Modeling and generating Web services metadata using Rational Software Architect

The functional aspect, WSDL, and the nonfunctional aspect, WS-Policy

Xi Ning Wang
Xin Peng Liu
薛 亮

May 15, 2007

Learn how to use IBM® Rational® Software Modeler/IBM® Rational® Software Architect and the Unified Modeling Language (UML) to model the standardized Web services metadata. Web services metadata contains both functional and nonfunctional aspects that provide essential information, such as the XML schema, the Web Services Description Language (WSDL), and the Web Services Policy Framework (WS-Policy). Although the UML-to-XSD transformation is one of several model-to-model and model-to-text transformations delivered with Rational Software Architect, this article focuses on modeling and generating the functional WSDL aspect and the nonfunctional WS-Policy aspect.

UML metadata context in this article

The introduction of the Unified Modeling Language (UML) was long-anticipated, because it contains certain design elements that lend themselves well to the specification of services. In UML models, a custom UML profile is a package that contains stereotypes and constraints that adapt the UML metamodel for a particular domain. A custom profile extends UML without changing the base metamodel.

In a service-oriented environment, service policies have a fundamental impact on interoperability. It’s important for a service provider to communicate with potential requesters when these Web service policies will be enforced. Service policies become a critical part of service descriptions, augmenting the basic WSDL functional description with a statement of nonfunctional service behaviors. As such, Web service policies support the development of service applications and provide the means to perform both development time and runtime service discovery and selection that are based on nonfunctional capabilities. For example, a service can be selected from among a list of functionally equivalent services, based on its support for a specific privacy policy or the security assurances that it provides.
The first set of Web service policy documents (WS-Policy, WS-PolicyAttachment, and WS-PolicyAssertions) was published in 2002. That was followed by an updated set of two documents, WS-Policy and WS-PolicyAttachment, which was published in September 2004. By 2006, the Web service policy documents (Web Services Policy 1.5 - Framework and Web Services Policy 1.5 - Attachment) had been submitted to World Wide Web Consortium (W3C). The Web Services Policy Working Group has since produced a Working Draft.

Web services metadata is a piece of the data puzzle for Web services because it provides essential information, such as the XML schema, the Web Services Description Language (WSDL), and the Web Services Policy Framework (WS-Policy). WSDL defines an XML-based syntax for describing network services as a set of endpoints that accept messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, but they are bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WS-Policy defines a base set of constructs that can be used and extended by other specifications for Web services to describe a broad range of service requirements, preferences, and capabilities.

The model that Figure 1 presents is intended to capture the set of concepts common to the design and architecture of service-oriented solutions.

As the diagram shows, these concepts can be grouped into these five categories (for details, please see the link to the Object Management Group. UML Profile and Metamodel for Services RFP, 2006 in Related topics):

- **Message**: The information model concerned with the messages flowing between services.
- **Service**: The specification of a service, it’s structural and behavioral contract.
- **Composition**: The manner in which services are composed into aggregate services and how services are distributed.
- **Collaboration**: The dynamic manner in which services interact, this can be seen as a choreography view.
- **Policy**: Affecting all of the four views above policy can be seen as a set of constraints and statements about the elements contained in them.

**Figure 1. The service-oriented model from 5 different views**

While following the steps that this article explains, you customize a profile for modeling WSDL. The profile provides the functional metadata and applies stereotype to UML elements for specifying WS-Policy, typically WS-Security Policy. You use UML modeling and IBM® Rational® Software Modeler to model the standardized Web services metadata, which includes both functional and
nonfunctional aspects. Although the UML-to-XSD transformation is one of several model-to-model and model-to-text transformations delivered with Rational Software Modeler, this article focuses on modeling and generating the functional WSDL aspect and the nonfunctional WS-Policy aspect.

Modeling Web services metadata based on UML

Web services are emerging as the perfect framework for application-to-application integration or collaboration, to make these applications available as Web services. To standardize the use of Web services, the World Wide Web Consortium (W3C) proposed the Web Service Description Language (WSDL) standard, an XML-based language that describes Web service functionality. Essentially, a WSDL file is a language-independent XML-based version of an IDL (Interface Definition Language) file that describes the operations offered by a Web service, as well as the parameters that these operations accept and return. Thus, WSDL has become the standard that supports the description of Web services: What they do, how they should be used, and where they are localized.


- The **WS-Policy** specification describes the syntax for expressing policy alternatives and for composing them as combinations of domain assertions. The WS-Policy specification also describes the basic mechanisms for merging multiple policies that apply to a common subject and the intersection of policies to determine compatibility.

- The **WS-PolicyAttachment** specification describes how to associate policies with a particular subject. It gives normative descriptions of how this applies in the context of WSDL and UDDI, (Universal Description, Discovery, and Integration), and it provides an extensible mechanism for associating policies with arbitrary subjects through the expression of scopes.

