ANALYSIS AND SIMULATION OF A JAIN SLEE-BASED PLATFORM FOR INTELLIGENT NETWORK SERVICES

Vytautas Pilkauskas¹, Vilius Panevezys²
¹Kaunas University of Technology, Department of Business Informatics, Studentu 50, Kaunas, Lithuania, vytpilk@ktu.lt
²JSC ELITNET, Pasiles 102, Kaunas, Lithuania, vilius@elitnet.lt

Abstract: Intelligent Network is a telephone network architecture in which the service logic for a call is located separately from the switching facilities, allowing services to be added or changed without having to redesign switching equipment. The Java Community, within the Java APIs for Integrated Networks activities, offers a set of standard frameworks and open protocol APIs for the creation of telecommunications services. Currently the only open source implementation of JSLEE specifications is Mobicents. This paper proposes a solution for protocol-agnostic service creation in Mobicents platform. In our proposed solution, services depend on an abstract API defined by specifications from the JAIN group and implemented by a custom JSLEE Resource Adaptor. Discrete-event simulation model was used, in order to evaluate the performance of the proposed solution.

Keywords: Intelligent network, JSLEE, Mobicents, simulation model.

1 Introduction

Basic call setup functions are not enough for an operator to survive in modern competitive markets. Telecoms strive to get revenue by providing value-added services to their customers. One of value-added service enablers is Intelligent Network (IN) [1]. Intelligent Network is a telephone network architecture originated by Bell Communications Research in which the service logic for a call is located separately from the switching facilities, allowing services to be added or changed without having to redesign switching equipment. IN eases deployment, management and significantly reduces “time to market” of innovative services. One of key IN services is prepaid, requiring online charging. Other popular IN services include Friends and Family Promo, Corporate User Group, Single IMSI multiple MSISDN and others.

The Java Community, within the Java APIs for Integrated Networks (JAIN) activities, offers a set of standard frameworks and open protocol APIs for the creation of telecommunications services. In particular, the JAIN SLEE (JSLEE) provides a set of specifications [2] for the implementation of a Service Logic Execution Environment (SLEE) container. A SLEE is a hosting environment for telecom applications, also known as application server (AS), specifically designed to support asynchronous services with real-time constraints. Being composed of several layers of abstraction, a SLEE AS simplifies service design and implementation by providing the non-functional features needed for service execution. The resulting benefit is that developers can focus on aspects related to the implementation of the service logic, since the underlying complexity is masked by high-level APIs [3]. This kind of AS is a major candidate for deploying telecom application services [4].

Currently the only open source implementation of JSLEE specifications is Mobicents [5], which relies on the JBoss AS hosting environment. JBoss offers capabilities for service and SLEE management through Java Management Beans (MBean), service deployment, and thread pooling [6].

Telecom services are intrinsically asynchronous and typically require short latency, high throughput and high availability. Their integration is not simple as they may rely on several network protocols and interoperate with different software and hardware platforms. [7] presents the results of an experimental campaign in which several SIP-based VoIP services were tested, in order to evaluate the performance and scalability of the Mobicents SLEE platform. This paper presents a solution for IN protocol integration with Mobicents JSLEE and a simulation model of the system.

The paper is organized as follows. In section 2, we give some background information. In section 3, we illustrate in detail our proposal. In section 4, we describe an example of IN service implementation of the proposed solution, and section 5, outlines the simulation model used to evaluate the platform performance, and the relevant results. Finally, section 6 reports our conclusions.

2 Background

2.1 JAIN SLEE specifications

The JSLEE is the Java standard specification for the creation of a SLEE application environment, as reported in JSR 22 (v.1.0) and JSR 240 (v.1.1) [1]. 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, which are composed into higher level and more sophisticated services. Well behaved applications may be deployed on any application environment that implements the JSLEE specification. Figure 1 presents a simplified view of the JSLEE Architecture.

JSLEE is an event driven component-based container technology, supporting high performance, asynchronous application services. The service logic is implemented in system components called SBB, managed as a pool. SBBs operate asynchronously by receiving, processing, and firing events. They can be attached to event streams called Activity Contexts (AC), by which they receive notifications from other entities. When SBBs are no longer used, they are passivated, that is they are put within an instance pool waiting to be re-used.

External network events are translated into internal Java events by JSLEE modules called Resource Adaptors (RAs), which constitute an abstract interface layer that allows JSLEE AS to access external resources. RAs, 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. Events are internally routed by a system component called Event Router.

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

![Figure 1. Simplified JSLEE Architecture](image)

2.2 Open source implementation of JSLEE

Mobicents is an open-source communication platform that includes a SLEE, a Media Server, a Presence Server and a SIP Servlet AS. Below the Mobicents SLEE will be referred to as MSLEE. At the time of this writing the most recent version of Mobicents was 2.2.1.FINAL based on JBoss 5.1.0.GA. JBoss offers capabilities for service and SLEE management through Java Management Beans (MBean), service deployment, and thread pooling. It represents a container for higher level service containers. MSLEE does not rely on the CMPs offered by the JBoss container.

