

WSODL – An object-oriented specification for RPC-based web services

Enric Jaen Villoldo¹, Joan Serrat-Fernandez²

¹ Computer Science Department
Universitat Autònoma de Barcelona
Bellaterra, Spain

enric.jaen@campus.uab.es

² Network Management Group
Universitat Politècnica de Catalunya
Barcelona, Spain
serrat@tsc.upc.edu

Abstract

RPC-based web services tries to abstract away XML at the implementation level, but fails to do it at the specification level. We propose WSODL, a specification language for web services based on IDL, CIM and WSDL concepts. This language allows to model web services as an object-oriented middleware.

1. Introduction

Web Services can be designed around two styles, namely RPC and document [4]. Meanwhile RPC web services abstract away XML by transparently binding operation parameters to native classes, document web services require the service implementation to access directly at the XML level.

However, despite these two different ways to use and implement web services, both styles rely on the same, XML-based, WSDL description. For RPC web services it would be desirable to describe services abstracting away XML. Nowadays the alternative is to use the code-first approach[1], where toolkits generate WSDL from the native language. This approach, however, generates interoperability issues; on one hand because of the *impedance mismatch* [2][3] between XSD types and the programming language, and on the other hand because vendor toolkits implement different XSD subsets[5].

Furthermore, WSDL models interfaces, not objects [6]. A client cannot create object instances, neither pass objects-by-value or reference. We consider this a limitation, as the object-oriented paradigm has been widely accepted for modeling and programming systems, as in the case of CORBA, DCOM or RMI middlewares. We think that a web service description should be capable to be modeled as objects, so the gap between object-oriented languages and the service specification is eliminated.

We therefore propose WSODL (Web Service Object Definition Language), an approach for specifying object-oriented web services that combines the features of three standard languages: the OMG's

IDL[7], the DMTF's CIM[8], and the W3C's WSDL [9]. IDL provides us structured datatypes, CIM provides qualified classes and attributes, and WSDL provides binding information.

In WSODL, a service is modeled as a set of accesspoint operations, plus a set of classes with their properties and operations. This way of modeling a service provides furthermore semantic protocol information as it will be explained later.

In our model, objects are passed either by reference or as shared objects. In the latter, properties are read locally, and operations executed remotely. A light synchronization mechanism for maintaining the state coherence is proposed.

The WSODL description maps to the target programming language, and messages are transported using SOAP. Our approach improves interoperability by standardizing the mapping between WSODL and XSD, so that the subset of XSD constructs is well-known by the toolkit vendors.

WSODL supports standard container types, such as maps or trees. Container types are valuetypes whose operations are executed locally. We propose a set of standard WSODL containers and their respective XSD mappings in a library called Standard Container Type Library (SCTL). This library must be implemented in the client's programming environment. Otherwise, the client will need to invoke a proxy that implements such library.

The paper is structured as follows: Section 2 explains the proposed web service model. Section 3 describes related work regarding IDL to XML, approaches to solve the impedance gap, and object-oriented interface languages. Finally section 4 concludes the paper.

2. Proposed web service model

2.1 Motivating example

The proposed model is explained around an example. Consider a web service, named DigitalSecretary, that

manages meeting reservations in a hotel. This service allows users to manage rooms and meetings. Figure 1 list the WSODL description, and the next section explains its constructs.

```

1. enum Month { jan, feb, ..., dec };
2. typedef long Year;
3. [Min(1),Max(31)]
4. typedef short Day;
5. struct Date{Year y; Month m; Day d};
6. typedef Map<string,int> Months;
7. class Room {
8.     [Max(100)]
9.     int capacity;
10.    image/mpeg photo;
11.    Meeting book (
12.        [IN] Date slot,
13.        [IN,Max(90),Units(min)] int duration );
14.    void cancelBooking ( [IN] Date slot );
15. };
16. typedef sequence<Room> Rooms;
17. typedef ref<Room> RoomName;
18. class Meeting { ... }
19. accesspoint DigitalSecretary {
20.     RoomName createRoom([IN] string name);
21.     Rooms getRooms([Props]);
22.     Meeting createMeeting(...);
23.     //...
24. };
25. //-----
26. [soap,document]
27. binding DigitalSecretary_BD1 {
28.     [literal] createRoom( [literal] name );
29.     [literal] getRooms();
30.     //...
31. };
32. service DigitalSecretary {
33.     DigitalSecretary_BD1;
34.     endpoint http://host/digitalsecretary;
35. };

```

Figure 1. Example of WSODL description.

2.2 WSODL

The Web Service Object Description Language defines classes, operations, datatypes, containers accesspoints, bindings and endpoints to characterize a service. The language is based on the OMG's IDL datatype syntax, which has been extended with the DMTF's CIM classes and qualifiers to add datatype semantics, and finally WSDL binding information. Meanwhile some qualifiers are standard, the service designer can create its own.

