Applying MDA Approach to B2B Applications: A Road Map

Jean Bézivin[(1)] Slimane Hammoudi[(2)] Denivaldo Lopes[(1)(2)] Frédéric Jouault[(1)(3)]

[(1)] Atlas Group, INRIA and LINA
University of Nantes
2, rue de la Houssinière - BP 92208
44322 Nantes Cedex 3, France

[(2)] ESEO
4, rue Merlet de la Boulaye, BP 926
49009 cedex 01 Angers, France

[(3)] TNI-Valiosys
120, rue René Descartes
Technopôle Brest Iroise - BP 70801
29608 Brest Cedex, France

{Jean.Bezivin, Frederic.Jouault}@lina.univ-nantes.fr {shammoudi, dlopes}@eseo.fr

Abstract

B2B applications are systems that evolve quickly and they are often developed using different technologies, such as XML/EDI, Distributed Components and Web Services. Many technologies supporting B2B applications exist and others will appear. Recently, Model-Driven Architecture (MDA) approach have been introduced to support the evolution of systems and the harmonization between different technologies. However, before this becomes a reality, some issues need solutions, such as the creation of mappings between meta-models. In this paper, we provide some insights into the definition of mappings and transformation rules.

1. Introduction

Business-to-Business (B2B) Applications on the Internet are often complex software systems. They are not only complex because they are large distributed systems but for some reasons:

- They are systems that evolve quickly.
- They are implemented using different technologies.
- They must often integrate legacy systems.

Many technologies such as Electronic Data Interchange (EDI), Transaction Processing Monitor (TP monitor), and Distributed Component have supported B2B applications [7]. Recently, Web Services [13] have been used to enable B2B applications on the Web. In order to describe a B2B process as a composite Web Service, business process languages were created such as Business Process Execution Language for Web Services (BPEL4WS) [1].

Technologies come and go, but business logics are longer lasting. Protecting software investments from obsolescence is an important goal. On the other hand, B2B applications are and will continue to be developed using multiple technologies. The integration and harmonization between different technologies for the development of business applications is another important goal.

In recent years, Model Driven Architecture (MDA^{TM})[1] [8] has been proposed to support the development of large software systems providing an architecture where systems can evolve, and technologies can be integrated and harmonized. To reach this challenge, some issues need be settled such as the definition of mappings between meta-models and the creation of transformation rules. In this paper, we concentrate our discussion to provide a solution for these two issues. For this purpose, we use UML [2] as a Platform-Independent Model (PIM); BPEL4WS, Web Service, Java and Java Web Services Developer Pack (JWSDP) [9] as Platform-Specific Models (PSMs); and Atlas Transformation Language (ATL) [2] as formalism to express transformation rules.

This paper is organized as follows. Section 2 is an overview of the B2B applications and Web Services in the context of MDA. Section 3 proposes some meta-models (i.e. a BPEL4WS, Web Service and Java meta-model), and mappings from UML into these platforms. The last section is the conclusion of our research.

2. Overview

In the 90's, UML was created as a common modeling language to visualize, specify, construct and document the artifacts of software systems. Rapidly, it was becoming a de facto industry standard. Thanks to UML, software enterprises and organizations have discovered the power of models.

Recently, the Object Management Group (OMG) has proposed and stimulated the model driven approach to develop the artifacts of software systems. This is becoming a reality through Model Driven Architecture (MDA) [8]. This

1 MDA^{TM} is a trademark of the Object Management Group.

2 In this research, we have used UML version 1.4, but afterwards we intend update to UML version 2.0

model driven approach is based on an architecture with four meta-layers: meta-meta-model (or M_3 layer), meta-model (or M_2 layer), model (or M_1 layer), and information (or M_0 layer) [4].

Figure 1 presents the B2B application design in the context of MDA. According to this figure, two techniques are possible: (a) marking and (b) weaving. Marking is based on UML Profiles to decorate a Platform-Independent Model (PIM) with aspects such as services and security. Weaving is based on the idea of making a texture between a PIM and a Platform-Description Model (PDM) [3] before generating a Platform-Specific Model (PSM). In our research, we have developed the weaving technique, but it is out of the scope of this paper.

The term Platform-Independent Model is used to refer to a model that has only the structure and functionality of a system and no information about implementation details. Platform-Specific Model is used to refer to models that have information about implementation details [8].

