A Model-Driven Development Approach to Creating Service-Oriented Solutions

Simon K. Johnson and Alan W. Brown

IBM Rational, 3039 Cornwallis Road, P.O. Box 12195, RTP, NC 27709, USA
{skjohn, awbrown}@us.ibm.com

Abstract. Many challenges face organizations as they describe their business domains from a services perspective and transform that understanding of their business into a specific realization targeting a solution infrastructure. However, one of the most pressing problems involves helping organizations to effectively transition to service-oriented design of applications. Great benefit could be gained by using a well-defined, repeatable approach to the modeling of business domains from a services perspective that supports the application of automated approaches to realize a service-based solution. In this paper we explore model-driven approaches to the realization of service-oriented solutions. We describe a services-oriented design approach that utilizes a UML profile for software services as the design notation for expressing the design of a services-oriented solution. We describe how a services model expressed in this UML profile can be transformed into a specific service implementation, and describe the design-to-implementation mapping. We then comment on how these technology elements play in an overall MDD approach for SOA.

Keywords: Software design, Model-driven development, Service-oriented Architecture, Unified Modeling Language.

1 Introduction

Many organizations are struggling to improve the flexibility and reduce the maintenance cost of the enterprise solutions that run their businesses. To help them in their task they are looking for ways to move toward solutions that are more readily assembled from existing capabilities, and to develop new capabilities that enable reuse. An approach gaining a lot of support in the industry today is based on viewing enterprise solutions as federations of services connected via well-specified contracts that define their service interfaces. The resulting system designs are frequently called ***Service Oriented Architectures (SOAs)***. Systems are composed of collections of services making calls on operations defined through their service interfaces. Many organizations now express their solutions in terms of services and their interconnections. The ultimate goal of adapting an SOA is to achieve flexibility for the business and within IT.

A number of important technologies have been defined to support an SOA approach, most notably when the services are distributed across multiple machines and connected over the Internet or an intranet. For example web service approaches

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 624 – 636, 2006.
© Springer-Verlag Berlin Heidelberg 2006

rely on intra-service communication protocols such as the Simple Object Access Protocol (SOAP), allow the web service interfaces (expressed in the Web Services Definition Language – WSDL) to be registered in public directories and searched in Universal Description, Discovery and Integration (UDDI) repositories, and share information in documents defined in the eXtensible Markup Language (XML) and described in standard schemas.

Of course, SOA is more than a set of standards and service descriptions. Indeed, it is possible to create an SOA that does not use web services technology, and it is possible to use web services technology in a way that would not be considered service-oriented. There is a great deal more that needs to be explored to understand why a service-oriented viewpoint adds value to the business, and how service-oriented solutions are designed, implemented, deployed, and managed. However, one of the most pressing areas involves helping organizations to effectively transition to service-oriented design of applications. Many challenges face organizations as they describe their business domains from a services perspective and transform that understanding of their business into a specific realization targeting a solution infrastructure. Industry analysts such as Gartner [1] and CBDi [2] place poor services design and the difficulties of educating software practitioners in the techniques of service design at the top of their list of inhibitors to success with developing SOA solutions.

Great benefit could be gained by using a well-defined, repeatable approach to the modeling of business domains from a services perspective that supports the application of automated approaches to realize a service-based solution. Fortunately, we have a rich tradition of modeling and model-driven approaches to software development on which we can draw. Models, modeling, and model transformation form the basis for a set of software development approaches that are known as ***Model-Driven Development (MDD)***. Models are used to reason about the problem domain and the solution domain for some area of interest. Relationships between these models provide a web of dependencies that record the process by which a solution was created, and help to understand the implications of changes at any point in that process. In fact, we can be quite prescriptive in the use of models in a software development process. If we define the kinds of models that must be produced, and apply some rigor to the precise semantics of these models, we can define rules for:

- Automating many of the steps needed to convert one model representation to another;
- Tracing between model elements;
- Analyzing important characteristics of the models.