WSODL, like WSDL, provides both an abstract (lines 1 to 24) and binding service information (lines 26 to 35). The abstract service is based on IDL and CIM. IDL types, attributes and parameters are qualified with square brackets, following the CIM notation. The service defines accesspoint operations and classes. The `accesspoint` construct (line 19)

equivalents to a port type in WSDL or interface in WSDL2. It defines those operations that a client can access at start-up. They have a static binding, i.e. their endpoint is well-known at start-up.

In addition, WSODL defines classes (`class` construct) which represent shared objects (lines 7 and 18). Classes define properties and operations. Objects (i.e. class instances) have dynamic binding, that is, their endpoints are created at runtime. Class properties are accessed locally, and operations are executed remotely. See sections 2.3 and 2.5 for more explanations about objects.

WSODL properties and datatypes are qualified. For example, the type `Day` (line 4) has a range 1 to 31. The method `book` of the `Room` class (line 11) has two input arguments: a `Date` and an integer whose qualifier indicates that the maximum duration for the meeting is 90 minutes.

It is possible to use standard containers, which are objects sent by value. For example, a `Map` is defined in line 6. See section 2.6 for more about container types.

WSODL allows to define MIME data types (line 10). These types are sent as attachments.

The `binding` construct describes the encoding of information transported. In this example, the `accesspoint` operations have a SOAP binding and use the document style (lines 26 to 31). If not specified, classes assume the same encoding as the service `accesspoint`. Operation parameters and return type can be qualified with the encoding use (literal in the example). Should all the types have the same encoding use, lines 28 and 29 could be omitted and the specified in line 26.

Finally, the service (`service` construct) is modeled as a list of bindings with their corresponding endpoints. In this example only one binding is defined.

2.3 Shared objects

A WSODL service can create object instances dynamically and pass them as shared objects. A shared object is a copy of a service object (see in section 2.5 how they synchronize). Properties are read locally, and operations executed remotely. A shared object has an intrinsic `ref` property, which is the URL of the service object. The object reference has the following notation:

```
http://host/service/className=identifier
```

For example, a reference for a `Room` could be:

```
http://h/digitalSecretary/Room=Einstein
```

When getting an object, clients can specify the properties which are interested in. The `Props` qualifier represents the list of properties that must be included (see line 21 for an example)

This qualifier is visible to clients as an ordinary operation parameter, but it is transparent to the service implementation, which simply returns the object and the service platform is responsible to serialize the selected properties. When getting an object, the client gets also its reference.

The service may define factory operations to allow users create new instances (like the `createRoom` operation at line 20). The service is responsible to control the creation and deletion of such objects.

2.4 Intrinsic operations and properties

A user can retrieve objects by invoking *intrinsic operations*. These are service accesspoint operations that are automatically generated and implemented during the WSODL compilation. For example, to get a Room object, the user would call the intrinsic operation:

```
Room getRoom([IN]string name, [IN]Props props)
```

Similarly, classes have intrinsic operations and properties. For example, the object reference is an intrinsic property, which is accessed through the `getEndpoint()` intrinsic operation.

2.5 Object synchronization

As explained before, when a client gets a shared object it is getting a copy of the service object. Therefore it is needed to maintain the coherence among copies. To avoid the service having to maintain such as coherence (which would increase the service load) the responsibility is given to the client. The mechanism we propose is the following: Each class has an intrinsic property named `epoch`. To update the user's copy, the user must call the `syncR()` intrinsic method, which requests the epoch of the service object, compares with its local epoch and, if different, the updated properties are downloaded and the local epoch updated. Similarly, to update the service object, the user calls `syncW()`, which sends only those properties updated locally.

2.6 Container types

A container type is a valuetype. Valuetypes are objects passed by value, whose operations are executed locally¹. Container types define classes that contain other types, like for example a map or a tree. Figure 2 describes a map in WSODL, and its mapping to XSD.

¹ Note the difference with regard WSODL objects, whose state is accessed locally but its operations executed remotely.

```
(a) container Map <key,val> {
    void put(key, val);
    val get(key);
};

(b) typedef Map <string, int> Months;

(c) <xs:complexType name="Months">
  <xs:sequence>
    <xs:element name="key" type="xs:string"/>
    <xs:element name="value" type="xs:int"/>
  </xs:sequence>
</xs:complexType>
```

Figure 2. Example of (a) container map specification, (b) WSODL definition, and (c) its XSD mapping.

We propose the creation of a standard container type library (SCTL), whose containers, their operations and XSD mapping are standardized. This library follows a similar concept as the STL (Standard Template Library) for C++.

When the client's environment implements the SCTL it will be able to unmarshal the data. Toolkits can map container types to language specific datatypes. For example the container type `Map` can be mapped to a `map` in STL, or a `java.util.HashMap` in Java. If the client programming environment doesn't implement the SCTL library, it can invoke a proxy service where these types be implemented. This proxy can be located locally or remotely. From the client viewpoint, the proxy acts as the service.