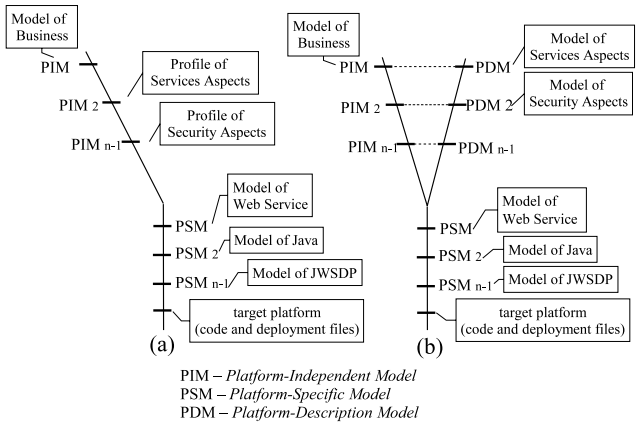

Figure 1. B2B Application Design and MDA

For example, UML can be considered as a PIM, but a model that depends on artifacts such as Web Services Description Language (WSDL) must be seen as PSM, therefore it integrates notions of the Web Service platform. In addition, we consider the existence of the term Platform-Description Model to refer to aspects such as services-oriented, security and availability. A business model is developed as a PIM (e.g. using UML), using aspects such as service, security, and availability. Afterwards, this PIM is transformed into a PSM (e.g. based on BPEL4WS, Web Service, Java and JWSDP) using transformation rules, until exported as code and deployment files.

When we employ the model driven approach, we do not forget the existence of other Technological Spaces(TS), i.e. *a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities* [6]. In fact, we have used MDA to harmonize different Technological Spaces. Figure 2 presents the relationships between MDA, Abstract Syntax (AS) and XML technological space.

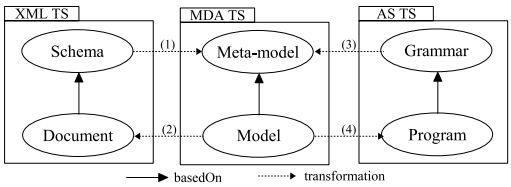

Figure 2. Technological Spaces

MDA should be understood as a technology which harmonizes and unifies the relationships between different Technological Spaces. In Figure 2, (1) we take an XML Schema, move it to MDA TS (i.e. meta-model based on MOF), create a model based on this meta-model (i.e. models) and afterwards (2) export the result again to XML TS as an XML document. Other similar steps (3)(4) can be used with the AS and MDA TS.

At this step, we can note some benefits from the MDA approach to the B2B applications:

- the same PIM can be used many times to generate models on different platforms (PSMs) [5].
- preservation of the business's logic against the changes or evolution of technology [8].
- a uniform manner for business models and for technologies to evolve together.
- enhancement of the reengineering [4], i.e. it assists the recuperation of a business's logic from source codes or from implementation environments.

The model driven approach seems to be a good solution for the development of large software systems, but some issues are still not settled, such as mappings between meta-models [5].

3. From PIM to PSM: A road map

In our research, UML model is considered as a PIM. For this purpose, the UML activity diagram is applied for modeling business process. On the other hand, static structure between the participants of a process is modeled using UML class diagram.

In this section, we will present and discuss some mappings needed to create a B2B application in the context of MDA. For this purpose, we begin showing a mapping from UML into BPEL4WS; afterwards, a mapping from UML into Web Services; and finally, a mapping from UML into Java Platform.

3.1. From UML to BPEL4WS

BPEL4WS [1] defines a model and a grammar for describing the behavior of a business process. It depends on the WSDL [12], i.e. a BPEL4WS Process makes references to portTypes (see section 3.2).

BPEL4WS has elements that enable the creation of abstract and executable business processes. In addition, BPEL4WS supports extensibility using namespace-qualified attributes and elements from other namespaces to appear within BPELWS elements. A BPEL4WS meta-model is presented in the right side of Figure 3. This meta-model is formed by elements, such as:

- `Process` - created using a series of activities, partners, correlation sets, fault handlers and compensation handlers.
- `PartnerLinks` - defines the different parties that interact within a business process in execution.
- `Partners` - defined as a set of `PartnerLink` (i.e. references), it introduces a constraint on the functionality that a business partner is required to provide.
- `Variables` - defines the data variables used by the process, and allows processes to maintain state data and process history based on messages exchanged.
- `Activity` - structured in some parts such as control flow, message flow (e.g. `Invoke` and `Receive`) and data flow.

