UML 2.0 Profile for Software Services

Simon Johnston

April 13, 2005

This article describes the 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. The profile has been implemented in IBM Rational Sofware 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.

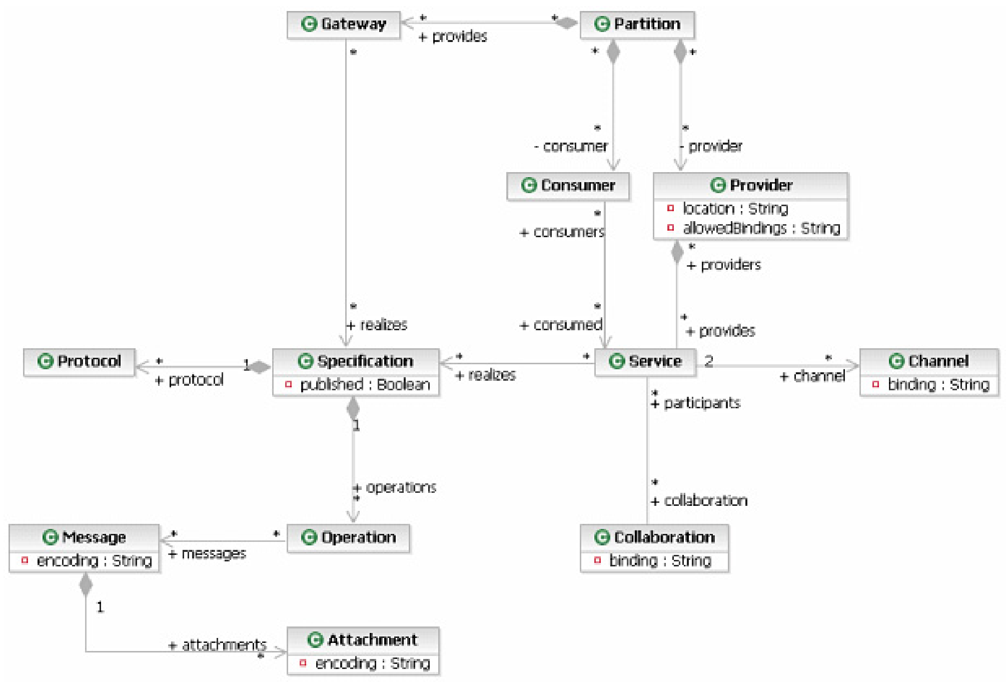
Overview of the UML profile for software services

The aim of the profile, implemented in IBM Rational Sofware Architect, is to provide a common language for describing services, one which covers a number of activities through the development lifecycle and also provides views to different stakeholders. For example, the profile provides capabilities for architects 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 services' clients and implementers. The *message* view provides the ability for designers to reuse information models for common service data definitions.

Conceptual model

Figure 1, following, is a diagram showing the important concepts in modeling services. As you can see the number of concepts is relatively small, and should be reasonably familiar to anyone who has worked on service-oriented solutions. The UML 2.0 profile this diagram represents has been implemented in IBM Rational Sofware 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. However, although the profile is a realization of this model, note that 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 UML 2.0 that the profile reuses without any ambiguity or further constraint.

Figure 1. Modeling services



Identified subset of UML 2.0

Table 1 lists the elements of the UML 2.0 meta model that are used as meta classes for stereotypes in the UML Profile.