Along with the boom of Web services and the thriving Model Driven Architecture (MDA), we must consider the growing significance and utility of modeling in the development of software and solutions. MDA, which was proposed by the Object Management Group (OMG), is a model-driven framework for software development that proposes to model the business logic with Platform-Independent Models (PIMs) to later transform them on Platform-Specific Models (PSMs) by using transformation guides between the different models. The main advantages of MDA are the ability to transform one PIM into several PSMs, one for each platform or technology in which the final system will be deployed, and the automatic code generation that implements the system for those platforms from the corresponding PSMs.

Because Web services are software components, the development of Web services must exploit the advantages of MDA. To apply the MDA principles in the development of Web services, a modeling process must be considered. According to MDA principles, this modeling activity should result in automatic code generation. If we want to abstract from the platform in which the Web service will be deployed, the code that should be generated is the WSDL document that contains the Web service description in a standard format.

UML profile for WSDL and WS-Policy

Deriving from the MDA methodology, WSDL modeling can be divided into two phases:
• One view represents the abstract portion of the WSDL, which includes:
  • Definitions
  • Service
  • PortTypes
  • Messages
  • Parts
  • PartTypes
• The other view completes the bindings section of the WSDL, which includes these modeled elements:
  • Service
  • Ports
  • Bindings

Those are the important concepts in modeling Web services metadata. UML and the extendable UML profile can be used to model Web services. For example, Web service operations (WSDL operations) can be described as UML operations, Groups of Web service structure requests (WSDL port types) can be depicted as UML interfaces or classes, as XML schemas by using UML class diagrams, and as Web services data structures by using UML class diagrams.

WS-PolicyAttachment defines various attachment points for policy. This article focuses on the example of WS-Security Policy to demonstrate how to model nonfunctional metadata for Web services. Figure 2 is a UML profile diagram for modeling WSDL and parts of WS-SecurityPolicy. It demonstrates the actual details of the profile and shows each stereotype with its metaclass, using the extension notation.

**Figure 2. UML profile applied to model WSDL and WS-SecurityPolicy**

The UML diagram in Figure 2 also shows the classes used for security bindings, supporting token assertions for operation policy subject and protection assertions for the message policy subject. The TransportBinding, SymmetricBinding, and AsymmetricBinding classes reflect the security bindings described in the WS-SecurityPolicy specification. Each of these three concrete bindings has one or more associations with the abstract TokenAssertion class. The complete set of token assertion classes is described in the WS-SecurityPolicy specification.
The supporting token assertions for the operation policy subject define four properties related to supporting token requirements. These may be referenced by a security binding:

[Supporting Tokens]

[Signed Supporting Tokens]

[Endorsing Supporting Tokens]

[Signed Endorsing Supporting Tokens]

The protection assertions for the message policy subject are used to identify what is being protected and the level of protection provided. These contain the following assertions:

[SignedParts Assertion]

[SignedElements Assertion]

[EncryptedParts Assertion]

[EncryptedElements Assertion]

[RequiredElements Assertion].

**Plug-in development based on Rational Software Architect**

This section shows you how to develop new plug-in features based on Rational Software Architect. It extends the com.ibm.xtools. presentation.paletteProviders extension point to provide the modeling function by dragging from the palette, as Figure 3 shows. It also extends the com.ibm.xtools.presentation.semanticProviders extension point to provide semantic for modeling. For example, when you drag a Service element from the palette, the new UML2 Package element is created and applied to the Service stereotype of the WSDL profile. And if you need to create an element for describing message, you will drag an Entity element from the palette and the UML2 class element will be created and applied to the Entity stereotype.
**Figure 3. Modeling environment**

![Modeling environment](image)

**Figure 4** shows the conversion from a WSDL document attached to a WS-Security Policy to a UML model. The left side of the figure shows the corresponding view of the UML model for a WSDL document. For clarity, the WSDL document is simplified by leaving out a few elements and attributes, as well as by removing all of the XML namespace information. The right side shows several UML elements modeling concrete WSDL document elements. The arrows link these elements with corresponding representation in the WSDL model structure.
Generating WSDL and WS-Policy artifacts

This article uses Model 2 Text, a conscript specification-based language, to specify the transformation rules between UML and WSDL (with WS-Policy). The "Object Management Group. Request for Proposal: MOF Model to Text Transformation RFP, 2004" (listed in Related topics) is the Softeam/SINTEF revised submission in response to this RFP.

UML2 WSDL (with WS-Policy) transformation rules


Listing 1. Detail of UML2 WSDL (with WS-Policy) transformation rules

```c++
// main
uml.Model::main () {
  self.ownedMember->forEach(p:uml.Package)
  p.interfacePackage()
}

/*
 * Wsdl Bindings
 */
uml.Interface::wsdlBindings() {
  <%
```
Plug-in development based on MTF

Based on these transformation rules, we extended Rational Software Architect Xtools transformation extensions to implement the UML to WSDL (with WS-Policy) transformation
feature. Rational Software Architect Model Transformation Framework (MTF) provides one extension point that you can use to extend the capabilities of UML transformation.

