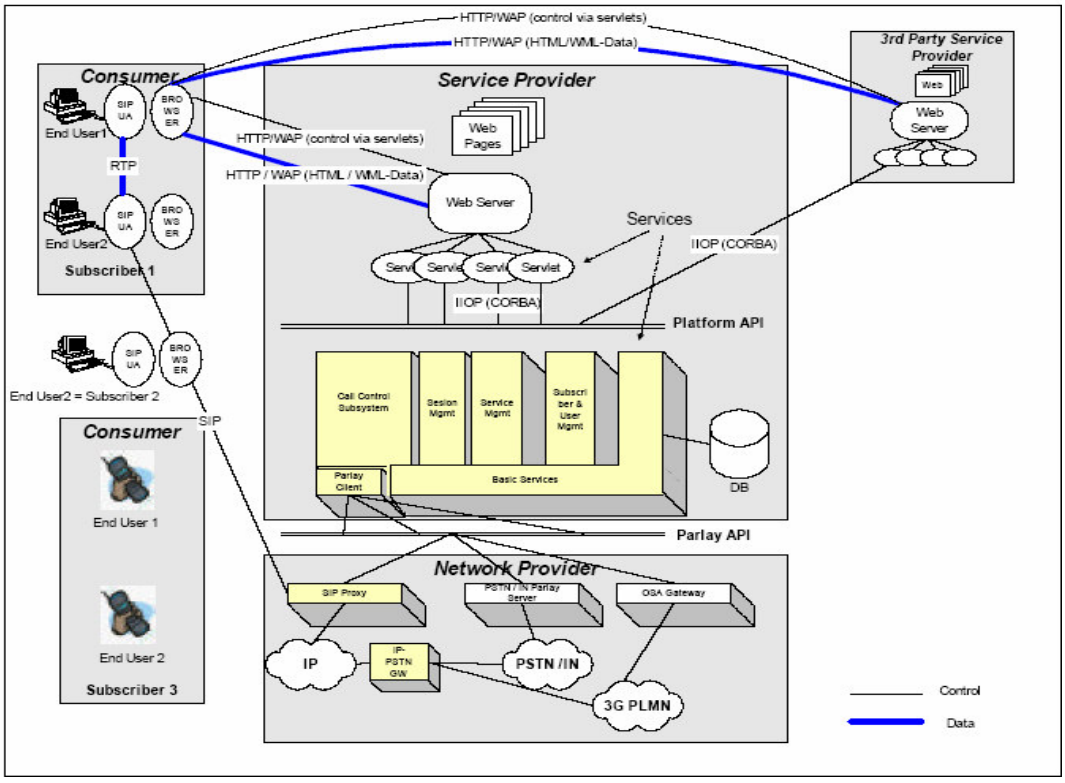

Figure 10 Service platform using Parlay API over SIP

This diagram shows that this service platform resides on a SIP Server separated by the Parlay interface. The SIP Server is usable not only for communication with the service Platform via Parlay, but also for the development of a SIP user agent. The platform has an additional CORBA interface towards a web server used by web Servlets to interact with the Platform. This enables a third party call set-up in the SIP Server triggered by a web interface e.g. Click to Dial. This interface is also used for management purposes and service modifications by the end user. [7]

2.5.2 An architecture for the integration of IP-Telecom services

[15] proposes an architecture for hybrid services that spans many network technologies, such as IP, PSTN and wireless networks. The service architecture (as shown in Figure 11) is composed of two elements: A SCE and a Service Infrastructure (SI). SCE and SI provide service life-cycle management capabilities, including service creation, deployment, activation, deactivation and removal. In the SCE, service logic is put together by the service creator using a set of service components as building blocks. The service logic is then sent to a service factory, which creates a new instance of the service. The service instance is organized and

uploaded into the SI. SI can activate and deactivate the service instance. The service instance can also be removed from SI. The Service Infrastructure is composed of a collection of controllers and the underlying system elements of the Intelligent Network, such as terminals, gateways, network nodes, etc. An event interface enables the data flow between the system elements and the service logic.

The service architecture currently supports two services: hybrid calendar and hybrid call forwarding. Hybrid calendar allow users to access their calendars from any terminal connected to an IP network, as well as expose and share portions of their calendars with other users. Hybrid call forwarding allows a customer to forward his incoming calls (addressed to his telephone set from any other telephone set) to a computer connected to an IP network.

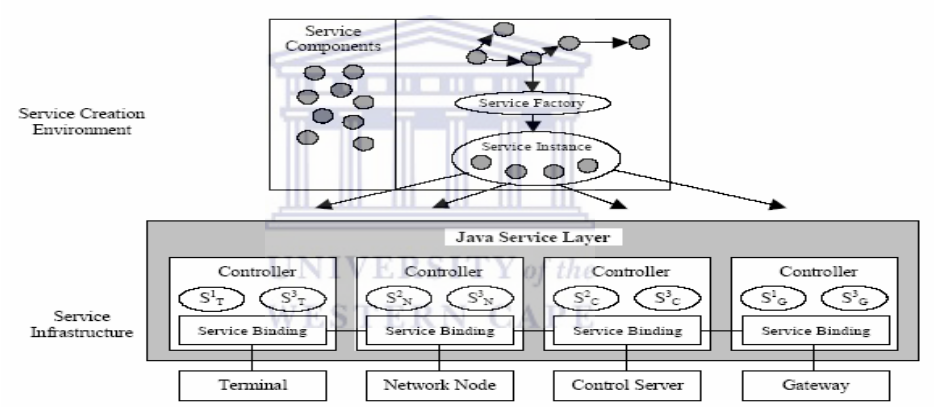

Figure 11 Service architecture based on a SCE and a SI

S^i_T = subsystem of service i running on terminals; S^i_N = subsystem of service i running on network nodes; S^i_C = subsystem of service i running on control servers; S^i_G = subsystem of service i running on gateways. [15]

2.6 Summary

It is important to provide IP communication applications with carrier grade characteristics to make them applicable in telecommunication networks. This thesis focuses on the provision of service life-cycle management, which is a basic carrier grade characteristic. The life-cycle management characteristic includes the deployment, activation, deactivation, uninstallation and online upgrade of IP communication applications.

For the issue of providing life-cycle management for IP communication applications, a service platform is built in [7] and service architecture is constructed in [15] to provide the life-cycle management for their IP-Telecommunication applications. However, these designs can only offer the characteristic to their own services, and therefore can not be applied to general IP communication applications. An NGN can also provide a solution for the issue. In an NGN, an Application Server provides the creation, execution and management environments for service life-cycle management. Generally, two open service APIs, JAIN and Parlay, provide their solutions to support an NGN. In JAIN, a JSLEE acts as an open Application Server, where the life-cycle management of IP-Telecommunication applications can be carried out [28] [39]. In Parlay, a logical entity named SCS, works as the server to implement the Parlay API. However, a Parlay application is deployed by a third-party server and only communicates with the SCS using CORBA, it is therefore independent from the Parlay itself. Thus, the life-cycle management of the application will not be considered by Parlay [18] [34].

For the issue of life-cycle management, a SoftBridge application is used as an example of an IP communication application to examine service life-cycle management. However, most work related to the SoftBridge does not explicitly address service life-cycle management, such as [29] [31]. They just focus on providing semi-synchronous multi-modal communication services.

Based on the above, an Application Server, in particular a JSLEE, should be strongly considered for putting carrier grade characteristics into a SoftBridge Application with respect to service life-cycle management.

Chapter 3 APPROACH AND RESEARCH METHODOLOGY

First, the research question is stated in this chapter. A research approach follows to suggest a solution to the question. Finally, research methodologies are listed out to introduce the way in which the research approach is performed.

3.1 Approach to research question

3.1.1 Research question statement

The main question of this research is**: "How to provide carrier grade characteristics to IP communication applications in terms of service life-cycle management?"** Using an IP-based SoftBridge application as an example of IP communication applications, the issue of providing life-cycle management for a SoftBridge application is broken down into two aspects:

- A SoftBridge application deployment, activation, deactivation and uninstallation. As a carrier grade service/application, a service execution and management environment should be used to deploy, activate, deactivate and uninstall a SoftBridge application.
- Online upgrade of a SoftBridge application. As a carrier grade service/application, a SoftBridge application in the running state cannot be interrupted when it is upgrading or version updating, e.g. the call, in a connected state, should be kept going during the process of a system upgrade.

3.1.2 Proposed approach

As the main functional entity in the service layer, an Application Server acts as a service creation, execution and management environment in an NGN [57]. An Application Server is therefore proposed to provide a service support environment for a SoftBridge application. Using a JSLEE as a representation of the Application

Server, a SoftBridge application is deployed, activated, deactivated and uninstalled in it, as shown in Figure 12. A JSLEE also enables a SoftBridge application to online upgrade.

In the JAIN API architecture, the JSLEE acts as an Application Server that provides the execution and management environment for the next generation Internet–Telecommunication services [28]. The JSLEE defines interfaces and requirements which are mandatory for telecom/Internet operations within carrier grade and Internet networks.

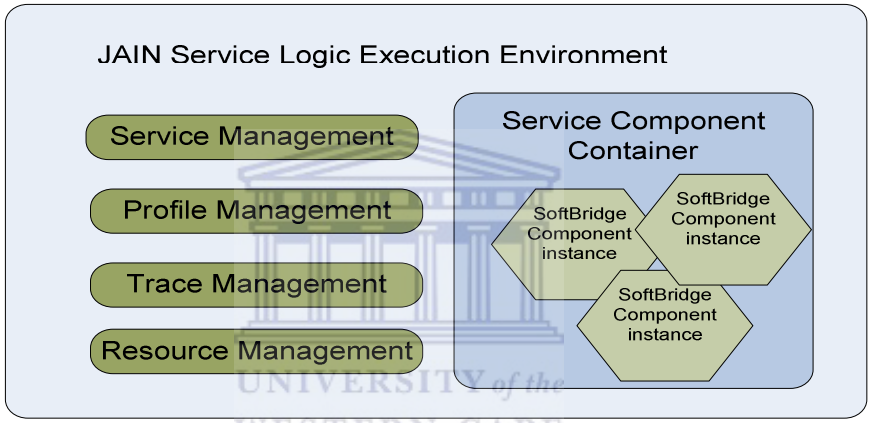

Figure 12 SoftBridge applications in JSLEE

Figure 12 shows that a SoftBridge application is deployed into a JSLEE that provides various management facilities. One or more instances of the SoftBridge application run in a Service Component Container of JSLEE.

