Photorealistic Semantic Web Service Groundings

Unifying RESTful and XML-RPC Groundings Using Rules, with an Application to Flickr

Dave Lambert and John Domingue

Knowledge Media Institute
The Open University
Milton Keynes, United Kingdom
d.j.lambert@open.ac.uk

Abstract. Semantic Web services achieve effects in the world through Web services, so the mechanism connecting the ontological representations of services with the on-the-wire messages—the grounding—is of paramount importance. The conventional approach to grounding is to use XML-based translations between ontologies and the SOAP message formats of the services, but these mappings cannot address the growing number of non-SOAP services, and step outside the ontological world to describe the mapping. We present an approach which draws the service's interface into the ontology: we define ontology objects which represent the whole HTTP message, and use backward-chaining rules to translate between semantic service invocation instances and the HTTP messages passed to and from the service. We show how this approach can be used to access the Flickr photo-sharing service through both its RESTful and XML-RPC interfaces.

1 Introduction

The field of semantic Web services uses ontologies to formally model the purpose and operation of Web services such that they can be intelligently used by machines. A crucial part of this is modelling how the Web service is invoked: the message format, and application protocol usage. The major semantic services frameworks—OWL-S [1], WSMO [2], and SA-WSDL [3]—assume services will be implemented using SOAP, and implemented brokers use XML mapping languages to translate between the XML serialisation of the ontology data and the on-the-wire messages exchanged with the Web service. This approach solves only the *data grounding* problem, and even then, solves it only for SOAP services.

Recent trends in Web services have shown that many developers, both on the client and server side, prefer to use techniques other than SOAP or even XML. In particular, XML-RPC is a popular lightweight alternative to SOAP which still uses XML, while RESTful interfaces [4] often eschew XML formats altogether in favour of JSON, or multimedia MIME types. An approach based on SOAP, and therefore XML, does not translate easily to non-SOAP flavours of Web service (Section 2).

In a previous paper [5] we introduced a method where the grounding was described entirely using an ontology language's frames and rules, and demonstrated how it could access Amazon's Simple Storage Service through its RESTful interface. In that paper we made but did not substantiate the claim that our approach could be extended to support other message formats, in particular XML-RPC and SOAP. In the current paper, we show how our technique can be used for XML-RPC. After recapping our approach (Section 3), we develop an ontological model for XML documents and XML-RPC messages (Section 4). We proceed to demonstrate upon Flickr—a popular picture sharing service, which happens to provide interfaces for REST, XML-RPC, and SOAP—presenting descriptions for some of Flickr's XML-RPC and RESTful services (Section 5). We compare our method to the usual approaches (Section 6) and conclude (Section 7).

2 Grounding Web Services

The objective of semantic Web services is to formally model the operation of Web services, so that intelligent agents can reason about them. A key part of the reasoning is concerned with the mechanism of service invocation. A semantic service broker must be able to convert an abstract 'invocation' expressed in terms of ontologies into the correct sequence of bytes sent to the correct network address to cause the service to operate, and to interpret the service's response. The process of translating between the ontological world of domain theories and the on-the-wire data formats and protocols is known as 'grounding' the service. For semantic Web services to be practical, brokers must be able to ground a comprehensive collection of real, actively used Web services.

For most of the lifetime of semantic Web services research, the de facto flavour of Web service was defined by XML [6], SOAP [7], and WSDL [8], which grew into the W3C's Web Services stack, colloquially known as 'WS-*'. As the complexity of the WS-* stack has increased, its popularity has waned, with many services now being offered using lighter weight alternatives. The genuinely simple protocol which inspired SOAP has re-emerged in its own right as XML-RPC [9]. More visibly, REST [4] has gained considerable mind-share: according to Amazon's Web services evangelist Jeff Barr, around 80% of invocations of Amazon's services are made through the REST interface.[1] Yahoo! does not provide a SOAP interface, and has no intention of adding one.[2] Flickr, a popular photo-sharing website, offers its API in SOAP, XML-RPC, and RESTful flavours. The SOAP interface does not have a WSDL description, and none of the the third-party bindings[3] for the most popular languages target the SOAP variant:

[1] http://www.jeff-barr.com/?p=96
[2] http://developer.yahoo.com/faq/#soap
[3] http://www.flickr.com/services/api/