2.3 Comparison with commercial JSLEE solutions

There are other tools and architectures used to simplify JSLEE service development, but they are bundled within commercial products. JNetX JSLEE [7] implementation Open Convergent Feature Server (OCFS) comes with an IDE tool called Telecom Service Studio that claims to ease the whole development process. OCFS includes implementations of the IN protocols. Service designer may use the implementations directly to build the service using the protocol API directly or alternatively an intermediate protocol abstraction layer may be used. The abstraction layer is a protocol independent API called Multi-Party Call Control (MPCC). It should be possible to reuse the same service built using MPCC with different supported protocols. MPCC is based on Parlay call control API [8].

Instead, the OpenCloud Rhino [7] platform is very well documented. It consists of an XDocket module and the FSM tool. The first one is used to write the deployment descriptor code directly inside Java class files, utilizing special tags without switching between different XML files (but the developer must yet write this type of data). The FSM tool is used for the creation of a finite state machine inside the service. It uses the VFMS description language to describe the state machine behavior and allows three different type of actions: entry, exit and input, executed when entering a state, leaving a state and corresponding to an event occurrence, respectively; obviously transitions can be defined as well. The tool parses this description and generates an SBB Java class
that defines the abstract behavior of the component. The developer must extend this class, implementing the actions with Java code and mapping actions with RAs callback methods. This type of approach, which delegates event handling to other methods and classes, is very similar to our SBB starting point/event router solution, but we think that our solution is more powerful. Opencloud’s VFSM language is relatively simple and with few pattern options, unable to create complex state machine diagrams. Instead, we integrate the JSLEE AS with a whole WFMS capable of advanced control-flow patterns (e.g. split, merge, synchronization) and features. In the OpenCloud solution, the communication channels between different components still relies on JSLEE features (SbbLocalObject interface or ACI), whereas we can overtake this issue through the usage of transient variables which are more easy and flexible to use. Finally, using only one SBB, we do not have to care of multiple deployment descriptor XML files. Rhino has support for IN protocols as well, though, does not provide any abstraction to ease service development and make the service logic portable to different protocols.

3 Proposed solution

Despite the ever growing interest in VoIP services telecom operators still heavily rely on existing IN infrastructure, which has required considerable investments in the past so it is very unlikely to be withdrawn heedlessly. This historical background suggests that MSLEE would only be attractive to telecom operators with reliable and convenient IN integration.

This problem was approached with these key qualities in mind: reliability, high-availability, scalability, deployment flexibility.

Figure 2. Mobicents IN integration solution scheme

Figure 2 depicts the basic scheme of the solution and how it aligns with IN architecture and SS7. IN Service Control Function is constituted of two components TCAP Gateway and MSLEE.

TCAP Gateway resides in a node equipped with a Dialogic® SIGTRAN-based SS7 stack (M3UA, SCCP-CL, TCAP). The component directly integrates with the protocol stack, takes care of TCAP dialogue management and acts as a gateway for higher-level protocol payloads. Flexible architecture of the solution allows this component to either be located on a separate node or co-located with MSLEE. This allows various deployment schemes and easy horizontal scalability for either of the components.

The second component is the MSLEE enhanced with a custom Resource Adaptor for Java Advanced Call Control that communicates with TCAP Gateway and implements logic of IN call control protocols such as CAP, INAP. Java Advanced Call Control is an API that builds upon existing specifications for Java call control from the JAIN group and introduces user interaction and other advanced call control functions.

TCAP Gateway directly communicates with Mobile Switching Center, a Service Switching Function in IN parlance, which is responsible for routing voice calls, SMS and other services. The other link maintained by TCAP Gateway is used to exchange messages of IN call control protocols with the JACC RA deployed in MSLEE. The link uses a specially designed efficient ASN.1-based protocol, TCP is used for transport.

The JACC RA integrates IN protocols to MSLEE and presents an easy to use, protocol-independent, standardized API for service logic composition.

4 Analysis of performance and scalability of the solution

In order to assess the call processing performance of the solution and calibrate the simulation model, timing probes have been injected into JACC RA to measure time taken for particular segments of a call processing scenario.

For this test a real world use case has been chosen based on the flow depicted in Figure 3. This is an example of a trivial subscriber service management service, for example an IVR-like service to configure voice mail service:

1. Subscriber dials a short code dedicated for voice mail management. JACC RA gets notified about the initiated call by InitialDP message from network.
2. JACC RA fires appropriate event to SLEE, which selects a service to handle this event and the event gets delivered to the service for voice mail configuration.

3. Service starts playing and announcement for the subscriber, e.g. “Press 1 to enable voice mail on busy, press 2 to enable voice mail when your phone is turned off or out of the radio coverage, press 3 to enable voice mail in both cases or press 4 to disable voice mail service.” The appropriate network node (gsmSRF) is instructed to do this by sending out messages ConnectToResource and PromptAndCollectUserInformation.