3.2 Research Methodology

Two methodologies are applied in this research: exploratory prototyping in the development of a SoftBridge application and comparison in the empirical evaluation of a SoftBridge application with regard to carrier grade capabilities.

3.2.1 Exploratory prototyping

Exploratory prototyping is the rapid development of a system, where an initial prototype is produced and refined through a number of stages towards the final system [20]. Exploratory prototyping applies to the development of those systems

with the best understood requirements. It is considered as a risk reduction activity. An exploratory prototyping model is illustrated in Figure 13.

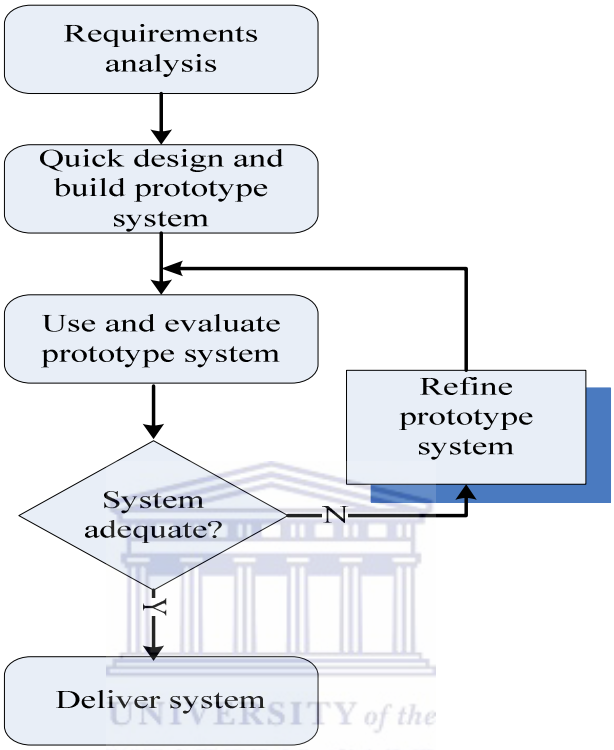

Figure 13 Exploratory prototyping process

Figure 13 shows main steps in an exploratory prototyping. After requirements analysis is finished, the first prototype system is quickly designed and implemented. The first prototype is evaluated and refined if it is not adequate. The final system is delivered until it is evaluated to be adequate.

The exploratory prototyping can be divided into the following phases:

1. Requirements analysis: Define the service provided by the SoftBridge application; analyze and clarify system requirements. Service definition and system requirements analysis document is also developed in this phase.
2. Quick design and build SoftBridge prototype system: According to the results obtained during the requirements analysis and service definition, the rapid design and implementation of the SoftBridge prototype are carried out. General design specification and user manual should be finished in this phase.

3. Using and verifying the prototype system: After the first prototype system is finished, it is used and evaluated by experimental users with the help of the user manual. Service definition and general design specification are used as standards to validate the adequacy of the system.
4. Refine the prototype system if the prototype system is not adequate. Refining is iterated until it is adequate enough to deliver the final system.
5. Finally, deliver of the final system and the related technical documents.

3.2.2 Comparison in empirical evaluation

An empirical evaluation of a SoftBridge application is used to prove the proposed approach [56]. The process of evaluation is illustrated in Figure 14.

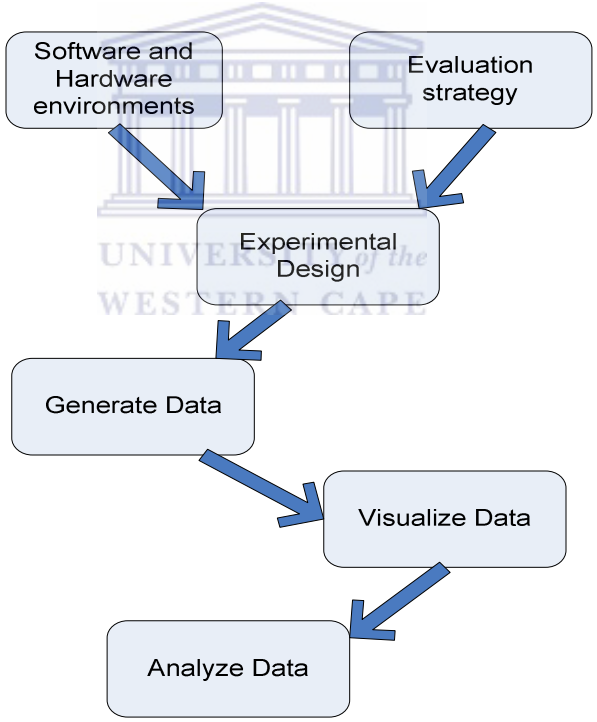

Figure 14 Evaluation process

Figure 14 shows main steps of an empirical evaluation. With software and hardware environments, the experiments are designed and carried out according to evaluation strategy, and then experimental data is collected, visualized and analyzed.

3.2.2.1 The software and hardware environment for the evaluation

The evaluation is carried out in an experimental environment. The hardware environment includes PCs, telephones, a PBX (Private Branch eXchange), LAN (local area network). The software environment involves operating systems (OS) including both Windows and Linux, a SoftBridge application – SIMBA (see Chapter 4.5), a JSLEE implementation – Rhino (see Chapter 4.5), client application and a simulator (as shown in Figure 15).

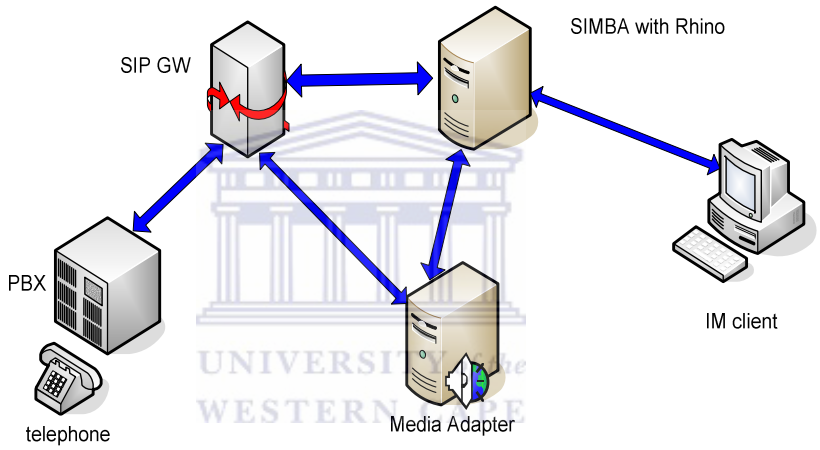

Figure 15 Software/hardware environment in the evaluation

Figure 15 shows some of the software and hardware involved in the evaluation. An IM client on a PC sends the message to a telephone on a PBX through the SoftBridge application (SIMBA) with a JSLEE (Rhino) and a SIP gateway.

3.2.2.2 Evaluation strategy

SIMBA with and without Rhino is evaluated in the same experiment. The test data with and without Rhino in the same case is recorded and analyzed in the form of tables. Through the comparison of these two systems, it is expected to show that a JSLEE can provide life-cycle management for a SoftBridge application. The evaluation strategy is illustrated in Figure 16.

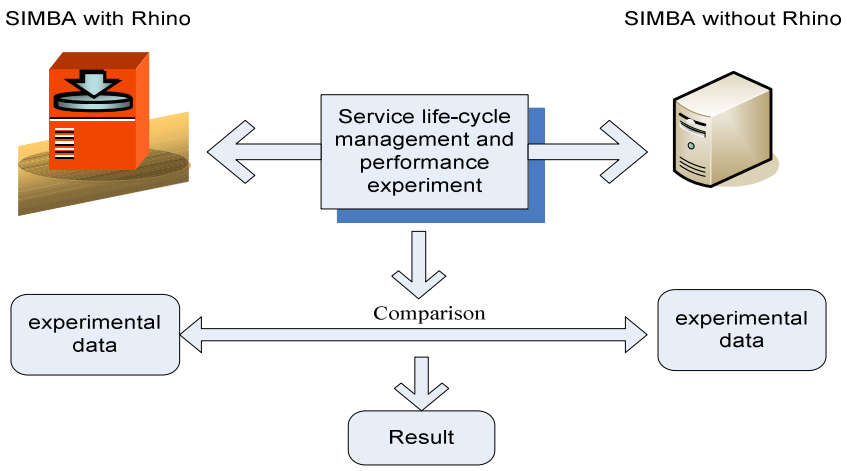

Figure 16 Comparison strategy

Figure 16 shows that service life-cycle experiment and performance experiments are carried out between a SoftBridge application (SIMBA) with and without a JSLEE (Rhino). The experimental data is compared and the experimental result is expected to show a SoftBridge application can be managed in a JSLEE.

3.2.2.3 Experiment involved in evaluation

Experiments in the evaluation address service life-cycle management and performance. The service life-cycle management experiment involves SIMBA deployment, activation, deactivation, uninstallation and online upgrade. The Performance experiment includes Registration per Second (RPS) and Call per Second (CPS) [43].

3.2.2.4 Evaluation steps:

The exploratory prototyping can be divided into the following phases:

1. Generate data: test the process of service life-cycle management of SIMBA using the HTTP (Hyper Text Transfer Protocol) management interface provided by Rhino. To determine the RPS and CPS values, the request rate from a simulator to SIMBA is increased until the failure probability increases to 5%. The highest sustained throughput is reported as the RPS and CPS values.

2. Visualize data: visualize the result data of life-cycle management of SIMBA in tabular form. The table lists the life-cycle management capabilities supported by SIMBA. Visualize the performance data with tables.
3. Analyze data: compare and analyze the visualized results of SIMBA with and without Rhino using tables and find out the advantage of the SoftBridge application in a JSLEE with regard to carrier grade capabilities.