Flickr Binding	*Language*	*API*	*Flickr Binding*	*Language*	*API*
Flickcurl	C	REST	Flickr-Upload	Perl	REST
flickrj	Java	REST	phpFlickr	PHP	REST
jickr	Java	REST	flickr.py	Python	REST
FlickrNet	.NET	REST	flickr-ruby	Ruby	REST
Flickr-API	Perl	REST	rflickr	Ruby	XML-RPC

In most cases, the Flickr user is ultimately motivated by the prospect of obtaining a picture. Flickr's API provides only a RESTful means for retrieving the image, requiring the construction of the image's URL in one of several forms, the following being the simplest and most restrictive:

```
http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg
```

The `farm-id`, `server-id`, `id`, and `secret` parameters are part of a photo's identity, and can be obtained by calling one of several other Flickr services. There is no SOAP mechanism for retrieving images, and no way to couch the URL creation as an XML transformation. Although WSDL 2 [8] and WADL [10] can form URLs based on templates, many of the URLs used in RESTful interfaces are much too complicated for this. For example, the Flickr URL shortening service (similar to the TinyURL service) produces URLs of the form:

```
http://flic.kr/p/{base58-photo-id}
```

where `base58-photo-id` is an algorithmically specified mapping from a photo's details to a base 58 encoded string. RESTful authentication mechanisms also typically produce URLs which are impossible to define with a simple substitution template. A general grounding mechanism must not only support the XML that underlies SOAP, but also describe complex URL schemes, and at least handle, if not manipulate, the multimedia data that constitutes much of the data in the HTTP stream.

3 Ground Rules

We previously introduced our approach to grounding which we believe achieves these objectives [5], and in this section we quickly review the technique. The scheme has been implemented in the Internet Reasoning Service (IRS),[4] a broker based on the Web Services Modelling Ontology (WSMO) [11]. The IRS uses OCML [12] as its knowledge representation language and reasoner. OCML is a frame language with a Lisp syntax and procedural attachment, and is comparable in expressiveness to the Ontolingua and Loom languages. Although we implmented our approach in OCML, the technique is broadly applicable to KR languages with backward-chaining rules.

Our method is to model in ontologies the actual HTTP messages sent and received—including the URL, headers, and the content—and use rules in the ontology language to manage the mapping between those messages and the service invocation objects that the broker deals with anyway in managing the invocation. At a high level, the process of invoking a semantic Web service is:

[4] `http://technologies.kmi.open.ac.uk/irs/`

1. A user invokes a semantic service by calling the broker with a goal description described ontologically.
2. After some processing by mediators, an ontological Web service invocation instance is created. The invocation object holds the service name, input parameters, and slots to hold the return values from the Web service.
3. A rule from the service's semantic description is called to create an HTTP message object based on the service invocation object, with its various slots' values set to reflect the parameters from the the service invocation object.
4. The HTTP message is passed to the broker, which then turns the HTTP object directly into a request on the network.

When the service replies, the same procedure is preformed in reverse. The novelty of our scheme lies in step 3. We define two entry points, or generic rules, which we call `lower` and `lift`. These respectively 'lower' the service request object to an implementation level and 'lift' it back. Concretely, the two rule heads are

```
(lower ?serviceType ?serviceInvocation ?httpRequest)
(lift ?serviceType ?serviceInvocation ?httpResponse)
```

Each Web service description can define its own version of `lift` and `lower`, which the broker distinguishes by unifying on the `?serviceType` parameter that names the service being invoked. The `lower` rule's successful fulfilment leads to the instantiation of `?httpRequest`, which can then be directly interpreted by the broker to call the Web service. When a response is received from the server, the `lift` rule runs on the the newly returned `?httpResponse`, modifying the original `?serviceInvocation` frame to record the return values. The HTTP ontology defines the general form of the HTTP messages in a simple way:

```
(def-class HttpMessage ()
  ((hasHeader :type HttpHeader)
   (hasContent :type String :max-cardinality 1)))
```

with `HttpRequest` and `HttpResponse` subclassed from `HttpMessage`. Our grounding ontologies are relatively simple, being built solely for the purpose of supporting groundings, but they could in principle be general purpose ontologies developed for other uses in the respective domains.