4. After the user has chosen one of the alternatives and pressed the corresponding digit, JACC RA receives ReturnResult message from network with the collected information. SLEE selects the service that is handling this call and delivers the second event. Service would then persist the new configuration defined by the subscriber to a data store and release the call (possibly after playing back another announcement to the subscriber to assure the new setting has been saved). Interactions with the data store are omitted for simplicity.

![Figure 3. Call flow of an interactive service](image)

Segments of this scenario have been timed to get experimental data for calibration of the simulation model.

**Table 1. Timed segments of call processing**

<table>
<thead>
<tr>
<th>Segment</th>
<th>Time taken (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>End</td>
</tr>
<tr>
<td>1</td>
<td>Initial event received from network</td>
</tr>
<tr>
<td>2</td>
<td>Initial event fired to SLEE</td>
</tr>
<tr>
<td>3</td>
<td>Start of service logic execution</td>
</tr>
<tr>
<td>4</td>
<td>Start of user interaction</td>
</tr>
<tr>
<td>5</td>
<td>End of user interaction</td>
</tr>
<tr>
<td>6</td>
<td>Result event received from network</td>
</tr>
<tr>
<td>7</td>
<td>Event fired to SLEE</td>
</tr>
<tr>
<td>8</td>
<td>Start of service logic execution</td>
</tr>
<tr>
<td>9</td>
<td>Start of release call operation</td>
</tr>
<tr>
<td>10</td>
<td>End of release call operation</td>
</tr>
</tbody>
</table>

Testing methodology used was the following:
1. Start SLEE with the deployed service and run a “warm-up” round of 50000 calls to avoid cold start effects of the JVM like JIT and other dynamic optimization techniques interfering with the test results.
2. Run 5 calls at 1 CPS rate.
3. Average results of the last 5 calls.

The tests have been performed using HP ProLiant DL360 G5 server.
5 Performance evaluation using simulation model

Discrete-event simulation modeling has become the most commonly used tool for performance evaluation of stochastic dynamic systems in science and engineering, including such complex systems as computer and communication systems [11]. Simulation modeling can be used both as an analysis tool for predicting the effect of changes to existing systems and as a design tool to predict the performance metrics such as average queue length and utilization of resources of new systems under varying sets of circumstances.

5.1 Simulation Model Description

In this paper we use a piece-linear aggregate (PLA) [12] formalism for creation of discrete event models [13]. In the application of the PLA approach for system specification, the system is represented as a set of interacting piece-linear aggregates. The PLA is taken as an object defined by a set of states \( Z \), input signals \( X \), and output signals \( Y \). A behavior of an aggregate is considered in a set of time moments \( t \in T \). State \( z \in Z \), input signals \( x \in X \), and output signals \( y \in Y \) are considered to be time functions. An execution of PLA corresponds to a sequence of internal and external events. Transition and output operator \( H \) must be known as well. PLA formalism has been used for creating object-oriented library for development of simulation models [14].

During research the formal (mathematical) model of MSLEE system realizing IN services was created. In the PLA system model five types of aggregates were developed (Figure 4):

- **Network** – for message stream generation of IN services in the model;
- **ResourceAdaptor** – for simulation of resource’s adapters which connects MSLEE container with the external devices;
- **SLEE** – for simulation IN service processing;
- **ProcUnit** – for simulation of functioning maintenance resources adapters and MSLEE container’s program components in server process unit.

![Figure 4. Structural scheme of PLA simulation model.](image)

Due to the formal model simulation modelling program was created. For program code creation of simulation models specified using PLA formalism, object-oriented imitation modeling library PLASim, created in KTU business department, was used as well as graphical redactor specialized for PLA simulation models creation. In programming model IN services specified using sequences diagram (Figure 3) are simulated. This sequences diagram is considered as the model’s input parameter. Created model allows imitation of several types of IN services processing with different sequence diagrams of such type.

5.2 Simulation Results

Simulation experiments with the program model were performed. The call processing scenario depicted in Figure 3 can be split into two phases:

1. Starting with call initiation indication from network (InitialDP message), ending with response from service logic (ConnectToResource and PromptAndCollectUserInformation messages)
2. Starting with ReturnResult message from network and ending with service logic release the call indicated to network by ReleaseCall message.
Graphs presented in this chapter (Delay1, Delay2) correspond to the call processing phases 1 and 2 respectively as described above.

IN service’s processing time dependencies for the execution of software components of MSLEE were gained (Figure 5, Figure 6).

6 Conclusions

In this paper we have analyzed the open source Mobicents JSLEE server scalability performance through a typical IN service. The proposed solution proved to be highly effective even without employing any scalability-oriented deployment scheme. Testing and simulation have shown that Mobicents is fully capable to be used for call processing in high traffic networks. Moreover, flexibility of the solution allows easy horizontal
scalability, which can be used to achieve both higher throughput and high availability. Having integrated IN protocols Mobicents becomes a suitable and cost-effective replacement for commercial IN platforms.

7 Acknowledgements

This work was supported in part by measure “IntelektasLT” of EU structural programme for Economic Growth in Lithuania. Project “Development of the third generation service delivery platform based on open source service logic execution environment for telecommunication intelligent networks” VP2-1.3-UM-02-K-01-058.

References


