Introduction to the SIP Modeling Toolkit

Jim Conallen

June 17, 2008
(First published August 07, 2007)

The SIP Modeling Toolkit is a set of domain extensions to the Rational modeling platform for the development of Session Initiation Protocol (SIP) services. It can be used in IBM® Rational® Software Architect, IBM® Rational® Systems Developer or IBM® Rational® Software Modeler. SIP is rapidly becoming a key protocol in the use of the internet as backbone for multi-media applications like voice over IP (VoIP), video conferencing, and instant messaging. The toolkit provides extensions to UML sequence diagrams to model SIP call flows, and can automatically create call flows by importing network log files. The functionality includes the ability to merge multiple call flows into state machines, making them easier to comprehend. Call flow diagrams can be exported to IBM® Rational® Performance Tester Extension for SIP thereby making it easier to create SIP test cases, but also help establish the connection between analysis, design and testing artifacts. The toolkit also provides extensions to class diagrams, enabling the modeling of JSR 116 compliant SIP servlet applications and the generation of deployment descriptors and skeletal code.

Introduction

The Session Initiation Protocol (SIP) is an Internet Engineering Task Force (IETF) specification (RFC3261) for connecting one or more multimedia participants (user agents) together over the Internet. User agents are typically phones (POTS, mobile phones, soft phones, IP-PBX devices, and so on), but can be any SIP-enabled device.

The protocol is responsible for creating, modifying, and terminating sessions, but is not actually involved with interpreting the media exchange. Once the devices are connected (that is, they are exchanging a media stream directly between each other), the SIP service is no longer involved. However, SIP services provide value beyond just connecting two agents. For example, while setting up or tearing down a connection, SIP services can also implement call blocking or forwarding, as well as interface with gaming and back office systems.

The SIP Modeling Toolkit adds SIP-specific extensions to the UML modeling and development platform provided by the Rational modeling platform. These extensions include domain-specific profiles, reference models, transforms, and other utilities for modeling and model-driven development (MDD) of next-generation communications services. The toolkit also provides code
generation extensions to the existing UML-to-Java™ transforms within Rational Software Architect that update the SIP-specific artifacts of a service implementation.

Therefore, this toolkit makes it easier to create design-level models that target code generation for SIP servlets. It does this by providing domain relevant reference models, as well as stereotyped elements and properties, that capture SIP-specific information (such as SIP filter patterns) in the servlet models. The toolkit then uses those models, elements, and properties to generate code transformations.

Using the SIP Modeling Toolkit, SIP and converged SIP-HTTP services can be created and deployed on the IBM® WebSphere® Application Server product as SIP servlets. It is also possible to customize the toolkit for deployment on other SIP application server platforms. The implementation of SIP services by Java™ Servlets is specified by the Java™ Community Process JSR 116. WebSphere Application Server Version 6.1 and later provides native support for JSR 116. This article introduces the toolkit and outlines its key features.

**Toolkit functionality**

The SIP Modeling Toolkit addresses three key areas of SIP service modeling:

- **Call flow modeling**: Rapid modeling of SIP call flows using enhanced sequence diagrams. Import network log files to populate actual SIP call flows. Combine call flows into emergent state machines.
- **Servlet modeling**: Design SIP and converged servlet applications that integrate in the context of your applications.
- **Test case integration**: Generate SIP test cases for Rational Performance Tester Extension for SIP from modeled call flow diagrams.

This toolkit does not presuppose any specific methodology for developing SIP-based services. Using call flow modeling or servlet modeling is optional, and whereas they can be used together successfully, there is no dependency between them. On the other hand, model-to-code transformations do depend on servlet models and the correct use of the related extensions.