Figure 3 presents a mapping from UML Activity Graph into BPEL4WS (fragment). This mapping is made by a set of transformation rules that specify the elements of the source meta-model, i.e. UML, which are equivalent to elements of the target meta-model, i.e. BPEL4WS meta-model. In this figure, we use a graphical notation to illustrate a mapping from UML into BPEL4WS. The UML meta-model is presented on the left side, the mapping in the center, and the BPEL4WS meta-model on the right side. This graphical notation has the following elements: connection (source and target), association, transformation rule and composition. A connection links one or more meta-model element(s) to a transformation rule. The association shows a relationship between rules. The composition shows a tight relationship between rules. The transformation rule takes a source element and generates the suitable target element.

This figure illustrates mappings one-to-one (i.e. each element from the source meta-model corresponds to one element from the target meta-model).

We have used ATL transformation language to define transformation rules [2]. According to Figure 3, we have the following mappings (we will present only a few ATL transformation rules):

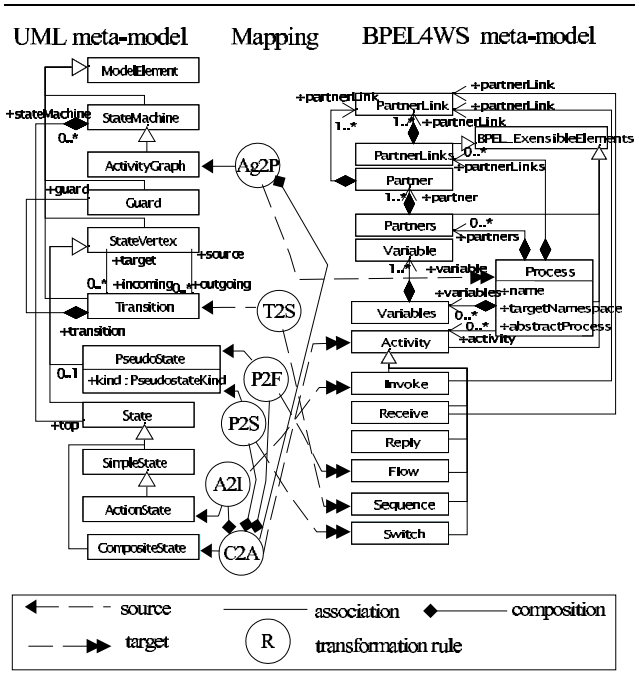

Figure 3. Mapping from UML Activity Graph into BPEL4WS

- The UML `ActivityGraph` is mapped into BPEL4WS `Process` through the rule `Ag2P`:

```
rule Ag2P{
  from ag: UML!ActivityGraph
  to pc: BPEL!Process
  mapsTo ag(
    name <- ag.name,
    targetNamespace <- "http://www."+ag.name+".com",
    abstractProcess <- false,
    xmlns <- "http://schemas.xmlsoap.org/" +
             "ws/2003/03/business-process/"
  )
}
```

- The UML `PseudoState` is mapped into BPEL4WS `Flow` through the rule `P2F`:

```
-- Helper pour P2F:
helper context UML!PseudoState def:
helperGetActivities(): Collection(BPEL!Activity) =
-- the body was omitted because it is so large
;
helper context UML!PseudoState def:
helperGetLinks(): BPEL!Links =
-- the body was omitted because it is so large
;
-- Rule P2F
rule P2F{
  from  ps: UML!PseudoState(ps.kind=#pk_fork)
  to    fl: BPEL!Flow
  mapsTo ps(
    name <- ps.name,
    links <- ps.helperGetLinks(),
    ref <- ps.helperGetActivities()
  )
}
```

3.2. From UML to Web Services

The concept of services was introduced before Web Service technologies. In fact, this concept has been used for a long time by DCE, CORBA, Java RMI, and DCOM. A service is an abstraction of programs, business process, and other artifacts of software defined in terms of what it does. Services can be organized in a Service Oriented Architecture (SOA) which is a form of distributed system architecture [13].