This style of MDD is called Model-Driven Architecture (MDA). Standards are emerging to support this approach. The primary driving force behind MDA approaches based on a standardized set of models, notations, and transformation rules is the Object Management Group (OMG). They provide an open, vendor-neutral basis for system interoperability via OMG's established modeling standards: Unified Modeling Language (UML), Meta-Object Facility (MOF), and Common Warehouse Meta-model (CWM). Platform-independent descriptions of enterprise solutions can be built using these modeling standards and can be transformed into a major open or proprietary platform, including CORBA, J2EE, .NET, XMI/XML, and Web-based platforms [3].

In this paper we explore model-driven approaches to the realization of service-oriented solutions. We describe a services-oriented design approach that utilizes a UML profile for software services as the design notation for expressing the design of a services-oriented solution. We describe how a services model expressed in this UML profile can be transformed into a specific service implementation, and define the design-to-implementation mapping. We then comment on how these technology elements play in an overall MDD approach for SOA.

2 Model-Driven Generation of Services and Service-Oriented Solutions

Defining and applying model transformations are critical techniques within any model-driven style of development. Model transformations involve using a model as one of the inputs in the automation process. Possible outputs can include another model, or varying levels of executable code. In practice there are three common model transformations: refactoring transformations, model-to-model transformations, and model-to-code transformations [4, 5].

1. *Refactoring transformations* reorganize a model based on some well-defined criteria. In this case the output is a revision of the original model, called the refactored model. An example could be as simple as renaming all the instances where a UML entity name used, or something more complex like replacing a class with a set of classes and relationships in both the metamodel and in all diagrams displaying those model elements.
2. *Model-to-model transformations* convert information from one model or models to another model or set of models, typically where the flow of information is across abstraction boundaries. An example would be the conversion of one type of model into another, such as the transformation of a set of entity classes into a matched set of database schema, Plain Old Java Objects (POJOs), and XML-formatted mapping descriptor files.
3. *Model-to-code transformations* are familiar to anyone who has used the code generation capability of a UML modeling tool. These transformations convert a model element into a code fragment. This is not limited to object-oriented languages such as Java and C++. Nor is it limited to programming languages: configuration, deployment, data definitions, message schemas, and others kinds of files can also be generated from models expressed in notations such as UML. Model-to-code transformations can be developed for nearly any form of programming language or declarative specification. An example would be to generate Data Definition Language (DDL) code from a logical data model expressed as a UML class diagram.

2.1 Applying Model Transformations

Having described different kinds of model transformations, we also note that in practice there are several ways in which model transformations can be applied. In model-driven approaches there are four categories of techniques for applying model transformations:

- *Manual*. The developer examines the input model and manually creates or edits the elements in the transformed model. The developer interprets the information in the model and makes modifications accordingly.[1]
- *Prepared Profile*. A profile is an extension of the UML semantics in which a model type is derived. Applying a profile defines rules by which a model is transformed.
- *Patterns*. A pattern is a particular arrangement of model elements. Patterns can be applied to a model and results in the creation of new model elements in the transformed model.
- *Automatic*. Automatic transformations apply a set of changes to one or mode models based on predefined transformation rules. These rules may be implicit to the tools being used, or may have been explicitly defined based on domain-specific knowledge. This type of transformation requires that the input model be sufficiently complete both syntactically and semantically, and may require models to be marked with information specific to the transformations being applied.

The use of profiles and patterns usually involves developer input at the time of transformation, or requires the input model to be "marked". A marked model contains extra information not necessarily relevant to the model's viewpoint or level of abstraction. This information is only relevant to the tools or processes that transform the model. For example, a UML analysis model containing entities with String types may be marked variable or fixed length, or it may be marked to specify its maximum length. From an analysis viewpoint just the identification of the String data type is usually sufficient. However, when transforming a String typed attribute into, say, a database column type, the additional information is required to complete the definition.

2.2 Models and Transforms

Transformations such as these can be used to enable efficient development, deployment and integration of services and services-oriented solutions. Practitioners create models specific to their viewpoint and needs which are used as the basis of analysis, consistency checking, integration, and automation of routine tasks. Model-driven approaches allow developers to create services and service-oriented solutions by focusing on logical design of services and to apply transformations to the underlying SOA technologies. Furthermore, as illustrated in the examples later in this paper, substantial improvements in the quality and productivity of delivered solutions is possible by automating substantial aspects of these transformations to service implementations**.**

3 Model-Driven Service Specification and Design

The Unified Modeling Language (UML) is the standard modeling notation for software-intensive systems [6]. Originally conceived over a decade ago as an integration of the most successful modeling ideas of the time, the UML is widely used by organizations, and supported by more than a dozen different product offerings. Its evolution is managed through a standards process governed by the Object Management Group (OMG).