**SIP Design Model template**

SIP design models require that the UML model have SIP reference types and a SIP profile. The SIP Modeling Toolkit installs a new reference model and profile that can be applied to any existing UML model. Or to simplify things, the toolkit includes a model template that can be used to create new model files with the reference model and profile already applied. The SIP reference model includes most of the SIP types used in a design model (from JSR116 and WebSphere Application Server extensions), along with pre-built SIP messages for use in call flow diagrams. The SIP profile includes stereotypes for servlets and UML collaborations used for call flow modeling. The SIP Design Model template is available from the list of templates available when you create a new UML model, as shown in Figure 1.
Call flow modeling

One of the most common ways to communicate a specific scenario of activity between user agents and SIP services is with a call flow diagram. These diagrams are, for the most part, UML sequence diagrams, as shown in Figure 2. The SIP User Agent (UA) typically initiates the call flow with an `INVITE` message to a SIP-aware service. This service usually forwards the messages on to other SIP services, or translates the messages and invokes other types of equipment (like the call control elements in the control plane).
Most of the information exchanged between the participants is in the header content. The SIP Modeling Toolkit provides a specialized user interface (UI) to capture this extra information (see Figure 3). User interface customizations make it easy to copy headers from other messages and reorganize header order. One button will create or update the content length header based on the actual content specified in the message body.

In addition to modeling SIP messages the SIP Modeling toolkit also supports the modeling of HTTP headers and content in call flows diagrams, independent of or mixed with SIP messages (Figure 4). For converged applications that involve both SIP and HTTP this helps visualize the coordination and interaction between these two related protocols.
Importing SIP call flows from network log files

A new feature in this second release of the SIP Modeling Toolkit is the ability to import call flows from actual network log files. Network traffic files stored in PCAP form can be directly imported into SIP call flows. PCAP files can be recorded with commonly available programs like UNIX’s tcpdump, or Wireshark (formerly known as Ethereal). Imported call flows include all the header and body content details that are actually transmitted. With this functionality actual running systems can be analyzed in a significantly easier way, than by looking only at the raw text based dumps of traffic. Since the call flows are no different than call flows created manually they too can be modified, and updated to represent the desired behavior of a service under development.

Importing SIP call flows from SIPp files

Another new feature of the SIP Modeling Toolkit is the ability to import the core scenarios from SIPp configuration files. SIPp is a popular open source traffic generator program that has been in use for a number of years. Its primary purpose is to generate traffic and evaluate the results on a pass/fail basic. SIPp is simple to use and very efficient. While it doesn’t offer all the full featured testing capabilities of a product like Rational Performance Tester Extensions for SIP, it is a very useful tool for unit testing and debugging.

The SIP Modeling Toolkit can import the majority of scenario information from SIPp configuration files into call flow diagrams. Since there are a few SIPp specific elements of functionality that are not part of SIP Modeling Toolkit managed call flows, not all of the information in the SIPp files is imported. The core scenario however is captured, and some heuristics are used to reconstruct the lifelines of the external participants of the scenario. While the modeling toolkit doesn’t claim to preserve all SIPp content it does capture enough of the scenario to provide a good visualization.
of the scenario in call flow form, and does provide a starting point for downstream design, development and testing activities.

**Call flow merging into state machines**

Call flows represent key scenarios that a SIP service and how it is expected to behave. A real life service behaves like many call flows simultaneously. One way to get a better understanding of how a single service might exhibit the combined behavior of multiple call flows is to merge them into a single state machine. That state machine would then represent the single service, behaving like each call flow.

Take as an example the two following call flow diagrams (Figure 5). In the first scenario an attempted call is blocked, and an instant message is sent to the intended caller. In the second the call flow the call is proxied. Some of the messages in the call flows are optional (indicated with a property in the custom property view). By tagging a lifeline with a common keyword in each diagram, we define the focus of the combined behavior. With the common lifelines tagged we can generate the emergent state machine. This machine (Figure 6) represents the combined behavior of all the call flow diagrams into a single machine.