Web Service is a good implementation of SOA. According to this architecture, Universal Description, Discovery, and Integration (UDDI) [10] is used to publish services using information registered by the service provider (e.g. provider name and endpoint) or to search for a service using the requirements of a service requester. This endpoint indicates the interface of the service that is described using Web Service Description Language (WSDL) [12]. The service uses Simple Object Access Protocol (SOAP) [11] as a communication protocol, and SOAP uses HTTP or FTP or SMTP as transport protocol. In order to simplify our discussion, we present only a mapping from UML into WSDL.

In the right side of Figure 4 is presented a simplified meta-model of WSDL. This meta-model has elements, such as: `Definition`, `TypesType`, `MessageType`, `ServiceType` and so on [12].

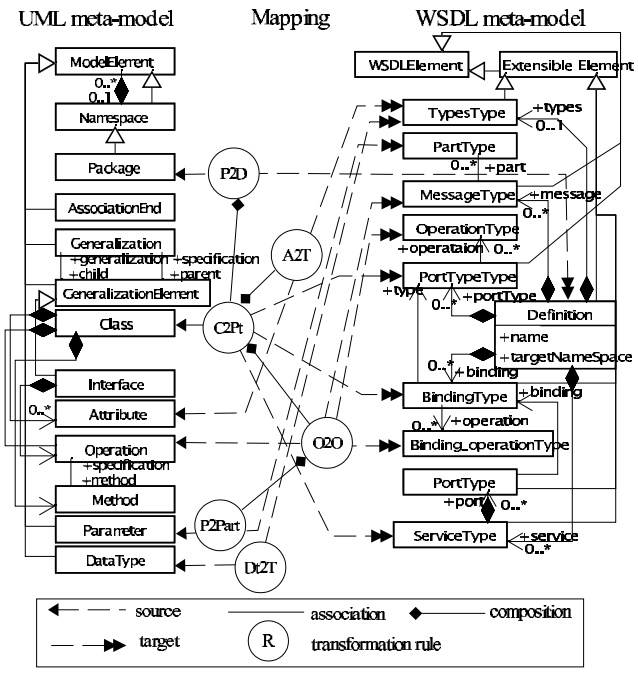

Figure 4. Mapping from UML into WSDL

Figure 4 presents a mapping from UML into WSDL. The `C2Pt` rule is of type one-to-many (i.e. one element from the source meta-model corresponds to two or more elements from the target meta-model), it maps a UML `Class` into a WSDL `PortTypeType`, a `BindingType` and a `ServiceType`. The rules are expressed in ATL language as follow (we will present only a few ATL transformation rules):

- The UML `Package` is mapped into WSDL `Definition` through the rule `P2D`:

```
-- The UML Package is transformed into WSDL
-- Definition through the rule P2D:
rule P2D{
  from pack : UML!Package
  to def :WSDL!Definition
  mapsTo pack(
    name <- 'Service_' + pack.name,
    targetNameSpace <- 'urn://' + pack.name + '.wsdl'
  )
}
```

- The UML `Class` is mapped into WSDL `PortTypeType`, `BindingType` and `ServiceType` through the rule `C2Pt`:

```
-- The UML Class is transformed into WSDL PortType,
-- BindingType and ServiceType through the rule C2Pt:
rule C2Pt{
  from c : UML!Class
  to pt : WSDL!PortTypeType
  mapsTo itf(
    name <- c.name,
    operations <- [O2O.wsdlop]
      itf.feature ->  select(e |
        e.oclIsKindOf(UML!Operation))
  ),
  bd : WSDL!BindingType(
    name <- c.name + 'Binding',
    type <- pt,
    operations <- [O2O.wsdlob]
      c.feature ->
         select(e | e.oclIsKindOf(UML!Operation))
  ),
  sv : WSDL!ServiceType(
    name <- 'Service' + c.name,
    port <- pport
  ),
  pport : WSDL!PortType(
    name <- c.name+'Port',
    binding <- bd,
    soap <- ssoap
  ),
  ssoap : WSDL!SOAP(
    location <- 'http://host:port/'  +
    context-path/url-pattern '
    -- to be replaced
  )
}
```

3.3. From UML to Java Platform

In order to develop Web Services using the Java Platform, two main components are taken into account: Java language and JWSDP[9].