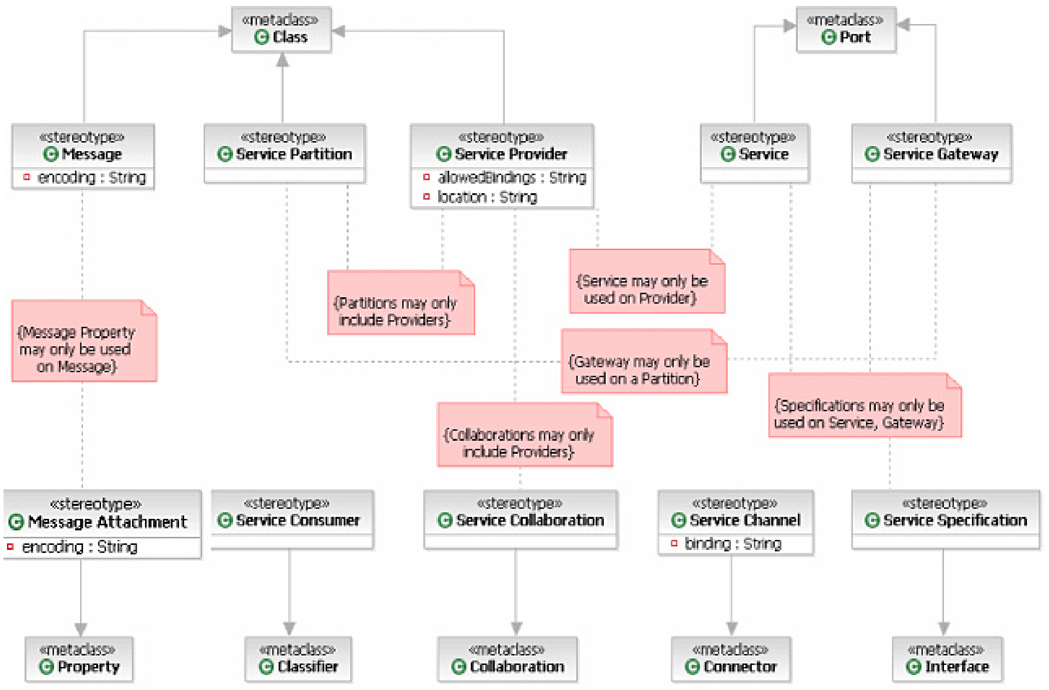
Table 1. Elements of the UML 2.0 meta model

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

The profile itself

Figure 2 (clickable) is a UML 2.0 profile diagram. It demonstrates the actual details of the profile, showing each stereotype with its meta class using the extension notation (filled arrow head).You can also see some of the constraints in the model, particularly those co-constraints between profile elements.

Figure 2. A UML 2.0 profile diagram


«metaclass» Class
«metaclass» Port
«stereotype» Message
encoding : String
«stereotype» Service Partition
«stereotype» Service Provider
allowedBindings : String
location : String
«stereotype» Service
«stereotype» Service Gateway
{Service may only be used on Provider}
{Partitions may only include Providers}
{Message Property may only be used on Message}
{Gateway may only be used on a Partition}
{Specifications may only be used on Service, Gateway}
{Collaborations may only include Providers}
«stereotype» Message Attachment
encoding : String
«stereotype» Service Consumer
«stereotype» Service Collaboration
«stereotype» Service Channel
binding : String
«stereotype» Service Specification
«metaclass» Property
«metaclass» Classifier
«metaclass» Collaboration
«metaclass» Connector
«metaclass» Interface

The following sections outline the elements of a UML 2.0 profile as it is currently defined. Each section outlines an individual stereotype, the details of which specify its meta class, properties, and any constraints which should be applied when using the profile.

Stereotype *message*

Extends

Class

Semantics

A message represents a concept as defined in the WSDL (Web Services Description Language) specification; that is, *it's a container for actual data which has meaning to the service and the consumer*. A message may not have operations, but it may have properties and associations to other classes (likely of some domain model). A message stereotype has a property to denote its assumed encoding form (for example, `SOAP-literal, SOAP-rpc, ASN.1`, and so on).

Properties

Table 2 lists the properties of a message stereotype.

Table 2. Properties of a message

Kind	Name	Type	Description

Property	encoding	String	Denotes the platform encoding mechanism to use in generating the schema for the message; examples might be `SOAP-RPC, Doc-Literal, ASN.1`, and so on.

Notation

Constraints

- Shall not have any owned operations
- Shall not have any owned behaviors

Stereotype *message attachment*

Extends

Property

Semantics

This stereotype is used to denote that *some component of a message is an attachment to it* (as opposed to a direct part of the message itself). In general, you're not likely to use this stereotype much in higher level design activities, but for many processes it's important to differentiate attached data from embedded message data. For example, a catalog service may return general product details as a part of the structured message, but return images as attachments to the message. This also allows you to denote that the encoding of the images is binary (as opposed to the textual encoding of the main message).

Properties

Table 3 lists the properties of a message attachment stereotype.

Table 3. Properties of a message attachment

Kind	Name	Type	Description
Property	encoding	String	Denotes the platform encoding mechanism to use in generating the schema for the message; examples might be `SOAP-RPC, Doc-Literal, ASN.1`, and so on.