**Figure 5. Multiple call flows of a common service**
The state machine can be examined in detail. Selecting the triggers on the events that cause state transitions shows the details of the SIP messages as they appear in the original call flow (Figure 7). The entry actions inside the states (Figure 8) provide a summary of the outgoing messages that the service may instigate (depending upon the specific scenario). This state machine is a key artifact of understanding in the design process of developing the service.
Servlet modeling

When you implement a SIP service, you usually begin with a set of requirements that may include both high-level text documents and abstract call flow examples. The design process refines these
artifacts, in part, by adding more detail to the header content in the call flows (suitable for direct use with testing), and incorporating external APIs and local framework classes to the design model. Some development teams may decide to use UML state machines as a mechanism to combine call flows and increase the design detail of a specific service. Other teams may decide to work directly from the call flows. In either case the implementation begins with a SIP servlet, and this is best modeled with a UML class. Because the call flows and state machine representations are also in UML models they can be traced directly to servlet classes. This can establish a traceability chain that will make future change and impact analysis a little easier.

**Tooling requirements**

When you look at the actual effort involved in implementing a SIP service, the majority of the activity takes place when you create the business logic, including accessing enterprise objects, databases, Web services, and other services. Very little of what is addressed by such services actually involves the SIP technology itself. This implies that when you choose the right development environment to build those services, you need to choose one that can work with all the relevant technologies, in addition to the SIP-specific ones.

The runtime components of a SIP service (implemented according to the JSR116 specification) require only a Java class to extend the SipServlet framework class, and to include a few elements in the sip.xml and web.xml deployment descriptors of a Web project. Create servlets from scratch in the model by using the tool palette item for Siplet (SIP servlet) or ConvergedServlet, and by creating a new stereotyped class in the model. Table 1 lists the class stereotypes. The palette includes tools for each of the four main class stereotypes in the SIP profile.

A stereotype for a UML model element enables you to associate additional semantics to the given model element. It also allows you to define extra properties that can be captured in the model and used during automated transformations.

### Table 1. Class stereotypes in the SIP Modeling Toolkit

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIPServlet</td>
<td>Applied to classes that respond to SIP messages. Classes stereotyped SIPServlet result in the generation of a Java class that extends SIPServlet.</td>
</tr>
<tr>
<td>HttpServlet</td>
<td>Applied to classes that respond to standard HTTP messages. Classes stereotyped HTTPServlet will result in the generation of Java servlet classes (that is, they extend HttpServlet).</td>
</tr>
<tr>
<td>ConvergedServlet</td>
<td>Applied to classes that respond to both SIP and HTTP messages. These services extend the WebSphere-supplied class com.ibm.wsspi.sip.converge.ConvergedServlet.</td>
</tr>
<tr>
<td>SIPListener</td>
<td>Applied to classes that listen to the server for SIP events. Instances of this class must implement the SIPListener interface.</td>
</tr>
</tbody>
</table>

When you use the palette to create a new Siplet or servlet, it will also ensure that the new class extends the proper superclass from the SIP reference model (see Figure 9). If you do not use the palette, you can still apply the stereotype to a class, and you have the opportunity to extend a different framework class or interface.
A realization of the service, as described by a set of call flow diagrams, begins in the design model with a single stereotyped class that extends the right framework class. Its methods receive requests by SIP clients and are responsible for communicating with a proxy to establish a session with one or more other devices.

The SIP Servlet exists in a larger design, one that includes a number of other classes and library APIs that are used in the implementation of the service. For all but trivial SIP services, the implementation exists in a larger application and will communicate with back end processes, databases, enterprise beans, and other services as shown in Figure 10.

The main reason behind the use of the stereotypes is to identify particular classes as being the primary implementations for the SIP service behavior. It is these classes that require additional meta information to "register" them with the Web application server. This additional information is associated with the stereotype and presented to the developer in the form of a special view (shown in Figure 11). This view is available as a new tab in the standard Properties view section.
In the case of a SIP Servlet, this information includes the pattern or filter criteria that defines the conditions under which this SIP Servlet should be activated by the container. The pattern is a simple tree structure that translates into an XML fragment and is used directly in the sip.xml deployment descriptor (as specified by JSR116).