[1] Apart from raw speed, the significant difference between manual and automated transformations is that automation is guaranteed to be consistent, while a manual approach is not.

One of the reasons for the success of UML is its flexibility. It supports the creation of a set of models representing both the problem domain and solution domain, can capture and relate multiple perspectives highlighting different viewpoints on these domains, enables modeling of the system at different levels of abstraction, and encourages the partitioning of models into manageable pieces as required for shared and iterative development approaches. In addition, relationships between model elements can be maintained across modeling perspectives and levels of abstraction, and specialized semantics can be placed on model elements through built-in UML extension mechanisms (i.e., stereotypes and tagged values bundled into UML profiles).

One recent effort at IBM has been to create a UML profile for software services, a profile for UML 2.0 which allows for the modeling of services, service-oriented architecture (SOA), and service-oriented solutions.[2] The profile has been implemented in IBM Rational Software Architect, used successfully in developing models of complex customer scenarios, and used to help educate people about the concerns relevant to developing service-oriented solutions. This profile is used as the basis for a model-driven approach in which services and service interactions are

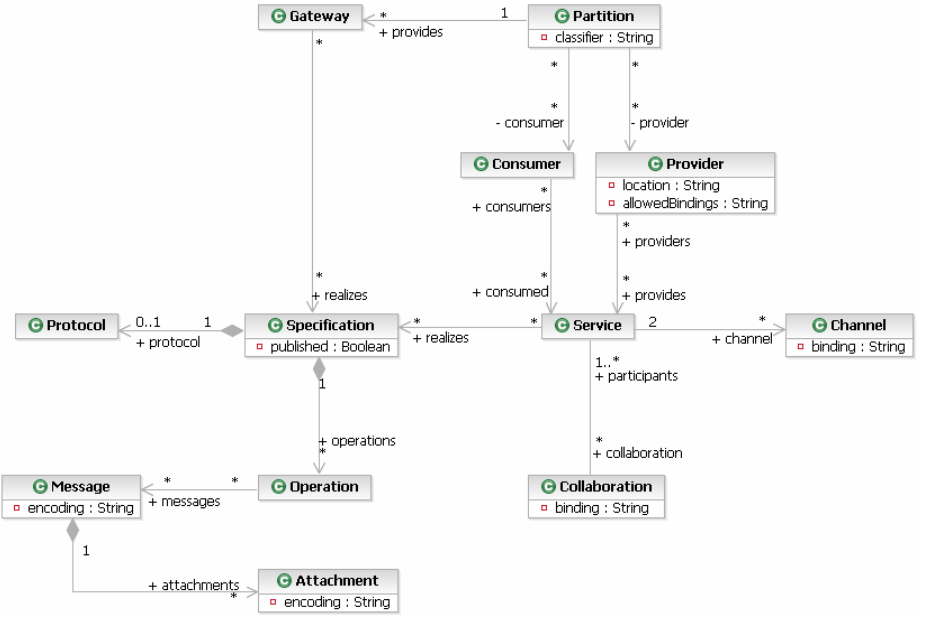

Fig. 1. Conceptual Model

described using the UML profile, and realizations of those services are then generated through an automated model transformation to Web Services Definition Language (WSDL), a standard language for web services implementation.

[2] Details of this UML profile and a downloadable version of the profile for Rational Software Architect are available at http://www.ibm.com/developerworks/rational/library/05/419_soa.

3.1 The UML Profile for Software Services

The UML profile for software service provides a common language for describing services that covers a number of activities through the development lifecycle and also provides views of those services to different stakeholders [7]. So, for example, the profile provides capabilities for the architect to map out services early in the lifecycle using logical partitions to describe the entire enterprise-wide service portfolio. This view is further detailed by designers who develop the service specifications, both structural and behavioral that act as the contracts between the service clients and implementers. The message view provides the ability for designers to reuse information models for common service data definitions. The profile has been implemented in Rational Software Architect and used successfully in developing models of complex customer scenarios and also in educating people to the concerns relevant to development of service-oriented solutions.