A Java meta-model is presented in the right side of Figure 5. The main elements of this Java meta-model are:

- `JavaPackage` - a container for `JavaClass`, `JavaInterface` and so on.

- `JavaClass` - a specialization of `JavaClassifier` which implements `JavaInterfaces`. In this paper, we do not address the transformation from a UML Class that contains multiple inheritance into a Java Class that has a single inheritance.
- `JavaInterface` - another specialization of `JavaClassifier`, but which has only prototypes of methods and final static attributes.
- `JavaField` - a composition of only one `JavaPrimitiveType` or `JavaClass` or `JavaInterface`.
- `JavaMethod` - has the body and the signature of an operation, i.e. a return, identifier and arguments.

Figure 5 presents a mapping from UML into Java (fragment). The rule `OM2M` is of type many-to-one, it maps a UML `Operation` and a `Method` into a `JavaMethod`.

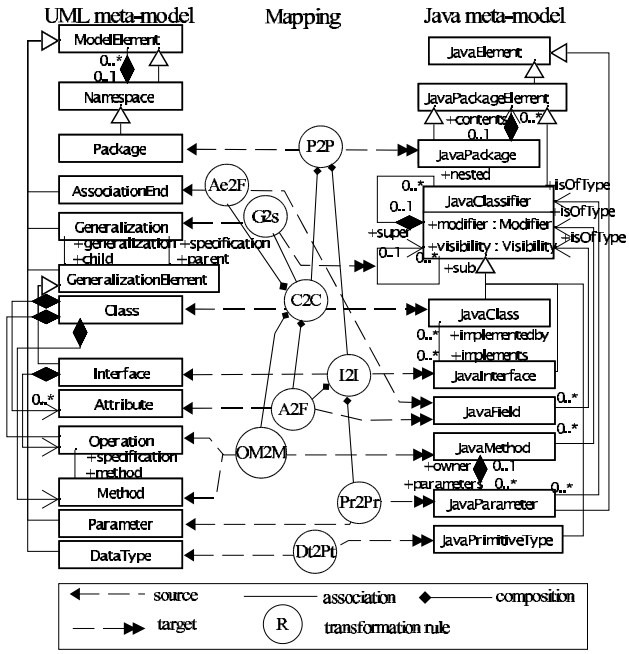

Figure 5. Mapping from UML into Java

According to Figure 5, we have the following mappings (we will present only a few ATL transformation rules):

- The UML `Package` is mapped into Java `Package` through the rule `P2P`:

```
-- The UMLPackage is transformed into the
-- JavaPackage through the rule P2P:
rule P2P{
  from pck : UML!Package
  to jp : Java!JavaPackage
  mapsTo pck(
    name <- pck.name
  )
}
```

- The UML `AssociationEnd` is mapped into Java `Field` through the rule `Ae2F`:

```
-- Helper for getOtherEnd()
helper context UML!AssociationEnd def: getOtherEnd():
UML!AssociationEnd = self.association.connection->
  select(e|e <> self)->first();

-- The AssociationEnd is transformed into the
-- JavaField through the rule Ae2F:
rule Ae2F{
  from ae : UML!AssociationEnd
  to  jf : Java!JavaField
  mapsTo ae(
    name <- ae.name,
    visibility <-
     if ae.visibility = #vk_public then
       #public
     else
       #private
     endif,
    isTransient <- false,
    isVolatile <- false,
    isFinal <- false,
    isOfType <- ae.participant,
    owner <- ae.getOtherEnd().participant
  )
}
```

- The UML `Class` is mapped into Java `Class` through the rule `C2C`:

```
-- The UML Class is transformed into the Java
-- Class through the rule C2C:
rule C2C{
  from c : UML!Class
  to jc : Java!JavaClass
  mapsTo c(
    name <- c.name,
    visibility <-
    if c.visibility = #vk_public then
      #public
    else
      #private
    endif,
  modifier <-
    if c.isAbstract then
      #abstract
    else if c.isLeaf then
      #final
    else
      #regular
    endif endif,
  isActive <- c.isActive,
  super <-
    c.generalization->first().parent,
  implements <-
    c.clientDependency ->
      select(e|e.hasSterotype('realize'))
       ->collect(e | e.supplier)
  )
}
```

After the transformation from UML into Java, we proceed transforming the obtained model into another model based on a JWSDP template and meta-model.

