RDF properties, but there are several deviations; notably in component identification, where the RDF uses URIs that conflate the usual QNames of components. Another interesting difference between the component model and the ontology is that a few components (Feature and HTTP Header components) are not present in the RDF form at all, replaced with direct links that should improve the usability of the ontology.

We are currently working on an XSLT stylesheet that will transform WSDL files into the appropriate RDF data. Our stylesheet cannot handle extensions not defined in the WSDL 2.0 specifications, and it has only limited support for imports and includes, therefore we do not plan to make this stylesheet a normative reference implementation.

4 Guidelines for standards-based ontologies

Based on our experience with the RDF mapping of WSDL, we propose the following two guidelines for creators of ontologies for standardized XML data languages:

1. the ontology should follow the *intent* of the language, not necessarily any particular formalization of it,
2. the ontology need not model all data constraints present in the underlying standardized language,

The subsections below explain the guidelines in more detail.

4.1 Intent, not formalization

Because we assume that standardized XML languages will have some kind of underlying formalization, it is tempting to translate such a formalization directly into an ontology, given that ontological data is intended for automated computer consumption. We suggest, however, that the ontology should eschew artifacts of the formalization, like component sets and the strict distinction between components and their properties in WSDL.

The rationale for this point lies in the differences in expressivity between the underlying language formalization model and the ontology model. The formalization model is often simple (like WSDL components with properties), whereas ontology models (for example RDF/OWL) can natively capture relationships like class membership and sub-class relations, and this power should be used.

For instance, we modeled properties like binding type or message direction as class membership (rdf:type), i.e., binding extensions (HTTP, SOAP or any new ones) are classes, and a given binding component (`<wsdl:binding>`) is an instance of one of these classes. Further, some enumerated properties were combined with container relationships, for example in the WSDL component model, a component can contain a `feature` which is required or optional, and in the ontology the component *offers* or *requires* the feature.

4.2 Not all constraints

Most languages, not only standardized ones, put many constraints on the data. The constraints can be either explicitly spelled out or implied by the structure of the language. In WSDL, for example, every Service component points to an Interface (explicit constraint), and an Interface Operation can only be defined as part of an Interface (implicit, structural constraint).

Not all of these constraints need to be modeled in the ontology. Language constraints are formalized for validation, so that documents can be automatically checked for correctness. But the mapping to ontological data is expected to work with valid documents, so any RDF graph resulting from the mapping of

a valid WSDL file will, by definition, be valid, rendering constraints like OWL cardinalities or even property ranges and domains redundant.

One could assume that all the language constraints should be present in the ontology in order to avoid unintended interpretations of the data; this is part of the principle of minimal ontological commitment. This principle is very useful if the ontological data stands alone; however when creating an RDF mapping for an XML-based language, the RDF data is effectively a read-only RDF view on the original XML data. XML is proven to be an interoperable basis for machine-processible languages, therefore if an application needs to check or access all the intricacies of the language, it should use the XML form.

The RDF form is intended for combining with other ontological data in the Semantic Web. Here, modeling all the available XML constraints could in fact be harmful. For instance, in the WSDL component model, a single Description component contains all the other components, however when we combine information from multiple (unrelated) WSDL documents, we can find one interface reused in multiple places. This is a real scenario that the WSDL specification considers out of its scope (in order to simplify the language).

In other words, when the RDF data is going to be backed by the original XML form, the ontology may beneficially drop some of the XML language constraints, so that the RDF data can better play its role in the Semantic Web. While ontologies are often viewed as complete domain models, we argue that ontologies for XML standards need not be complete in this regard.

5 WSDL RDF mapping case study

Having formulated the guidelines in the previous section, we can now re-apply them to the WSDL RDF mapping. Even though the guidelines stem from our RDF mapping experience, it is useful to revisit them in an iterative process and show where they have helped, and whether they might help even more.