Figure 1 is a model showing the concepts important in modeling services. As you can see the number of concepts is relatively small and should be reasonably familiar to anyone having worked on service-oriented solutions.

Note, however that although the profile is a realization of this model a number of the concepts are not explicit stereotypes in the profile. For example there is no stereotype for operation or for protocol as these are existing notions in the UML 2.0 that the profile reuses without any ambiguity or further constraint.

The following table lists the elements of the UML 2.0 meta model that are used as meta classes for stereotypes in the UML Profile.

UML 2.0 Meta Class	Stereotypes
Class	Message, Service Partition, Service Provider
Classifier	Service Consumer
Collaboration	Service Collaboration
Connector	Service Channel
Interface	Service Specification
Port	Service, Service Gateway
Property	Message Attachment

3.2 The Profile Defined

To describe the concepts supported by the UML profile for software services we can begin by looking at the details of the profile itself. Figure 2 is a UML 2.0 profile diagram, it illustrates the details of the profile with each stereotype, its meta class using the extension notation (filled arrow head). Additionally, a number of constraints in the model are described, particularly those co-constraints between profile elements.

By reference to Figure 2 we can review the key elements of how services and service interactions are defined in the profile, and some of the expected uses and constraints when modeling services and service interactions using the profile.

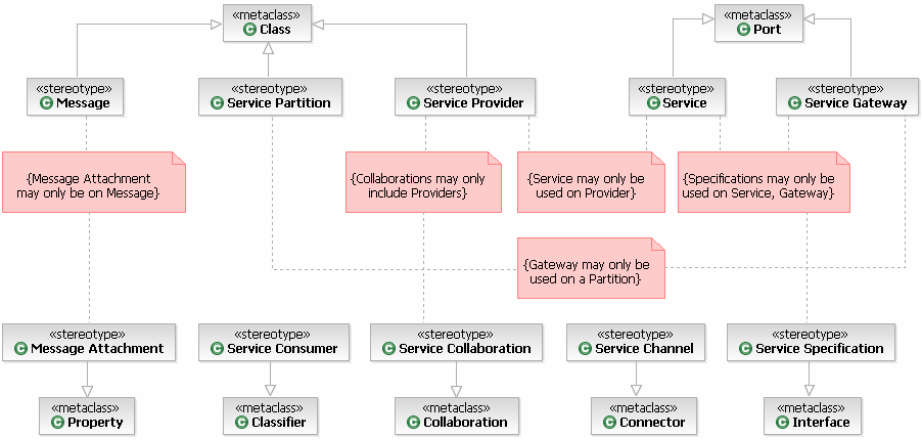

Fig. 2. UML 2.0 profile

- **Message:** A message represents the concept of a container for actual data which has meaning to the service and the consumer. A message may not have operations, it may have public properties and associations to other classes (one assumes classes of some domain model). A message stereotype has a property to denote it's assumed encoding form (i.e. "SOAP-literal", "SOAP-rpc", "ASN.1", etc.). The use of this element may be optional in a tool for two reasons. Firstly the modeler may simply wish to use elements from a domain model directly as the parameters to an operation rather than specifying a message. Secondly the modeler may wish to use the convention of specifying a set of input and output messages on an operation, in which case the modeling tool would have to construct an input and output message matching the parameters when generating service descriptions in WSDL.
- **Message Attachment:** This is used to denote that some component of a message is an attachment to the message as opposed to a direct part of the message itself. In general this is not likely to be used greatly in higher level design activities, but for many processes attached data is important to differentiate from embedded message data. A message attachment stereotype may only be used on properties owned by classes stereotyped as message. The stereotype also has a property to denote its assumed encoding form (i.e. "SOAP-literal", "SOAP-rpc", "ASN.1", etc.). For example, a catalog service may return general product details as a part of the structured message but images as attachments to the message; this also allows us to denote that the encoding of the images is binary as opposed to the textual encoding of the main message.
- **Service:** The service model element provides the end-point for service interaction (in web service terminology) whereas the definition of these interactions is a part of the service specification. In the model a service not only identifies the provided interface, but may also identify required interfaces (such as callback interfaces). This stereotype may only be used on ports owned by Classes or Components stereotyped as Service Provider. A service has an additional property that denotes the binding to be used, such as "SOAP-HTTP", "SOAP-JMS", etc.