As an example, consider the operation to retrieve an image (Section 2). Recall that this consists of performing an HTTP `GET` operation on a URL of the form:

```
http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg
```

If an invocation of that service had a `hasPhoto` slot which held the identifying features of the picture, the `lower` rule could be written thus:

```
(def-rule lowerGetFlickrImageService
  ((lower GetFlickrImageService ?invocation ?httpRequest) if
   (hasPhoto ?invocation ?photo)
   (hasFarmId ?photo ?farmId)
   (hasServerId ?photo ?serverId)
   (hasId ?photo ?id)
   (hasSecret ?photo ?secret)
   (= ?httpRequest (new HttpRequest))
   (= ?url (concatenate "http://farm" ?farmId ".static.flickr.com/"
                        ?serverId "/" ?id "_" ?secret ".jpg"))
   (assert (hasUrl ?httpRequest ?url))
   (assert (hasMethod ?httpRequest "GET"))))
```

We use `asserts` here because it is OCML's 'house style' as a mutation-oriented frame language, but we believe the rules would be clearer still in an immutable language. Once `lowerGetFlickrImageService` has succeeded, `?httpRequest` is bound to an instance of `HTTPRequest` whose method field is `GET`, and the URI field is the correctly constructed location of the photo resource.

4 Working with XML and XML-RPC

In many cases, the content of the HTTP messages will be XML, in particular, XML-RPC or SOAP messages. In our lift and lower rules, we could directly manipulate the string representations of XML, but this becomes cumbersome, prone to error, and fails to 'model' in any meaningful way the transformations. Instead, we introduce simple ontologisations of XML and XML-RPC, and using these we can then write simple lifting and lowering rules.

Our ontologies are straightforward. XML is modelled with a `Document` concept, which has a `rootElement` of type `Element`. In turn, `Element`s have children which are `Element`s or `Text`, and they also have lists of `Attribute`s. Such structures can be transformed into their corresponding strings, and back again, with the relation `serialiseXml`. The essentials are shown in Figure 1.

```
(def-class Document ()
  ((rootElement :type Element)))

(def-class Element ()
  ((tag :type string)
   (attributes :type Attributes)
   (contents :type Contents)))

(def-class Attributes () ?attributes
  :iff-def
  (and (listp ?attributes)
       (every ?attributes Attribute)))
```

```
(def-class Attribute ()
  ((name :type string)
   (value :type string)))

(def-class Contents () ?contents
  :iff-def
  (and (listp ?contents)
       (every ?contents
              (or Element Text))))

(def-class Text ()
  ((value :type string)))
```

Fig. 1. XML ontologisation in OCML.

An XML-RPC [9] message is a simple serialisation of a remote procedure call, for example:

```
<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
      <param><value><i4>41</i4></value></param>
   </params>
</methodCall>
```

The content of an XML-RPC service invocation is a `MethodCall`, comprising a `MethodName` and a list of parameters. Each parameter `Param` has a typed value, which may be a simple scalar, an array, or a structure. These are represented by matching concepts in the ontology, shown in Figure 2.

```
(def-class MethodCall ()
    ((methodName :type MethodName
                 :cardinality 1)
     (params :type ListOfParam)))

(def-class ListOfParam () ?list
  :iff-def (and (listp ?list)
                (every ?list Param)))

(def-class MethodName ()
    ((value :type string)))

(def-class Params ())

(def-class Param ()
    ((value :type Value)))

(def-class Value ())
```

```
(def-class scalarValue (Value))

(def-class I4 (scalarValue)
    ((value :type integer)))

(def-class Boolean (scalarValue)
    ((value :type boolean)))

(def-class String (scalarValue)
    ((value :type string)))

(def-class Struct (Param)
    ())

(def-class Member ()
    ((name :type string)
     (value :type Param)))
```

Fig. 2. XML-RPC ontologisation in OCML (an illustrative subset only).

The relation `xmlrpcToXml` deals with mapping from the XML-RPC ontological objects to XML ontological objects, which can then be serialised to a string of XML with `serialiseXml`.

