

JSLEE and SIP-Servlets Interoperability with Mobicents Communication Platform

Jean Deruelle

Jboss R&D, a division of Red Hat

jderuell@redhat.com

Abstract

JSLEE is a more complex specification than SIP Servlets, and it has been know as heavyweight and with a steep learning curve. However JSLEE has standardized a high performing event driven execution

environment with a good concurrency model and powerful protocol agnostic capabilities covering a variety of telecommunication protocols.

SIP Servlets on the other hand is simpler and easier to get started with for developers with Java EE experience. Its focus is on extending the HTTP Servlets and Java EE hosting environments with SIP capabilities. Hence SIP Servlets is more of a SIP programming framework while JSLEE is a complete, self sufficient application environment, a platform.

JSLEE and SIP Servlets target different audiences with different needs but they can be complementary in a number of real world cases.

This paper will present a converged application built on the Open Source Mobicents Communication Platform. It will show how JSLEE and SIP Servlets can be used together to provide converged applications integrating with network elements via HTTP, SIP and other standard protocols.

1. Introduction

The telecommunications industry is currently in a period of unprecedented change and opportunity : Convergence is becoming a reality. Wireless and wireline networks are being linked to create a seamless and rich end user experience. IP-based communications represent major opportunities for today's network operators, with myriad of new services that can be offered to customers. Services that combine web, voice, and video in interesting new ways are beginning to emerge.

As the telecommunications industry is changing rapidly, remaining competitive and profitable in this dynamic environment will require from operators to embrace innovation, while controlling costs and avoiding solutions that are overpriced.

In this context, new open standards, specifications, frameworks and architectures have sprang to life and are reaching maturity levels allowing mainstream development. For operators, open platforms offer value as flexible, cost effective, and faster to deploy than traditional proprietary products.

The Mobicents Communications Platform [1] is one such Open Platform that aims to drive *Convergence* by bringing to the industry the infrastructure required for a wide variety of intelligent web and multimedia applications.

This paper is organized as follows: section 2 will present the two most popular Java specifications for next generation communication applications. section 3 will then present our proof of concept application to show how we succeeded to make those specifications interoperate. We will also explore other possible approaches and finish by peeking into research work to unify both specifications.

2. Java in Next Generation Networking

Telecommunications applications are moving to component based architectures on Java containers that provide essential infrastructure services such as transactions, resource pooling, security... and let developers focus on value-add application logic. They can benefit from a robust, scalable middleware environment and a number of productivity tools, tests suites, and a large community knowledge base. These benefits reduce time to market and cut development costs.

Regarding NGN, a specific group called Java Advanced Intelligent Network (JAIN) [2] has been defined. Its most widely recognized specifications are JAIN-SIP [3], JSLEE [4] and SIP Servlets [5].

The motivation behind JAIN SIP was to develop a standard interface to the SIP protocol that can be used independently to program SIP applications or be used by higher level programming entities and container based environments. The latter is how JAIN SIP is utilized by the Mobicents Communications Platform in being the underlying API at the core of the

Mobicents SIP Servlets implementation and Mobicents JSLEE SIP Resource Adapter.

Let's now focus more on JSLEE and the SIP Servlets specifications.

2.1. JSLEE

SLEE stands for Service Logic Execution Environment, which is a high throughput, low latency event processing environment targeted at Java communications applications.

Figure 1 presents a simplified view of the JSLEE Architecture.

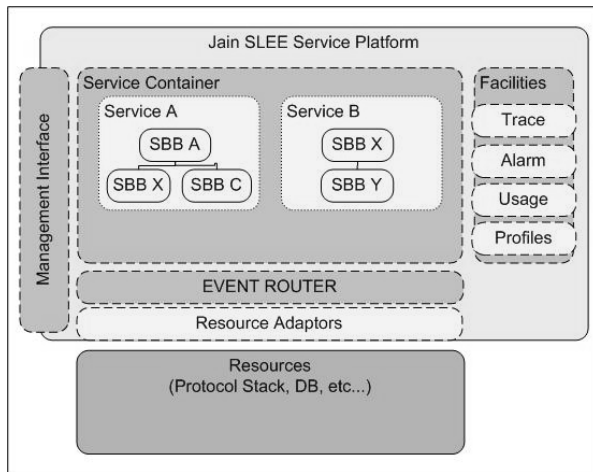


Figure 1 : Simplified JSLEE Architecture

The specification includes a component model for structuring the application logic of communications applications. It views such applications as a collection of reusable object-orientated components, that are composed into higher level and more sophisticated services. Well behaved applications may be deployed on any application environment that implements the JSLEE specification.