- **Service Channel:** A channel represents the communication path between 2 services, importantly it is the channel over which interaction may occur and does not represent any particular interaction. In the web services world each service denotes the binding(s) associated with it so that a client may access it. In the modeling profile we denote binding on the communication between services or between a service and consumers. In this way we can be flexible in understanding the binding requirements. The stereotype has a "channel" property denoting the platform binding mechanism to use in generating the service binding in WSDL; examples might be SOAP-RPC, SOAP-Doc, HTTP-Get, and so on.
- **Service Collaboration:** A service collaboration is a way of specifying the implementation of a service as a collaboration of other services. From a web services point of view this corresponds to the use of BPEL4WS in specifying service implementation. A service collaboration is used as the behavior of a service and, if it is intended to generate to a language such as BPEL it may have other implementation-specific constraints. Participants in a Service Collaboration may be Service Consumers, Service Providers or Service Specifications.
- **Service Consumer:** Any classifier (Class, Component, etc.) may act as the consumer of a service, and that includes another service. While this stereotype is most definitely optional it may be useful in identifying elements of a model that are not services themselves as clients of services. On the other hand it may be overhead and not used.
- **Service Gateway:** A service gateway looks like a service but is only available for use on partitions and not service providers. A gateway acts as a proxy service and can be used to mediate protocols or denote the interface available to a partition. This stereotype may only be applied to a port owned by a Service Partition. For example, we might denote that although a number of services are implemented within a partition only some are available for use outside the partition and so gateways are provided for these services. This disallows other services or partitions from communicating to services that are not exposed via gateways.
- **Service Model:** The Service Model is an architectural view of an SOA highlighting the major elements of a system based on an SOA style.
- **Service Partition:** A service partition represents some logical or physical boundary of the system. It is optional to model partitions, but frequently this concept is very useful. For example, partitions could be used to represent the web, business and data tiers of a traditional n-tier application. Partitions might also be used to denote more physical boundaries such as my primary data center, secondary site, customer site, partners etc. in which case the crossing of partitions may have particular constraints for security, allowed protocols, bandwidth and so on. A partition may only have properties that represent nested parts, be they services or other partitions (this is a constraint - no other elements may currently be represented in a partition). A partition may not have any owned properties, operations or behaviors. It may have parts that are either Service Providers or Service Specifications. This stereotype also has a property "classifier" which is used to group partitions that represent similar concepts. A partition also has the notion of being "strict", if a partition denotes that all communication between it and other partitions must be directed through typed gateways then it is said to be a strict partition.

- **Service Provider:** The Service Provider is a software element that provides one or more services. In modeling terms one would most usually expect to see a UML component here, however such a restriction seems arbitrary and so the metaclass is noted as Class for more flexibility. A service provider has a property that captures information about its location although the meaning of this is implementation dependent. The Class acting as the service provider may not expose any attributes or operations directly, only public ports may be provided (stereotyped as service) and these are typed by service specifications (or classes realizing service specifications).
  A Service Provider may not have any directly owned properties, operations or behaviors. All operations are provided through the ports on the provider. The location property, while implementation/platform specific is useful in generating service endpoint names. For example with WSDL the location may be `http://svc.myco.com/` and a service might be called CustInfo, in which case the endpoint name for the service could be generated as `http://svc.myco.com/CustInfo`.
- **Service Specification:** The use of an interface denotes a set of operations provided by a service; note that a service may implement more than one interface. By convention it is possible to attach a protocol state machine or UML 2.0 Collaboration to such a specification to denote the order of invocation of operations on a service specification. With such a behavioral specification any implementing service can be validated against not only a static but dynamic specification of its structure and behavior. Note that the service specification may only provide public operations. The service specification may not have owned properties and all operations shall be public. The stereotype also has a property "published" that denotes whether the service is assumed to be published into a service repository; this is a different notion from the public/private property provided by UML.