3.3 Summary

In this chapter, the proposed approach to the research question is presented. Two methodologies used in this research are described, including exploratory prototyping in the development of a SoftBridge application – SIMBA (see Chapter 4.5), and comparison in an empirical evaluation to SIMBA with regard to service life-cycle management capabilities. Using exploratory prototyping methodology, SIMBA can be designed and implemented. An empirical evaluation of SIMBA includes the experiments with service life-cycle management and performance. By comparing the experimental results of between the SoftBridge application with and without a JSLEE – Rhino (see Chapter 4.5), it is expected that a JSLEE can provide life-cycle management for SoftBridge applications.

Chapter 4 SYSTEM DESIGN

For this thesis, a SoftBridge application is built according to the service development mode of the JSLEE specification. Here the SoftBridge application is called SIMBA (SoftBridge for Instant Messaging Bridging Application). Although the Deaf Telephony service is similar to a previous Deaf Telephony SoftBridge [26], SIMBA's SoftBridge design and implementation are unique to this thesis. SIMBA is used as an experimental representation of a SoftBridge application in the evaluation to show that a JSLEE is able to provide service life-cycle management for SoftBridge applications.

4.1 Service definition

SIMBA enables a text-based IM client to communicate with an IP phone, a telephone or a cellular phone. Similar to the Deaf Telephony SoftBridge, SIMBA provides a bridging service, which enables a Deaf user with a text-based IM client to communicate with a hearing user with a telephone or a cellular phone [26]. Using an IM client, a Deaf user sends a text message to a telephone user through SIMBA. SIMBA establishes a call to the telephone user and converts text messages to speech via a Media Adapter Server (MAS). When the called user picks up the phone, he/she hears the synthetic voice and speaks to the Deaf user. After receiving audio from the hearing user, SIMBA then controls the MAS to convert the incoming audio stream to text and sends the text message to the Deaf user. The conversion between text and speech is transparent to both hearing and Deaf users. The whole process is shown in Figure 17.

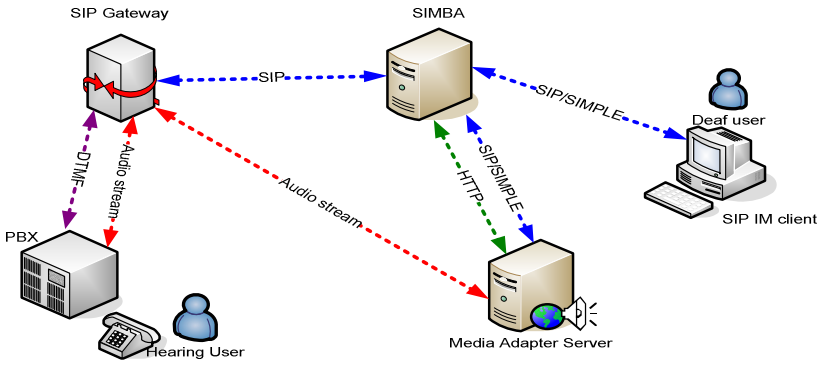

Figure 17 SIMBA service flow

Figure 17 shows how SIMBA bridges a Deaf user with an IM client to communicate with a hearing user using a telephone. SIMBA receives a text message from an IM client and then establishes a call to the telephone on a PBX. After the call is established successfully, SIMBA notifies Media Adapter Server (MAS) to open an audio stream to the telephone. MAS gets the text message from SIMBA, converts to voice and transmits the voice to the telephone. After receiving the voice from the telephone, MAS converts it to text and sends a text message to the IM client through SIMBA.

4.2 System requirements

The system requirements are as follows:

- Support SIP and SIP Extension for Instant Messaging and Presence (SIMPLE) [11] [35] [40] [41]. This system should function as a SIP Proxy, registrar and presence server. As a presence server, it is able to process NOTIFY and SUBSCRIBE requests. As a proxy, it is able to add and process the RECORD-ROUTE and ROUTE headers specified in the request besides basic proxy capability. The proxy can authenticate all requests that it receives and fork the INVITE requests it receives.
- Convert between two media types: text and voice. Two modalities of media bridging are required: text to speech bridging and speech to text bridging.
- Support IM clients based on SIP/SIMPLE. There are two messaging modes: page mode and session mode, but the system only supports page mode so far.

- Support multiple users. This system should support multi-user, parallel call requests and set up multi-call to telephones or SIP clients.
- Service execution and management environment: the system can be deployed activated, deactivated and uninstalled in a JSLEE.
- Support online upgrade. An upgrade can be carried out during execution. The "connected" call can continue during the upgrade and terminate as normal.
- Support profiles configure and log/trace.

4.3 System functionality

This system includes two independent parts: SIMBA and MAS, as shown in Figure 18. SIMBA includes five main functional modules: Proxy, Registrar, Presence, Bridging and Communicator. MAS includes three functional modules: Media adapter Servlet, TTS/ Automatic Speech Recognition (ASR) and Media Send/Recv. SIMBA communicates with MAS through the HTTP interface. [13].

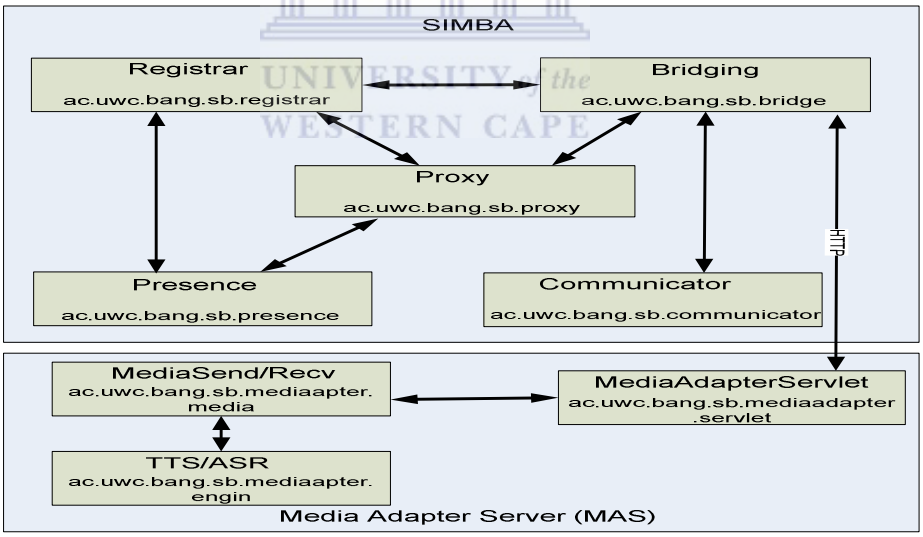

Figure 18 Functional modules architecture

Figure 18 shows the main functional modules in SIMBA and MAS and their relationships. SIP messages are processed by Proxy, Registrar and Presence modules. Bridging module invokes Communicator module to establish and terminate a SIP call. Bridging module controls MediaAdapterServlet module to open, update and close a media stream by invoking Media Send/Recv module. TTS/ASR module performs the conversion between text and voice.

4.3.1 Proxy

The proxy function acts as a SIP proxy, which routes SIP requests to user agent servers (UAS) and routes SIP responses to user agent clients (UAC). A request may traverse several proxies on its way to a UAS. Each proxy will make routing decisions, modifying the request before forwarding it to the next element. Responses are routed through the same set of proxies traversed by the request in the reverse order. According to [41], Proxy behavior of processing a request includes the following steps:

1. Validate the request;
2. Preprocess routing information;
3. Determine target(s) for the request;
4. Forward the request to each target;
5. Process all responses.

4.3.2 Registrar

The registrar function acts as a SIP registrar, which accepts REGISTER requests and places the information it receives from those requests into the location service for the domain it handles. According to [41], Registrar behavior of processing a request includes:

- Authenticate the UAC;
- Add, remove, update the bindings for the domain it handles;
- Manage the expiration time.

4.3.3 Presence server

The presence server function acts as a SIP presence server, also known as a presence agent (PA) [35]. A PA is a SIP user agent, which is capable of receiving SUBSCRIBE requests, responding to them, and generating notifications of changes in presence state (e.g. online, offline, busy and away). A PA must have knowledge of the presence state of a presence entity. The way to access presence data of a presence entity is by co-locating the PA with the proxy/registrar.

4.3.4 Bridging

The bridging function controls the bridging between a *Text* message session and a *Voice* media session. Bridging behavior when processing a request includes the following steps:

1. Process a MESSAGE request whose destination is a telephone user (e.g. 9110@sipgateway.com).
2. Setup a call with the telephone user by invoking call methods of the Communicator module and indicate the IP address and port number of the media adaptor server in the Session Description Protocol (SDP) data with an INVITE request [19].
3. Notify MAS to start a RTP transmission session and send the text body of the MESSAGE request to it.
4. Receive the recognized text message from the IM agent of MAS and forward it to the client, who sends the MESSAGE request.

4.3.5 Communicator

The Communicator function acts as a SIP user agent (UA). It performs the function of establishing, modifying and terminating a SIP call to the callee (telephone or SIP UA). It is invoked by the bridging module when a MESSAGE request is sent, whose destination is a telephone user. It includes two parts: a user agent client (UAC) and user agent server (UAS).

- The UAC creates an SIP request, e.g. INVITE, and then uses the client transaction state machinery to send it.
- The UAS generates a response to a SIP request. The response accepts, rejects, or redirects the request.

4.3.6 Media Adapter Server

MAS is built in conjunction with a Web Server [53], whose functions include: media transmissions and media adaptation between text and speech, including TTS and ASR. MAS is independent of SIMBA and receives HTTP request from SIMBA to start a

RTP media session, send synthesized voice to the telephone and send the recognized text message to the IM client through SIMBA. The media adaptor server includes three parts: Media Adaptor Servlet, media transmitter/receiver and TTS/ASR Engine.

4.3.6.1 Media adapter Servlet

After receiving the HTTP request from a SoftBridge or other clients, the Media Adapter Servlet opens, updates or closes a media Real-Time Transport Protocol (RTP) session between the MAS and the SIP GW or the SIP client [42]. The Media Adapter Servlet extracts the content of the text parameter received from a HTTP request and invokes a TTS/ASR engine to perform the conversion between text and speech.