5 Flickr

We now illustrate the use of rules to create groundings for both RESTful and XML-RPC interfaces to Flickr services. Flickr is a commerical website offering 'freemium' hosting of users' photographs, combined with social-networking facilities for the structured sharing of those images. Flickr provides a sizable API to examine the metadata around the images themselves. We have already seen how an image can be retrieved from Flickr (Section 3), but to do that, we must discover the details about an image required to construct the URL.

We will skip the intricacies of creating an account, setting up API keys and the like, although we have create groundings for those services, too. For this paper, our concern is with *a*) getting a list of recently changed photos in a user's account *b*) getting a list of sizes in which those are available . The relevant services are `flickr.photos.recentlyUpdated` and `flickr.photos.getSizes`. The Flickr API provides interfaces for SOAP, XML-RPC, and REST, but for all three, services are defined in terms of a Flickr-specific abstraction for passing arguments. Each method takes a set of name/value pairs, and this is then cryptographically signed to ensure security. The `flickr.photos.recentlyUpdated` method takes three arguments, `api_key`, `auth_token`, and `min_date`, while `flickr.photos.getSizes` takes arguments `api_key`, `auth_token`, and `photo_id`. The `api_key` identifies the application making the request, and the `auth_token` is obtained when a user grants a particular application access to their account. Figure 3 shows the rule for signing an argument set with an account key, which amounts to taking the MD5 sum of a string which is the concatenation of the application's 'secret' key, and the name/value pairs of the arguments (this ordering is arranged by

the `canonicalArgumentsString` relation, which for brevity is not shown). The `signArguments` rule is used by lowering rules where required.

```
(def-rule signArguments
    ((signArguments ?flavour ?arguments ?account) if
     (hasSecret ?account ?secret)
     (hasValue ?secret ?secret-string)
     (canonicalArgumentsString ?flavour ?arguments ?canonical-args)
     (= ?to-be-signed (concatenate ?secret-string ?canonical-args))
     (= ?signature (md5sum ?to-be-signed))
     (addArgument ?args "api_sig" ?signature)))
```

Fig. 3. Signing a call.

```
(def-rule lower-for-photosRecentlyUpdatedRestService
    ((lower photosRecentlyUpdatedRestService ?invocation ?http-request) if
     (= ?account (wsmo-role-value ?invocation hasAccount))
     (= ?token (wsmo-role-value ?invocation hasToken))
     (= ?min-date (wsmo-role-value ?invocation hasMinimumDate))
     (hasKey ?account ?apikey)
     (hasValue ?apikey ?apikey-string)
     (hasValue ?token ?token-string)
     (= ?args (new-instance Arguments))
     (addArgument ?args "method" "flickr.photos.recentlyUpdated")
     (addArgument ?args "api_key" ?apikey-string)
     (addArgument ?args "auth_token" ?token-string)
     (addArgument ?args "min_date" ?min-date)
     (signArguments rest ?args ?account)
     (argsToRestRequest ?args ?http-request)))
```

Fig. 4. Lowering rule for RESTful `photos.recentlyUpdated`.

The lowering rule for the RESTful `flickr.recentlyUpdated` service is shown in Figure 4. The rule is straightforward, with the first half concerned with extracting from the `?invocation` the necessary argument values, and the second half constructing an `Arguments` instance using those values. The last two lines ensure the arguments are signed, and then pack the arguments into an HTTP request. The conversion from an argument set to an HTTP request is handled by the `argsToRestRequest` rule, shown in Figure 5.

```
(def-rule argsToRestRequest
  ((argsToRestRequest ?arguments ?http-request) if
   (= ?args (setofall ?pairs
                      (and (hasArgument ?arguments ?argument)
                           (hasName ?argument ?name)
                           (hasValue ?argument ?value)
                           (= ?pairs (?name ?value)))))
   (listAsQuery ?args ?query)
   (= ?url (concatenate "http://api.flickr.com/services/rest/?" ?query))
   (rfc2616:set-url ?http-request ?url)
   (rfc2616:set-method ?http-request "GET")))
```

Fig. 5. Translation from an argument set to a RESTful HTTP GET message.