**Figure 11. SIP Servlet property view**

The rest of the view provides a convenient way to add method overrides for common SIP servlet methods. The **Load On Startup** property is a standard web.xml deployment descriptor field. It indicates a relative measure of importance that this servlet should be activated during server startup. Create common method overrides by selecting the checkboxes in the view.

When the design model is near completion, it is time to transform the design into Java code and deployment descriptors. The SIP Modeling Toolkit includes a transformation extension to the existing UML-to-Java transformations (both Java 1.4 and Java5). Therefore, the same transformation that you would use to generate generic Java code is used in exactly the same manner to generate SIP and HTTP Servlet code and extensions. You don’t have to do anything different has to generate the required SIP content.

**Extracting existing SIP deployment information**

For existing SIP applications, it is possible to transform existing sip.xml and web.xml deployment descriptor information into the SIP Design Model. You can do this by creating a new transformation configuration, similar to how one is created for the UML-to-Java process. This new type of transformation configuration, provided by the SIP Modeling Toolkit, is called the **SIP to UML Transformation**, as shown in Figure 12.
Figure 12. Creating a new reverse transformation

The only parameters to this transformation are the Web project and the target model. Invoking this transformation will update the model by creating the SIP servlet, HTTP servlet, or Converged servlet classes, and by updating their SIP-specific properties from the deployment descriptors. This transformation only updates the deployment descriptor content. It does not update the class' attributes, operations, or associations. Along with the rest of the Java content, you can accomplish these activities by using the existing Java-to-UML transformation that is supplied out-of-the-box with Rational Software Architect.

**SIP test case generation**

Rational Performance Tester Extension for SIP allows SIP services to be tested in the same as Web applications and Web services. Each test case is a call flow of activity that represents a user agent dialog. The SIP Modeling Toolkit includes a transformation that will produce Rational Performance Tester Extension for SIP test cases from call flow diagrams. The resulting test cases provide the starting point for the test engineer who will combine the test cases into test schedules and integrate them with data pools and other mechanisms that a full featured testing tool like Rational Performance Tester provides.

To transform a SIP design model with call flow diagrams into a set of Rational Performance Tester Extension for SIP test cases, you first need to tag those specific lifelines in all the call flows that
you want a test agent scenario to be generated around SIP test cases. The generated test case is executed from the point of view of that tagged lifeline in the call flow diagram. The more lifelines that are tagged the more test cases get generated.

**Figure 13. SIP to Rational Performance Tester test case transformation**

**Design activity overview**

The overall design work flow begins with input from the analysts, usually in the form of call flow diagrams, which can be captured as sequence diagrams. Some designers may want to combine these into a UML state machine or a UML activity as a way to consolidate the emergent behavior of all the scenarios. The state machine or activity would then represent a single behavioral component that, when implemented as a SIP service, would support all the call flows. This is also a good analysis mechanism to identify ambiguities and conflicts in expected scenarios.

The service solution design begins with the creation of a «Siplet» stereotyped class in the UML model. This class will map directly to the main SipServlet implementation class. The designer sets SIP domain-specific information through the custom Properties views, and continues the design by adding additional classes, connections to Web services, databases, enterprise beans, and so on. As mentioned previously, most of the work of designing and implementing a SIP service in actual development takes place when you create the business logic with these other classes and technologies.

In parallel to development of the implementation the same call flows that are driving the servlet design are used as the base for test cases to be executed in Rational Performance Tester for SIP. This re-use or sharing of development lifecycle work products is a significant benefit and helps ensure that the team is working towards the same solution. Of course in an iterative environment
changes can be made anywhere, however with the automation support supplied in the SIP Modeling Toolkit and the base functionality of the Rational modeling and testing products these changes can be quickly absorbed and acted upon. Developing real life services is a team sport, and common set of integrated tooling is a key factor in productivity.

© Copyright IBM Corporation 2007, 2008
Trademark
(www.ibm.com/developerworks/ibm/trademarks/)