4.3.6.2 Media transmitter/receiver

The media transmitter/receiver performs the following functions:

- Start, update or close a media RTP session,
- Send the synthesized voice,
- Receive audio data and play it through a speaker.

Based on RTP and Real-Time Transport Control Protocol (RTCP), the media transmission is built using the JMF API [48].

4.3.6.3 TTS and ASR

The TTS engine performs the function of converting text to speech [55]. It provides the interface for the Media Adapter Servlet and saves the synthetic speech into an audio file.

The ASR engine: a Wizard of Oz (WoOz) is used to replace the poor performance of ASR engine [16]. A person called the “ASR agent” is used to perform the function of speech recognition. When the ASR agent receives audio from a speaker, he/she converts the audio into a text message and sends it to the IM client. An ASR agent is responsible for five bridging sessions.

4.4 Life-cycle Management of SIMBA in a JSLEE

4.4.1 Design strategy for SIMBA deployment and uninstallation

4.4.1.1 Event driven SBB

The JSLEE specification defines that an application is composed of one or more components, which are known as Service Building Block (SBB) components [27]. SBB components are event driven components that receive requests in the form of events. Each SBB component identifies the event types accepted by the component. It has event handler methods that contain application code that processes events of these event types. In addition, an SBB component may have an interface for synchronous method invocations. SIMBA is built with SBB components receiving SIP events, including SIP request events, response events and transaction management events.

4.4.1.2 Deployable unit

The JSLEE specification defines a deployable unit as a Jar file that can be installed in the SLEE and uninstalled from the SLEE [27]. A deployable unit may contain SBB jar files, event jar files, profile specification jar files and resource adaptor jar files. Each of these jar files contain the Java class files and the deployment descriptors of one or more of these components.

4.4.1.3 SIMBA packaging for deployment and uninstall

To a deployable unit of SIMBA, the Jar packages involved are shown at Table 1.

Jar Package Name	Items included in Jar
JAIN-SIP resource adaptor (RA) Jar	1. JAIN-SIP RA Jar deployment descriptor 2. Class files of JAIN-SIP RA. 3. JAIN-SIP RA event Jar deployment descriptor 4. Class files of JAIN-SIP RA event 5. JAIN-SIP RA type Jar deployment descriptor 6. Class files of JAIN-SIP RA type

SIMBA SBB Jar	1. SIMBA SBB Jar deployment descriptor 2. Class files of SIMBA SBB specified by the sbb elements of the sbb-jar element in 1.
SIMBA SIP event Jar	1. SIMBA SIP event Jar deployment descriptor 2. Class files of SIMBA SIP event specified by the sbb elements of the sbb-jar element in 1.
SIMBA custom event Jar	1. SIMBA custom event Jar deployment descriptor 2. Class files of SIMBA custom event specified by the sbb elements of the sbb-jar element in 1.
SIMBA profile Jar	1. SIMBA profile Jar deployment descriptor 2. Class files of SIMBA profile specified by the sbb elements of the sbb-jar element in 1.

Table 1 Deployable unit in SIMBA

Table 1 lists the Jar packages in a SIMBA deployment unit. Each Jar package includes a deployment descriptor and the class files in the Jar package.

4.4.1.4 SIMBA deployment and uninstallation in a JSLEE

In the JSLEE specification, the DeploymentMBean interface defines the management API for installing and removing components [27] :

- The Install method. This method installs the deployable unit jar file identified by the URL argument. Each component jar file contained in the deployable unit is installed by the SLEE. If all of these components are installed successfully, the JSLEE then installs any services contained in the deployable unit jar file. The deployable unit is only successfully installed if all of its included component jars and services install successfully. Thus the deployable unit for SIMBA can be deployed into JSLEE through the install method in the DeploymentMBean interface.
- The Uninstall method. This method uninstalls the deployable unit specified by the ID argument. Uninstalling a deployable unit causes all the components in the deployable unit to be uninstalled. A deployable unit cannot be uninstalled if any

component contained in any other deployable unit depends on a component contained in the deployable unit being uninstalled. Thus the deployable unit for SIMBA can be removed from JSLEE through the uninstall method in the DeploymentMBean interface.

4.4.2 Design strategy for SIMBA activation and deactivation

4.4.2.1 States in the life-cycle of SIMBA in JSLEE

After deployment into the JSLEE, SIMBA can be in one of the following three operational states (as shown in Figure 19) in the life cycle.

- Inactive. SIMBA has been installed successfully and is ready to be activated. SIMBA is not running, i.e. root SBB entities of the SIMBA's root SBB will not be created to process events.
- Active. SIMBA has been activated, i.e. it is running. The JSLEE will create root SBB entities of SIMBA's root SBB to receive initial events and invoke SBB entities in the SBB entity trees of the Service.
- Stopping. SIMBA is being deactivated. However, some SBB entity trees of SIMBA still exist in the JSLEE and have not completed their processing. The JSLEE waits for the SBB entities in these SBB entity trees to complete processing so that they can be reclaimed. A SBB entity has completed processing and can be reclaimed when it and all of its child SBB entities are no longer attached to any Activity Context.

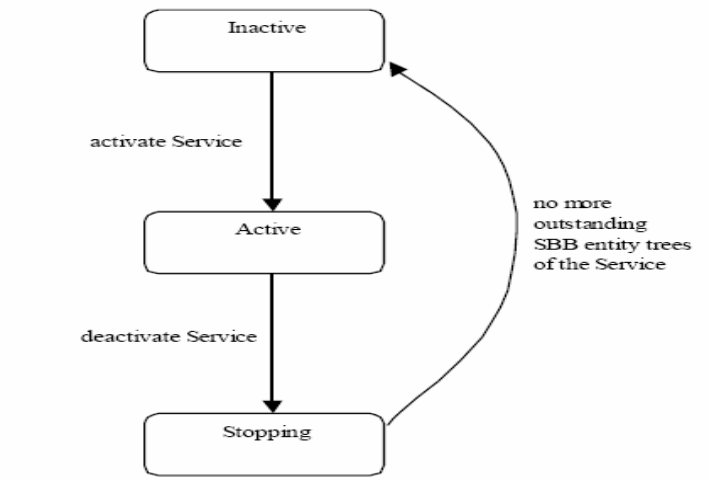

Figure 19 Operational states of SIMBA

Figure 19 shows three states of SIMBA in a JSLEE and the actions causing the state conversion [27]. SIMBA is in "inactive" state after it is installed in the JSLEE. The state of SIMBA changes to "active" after SIMBA is activated. The "stopping" state means that SIMBA is being deactivated.

4.4.2.2 SIMBA activation and deactivation in a JSLEE

In the JSLEE specification, the ServiceManagementMBean interface defines the management API for Services installed in the SLEE [27]. It can be used to change the operational state of SIMBA.

- The Activate method. This method activates the service specified by the ID argument. It can only be invoked when the service is in the inactive state. If it returns successfully, the service has transitioned to the Active state. Thus, SIMBA can be activated by the Activate method in the ServiceManagementMBean interface.
- The Deactivate method. This method deactivates the service specified by the id argument. It can only be invoked when the service is in the Active state. If it returns successfully, the service has transitioned to the stopping state. Eventually the service transitions to the inactive state when all SBB entities executing for the service have completed processing. Thus, SIMBA can be deactivated by the Deactivate method in the ServiceManagementMBean interface.

4.4.3 Design strategy for online upgrade of SIMBA

4.4.3.1 The process of SIMBA online upgrade in JSLEE

In the JSLEE specification, the ServiceManagementMBean interface defines the DeactivateandActivate API [27]. It can be used for online upgrade of SIMBA. The DeactivateandActivate method takes on two arguments - the service to deactivate, and the service to activate in its place. This operation is performed in a single transaction. To make use of this, a new version of SIMBA needs to be installed. The new one should have a different name or version number in the service deployment descriptor. The deactivateAndActivate method is used to pass the old SIMBA to the new version. The old version will go into the "deactivating" state, and any activities that it is processing are allowed to "drain". It will be deactivated when all its activities and SBB entities are finished. Meanwhile the new version of SIMBA is activated and any new initial events will go to the new one. Therefore no events will be lost. They will either go to the old or new version of SIMBA.

4.4.3.2 The issue of call interruption during the online upgrade

A complete call session starts when an IM client sends the first MESSAGE request to a telephone user and SIMBA sets up a call to the telephone; it ends when the IM client send a MESSAGE request with the text body of "bye" and SIMBA terminates the call.

An online upgrade requires that a call session existing in old SIMBA cannot be interrupted when the new SIMBA replaces the old SIMBA. The main problem is that the MESSAGE event from the IM client is the initial event; therefore any new MESSAGE events will be dispatched to the SBB entity of a new SIMBA. It will cause the new SIMBA to set up a new call to the same telephone user, but actually the telephone is already in the “working” status. The existing call session is interrupted due to the operation of an upgrade. The process of call interruption is shown in Figure 20.

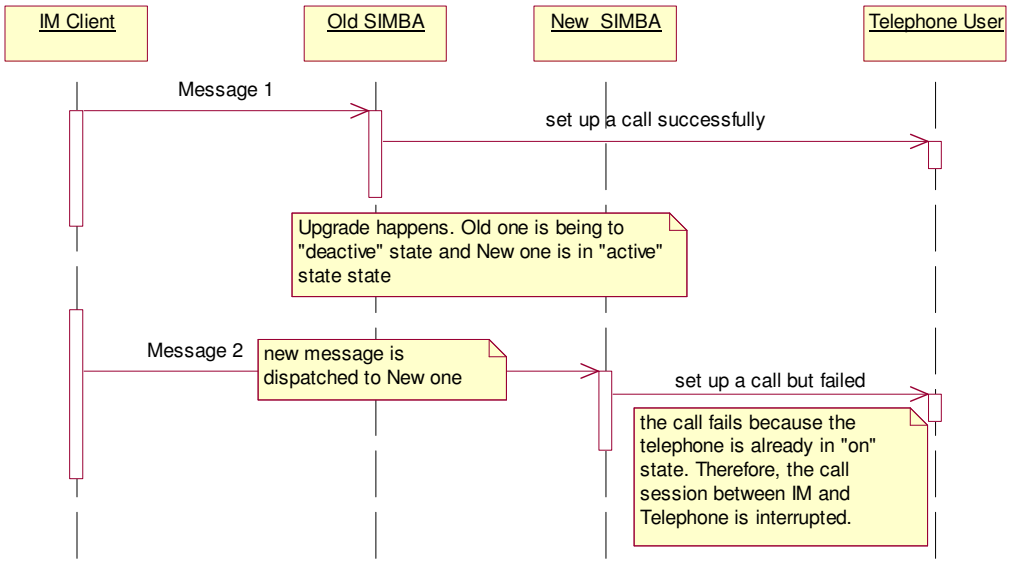

Figure 20 Call interruption during the online upgrade

Figure 20 describes the process of call interruption during the online upgrade. When online upgrading, old SIMBA is deactivated and new SIMBA is active. New SIMBA cannot keep the state of existing call session and cause the existing call session interruption.