Notation

Constraints

- Shall only be used on properties owned by classes stereotyped as <<Message>>

Stereotype service

Extends

Port

Semantics

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 stereotype. In your model, a service not only identifies the provided interface, but may also identify required interfaces (such as callback interfaces). A service has an additional property that denotes the binding to be used, such as SOAP-HTTP, SOAP-JMS, and so on.

Properties

None

Notation

Constraints

- Shall only be used on a class stereotyped as <<Service Provider>>
- Shall be typed by an interface stereotyped as <<Service Specification>>

Stereotype service channel

Extends

Connector

Semantics

A channel represents *the communication path between two services*. It is important to note that interaction may occur over a channel, but the channel 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 a modeling profile, you denote binding either on the communication between services or between a service and consumers. In this way, you can be flexible in understanding the binding requirements.

Properties

Table 4 lists the properties of a service channel stereotype.

Table 4. Properties of a service channel

Kind	Name	Type	Description
Property	binding	String	Denotes 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.

Notation

Constraints

- At *least* one end of the connector shall connect to a port stereotyped as <<Service
- At *most* one end of the connector may connect to a port stereotyped as <<Service Gateway>>, a class stereotyped as <<Service Provider>>, or a classifier stereotyped as <<Service Consumer>>

Stereotype service collaboration

Extends

Collaboration

Semantics

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 (Business Process Execution Language for Web Services) in specifying service implementation. So, this means that you use a service collaboration 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.

Properties

None

Notation

Constraints

- Collaboration participants shall only be classes stereotyped as <<Service Provider>>

Stereotype service consumer

Extends

Classifier

Semantics

Any classifier (class, component, and so on) *may act as the consumer of a service*, and that includes another service. While this stereotype is most definitely optional, it may help you identify elements of a model -- that are not services themselves -- as *clients* of services. On the other hand, it may just be overhead and you don't need to use it.

Properties

None

Notation

Constraints

None

Stereotype service gateway

Extends

Port

Semantics

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 you can use it to mediate protocols or to denote the interface available to a partition. For example, you might denote that -- although a number of services are implemented within a partition -- only some of them are available for use outside the partition, and therefore gateways are provided for these services. This disallows other services or partitions from communicating to services that are not exposed via gateways.

Properties

None

Notation

Constraints

- Shall only be used on a class stereotyped as <<Service Partition>>
- Shall be typed by an interface stereotyped as <<Service Specification>>

Stereotype service partition

Extends

Class

Semantics

A partition represents *some logical or physical boundary of the system*. It is optional but useful to model partitions. For example, you could use partitions to represent the web, business, and data tiers of a traditional n-tier application. You might also use partitions to denote more physical boundaries (such as my primary data center, secondary site, customer site, partners, and so on), 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. Note that *this is a constraint* -- no other elements may currently be represented in a partition.

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*.

Properties

None

Notation

Constraints

- Shall not have any owned properties
- Shall not have any owned operations
- Shall not have any owned behaviors
- Any owned part shall be a class stereotyped <<Service Provider>>

Stereotype service provider

Extends

Class

Semantics

The service provider is *a software element that provides one or more services*. In modeling terms, you 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.

The location property -- while implementation- or 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.

Properties

Table 5 lists the properties of a service provider stereotype.

Table 5. Properties of a service provider

Kind	Name	Type	Description
Property	allowedBindings	String	Denotes the allowed platform binding mechanism a channel may use in connecting to the service; examples might be `SOAP-RPC, SOAP-Doc, HTTP-Get`, etc.
Property	location	String	The location of the provider, this may be used by generators to create endpoint names.

Notation

Constraints

- Shall not have any owned properties
- Shall not have any owned operations
- Shall not have any owned behaviors
- Any owned port shall be stereotyped <<Service>>

Stereotype service specification

Extends

Interface

Semantics

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, and each operation should only consume at most one message and produce at most one message.

Properties

Table 6 lists the properties of a service specification stereotype.

Table 6. Properties of a service specification

Kind	Name	Type	Description
Property	published	Boolean	This property 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.

Notation

Constraints

- Shall not have any owned properties
- All operations shall be marked as `public`
- May only be used as the type for a port stereotyped as <<Service>> or <<Service Gateway>>