2.7 Semantic protocol information

A desired aspect of web services is the capability to give information about message ordering. The order of execution is established according two temporal relationships: sequence and choice. WSDL lacks of this feature. An individual operation gives us a sequential ordering: there is a request and then a set of possible responses. The information about the possible responses is given in an enumeration, as a type or qualifier. The accesspoint operations can be executed in parallel, i.e. they have a choice relationship. When an operation returns an object, this operation is giving us a temporal relationship: to invoke an object operation, first you must invoke the operation that gives the object reference. A similar temporal relationship happens when an operation has an object reference as a parameter. Finally, the operations inside an object can be executed in parallel; i.e. they have a choice relationship.

To illustrate this, consider a purchase order example. The abstract service is modeled in Figure 3. The user gets initially a list of products. Each product is an object. Given a product, the user can get the price, a description or make an order. The order may have several responses, given by the `ValueMap` qualifier. In case of an `Ok` response, the order returns an `Order`

object. The user can then confirm, cancel, or get the status of the order. The possible status of an order are given in an enumeration type.

```
enum OrdStatus {processing,blocked,cancelled,...};
class Order {
    confirm();
    cancel();
    OrdStatus status();
};
class Product {
    Order order([IN]int qty,[OUT, ValueMap
(Ok,noStockAvail,...)] int result);
    string desc;
    int price;
};

accesspoint ShoppingService {
    Product[] getProducts(...[Props]);
};
```

Figure 3. Example of WSODL service showing semantic protocol information.

4. Related Work

This section describes relevant work regarding IDL to XML mapping, approaches to solve the impedance gap, and object-oriented interface languages. Due to lack of space a brief description is given.

The OMG has published three specifications that map XML and IDL; A mapping from XML to IDL valuetypes using DTD [10]. Seconds, a mapping from WSDL to IDL [11], but some datatypes have semantic loss. Finally, a mapping from IDL to WSDL [12]. Our approach differs from the last in that we don't generate WSDL, but we directly map to the programming language.

Xen [2] is a programming language that solves the impedance gap by adding XML capabilities into an object-oriented language, such as C#.

XL (Xml Language) [13] is a programming model and platform. Web service applications are constructed using XML for the entire stack, instead of dealing with the current three different paradigms, namely, from bottom to top: XML, OO and SQL.

CIM (Common Information Model) [8] is an object-oriented modeling specification and framework for defining management classes. Classes define methods and properties. Properties can be qualified. Users get instances by-value, and invoke class methods remotely, through the XML-based WBEM protocol.

MIDL/ODL [14] describes COM interfaces. It is more verbose than IDL, and qualifiers are allowed but predefined. It is not possible to specify value constrains. Objects are sent by reference, and their properties accessed remotely. Only read objects can be sent by value. There is no notion of MIME datatypes.

5. Conclusions

We consider that WSDL is more targeted towards document-based services, and that is missing a web-based middleware that defines and implements an object-oriented middleware, like CORBA and DCOM.

In this paper we have presented WSODL, an object-oriented web service description language focused on RPC-based web services. This language has been created to overcome three issues: The interoperability problems that current toolkits face with RPC web services, the incapacity of WSDL to model object-oriented services, and the inconvenience to use XML to describe RPC web services.

References

- [1] W. Provost, WSDL First, O'Reilly webservices.xml, 22 July 2003.
- [2] E. Meijer, W. Schulte, G. Bierman, Programming with Circles, Triangles and Rectangles, XML Conference & Exposition, Phyladelphia, USA, Dec 2003.
- [3] Loughran, Steve; Smith, Edmund, Rethinking the Java SOAP stack, HP tech. Report. HPL-2005-83, May 2005.
- [4] Mitch Gitman, Keep up with the Web service styles (and uses), Javaworld, Oct 2003.
- [5] Yahoo SOAPBuilders Mailing List.
- [6] Vogels, Werner, Web Services Are Not Distributed Objects, Internet Computing, Nov 2003, vol 7 num 6.
- [7] OMG, OMG IDL Syntax and Semantics - CORBA 3.0 Specification, chapter 3, July 2002.
- [8] DMTF, Common Information Model Specification, version 2.2, June 1999.
- [9] W3C, Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, working draft, <http://www.w3.org/TR/wsdl20>, May 2005.
- [10] OMG, XML/Valuetype Language Mapping Specification, Version 1.1, Apr 2003.
- [11] OMG, WSDL-SOAP to CORBA Interworking Specification, July 2003.
- [12] OMG, CORBA to WSDL/SOAP Interworking Specification. Version 1.1, Feb 2005.
- [13] D. Florescu, A.Grunhagen, D.Kossmann, XL: A platform for Web Services, 1st Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, Jan 2003.
- [14] Microsoft Corporation, Microsoft Interface Definition Language (MIDL) , MSDN Library, 2005.