4.4.3.3 The solution to avoid call interruption during the online upgrade

A custom event is used to sort out the issue of call interruption in the online upgrade. The custom event is a low level event in the JSLEE. A SBB entity can receive custom events even if it is in the “stopping” state.

Assume that a SBB entity of the old SIMBA (call it SBB1) has received the initial MESSAGE from the IM client. To make sure it stays alive for the whole session, SBB1 must be attached to an activity the whole time. There is no single activity at the protocol level that it will stay attached to (ie. all the SIP transactions are separate, short-lived activities). Thus SBB1 creates a null activity and attaches to it. The activity is assigned a unique name–the SIP URL of the IM client (call it Session-SBB1).

SBB1 does the INVITE transaction with the telephone through the gateway. Upgrades occurring at this stage have no influence here because SBB1 is attached to

the ClientTransaction activity, so SBB1 will automatically get the response. The response is not an initial event so no other SBBs are created yet.

The operation of upgrade is implemented at the moment. Old SIMBA transfers to the “stopping” state, which is leading to the inactive state. A new MESSAGE arrives from the IM client. SBB1 does not see it because it is an initial event, and SBB1 is not attached to the new SIP ServerTransaction activity. The SLEE creates a new SBB entity of new SIMBA (call it SBB2) to handle the MESSAGE event.

SBB2 can look up the activity named "Session-SBB1" in the AC naming facility. If it finds that it exists, SBB2 can fire a custom event on it. SBB1 will receive the custom event on its activity. The custom event object could take the details of the message from the IM client, such as text body of the message and SIP URL of the IM client. SBB1 handles the message with normal processing. If the text body of the message is “bye”, SBB1 would terminate the telephone call. SBB1 can then detach from the null activity and clean it up. Further MESSAGE requests from the IM client or other IM clients will cause new SBB entities of the new SIMBA, to be created. The process of avoiding call interruption in the online upgrade is described in Figure 21.

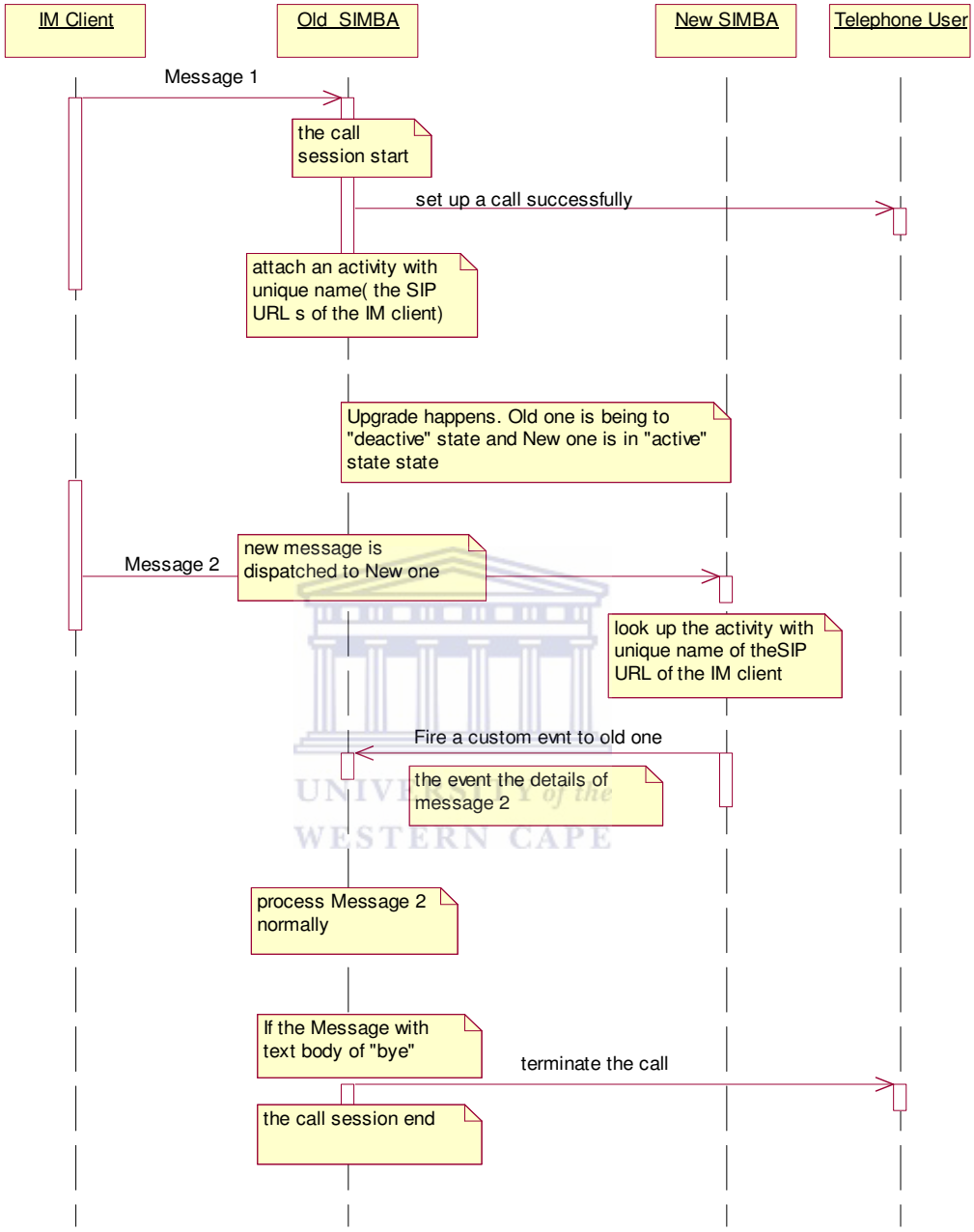

Figure 21 Solution to avoid call interruption during the online upgrade

Figure 21 shows that new SIMBA forwards the message of the existing call session to old SIMBA via custom event. Old SIMBA processes the message of existing call session until the existing call session is ended normally. New SIMBA is responsible for processing the new message. It enables SIMBA to avoid call interruption during the online upgrade.

4.5 System implementation

SIMBA provides similar services to the Deaf Telephony SoftBridge that was built on an open Jabber IM platform [26]. The bridging services were redesigned for SIMBA. In addition, the implementation of SIMBA is based on a JAIN-SIP Proxy, which is an open source Java project distributed by National Institute of Standards and Technology (NIST) [37]. The JAIN-SIP Proxy system acts as a SIP proxy, registrar and presence server. SIMBA implements bridging between text and voice on the JAIN-SIP Proxy system. Current SIMBA only supports the call established by an IM client to a telephone and does not handle the call initiation from a telephone.

MAS is built as a Java Servlet and communicates with SIMBA using an HTTP interface. The FreeTTS engine is embeded into MAS to perform the conversion from text to speech [55]. The ASR agent is responsible for converting voice into text and then SIMBA forwards the text message to the IM client.

Two versions of SIMBA have been developed for the evaluation: SIMBA with and without a JSLEE. These two versions of SIMBA provide the same service, but one is executed and managed within a JSLEE, the other is executed in a stand alone mode. Rhino is used as an implementation of JSLEE standards to provide life-cycle management for SIMBA [39]. SIMBA with and without Rhino implement the same source code for core functions, such as SIP proxy, registrar, presence server and bridging. Comparing with SIMBA without Rhino, SIMBA with Rhino implements the interfaces to be deployed into Rhino so that Rhino can provide life-cycle management of SIMBA.

Rhino provides a management web interface to manage the life-cycle of SIMBA, including deployment, uninstallation, activation, inactivation and online upgrade. To integrate SIMBA with Rhino, SIMBA is packaged into four deployable Jars: SIMBA SBB Jar, profile Jar, SIP event Jar, custom event Jar. These Jars consist of class files and corresponding deployment descriptor files. Through the Rhino deployment web interface, these four Jars can be deployed into Rhino and also be uninstalled from Rhino (as shown in Figure 22). After SIMBA Jars are installed into Rhino successfully, SIMBA can be activated through the Rhino service management web

interface (as shown in Figure 23). “Active” SIMBA can process the call request from an IM client to a telephone. SIMBA can also be deactivated and “inactive” SIMBA still resides in Rhino, but cannot process the call request.