As one of these extension points, com.ibm.xtools.transform.core.transformationProviders facilitates the configuration of providers for the transformation service (com.ibm.xtools.transform.core.services.TransformationService). The transformation service enables Xtools clients to register model transformations. You can use these transformations to convert the data from one model to another. Typically, the source model is UML-based with transformation-specific profiles applied. The target model is typically a code model, such as Java or XML. You can choose which transformation to apply to your source model.

Listing 2 provides the source code for the UML2 WSDL (with WS-Policy) transformation provider extension, which you can use to apply UML to WSDL (with WS-Policy) transformation:

Listing 2. UML2WSDL (with WS-Policy) transformation provider extension

```xml
<extension point="com.ibm.xtools.transform.core.transformationProviders">
  <TransformationProvider
    class="uml2wsdl.UML2WSDLTransformationProvider">
    <Priority
      name="Highest">
    </Priority>
    <Transformation
      groupPath="UML2WSDLTrans"
      profiles="WSDLProfile"
      targetModelType="Resource"
      transformGUI="uml2wsdl.TransformGUI"
      description="UML to WSDL"
      name="UML to WSDL"
      author="wangxn"
      id="uml2wsdl.transformation.uml2wsdl.root"
      version="1.0.0"
      sourceModelType="UML">
      <Property
        name="Overwrite output WSDL files"
        value="false"
        id="overwriteOutputFiles">
      </Property>
    </Transformation>
  </TransformationProvider>
</extension>
```

Creating a new plug-in project

Start by creating a new plug-in project.

1. From the File menu, select **New > Plug-in Project**.
2. In the resulting dialog, specify the project name and basic properties.
3. **Figure 5** shows the predefined templates. Select **Plug-in with Transformation** from the available templates, and then click **Finish**. This creates a standard plug-in project with one transformation provider that is used to define new transformations.
4. Extend com.ibm.xtools.transform.core.transformationProviders extension point, and add the transformation provider: `uml2wsdl.UML2WSDLTransformationProvider`. Figure 6 shows this extension and implementation class.

**Figure 6. Creating UML2WSDLTransformationProvider**

Tip:
You can apply either a horizontal or a vertical orientation layout by clicking the 🔊 or 🔊 icon.
5. After you create the transformation provider, you can add the new transformation, as Figure 7 shows. This specifies the UML2 as the source model type and Resource as the target model type. In the details of this extension element, you also can specify the UML profiles to apply to UML model. Otherwise, if you need to redesign the transformation GUI, you will specify one new TransformGUI class.

**Figure 7. UML to WSDL transformation details**

![UML to WSDL transformation details](image)

6. The other concrete implementations that you need to do include the plug-in implementation class, the utility class, and the transformation rules about the SetupTarget rule, the Package rule, the SaveOutput rule, and a template class, such as the WSDLPortTypeTemplate which generates the Port Type part of WSDL. The following Figure 8 shows the full implementation classes and their dependencies.
The Web Services Interoperability Organization (WS-I) has developed a supply chain management business scenario to demonstrate the features of the WS-I Basic Profile 1.0. The WS-I sample business scenario and the technical solution overview are described in the following documents:

- WS-I Supply Chain Management Use Cases 1.0
- WS-I Usage Scenarios 1.0
- WS-I Supply Chain Management Technical Architecture 1.0

For full details, see the Web Services Interoperability Organization Web site (listed in Resources).

We use that business scenario to demonstrate the UML to WSDL (with WS-Policy) Transformation feature. Our supplied business scenario sample represents a simplified SCM solution that is based on the WS-I SCM sample scenario. In Figure 9, the supply chain management application makes requests to the retail system to help customers buy electronics goods online. The retailer fulfills stock from the warehouse, and the warehouse replenishes stock from the manufacturers.

First, we use the UML-based modeling feature (described previously, under Modeling Web services metadata based on UML) to model all of the functional aspects (WSDL) and nonfunctional...
aspects (Security, WS-Security Policy) of this scenario’s Web services. Figure 10 shows the metadata structure for all of these services as modeled by UML, which includes the four services.

**Figure 10. UML-based modeling of these Web services**

![UML-based modeling of these Web services](image)

Figure 11 depicts the stereotypes applied to Retailer Service Interface which include TransportBinding assertion of the WS-Security Policy.