This template is a mold for implementing Web services using JWSDP. In fact, the template uses the Application Programming Interfaces (APIs) of JWSDP which we have classified as a model (i.e. model layer or M_1 layer). A Web Service implemented using JWSDP uses a `JavaClass` that implements a `JavaInterface`. This interface must extend `java.rmi.Remote` and its methods can launch `java.rmi.RemoteException`. The `JavaClass` has the

service and the `JavaInteface` is used by the client to call up the service.

JWSDP meta-model is an extension of the Java meta-model. Thus a transformation from Java into JWSDP implies a transformation from a Java model into another Java model, and also the creation of deployment files and interfaces implemented for each class that has services. This JWSDP meta-model was extracted from DTD and XML Schema of the JWSDP version 1.3 [9]. This JWSDP meta-model presents the deployment files needed to create, deploy and consume a Web Service. These deployment files are `config-interface.xml`, `jaxrpc-ri.xml`, `web.xml` and `config.xml`.

4. Conclusions

In this paper, we discussed the mapping between meta-models such as a mapping from UML into BPEL4WS. We have briefly presented ATL transformation language to specify the model transformation. Consequently, we have come to the following conclusions:

- **The size of the gap between two meta-models has a significant influence on the complexity of the mapping** - The gap between UML and BPEL4WS meta-models is so large, thus the mapping was complex.
- **A pivot meta-model to help the transformation from UML to BPEL4WS model is necessary** - the difficulty in creating transformation rules to transform UML into BPEL4WS can be transposed using an pivot meta-model which will operate like a intermediary between UML and BPEL4WS.

In order to create Web Services, we needed to use a specific API. In our case, we have used JWSDP APIs. In this research, we have considered APIs as models (i.e. $M1$ layer) and used them in model transformation. So, transformation rules take a source model and give a target model. In our experiments, this target model is incremented by a model of JWSDP APIs. The representation and manipulation of APIs in the context of MDA are open issues and they merit so much discussion, but they are out of scope of this paper.

In future research, we envisage create a pivot meta-model to facilitate the transformation from UML into BPEL4WS, develop a case study to validate our approach, and study APIs in the context of MDA.

References

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *Business Process Execution Language for Web Services (BPEL4WS) version 1.1*, May 2003.

[2] J. Bézivin, G. Dupé, F. Jouault, G. Pitette, and J. E. Rougui. First experiments with the ATL model transformation language: Transforming XSLT into XQuery. *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, October 2003.

[3] J. Bézivin and S. Gérard. A Preliminary Identification of MDA Components. *OOPSLA 2002 Workshop on Generative Techniques in the context of Model Driven Architecture*, November 2002.

[4] J. Bézivin and N. Ploquin. Tooling the MDA Framework: a new Software Maintenance and Evolution Scheme Proposal. *Journal of Object-Oriented Programming (JOOP)*, December 2001.

[5] A. Gerber, M. Lawley, K. Raymond, J. Steel, and A. Wood. Transformation: The Missing Link of MDA. *First International Conference on Graph Transformation (ICGT2002)*, October 2002.

[6] I. Kurtev, J. Bézivin, and M. Aksit. Technological Spaces: An Initial Appraisal. *CoopIS, DOA'2002 Federated Conferences, Industrial track*, 2002.

[7] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. K. Elmagarmid. Business-to-business interactions: issues and enabling technologies. *The VLDB Journal The International Journal on Very Large Data Bases*, 12(1):59–85, 2003.

[8] OMG. *Model Driven Architecture (MDA)- document number ormsc/2001-07-01*, 2001.

[9] Sun Microsystems. Java Web Services Developer Pack, March 2004. Available at http://java.sun.com/webservices.

[10] UDDI.ORG. *Universal, Description, Discovery and Integration (UDDI) Version 3.0*, July 2002. Available at http://www.uddi.org.

[11] W3C. *Simple Object Access Protocol (SOAP) 1.1*, May 2001. Available at http://www.w3.org/TR/SOAP.

[12] W3C. *Web Services Description Language (WSDL) 1.1*, March 2001. Available at http://www.w3c.org/tr/wsdl.

[13] W3C. *Web Services Architecture (WSA)*, February 2004. Available at http://www.w3c.org/TR/2004/NOTE-ws-arch-20040211/.