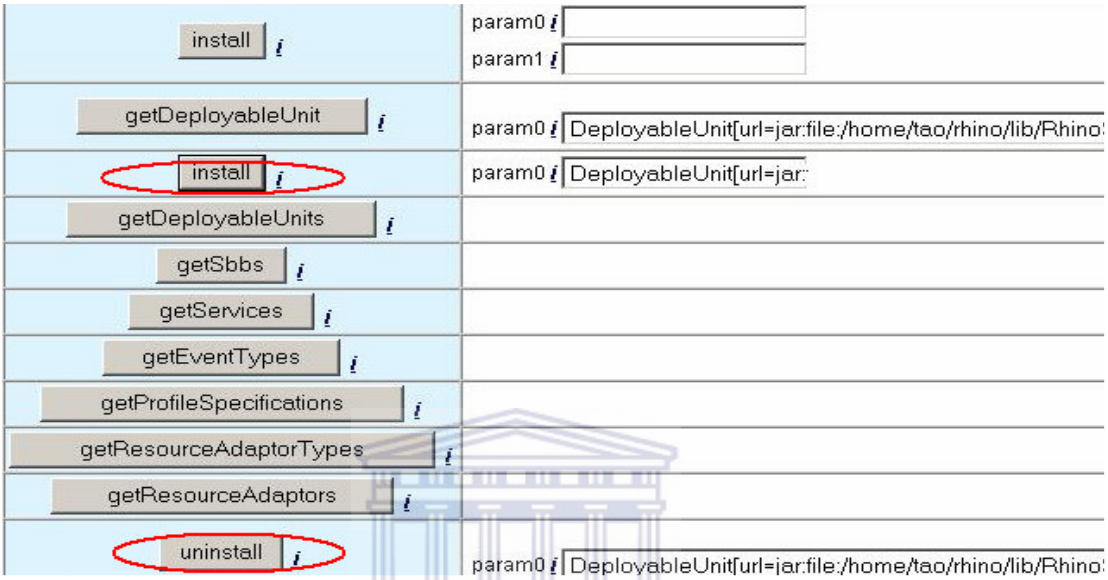

Figure 22 Rhino deployment interface to install and uninstall SIMBA

Figure 22 is a capture of the Rhino web UI. It shows the Rhino deployment web interface. The “install” button with a circle is used to deploy SIMBA into Rhino and the “un install” button with a circle is used to unintall SIMBA from Rhino.

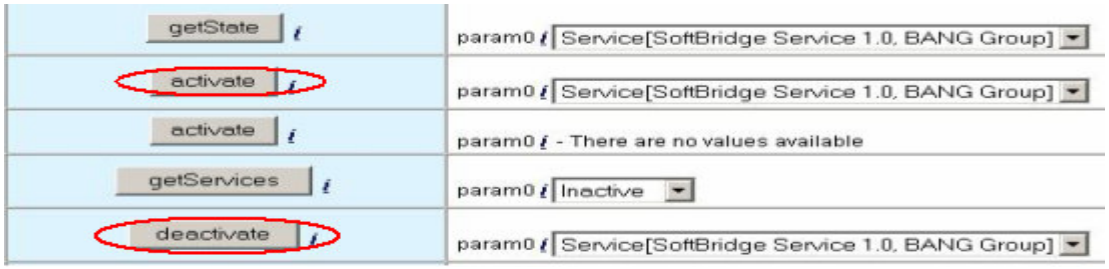

Figure 23 Rhino service management interface to activate and inactivate SIMBA

Figure 23 is a capture of the Rhino web UI. It shows the Rhino management web interface. The “activate” button with a circle is used to activate SIMBA and the “deactivate” button with a circle is used to deactivate SIMBA.

An Online upgrade requires that a call in-process does not get interrupted during the upgrade operation. That means a call is already established when the upgrade operation occurs. The call can continue and terminate as normal. It is assumed that

SIMBA 1.0 is activated by Rhino and processes the request from the IM client. After the call is already established, SIMBA 1.1 is deployed into Rhino. Through the service management web interface (as shown in Figure 23), SIMBA 1.0 is deactivated and version 1.1 is activated. SIMBA 1.1 begins to process the new request and version 1.0 continues to process the call until it terminates as it normally does.

Figure 24 Rhino service management interface to upgrade SIMBA online

Figure 23 is a capture of the Rhino web UI. It shows the Rhino management web interface. The "deactivateAndActivate" button with a circle is used to activate SIMBA 1.1 and deactivate SIMBA 1.0.

4.6 Summary

This chapter describes the design and implementation of a SoftBridge application called SIMBA. SIMBA enables a Deaf user with a text-based IM client to communicate with a hearing user with a telephone or a cellular phone. The system functional architecture includes two independent parts: SIMBA and MAS. SIMBA is built upon an open JAIN-SIP Proxy that acts as a SIP proxy, registrar and presence server. SIMBA implements bridging between text and voice on the JAIN-SIP Proxy system. SIMBA communicates with MAS through the HTTP protocol. SIMBA life-cycle management in a JSLEE includes deployment, activation, deactivation, uninsatllation and online upgrade. SIMBA is packaged in a deployable unit, and then installed or uninstalled in Rhino, a JSLEE. Through the Rhino service management interface, SIMBA can be activated and deactivated in Rhino. An Online upgrade requires the "connected" call to continue during an upgrade and terminate as normal. SIMBA adopts the solution of a custom event to avoid call interruption during an online upgrade in Rhino.

Chapter 5 EXPERIMENTAL DESIGN

An empirical evaluation of SIMBA is carried out. The evaluation involves experiments with both SIMBA life-cycle management and performance. SIMBA with a JSLEE and SIMBA without a JSLEE are evaluated in the same experiments. Through comparing the experimental results between SIMBA with and without a JSLEE, it is expected that a JSLEE can provide SIMBA life-cycle management without causing a significant decrease in performance.

5.1 Software and hardware environments in the experiment

5.1.1 Software environment

The software environment in the experiment includes operating system (OS) software and applications software, all of which are detailed in Table 2.

Name	Description
OS	Microsoft Windows 2000, Service Pack 3.
	Linux RedHat 7.3 Kernel 2.4.18.
J2SE1.4.2 [49]	Java(TM) 2 SDK, Standard Edition version 1.4.2. A development environment for building applications, applets, and components that can be deployed on the Java platform.
Rhino 1.3.0-beta5 [39]	Open Cloud Rhino is a JSLEE standards compliant service logic execution environment (SLEE) for carrier grade implementations. It includes everything in the SLEE category, SIP and JCC Resource Adaptors, Enterprise Integration features and example SIP and JCC applications for service development.
SIMBA	SIMBA with Rhino, which runs in Rhino 1.3.
	SIMBA without Rhino, which runs in J2SE 1.4.2.
MAS	A Web Server, whose functions include two parts: media transmission and media adaptation (between text and speech).
NIST-SIP IM client [36]	A JAVA based SIP IM client (open source) built on top of the JAIN-SIP-1.1 API. It supports:

	▪ IM capability: process MESSAGE requests ▪ Presence information capability: process PUBLISH, NOTIFY and SUBSCRIBE requests ▪ Registration capability: Register and unregister itself to a proxy
SIPRG [50]	The SIP Residential Gateway (SIPRG) is an open source application based on SIP. The SIPRG is an IP Telephony Gateway that allows a SIP User Agent to make and receive calls between PSTN/PBX and a SIP-based network. The SIPRG is developed with the VOVIDA SIP stack version 1.3.0, and uses a QuickNet LineJACK card for connecting a telephone line in a PBX. Currently, it supports only a single LineJACK card and is therefore a single-line gateway.
Sipp [14]	Sipp (open source) is a performance testing tool for the SIP protocol. Its main features are basic SIPStone scenarios, TCP/UDP transport, customizable (xml based) scenarios, dynamic adjustement of call-rate and a comprehensive set of real-time statistics. Sipp can be used to test many real SIP equipment like SIP proxies, B2BUAs, SIP media servers, SIP/x gateways, SIP PBX. It is also very useful to emulate thousands of user agents calling the SIP system or receive thousands of SIP calls from a SIP system.
IM Simulator	IM simulator is a performance testing tool for SIP/SIMPLE based IM system. It can emulate thousands of MESSAGE and REGISTER requests to the SIP system like proxy.
Apache Ant 1.6.1 [2]	Apache Ant is a Java-based build tool, which is used to build SIP Resource Adaptor.
PostgreSQL 7.4.2 [51]	PostgreSQL is an enhancement of the POSTGRES database management system, which is used as a background database server for the Rhino system.

Table 2 Software environment

5.1.2 Hardware environment

The hardware equipment being used in the experiment includes PC, speaker, microphone, voice card, PBX, telephone and LAN, all of which are detailed in Table 3. The experimental network topology is shown in Figure 25.

Name	Description
PC (as shown in Figure 25)	Intel Pentium 4 CPU 1.8Ghz, 416M RAM, 40G Hard driver. PCa: Microsoft Windows 2000 , j2SE 1.4.2. PCb: Linux RedHat 7.3 , j2SE 1.4.2.
	Intel Pentium 2 CPU 400Mhz, 416M RAM, 10G Hard driver PCc: Linux RedHat 7.3.
QuickNet LineJACK	QuickNet LineJACK is a single line, VoIP (Voice over IP) gateway card, whose PSTN port can connect a telephone line of a PBX. It runs with a VoIP gateway, such as SIPRG.
Siemens Hicom 150H (PBX)	Hicom 150 H is a PBX for small and medium-sized enterprises, complete with in-built support for all the requirements associated with an IP-based corporate network.
Siemens euroset 805 S (telephone)	Telephone, used to set up or receive a PSTN call.
Ethernet	100M bps, support TCP(Transmission Control Protocol) /IP 100M network adapter.
Speaker/Micro phone	Speaker/Micro phone are used as input and output voice.