4 The Role of a Service Model in MDD

A description of services and service assemblies using the UML profile for software services is useful in that it provides:

- A consistent set of concepts, notations, and semantics for modeling services and service interactions.
- A first-class "domain-specific language" for service modeling to support designers of SOAs as they design and reason about their solution.
- A language that can act as the target for transformations from more abstract, business-focused analysis models that will result in a logical service design model used as the basis for a service-oriented solution.
- A consistent basis for generating service realizations from the logical service model.

These observations are particularly important in terms of supporting a model-driven approach to service-oriented solutions. The classical approach of mapping from Computation Independent Model (CIM) to Platform Independent Model (PIM) to Platform Specific Model (PSM) can be instantiated by equating these three levels

to a business analysis model, services model, and services implementation, respectively. Mappings between these models can be defined as the basis for automation in the form of model transformations. Here, we briefly discuss the CIM-PSM mapping before a more offering a more detailed concrete example of a PIM to PSM mapping for model-driven design of services-oriented solutions.

4.1 Generation of a Service Model from an Analysis Model

In many projects detailed business-level analysis takes place to understand the business context within which any IT solution must operate. The resulting business analysis model is useful as documentation of the business needs and goals. However, in addition service models can be generated from business analysis models. A simple example of a business analysis model consistent with the business modeling discipline defined in the Rational Unified Process (RUP) is illustrated in Figure 3.

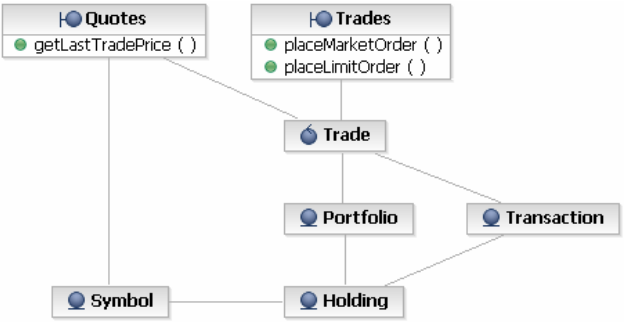

Fig. 3. RUP Analysis Model for Stock Portfolio

A transformation from this model to the service model shown later in Figure 4 is relatively easy to specify. A number of interesting aspects of that transformation are of note. For example, the transformation crosses abstraction boundaries and so we would expect transformations of types and some structures. In this case we would expect the boundary elements to be transformed to service specifications and service providers with Entities that are operation parameters being transformed as messages. Control elements and internal Entities would not be transformed to the service model, they are appropriate to the implementation of the service and not its specification.

4.2 Generation of WSDL from the Service Model

Based on this profile we are able to define specific model transformations that convert service models expressed using the profile into various target languages for service realization. Of particular interest is the Web Services Definition Language (WSDL), a standard maintained by the World Wide Web Consortium (W3C). WSDL is "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information."[3] Improvements

[3] See the WSDL description at the W3C website: http://www.w3.org/TR/wsdl.

in the design of service implementations can be gained by being able to model the logical design of services-oriented solutions and generating the WSDL service specifications from these models by applying a well-defined set of model transforms.

Figure 4 illustrates a simple stock quote service example (derived from an example discussed in the WSDL specification document). In this example the Service Specification provides the structural interface specification, the Message elements model the types of the structures passed into and returned from the operations, and a service provider realizing the specification.

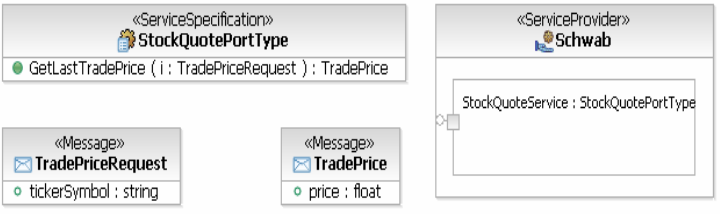

Fig. 4. UML Model for Stock Quote Service

From this model we can, with a relatively simple set of transformation rules, generate a WSDL interface definition. A number of possible transformation approaches are possible, but a specific example is shown in Listing 1 below.

```
<?xml version="1.0"?>
<definitions name="StockQuote"
  targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd1="http://example.com/stockquote.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
        <complexType>
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="TradePrice">
        <complexType>
          <all>
            <element name="price" type="float"/>
          </all>
        </complexType>
      </element>
    </schema>
  </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>
```