JSLEE features an event distribution mechanism that maps external network events to handler methods of hosted components. The event model as well defined transactional and concurrency semantics.

One of JSLEE's most powerful feature is its Resource Adapter Architecture. JSLEE Resources are external entities that interact with other systems outside of the JSLEE container, such as legacy network elements, remote user directories and various data sources. A Resource Adapter, as its name suggests adapts particular properties and behavior of an external Resource into normalized, portable Java interfaces and events visible to application code inside the JSLEE container.

In addition to the application component model, JSLEE also defines management interfaces and a set of built-in Facilities .

Mobicents is the first and so far only Open Source Platform certified for JSLEE compliance.

Let's move on to the next important Java specification for NGN : SIP Servlets.

2.2. SIP Servlets

The SIP Servlets specification defines an environment for execution of network based SIP applications. It provides the ability to mix SIP Servlets with HTTP Servlets API [6] or Java Enterprise Edition (Java EE) [7] components to implement rich media interactions to new or existing enterprise applications. It is leveraging the popular Servlets model, which is familiar to HTTP Servlets developers. It uses the same development model and defines high level SIP objects , similar to the HTTP Servlets ones, and helper objects (such as Proxy and Back to Back User Agent) to ease the development of SIP applications. Moreover it works together with HTTP Servlets in a way that allow them to be able to initiate calls or share data. It's however different from HTTP Servlets because SIP Servlets are asynchronous, don't have context root and multiple SIP Servlets can receive the same SIP message.

As opposed to non-container SIP applications, SIP Servlets containers provides value-adds such as protocol stack, state management, security or distributable applications.

The specification also defines a powerful application selection and composition model through a logical entity called the Application Router.

Figure 2 shows how SIP Servlets fit into the Java EE architecture.

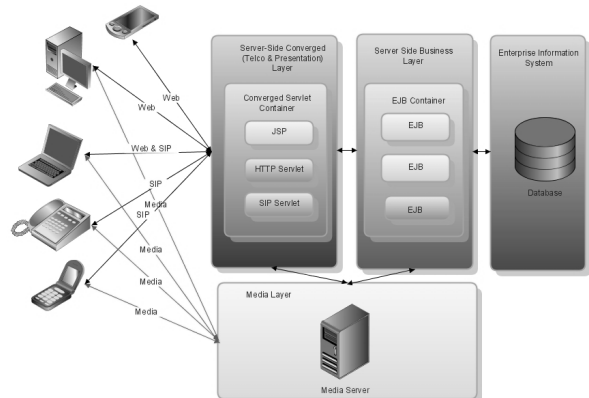


Figure 2 : SIP Servlets in the Java EE Architecture

2.3. JSLEE vs SIP Servlets

Some view SIP Servlets and JSLEE as competing specifications and try to choose between one or the other because both aim at creating next generation Java communication applications.

Table 1 compares briefly both standards :

Table 1 : JSLEE/SIP Servlets Comparison Table

SIP-Servlets	JSLEE
Application Architecture	
Based on HTTP Servlets. Unit of logic is the SIP Servlets	Component based, Object Orientated architecture Unit of logic is the Service Building Block
Composition through Application Router	Composition through parent-child relationship
Application State	
Servlets are stateless	SBBs may be stateful.
Shared state stored in a session Shared state is visible to all Servlets with access to the session	SBB state is transacted and a property of the SBB itself. Shared state may be stored in a separate ActivityContext via a type safe interface
Concurrency Control	
Application managed : use of Java monitors	System Managed : isolation of concurrent transactions
Facilities (Utilities for Applications)	
Timer, Listeners	Timer, Trace, Alarm, Statistics, Profiles.
Protocol Support	
SIP and HTTP	Protocol agnostic. Consistent event model, regardless of protocol/resource
Availability Mechanisms	
Container managed state (session object) that can be replicated	Container managed state (SBB CMP, Facility, ActivityContext) that can be replicated
No transaction context for SIP message processing	Transaction context for event delivery
Non transacted state operations	Container managed state operations are transacted
Facilities are non transacted	Facilities, timers, are transacted
No defined failure model	Well defined and

	understood failure model via transactions
Management	
No standard management mechanisms defined	JMX Interface for managing applications, life cycle, upgrades, ...