Table 3 Hardware environment

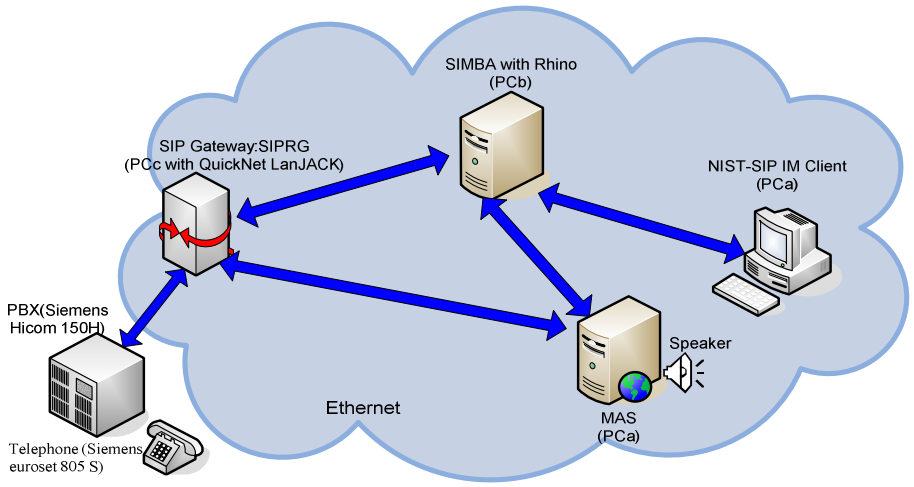

Figure 25 Experimental network topology

Figure 25 *shows SIMBA with Rhino on PCb, MAS on PCa, NIST-SIP IM client on PCa and SIPRG on PCc, connected through an Ethernet. A Siemens Hicom 150H PBX connects with a SIPRG gateway through a QuickNet LanJACK card.*

5.2 Experiment design

5.2.1 SIMBA life-cycle management experiment

5.2.1.1 Experiment strategy

The aim of this experiment is to show that a JSLEE (Rhino) can provide life-cycle management for a SoftBridge application (SIMBA). For the sake of comparison, SIMBA with Rhino and SIMBA without Rhino are involved in this experiment. The life-cycle management experiment includes deployment, activation, deactivation, uninstallation and online upgrade. The experimental results of SIMBA with and without Rhino are compared and analyzed. It is expected to show that advanced SIMBA with Rhino supports all life-cycle management capabilities.

5.2.1.2 SIMBA deployment and uninstallation in Rhino

This experiment includes SIMBA with Rhino on PCb. Rhino provides a management web interface to manage the services. SIMBA can be deployed into Rhino through the deployment web interface, including service jar package, event jar package and

profile jar package. Using the profile provisioning web interface, the profile of SIMBA can be created, modified and removed. When SIMBA is in the inactive state, it can be uninstalled from Rhino through the deployment web interface.

5.2.1.3 SIMBA activation and deactivation in Rhino

After SIMBA is deployed into Rhino successfully, it can be activated through the service management web interface. An IM client can communicate with a telephone client through an "active" SIMBA and MAS. SIMBA can also be deactivated through the service management web interface.

5.2.1.4 SIMBA online upgrade in Rhino

An online upgrade requires that a call in-process does not interrupt during the upgrade operation. That means a call is already established when the upgrade operation occurs. The call can continue and terminate as normal. In this experiment, an old version of SIMBA (called version 1.0) is activated by Rhino and processes the request from the IM client. After the call is already established, a new version of SIMBA (called version 1.1) is deployed into Rhino. Through a service management web interface, version 1.0 is deactivated and version 1.1 is activated. Version 1.1 begins to process the new request and version 1.0 continues to process the call until it terminates as it normally does. The online upgrade capability can be verified if the process is successful.

5.2.1.5 Life-cycle management of SIMBA without Rhino

SIMBA without Rhino performs life-cycle management by itself. Through a SIMBA UI, it can be started and stopped. For the upgrade operation, the operator must stop the old version and then start the new version. Therefore it cannot keep the established call going during the upgrade operation.

5.2.2 SIMBA performance experiment

The aim of the performance experiment is to show that service life-cycle management with a JSLEE does not cause significant decrease in performance. The performance experiment includes two test cases: RPS and CPS.

Registrations per second (RPS): Registrations per second is defined as the average number of successful registrations per second during the measurement interval.

Calls per second (CPS): Calls per second is defined as the average number of calls per second completed with a 2xx or 4xx response during a measurement interval. For a SIMBA test case, a single call includes both the INVITE and corresponding BYE transaction.

5.2.2.1 Performance experiment strategy

Both SIMBA with Rhino and SIMBA without Rhino are involved in this experiment. By comparing the performance results of implementing an experiment under each system, no significant performance difference is expected to exist between SIMBA with Rhino and SIMBA without Rhino even though SIMBA with Rhino performs a bit worse than SIMBA without Rhino.

To determine the RPS and CPS values, the request rate is increased by 10 (10, 20, 30, 40…) until the successful registration or call probability decreases to 95% that is generally accepted as the benchmark of evaluating RPS and CPS for carrier grade requirements [43]. To each request rate, the test is carried out five times and the value is determined by the average of five time test result.

A statistical approach, *t-test*, is used to compute the significance of the performance variance between these two systems. Even though the *t-test* does not offer a definitive measure, it does provide a good idea of the significance of relationships. The *t-test* is typically used to compare the means of two populations [52]. Therefore, to compare the variance of the CPS and RPS of two systems, SIMBA with Rhino and SIMBA without Rhino, t-test takes the responsibility.

5.2.2.2 RPS experiment

This experiment includes an IM Simulator on PCa and SIMBA with Rhino on PCb or SIMBA without Rhino on PCa. The register request rate (per second) from the IM simulator to SIMBA is increased by 10 (10, 20, 30, 40…). The numbers of successful or failed registrations are recorded. When the successful registrations probability decreases to 95%, the value of RPS is determined. The experiment is shown in Figure 26.

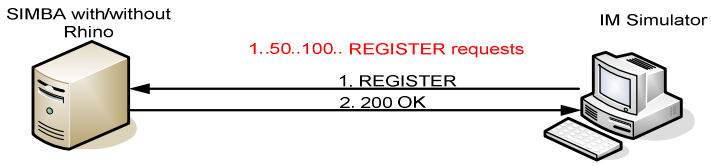

Figure 26 RPS experiment

Figure 26 shows that an IM simulator sends an increasing number of REGISTER requests (per second) to SIMBA and gets successful responses from SIMBA.

5.2.2.3 CPS experiment

This experiment (as shown in Figure 27) includes an IM simulator on PCa, a SIP UAS simulator (Sipp) on PCc, SIMBA with Rhino on PCb or SIMBA without Rhino on PCa. The call request rate (per second) from the IM simulator to SIMBA is increased by 10 (10, 20, 30, 40…). SIMBA then sets up the call to the SIP UAS simulator at the same increasing rate. A completed call includes both the INVITE and corresponding BYE transaction. The call holding time, after successful INVITE transaction is set to zero and the BYE transaction follows successful INVITE transaction immediately. The media transmissions are discarded. The number of successful or failed calls is displayed through the UI of Sipp (as shown in Figure 28). When the successful call probability decreases to 95%, the value of RPS is determined.

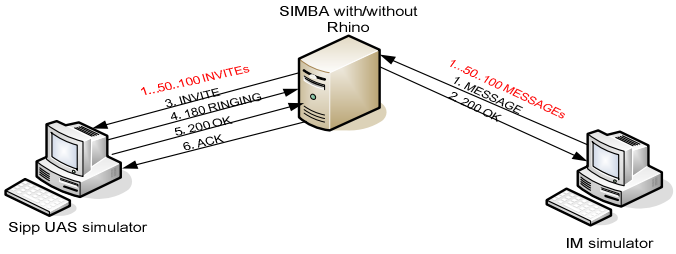

Figure 27 CPS experiment

Figure 27 shows that an IM simulator sends an increasing number of MESSAGE requests (per second) to SIMBA, which cause the same number of SIP calls to Sipp UAS simulator.

```
------------------------------ Scenario Screen -------- [1-4]: Change Screen --
  Port   Total-time  Total-calls  Transport
  5060      33.20 s           80  UDP

  0 new calls during 1.010 s period      20 ms scheduler resolution
  1 concurrent calls                     Peak was 51 calls, after 10 s
  1 open sockets

                                 Messages  Retrans   Timeout   Unexpected-Msg
  ----------> INVITE                80        0                  0

  <---------- 180                   80        0
  <---------- 200                   80       58        0
  ----------> ACK           E-RTD   79        0                  0

  ----------> BYE                   79        9                  0
  <---------- 200                   79        9
  [  4000 ms]
------------------------------ Sipp Server Mode -------------------------------
```

Figure 28 Sipp UI

Figure 28 shows that Sipp UI displays the number of successful INVITE transactions and BYE transactions, therefore, the successful and failed call numbers can be easily viewed. The interval between INVITE transaction and BYE transaction is set to zero.

5.2.2.4 *T-test* procedure

In order to compare the performance tests with and without the JSLEE, a standard *t-test* is performed. Assume that the null hypothesis is H_0: $\mu_1=\mu_2$, whereby μ_1 is the mean performance of CPS or RPS of SIMBA with Rhino and μ_2 is the mean performance of CPS or RPS of SIMBA without Rhino. This equation means that there is no significant variation between the performance of CPS or RPS of SIMBA with Rhino and without Rhino. Meanwhile, a one-sided alternative hypothesis H1: $\mu_1<\mu_2$ is assumed because SIMBA with Rhino will have little negative influence on the performance. Thus the significance level t of the performance variance of the two systems can be expressed as [52]:

$$t = \frac{\mu_1 - \mu_2}{s \cdot \sqrt{1/n_1 + 1/n_2}}$$

where s is obtained from s^2, the pooled estimate of the variance and calculated as:

$$s^2 = \frac{\sum_1^{n_1}(x_i - \mu_1)^2 + \sum_1^{n_2}(y_i - \mu_1)^2}{(n_1 - 1) + (n_2 - 1)} \quad and \quad s = \sqrt{s^2}$$

where the samples in group x and group y individually represent the values as to the performance of SIMBA with Rhino and SIMBA without Rhino. For example, the CPS values of five tests in SIMBA with Rhino are set as the samples of group x, while the samples in group y are the CPS values of five tests in SIMBA without Rhino. n_1, n_2 is the number of samples in group x and group y, and μ_1, μ_2 refers to the respective mean performance of group x and group y. Moreover, it is customary to carry out this one-way experiment at the 5% level of significance ($\alpha = 0.05$) and the degree of freedom is calculated as $df = (n_1 - 1) + (n_2 - 1) = 8$. Thus we can get $t_{\alpha/2} = t_{0.025} = 2.306$ for 8 df from table in [52].

According to the t-test principle, when the resultant t value is within (-2.306, 2.306), H_0 is accepted and the performances of SIMBA with Rhino and SIMBA without Rhino do not have significant variance. Vice versa, if the calculated t is out of the range, the two systems have significant variance in terms of performance.