**Figure 11. Stereotypes that applied to Retailer Service Interface**

![Stereotypes that applied to Retailer Service Interface](image)

To run the UML to WSDL transformation:

- Right-click the model applied to the Interface Model stereotype in the Model Explorer, and then click **Transform > Run Transformation > UML to WSDL (with WS-Policy)**, as shown in Figure 12.
- When you select a transformation, you will be presented with a dialog where you may enter additional transformation options.
- The first time you run the transformation, be sure to set any Eclipse project in the current workspace as the target of the transformation.
The generated WSDL artifacts for the model used here include the four related WSDL files:

- SCMService.wsdl
- RetailerService.wsdl
- WarehouseService.wsdl
- ManufacturerService.wsdl

We chose the more complex RetailService WSDL as an example to demonstrate UML to WSDL (with WS-Policy) transformation capabilities. The generated RetailerService.wsdl file looks like the code that Listing 3 shows.

**Listing 3. Generated RetailerService WSDL with WS-Policy file**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="RetailerService"

targetNamespace= "http://www.ibm.com/RetailerService.wsdl"


xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:types= "http://www.ibm.com/RetailerService/types/"


xmlns:sp= "http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"


<xsd:complexType>

<xsd:all>

<xsd:element name="CatalogItem"

<xsd:complexType>

<xsd:all>

<xsd:element name="name"

<xsd:element name="description"

<xsd:element name="productNumber"

<xsd:element name="category"

<xsd:element name="brand"

<xsd:element name="price"

</xsd:all>

</xsd:complexType>

</xsd:complexType>

</xsd:complexType>

</xsd:element>

</xsd:complexType>

</xsd:complexType>

</xsd:element>
```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Customer">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="country" type="xsd:string"/>
        <xsd:element name="zip" type="xsd:string"/>
        <xsd:element name="state" type="xsd:string"/>
        <xsd:element name="city" type="xsd:string"/>
        <xsd:element name="street2" type="xsd:string"/>
        <xsd:element name="street1" type="xsd:string"/>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="custnbr" type="xsd:string"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="OrderResponse">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="comment" type="xsd:string"/>
        <xsd:element name="price" type="xsd:integer"/>
        <xsd:element name="quantity" type="xsd:integer"/>
        <xsd:element name="productNumber" type="xsd:integer"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>

  <message name="Retailer_getCatalog_Response">
    <part element="types:CatalogItem" name="response"/>
  </message>
  <message name="Retailer_submitOrder_Request">
    <part element="types:Customer" name="customer"/>
    <part element="types:Order" name="order"/>
  </message>
  <message name="Retailer_submitOrder_Response">
    <part element="types:OrderResponse" name="response"/>
  </message>

  <portType name="Retailer_PortType">
    <operation name="getCatalog">
      <output message="tns:Retailer_getCatalog_Response"/>
    </operation>
    <operation name="submitOrder" parameterOrder="customer order">
      <input message="tns:Retailer_submitOrder_Request"/>
      <output message="tns:Retailer_submitOrder_Response"/>
    </operation>
  </portType>
  <binding name="Retailer_Binding" type="tns:Retailer_PortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    </wsp:Policy>
  </binding>
</xsd:schema>
<wsp:PolicyReference URI="#uuid14cb490c-010a-e5e2-b853-f25c1590aec1"/>
</wsp:Policy>
<operation name="getCatalog">
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body parts="response" use="literal"/>
</output>
</operation>
<operation name="submitOrder">
<input>
<soap:body parts="customer order " use="literal"/>
</input>
<output>
<soap:body parts="response" use="literal"/>
</output>
</operation>
</operation>
</binding>
<service name="Retailer">
<port name="Retailer_Port" binding="tns:Retailer_Binding">
<soap:address location="http://localhostRetailer"/>
</port>
</service>
<wsp:UsingPolicy Required="true"/>
<wsp:Policy Id="#uuid14cb490c-010a-e5e2-b853-f25c1590aec1">
<wsp:ExactlyOne>
<wsp:All>
<sp:TransportBinding ...>
<wsp:Policy>
<sp:TransportToken ...>
<wsp:Policy> ... </wsp:Policy>
...</sp:TransportToken>
<sp:AlgorithmSuite ...> ... </sp:AlgorithmSuite>
<sp:Layout ...> ... </sp:Layout>? 
<sp:IncludeTimestamp .../> ? 
... 
<wsp:Policy> ... </wsp:Policy>
</sp:TransportBinding>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</definitions>

Future articles

This article described the use of UML modeling to model the standardized Web services metadata, which contains functional and nonfunctional aspects. As the introduction explained, there are five different views in a service-oriented model. This article described three views (Message, Service, and Policy). A future article will cover the other two views and their combinations.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample code for this article</td>
<td>plugin.zip</td>
<td>89KB</td>
</tr>
</tbody>
</table>
Related topics

- IBM Rational Software Architect trial version
- Modeling Web Services. Part 2: Modeling and Generating WSDL

© Copyright IBM Corporation 2007
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)