To summarize, SIP Servlets focuses on SIP and its integration with Java EE. It is also more of a SIP framework within Java EE while JSLEE is an event driven application server with protocol agnostic architecture, spanning any legacy or potential future protocols. SIP Servlets applications are generally simpler to implement and accelerate time to market for Web and SIP deployment scenarios. JSLEE has a steeper learning curve and covers a wider set of target deployment environments. [8] [9]

We will now go on to show how developers can mix both together by using the best of both worlds.

3. JSLEE and SIP Servlets interoperability

As JBoss is the only vendor to implement both specifications through its Mobicents Communication Platform, this makes it a natural fit to build converged and interoperable JSLEE/SIP Servlets applications that are able to comply with standards in a portable manner. Here is the description of the application we implemented and its architecture to demonstrate the previously mentioned goal.

3.1. Application Description

Our application is aimed to receive calls, play announcement and ask for user input followed by a pound sign; then it waits for DTMF input. When pound sign is pressed, another announcement is played repeating numbers that have been pressed by the user.

We used media control here as a proof of concept of a typical need for SIP Servlets applications that is outside of the SIP Servlets Specification and support for it is really up to vendors thus the application, if built using only SIP Servlets, would become non portable between vendors and application servers.

The same kind of application could be built for accessing other standards (such as Diameter offered by most vendors in the same proprietary way) or legacy protocols through JSLEE in a portable way without to resort to vendor proprietary SIP Servlets extensions.

Figure 3 presents the architecture and the call flow of our application

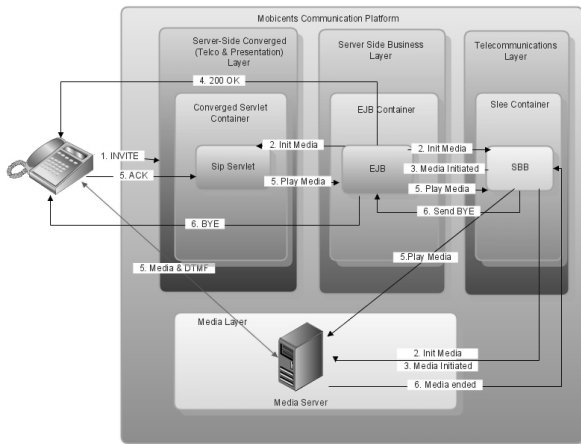


Figure 3 : Application Architecture & Call Flow

[1] The User Agent (UA) calls (sends INVITE) to an information number (sip:info@127.0.0.1:5080) on SIP Servlets converged application.

[2] The SIP Servlets converged application initiates the Media Connection by firing up an event into the JSLEE container through a stateful CallManager Enterprise JavaBean (EJB).

[3] When the Media connection has been initiated, the JSLEE application notifies back the stateful CallManager EJB with the SDP provided by the Media Server.

[4] CallManager EJB sends the OK response with SDP from the Media Server to UA.

[5] When ACK request is received from UA, the SIP Servlets converged application notifies the JSLEE container through the CallManager EJB that the Media Server should play the announcement and listen for DTMF.

[6] When the UA has entered the different numbers followed by the pound sign. A confirmation announcement is played and the JSLEE application tear down the call by calling back the CallManager EJB that will send the BYE request.

3.2. Application Implementation

The working proof of concept we built is available for testing in [10] and was built on top of JBoss using Seam, Mobicents SIP Servlets, Mobicents Media Server and Mobicents JSLEE with Media and Text To Speech Resource Adapters.

In this application, Mobicents SIP Servlets is used to handle the SIP protocol. It is tightly integrated with JBoss Java EE capabilities through a stateful EJB which is used as a gateway to communicate between SIP Servlets and JSLEE worlds.

Indeed, the SIP Servlets converged application contains a stateful EJB which will store the initial SIP Servlets request and fire events into Mobicents JSLEE through a SlesConnection.

As specified in the JSLEE specification, an EJB component may pass an event to the SLEE in order to drive application logic contained within the JSLEE container. The stateful EJB will gain a connection factory instance through JNDI and use this factory to obtain SlesConnection instances representing connections to the JSLEE container. A SlesConnection instance allows an EJB component to fire events to the JSLEE container.

A custom event has been created by our application containing a handle to the stateful EJB so that the JSLEE Service can call back the EJB in a synchronous manner.