Listing 1. Generated WSDL for Stock Quote Service

```
  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>

  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetLastTradePrice">
        <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
        <input>
          <soap:body use="literal"/>
        </input>
        <output>
          <soap:body use="literal"/>
        </output>
      </operation>
    </binding>

    <service name="StockQuoteService">
      <documentation>My first service</documentation>
      <port name="StockQuotePort" binding="tns:StockQuoteBinding">
        <soap:address location="http://example.com/stockquote"/>
      </port>
    </service>

</definitions>
```

Listing 1. *(Continued)*

The mapping demonstrated here relies on some relatively simple translation rules that can readily be automated using a model transformation engine, such as the one embedded in the IBM Rational Software Architect product:

- The Service Provider is used to denote a logical grouping of services and therefore it provides the scope for the outer WSDL definitions element.
- Each Service Specification becomes a WSDL port type, all operations are transformed to WSDL operations defined for the port type.
- All UML2 ports on the service provider become WSDL service definitions, the port types already derived above.
- A single message is created for input parameters and a single message created for returned data; each message has a single associated XML Schema element that aggregates any parameters on the original operation. Best practice is still to define a single class in the model for all information required by an operation (using the Message stereotype).

The example above does not demonstrate how the WSDL bindings are derived – these are derived from the connectors between services or services and their consumers modeled with UML collaborations.

5 Summary

Creating solutions for SOA means rethinking the kinds of systems being built today, reconsidering the skills in an organizations, and redefining the ways in which members of teams collaborate. Most importantly, adopting a service-oriented to development of solutions requires a broader review of its impact on how solutions are designed, what it means to assemble them from disparate services, and how deployed services-oriented applications are managed and evolved.

In this paper we have focused on UML profile offering a common notation and semantics for software service modeling, and the use of this profile for generating service implementations in WSDL. This technology is part of a much broader set of best practices for creating service-oriented solutions, captured as process guidance in the Rational Unified Process (RUP), and supported though automated tools in the IBM Rational Software Development Platform. As we gain experience in the design and realization of service-oriented solutions, we are continually documenting them for external validation (e.g., [8, 9]), adding new modeling support for service design in IBM products, and describing new techniques that help organizations repeatably deliver high quality solutions.

References

1. Daryl Plummer, “Six Missteps That Can Result in SOA Strategy Failure”, Gartner Research Report, June 2005.
2. John Dodd, “Practical Service Specification and Design”, CBDi Series, www.cbdiforum.com, May 2005.
3. OMG, “MDA Guide v1.0.1”, Available at http://www.omg.org/docs/omg/03-06-01.pdf 12th June 2003.
4. A.W. Brown, J. Conallen, D. Tropeano, "Models. Modeling, and Model Driven Development", in *Model-Driven Software Development*, pages 1-17, S. Beydeda, M. Book, V. Gruhn (Eds.), Springer Verlag, 2005.
5. A.W. Brown, J. Conallen, D. Tropeano, "Practical Insights into MDA: Lessons from the Design and Use of an MDA Toolkit", in *Model-Driven Software Development*, pages 403-432, S. Beydeda, M. Book, V. Gruhn (Eds.), Springer Verlag, 2005.
6. Jim Rumbaugh, Grady Booch, Ivar Jacobsen, “The UML 2.0 Reference Manual”, Second Edition, Addison-Wesley, 2005.
7. S. Johnston, “Modeling Service-oriented Solutions”, IBM Developerworks, http://www.ibm.com/developerworks/rational/library/jul05/johnston/, July 2005.
8. A.W. Brown, S. Iyengar, S.K. Johnson, "A Rational Approach to Model-Driven Development", IBM Systems Journal, pages 463-480, Vol. 44, #4, July 2006.
9. A.W. Brown, M. Delbaere, P. Eeles, S. Johnston, R. Weaver, "Realizing Service oriented Solutions with the IBM Software Development Platform", IBM Systems Journal, pages 727-752, Vol. 44, #4, October 2005,