As a good example of overformalization (Guideline 1), our initial draft modeled a Description component with a single property `interfaces` which pointed to a set containing Interface components (description hasInterfaces X; X contains interface1 etc.), which is how the component model is specified. The current draft takes advantage of the RDF set-of-triples model and simply uses multiple `interface` properties on Description, removing the indirection through an explicit set, making the data easier to use.

The current version of the WSDL ontology still contains some cardinality constraints. As discussed in Guideline 2, this may be redundant, making the ontology more complex than is really necessary. We have already identified some of them to be removed in the next version, as direct benefit of having the guideline written down in this paper, yet some of the cardinality constraints seem to remain useful even in the RDF data. In particular, some properties have local cardinality restricted to 1, indicating that the property will always be present, simplifying the use of the ontology.

6 Related work

Even though ontologies have been created for pre-existing standards like RosettaNet and eCl@ss, we are not aware of any directly related work suggesting a methodology for creating ontologies for reuse of formalized XML languages. However, in the general scope of ontology engineering methodologies, there is some overlap and even conflict with our first two suggestions.

The first guideline could be rephrased as "the existing formalization most probably wasn't intended for reuse and inference, so it may be improper as the basis for an ontology." While some could consider it unnecessary effort, we suggest that redoing the conceptualization and formalization of the underlying language, with focus on ontological modeling, will yield positive results.

The second guideline seems to be contrary to conventional ontological engineering where more information in the ontology is beneficial, but we favor ontology simplicity (increasing the chances of reuse) to complete inference power for the constraints, especially since we assume the constraints are enforced in validation before a piece of data is transformed into the ontological form.

7 Conclusions

In this paper, we described our work on the RDF mapping of the Web Service Description Language (WSDL), and we share the guidelines that we learned from this work, for the benefit of others who map standardized languages into an ontological form, which need not necessarily be limited to RDF/OWL.

Standardized languages share one specific property: they are commonly based on some formalization intended to limit or eliminate ambiguities in interpretation. While very beneficial for ensuring interoperability, such formalizations are not necessarily very convenient for working with actual data, therefore our first guideline suggests that the ontology should focus on the intent of the language, reusing the appropriate constructs available in the ontology language (e.g. multiple class membership and subclassing) even if the underlying formalization does not make use of such constructs.

The second guideling proposes that the ontology need not repeat all the data constraints of the standardized language, because such constraints are in fact enforced before a document is mapped into the ontological form, and may in fact be harmful when combining data from multiple documents.

Even though our work on WSDL RDF mapping is not finished yet, we believe our proposed guidelines can already be useful. If our guidelines are followed, it should be easier to seed the Semantic Web with useful data coming from existing standard languages.

8 Acknowledgements

This work is partially funded by the European Commission under the project DIP.

References

1. XML Schema Part 1: Structures. Recommendation, W3C, October 2004. Available at http://www.w3.org/TR/xmlschema-1/.
2. Web Services Description Language (WSDL) Version 2.0: RDF Mapping. Working Draft, W3C, November 2005. Available at http://www.w3.org/TR/2005/WD-wsdl20-rdf-20051104/.
3. Web Services Description Language (WSDL) Version 2.0: Adjuncts. Candidate Recommendation, W3C, January 2006. Available at http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060106/.
4. Web Services Description Language (WSDL) Version 2.0: Primer. Candidate Recommendation, W3C, January 2006. Available at http://www.w3.org/TR/2006/CR-wsdl20-primer-20060106/.
5. M. Fernandez-Lopez, A. Gomez-Perez, and N. Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In *Workshop on Ontological Engineering, Spring Symposium Series, AAAI97, Stanford, USA*.
6. D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Recommendation 10 February 2004, W3C, 2004. Available at http://www.w3.org/TR/owl-features/.
7. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 1992.
8. Web Services Description Language (WSDL) Version 2.0. Candidate Recommendation, W3C, January 2006. Available at http://www.w3.org/TR/2006/CR-wsdl20-20060106/.