The stored request in the stateful EJB will be used to create responses and subsequent requests.

Mobicents JSLEE is invoked here to control Media through Media and Text to Speech Resource Adapters and Mobicents Media Server is used to handle the media streams and DTMF recognition.

The drawback of this approach is that it stores the SIP Servlets request into the EJB and to handle correctly activation/passivation of EJB under load, the SIP Servlets request should be Serializable and this might be container dependent.

Other approaches may exist as a way of achieving interoperability. Let's review them and compare them to our proposed solution.

3.4. Another approach to interoperability : SIP

Since SIP is a protocol that is supported in a standard and portable way by both platform, this would seem logical to try to use it as a way to inter operate with the JSLEE application instead of EJB and SlesConnection.

In this approach the SIP Servlets application will act as a Back to Back User Agent (B2BUA) and transmit the SIP messages to the JAIN SLEE container. The call flow for this approach is pictured in Figure 4

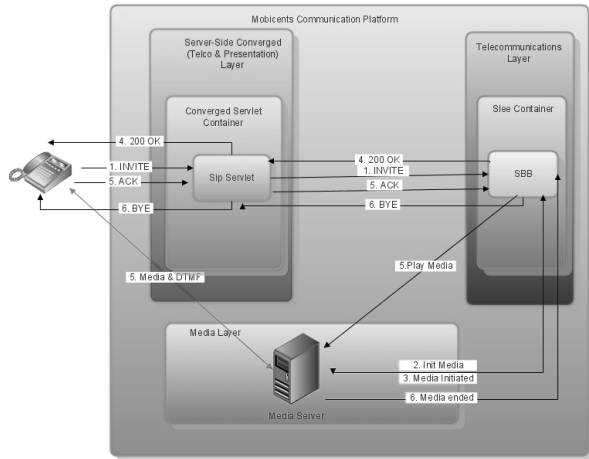


Figure 4 : SIP Based approach to SIP Servlets and JSLEE interoperability

[1] UA calls (send INVITE) to an information number (sip:info@127.0.0.1:5080) represented by our SIP Servlets application.

[2] The SIP Servlets application acts as a B2BUA and creates a call leg to the JSLEE Container in sending an INVITE to it containing the same SDP as the one it received in the INVITE on the ingress side.

[3] The JSLEE container receives the INVITE through its SIP Resource Adapter and route it to the JSLEE application which then initiates the media connection. Once it has been initiated, the JSLEE application notifies back the SIP Servlets application in sending to it a 200 OK response containing the SDP provided by the Media Server.

[4] The SIP Servlets application transmits the OK response to the UA.

[5] When ACK request is received from UA, the SIP Servlets application sends a ACK on the egress call leg to the JSLEE container.

[6] The JSLEE application, upon ACK reception, notifies the Media Server that it should play the announcement and listen for DTMF.

[7] When the UA has entered the different numbers followed by the pound sign, a confirmation announcement is played and the JSLEE application tear down the call by sending the BYE request that will go through the SIP Servlets application and then to the UA.

With regard to our proposed solution, this approach even if also portable and completely asynchronous has the following drawbacks :

Since the JSLEE container has also to implement the SIP logic, another Resource Adapter (SIP) has to be installed but the problem here is that it duplicates the work of the SIP Servlets application. Actually the work needed for the JSLEE application to support SIP will be more important than in SIP Servlets since SIP Servlets handles many low level concerns automatically.

Moreover, the UA could very well send the INVITE directly to the JSLEE container bypassing completely the SIP Servlets application. The SIP Servlets application existence might only be justified in this case if there is a need to interact with Web content or some business logic contained in EJBs.

3.4. Another approach to interoperability : Java Messaging Service

Another approach would be to use Java Messaging Service (JMS) [11] to communicate with JSLEE.