`argsToRestRequest` constructs a URL with the arguments in the form

```
param1=value1&param2=value2...
```

and sets the `?http-request` fields appropriately. The lifting rule is shown in Figure 6.

```
(def-rule lift-for-photosRecentlyUpdatedRestService
    ((lift photosRecentlyUpdatedRestService ?http-response ?invocation) if
     (rfc2616:get-content ?http-response ?http-content)
     (xml:serialiseXml ?xml ?http-content)
     (xml:rootElement ?xml ?root-element)
     (xml:elementByName ?root-element ?photos-element "photos")
     (extractXmlPhotoList ?photos-element ?photolist)
     (set-goal-slot-value ?invocation hasPhotoList ?photolist)))
```

Fig. 6. Lifting rule for RESTful `photos.recentlyUpdated`.

In lifting the response, we extract the content from the `?http-response`, convert it into an ontological model of the XML, and use a rule `extractXmlPhotoList` (not shown) to extract the appropriate fields and build an ontological list of the contents. With this list of photographs, we are now ready to invoke the second service, `flickr.photos.getSizes`. We will invoke this service in XML-RPC.

```
(def-rule lower-for-photosGetSizesXmlrpcService
    ((lower photosGetSizesXmlrpcService ?invocation ?http-request) if
     (argsForPhotosGetSizes ?invocation ?args)
     (= ?account (wsmo-role-value ?invocation hasAccount))
     (signArguments xmlrpc ?args ?account)
     (argsToXmlrpcRequest ?args ?http-request)))
```

Fig. 7. Lowering rule for `flickr.photos.getSizes`.

Figure 7 shows the lowering rule for this XML-RPC service. This time, the conversion from the invocation object to the argument pairs used by Flickr is done

in another rule `argsForPhotosGetSizes` (not shown). The use of a rule for this means we could share the logic between the XML-RPC version shown here, and a REST or SOAP version. The argument set is again signed using `signArguments`, and then passed to a new rule `argsToXmlrpcRequest`, shown in Figure 8.

```
(def-rule argsToXmlrpcRequest
    ((argsToXmlrpcRequest ?args ?http-request) if
     (getArgument ?args "method" ?method)
     (= ?nonmethodargs
        (setofall ?member
                  (and (hasArgument ?args ?arg)
                       (hasName ?arg ?name)
                       (not (= ?name "method"))
                       (hasValue ?arg ?value)
                       (= ?member (xmlrpc:Member ?name
                                                 (xmlrpc:String ?value))))))
     (= ?xmlrpc
        (xmlrpc:MethodCall ?method
                           (xmlrpc:Param (xmlrpc:Struct ?nonmethodargs))))
     (xmlrpc:mapToXml ?xmlrpc ?xmlmodel)
     (xml:serialiseXml ?xmlmodel ?xmlstring)
     (rfc2616:set-content ?http-request ?xmlstring)
     (rfc2616:set-method ?http-request "POST")
     (rfc2616:set-url ?http-request
                      "http://api.flickr.com/services/xmlrpc/")))
```

Fig. 8. Translation from argument set to an XML-RPC message.

In `argsToXmlrpcRequest` we see similar machinary to that in `argsToRestRequest` for converting the argument set, but this time we create an XML-RPC message with a `Struct` to hold the pairs, rather than embedding them in a URL.

6 Related Work

WSDL [13] has been the de facto means of specifying Web service interfaces since the birth of Web services. Both OWL-S [1] and WSMO [11] define their groundings by pointing at the WSDL of their targets, but the mapping to the syntactic content of the messages is something of a grey area. The OWL-S WSDL document [14] suggests that OWL-S services should require Web services to use an OWL specific encoding in their implementation. The semantic annotation extensions for WSDL—WSDL-S and then SA-WSDL (Semantic Annotations for Web Service Description Language) [3]—provide a vocabulary to link the WSDL descriptions to mapping schemas to handle the lifting and lowering, but the mechanism of the schemas themselves is not specified, and is XML-centric. The IRS previously used XPath expressions to generate OCML relations which performed the lifting and lowering. Another WSMO based broker, the Web Services Execution Environment (WSMX) uses service-specific 'adaptors', written in Java, to connect to services.