In this approach the SIP Servlets converged application will send JMS messages to the JSLEE application to control the Media Server. The JSLEE application will send back JMS messages to the SIP Servlets application to control the call flow, the call flow for this approach is depicted in Figure 5

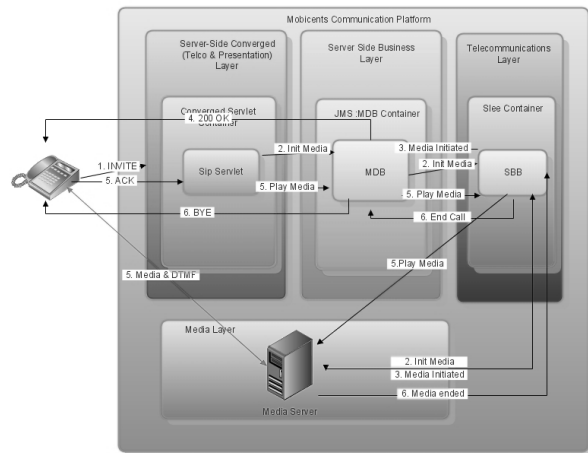


Figure 5 : JMS Based approach to SIP Servlets and JSLEE interoperability

[1] The UA calls (send INVITE) to an information number (sip:info@127.0.0.1:5080) represented by our SIP Servlets application.

[2] The SIP Servlets application store the request it just received as an attribute of its SIP session and sends a JMS message to the JSLEE application containing the identifiers of the SIP session and Sip application session and a message for controlling the Media Server.

[3] The JSLEE container application receives the JMS message through its JMS Resource Adapter and route it to the JSLEE application which initiates the media connection. Once it has been initiated, the JSLEE application notifies back the SIP Servlets application in sending back to it a JMS Message containing the SIP session and SIP application session identifiers and the SDP provided by the Media Server.

[4] The SIP Servlets converged application's Message Driven Bean (MDB) gets the JMS message on its queue. The MDB, which has access to the

SipSessionsUtil utility, retrieves the SIP application session reference using the identifier present in the message. Since the MDB also knows the SIP Session identifier, it gets to the right SIP Session from the SIP application session and can retrieve the sip request previously stored. It then creates the OK response through the SIP request, set the SDP from the JMS message and sends the OK response to the UA.

[5] When ACK request is received from UA, the SIP Servlets application sends a JMS message to the JSLEE application with a message in it instructing it to play the announcement.

[6] The JSLEE application, upon reception of the new JMS message, notifies the Media Server that it should play the announcement and listen for DTMF.

[7] When the UA has entered the different numbers followed by the pound sign. A confirmation announcement is played and the JSLEE application sends a JMS message to the SIP servlets converged application instructing it to tear down the call.

[8] The MDB gets the JMS message and retrieve the SIP session in using the same mechanism as described in step {4}. Through this SIP session it will create the BYE request and send it to the UA.

This approach has the same drawback that our approach, i.e. it stores the SIP Servlets request into the SIP session and to handle correctly activation/passivation of SIP Sessions under load, the SIP Servlets request should be Serializable and this might be container dependent.

Moreover, by using JMS, it adds complexity to the architecture and more application development time for no real gain.

4. Conclusion and Future Work

This paper presented an overview of the disruption in the telecommunications landscape and two of the key Java Platform standards that enable developers to ride the next wave.. We examined JSLEE and SIP Servlets and showed how they can be applied in a practical example deployed on Mobicents.

The next goal currently under active research at JBoss is to propose a unified environment and programming model that will integrate best of both worlds. If successful, it will preserve the protocol agnostic strength of JSLEE as well as its transactional and concurrency semantics, while offering a simpler way to harness its power that is in line with the current best practices of Java EE. JBoss SEAM is currently viewed as one starting point of achieving our goal.

5. References

[1] Jboss, "Mobicents Communications Platform", See <http://www.mobicents.org>.

[2] Sun Microsystems, "Java Advanced Intelligent Networking", See <http://java.sun.com/products/jain/>

[3] Sun Microsystems, "JAIN SIP Specification - JSR 32", 2002. See <http://jcp.org/en/jsr/detail?id=32>.

[4] Sun Microsystems, "JSLEE Specification - JSR 240", See <http://jcp.org/en/jsr/detail?id=240>.

[5] BEA, Sun Microsystems, "SIP Servlet Specification - JSR 289", 2008. See <http://jcp.org/en/jsr/detail?id=289>.

[6] Sun Microsystems, "Java Servlet Specification - JSR 154", 2002. See <http://jcp.org/en/jsr/detail?id=154>.

[7] Sun Microsystems, "Java Enterprise Edition Specification - JSR 244", 2006. See <http://jcp.org/en/jsr/detail?id=244>.

[8] <http://jainslee.org/othertechnologies/sleevsipservlet.html>

[9] <http://java.sun.com/products/jain/JSLEE-SIPServlet.pdf>

[10] <http://www.mobicents.org/jslee-sips-interop-demo.html>

[11] <http://java.sun.com/products/jms/>