Although the principle of 'lifting and lowering' the XML serialisation is well established, it does not address aspects of the HTTP protocol like the

`Authorization` header that many services require for authentication. Moreover, assuming XML translation precludes the use of services that do not employ XML at all. Although WSDL and SOAP are products of the W3C standards, there is significant disquiet amongst developers about their complexity, interoperability, and the way they ignore the Web's architecture. Personal experience has made us skeptical of the quality of WSDL and XSD descriptions, even, or perhaps especially, machine generated ones. Finally, using an XML mapping scheme like XSLT forces the ontology engineer to leave the semantic realm to work on the groundings, and to consider the domain objects in terms of their XML serialisation. In contrast, our groundings unify the lifting and lowering with the management of the HTTP protocol, and are declarative and wholly within the ontology language, modulo the small number of operational primitives such as cryptography functions. Since there are relatively few data encoding and cryptography schemes—many orders of magnitude fewer than there will be Web services—it makes sense to embed them in semantic brokers, and make them available to rules at the ontology level. Such facilities will need to be present one way or another: in our scheme, they are available as reusable primitives, and the number of them therefore kept to a minimum. With these hooks in place, we can encode groundings to a large number of important, real-world Web services in a unified, ontology-based manner.

7 Conclusion

Semantic Web services are about Web services as well as semantics. In this paper, we extended an approach to groundings rooted in ontologies and rules, adding support for XML content and XML-RPC Web services. As an illustration, we described some of the services provided by Flickr, the popular photo sharing site. We have implemented the ontologies discussed in the IRS broker.

We see this approach as a useful low-level implementation platform: it is sufficiently powerful to connect to any kind of HTTP service, and yet is fully accessible from the ontological level. It is general enough, for instance, that it could support multi-part MIME messages. The resulting rules are simple from an engineering perspective, allowing easy reuse of component rules, and declarative description of the necessary operations. For the Flickr services described here, a handful of rules capture the general invocation pattern, and lifting or lowering rules for individual services are compact. Future work will involve a translation to RDF, reusing vocabularies like the W3C's RDF schema for HTTP.[5] We also intend to support SOAP messages directly at the ontological level, in a similar fashion to the XML-RPC support developed here. We intend to create ontologies to model WSDL descriptions, and from there to permit some level of automatic lifting and lowering based on the available information from the WSDL. The same could be done for WADL, and SA-WSDL. Since other semantic Web services work has established XSLT as a means for performing these translations, we should be able to reuse existing XSLT mappings within rules when available.

[5] `http://www.w3.org/TR/HTTP-in-RDF/`

Acknowledgements

This research was funded by the European Union projects Living Human Digital Library (FP6-026932) and SOA4All (FP7-215219). We thank Marta Sabou and Barry Norton for interesting discussions.

References

1. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services. W3C member submission, World Wide Web Consortium (W3C) (November 2004)
2. Lausen, H., Polleres, A., Roman, D.: Web Service Modeling Ontology (WSMO). W3C member submission, World Wide Web Consortium (W3C) (June 2005)
3. Farrell, J., Lausen, H.: Semantic Annotations for WSDL and XML Schema. W3C recommendation, World Wide Web Consortium (W3C) (August 2007)
4. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
5. Lambert, D., Domingue, J.: Grounding semantic web services with rules. In Gangemi, A., Keizer, J., Presutti, V., Stoermer, H., eds.: Proceedings of the 5th Workshop on Semantic Web Applications and Perspectives (SWAP2008). Volume 426 of CEUR Workshop Proceedings. (December 2008)
6. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible Markup Language (XML) 1.0. W3C recommendation, World Wide Web Consortium (W3C) (2008)
7. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S., Winer, D.: Simple Object Access Protocol (SOAP) 1.1. W3C recommendation, World Wide Web Consortium (W3C) (May 2000)
8. Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C recommendation, World Wide Web Consortium (W3C) (June 2007)
9. Winer, D.: XML-RPC Specification (June 1999) Online at `http://www.xmlrpc.com/spec`.
10. Hadley, M.J.: Web Application Description Language (WADL). W3C member submission, World Wide Web Consortium (W3C) (November 2006)
11. Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services. Springer (2006)
12. Motta, E.: An Overview of the OCML Modelling Language. In: 8th Workshop on Knowledge Engineering: Methods & Languages KEML 98. (1998)
13. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. W3C recommendation, World Wide Web Consortium (W3C) (2001)
14. Martin, D., Burstein, M., Lassila, O., Paolucci, M., Payne, T., McIlraith, S.: Describing Web Services using OWL-S and WSDL