5.3 Summary

In this chapter, the experimental design of SIMBA evaluation is described in detail, including software and hardware environments, the service life-cycle experiment design and the performance experiment design. For the sake of comparison, SIMBA with Rhino (a JSLEE representation) and SIMBA without Rhino are involved in the experiments. The life-cycle management experiment is used to show that Rhino is able to provide life-cycle management to SIMBA, including deployment, activation, deactivation, uninstallaton and online upgrade. The performance experiment is used to show that service life-cycle management capabilities provided by Rhino do not cause a significant decrease in performance.

Chapter 6 DATA COLLECTION AND RESULTS

This chapter details how the experimental data is collected, including SIMBA life-cycle management and performance experiments. The experimental data is displayed in tabular form. Using these tables and figures, the experimental results are analyzed.

6.1 SIMBA life-cycle management experimental data

6.1.1 Deployment and uninstallation in Rhino

Through the Rhino deployment web interface, SIMBA service jar package, event jar package and profile jar package are installed into Rhino (as shown in Figure 29). After successful deployment, the jar files of SIMBA are removed from Rhino (as shown in Figure 30). The figures show that Rhino is able to deploy and uninstall SIMBA.

Array [javax.slee.management.DeployableUnitID], length == 5 <expand>	
Position	Value
0	DeployableUnit[url=jar:file:/home/tao/rhino/lib/RhinoSDK.jar!/javax-slee-standard-types.jar]
1	DeployableUnit[url=file:///home/tao/rhino/examples/sip/sbjar/softbridge-event-1.0.jar]
2	DeployableUnit[url=file:///home/tao/rhino/examples/sip/sbjar/jainsip-1.1-local-ra.jar]
3	DeployableUnit[url=file:///home/tao/rhino/examples/sip/sbjar/softbridge-config-profile.jar]
4	DeployableUnit[url=file:///home/tao/rhino/examples/sip/sbjar/softbridge-1.0-service.jar]

Figure 29 Deploy SIMBA into Rhino

Figure 29 is a capture of the Rhino web UI. It shows the deployed units in Rhino. These units with a circle are the jar packages of SIMBA, including event, profile and service jar packages.

Array [javax.slee.management.DeployableUnitID], length == 2 <expand>	
Position	Value
0	DeployableUnit[url=jar:file:/home/tao/rhino/lib/RhinoSDK.jarl/javax-slee-standard-types.jar]
1	DeployableUnit[url=file:///home/tao/rhino/examples/sip/sbjar/jainsip-1.1-local-ra.jar]

Figure 30 Uninstall SIMBA from Rhino

Figure 30 is a capture of the Rhino web UI. Comparing with figure 28, the number of deployed units decreases from 5 to 2. It shows those deployed units of SIMBA have been removed from Rhino after the uninstallation operation.

6.1.2 Activation and deactivation in Rhino

Through the Rhino service management web interface, SIMBA is activated and is ready to process the request from the IM client (as shown in Figure 31). After the successful activation, SIMBA is deactivated through the Rhino service management web interface (as shown in Figure 32). The figures show that Rhino is able to activate and deactivate SIMBA.

Managed Object Invocation Result	
Object	SLEE:name=ServiceManagement
Operation	getState(javax.slee.ServiceID)
Return type	javax.slee.management.ServiceState
Description	

Service Active

Figure 31 Activate SIMBA in Rhino

Figure 31 is a capture of the Rhino web UI. It shows that the state of SIMBA is "active" after the activation operation. "Active" SIMBA is ready to perform the bridging service between IM client users and telephone users.

Managed Object Invocation Result	
Object	SLEE:name=ServiceManagement
Operation	getState(javax.slee.ServiceID)
Return type	javax.slee.management.ServiceState
Description	

Service Inactive

Figure 32 Deactivate SIMBA in Rhino

Figure 32 is a capture of the Rhino web UI. It shows that the state of SIMBA is "inactive" after the deactivation operation.

6.1.3 Online upgrade in Rhino

Through the Rhino service management web interface, SIMBA 1.0 is activated. It received the request from the IM user "tao" and set up a call to the telephone user "Bill". When the user "tao" and "Bill" are in "talking" state, SIMBA 1.1 is activated and SIMBA 1.0 is deactivated through the Rhino service management web interface (as shown in Figure 33). The call between user "tao" and "Bill" is not interrupted and ended as normal. When a new request from user "tao" is sent, SIMBA 1.1 began to process it. The experimental result shows that SIMBA is able to be upgraded during runtime without call interruption.

Figure 33 Online upgrade SIMBA in Rhino

Figure 33 is a capture of the Rhino web UI. It shows the online upgrade operation. Through the Rhino service management web interface, the state of SIMBA 1.0 and 1.1 can be checked (refer to Figure 31 and Figure 32) after the online upgrade operation.

6.1.4 Life-cycle management of SIMBA without Rhino

SIMBA without Rhino is started and stopped through its UI (as shown in Figure 34 and Figure 35). For the upgrade operation, the old SIMBA must be stopped and then the new one started. The call in the old version cannot be kept going during the upgrade operation.

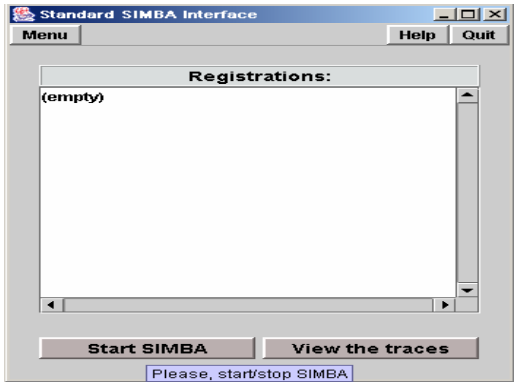

Figure 34 Start SIMBA without a JSLEE

Figure 35 Stop SIMBA without a JSLEE

Figure 34 and 35 are captures of SIMBA without Rhino. They show that SIMBA without Rhino starts and stops through its user interface.

6.1.5 The experimental results of SIMBA life-cycle management

In the life-cycle management experiment, a comparison between SIMBA with a JSLEE (Rhino) and SIMBA without a JSLEE (Rhino) is shown in Table 4.

Life-cycle management capabilities	SIMBA with a JSLEE (Rhino)	SIMBA without JSLEE (Rhino)
Deployment	Yes	No
Uninstallation	Yes	No
Activation	Yes	Yes
Deactivation	Yes	Yes
Online upgrade	Yes	No

Table 4 Comparison between SIMBA with a JSLEE and without a JSLEE

SIMBA with a JSLEE supports all five features of life-cycle management capability. However, SIMBA without a JSLEE only supports two features. A JSLEE provides a SIMBA execution and management environment. Therefore, SIMBA can be deployed, uninstalled, activated and deactivated in the JSLEE. Compared with SIMBA with a JSLEE, SIMBA without a JSLEE is not supported by a service execution and management environment, therefore it only supports activation and deactivation features.

During an online upgrade, a JSLEE enables SIMBA to be upgraded during execution and the call in "connected" state keeps going without corruption during the upgrade. However SIMBA without a JSLEE must stop the old version and then start the new version. The call in "connected" state is corrupted because of stopping the old version. Therefore SIMBA without a JSLEE does not support online upgrade.

6.2 SIMBA performance experimental data

6.2.1 RPS experimental results

The RPS experimental data of SIMBA both with and without Rhino is shown in Table 5. For SIMBA with Rhino, when the number of Registration Attempts per Second (RAPS) increases to 120, the probability of successful registrations decreases to 93.5%. This is the first instance where the value is less than 95%, the benchmark for acceptable RPS in carrier grade requirements [43]. The RPS value of SIMBA with Rhino is therefore determined to be 110 where the probability of successful registration is 96.5%. For SIMBA without Rhino, the RPS value is 120 where the probability of successful registration is 95.5%. The RPS experimental data from 10 to 90 is not shown in Table 5 because all the probability values of successful registration are 100%. The change of successful RPS with the increasing RAPS is shown in Figure 36, including both SIMBA with and without Rhino.

Registration Attempts per Second (RAPS)		100	110	120	130	140	150
the mean value of five tests (successful registrations)	SIMBA with Rhino	98.6	106.2	112.2	118.2	125.2	128.4
	SIMBA without Rhino	99	107	114.6	120.8	127.6	132.8
the probability of successful registrations	SIMBA with Rhino	98.6%	96.5%	93.5%	90.9%	89.4%	85.6%
	SIMBA without Rhino	99%	97.3%	95.5%	92.9%	91.1%	88.5%

Table 5 RPS experimental data

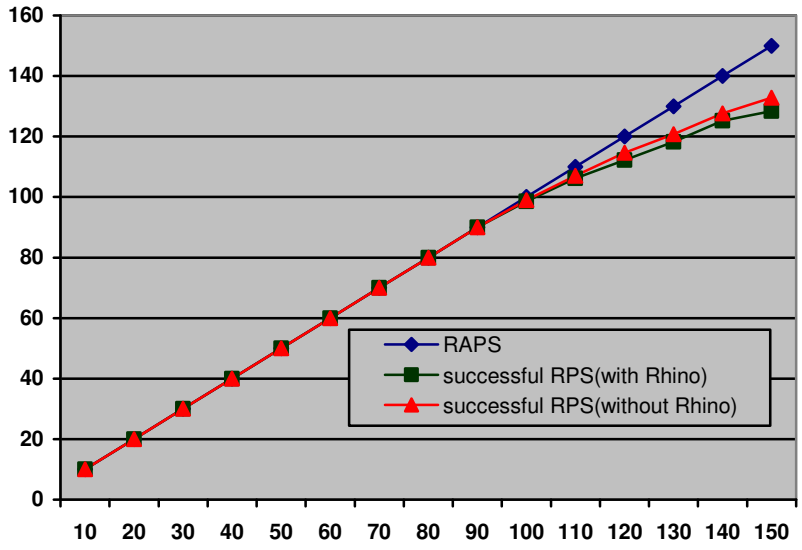

Figure 36 Successful RPS comparison between with and without Rhino

The square mark shows the change of successful RPS with the increasing of RAPS in SIMBA with Rhino. The triangle mark shows the change of successful RPS with the increasing of RAPS in SIMBA without Rhino. The RPS change of SIMBA with Rhino is close to that of SIMBA without Rhino.

6.2.2 CPS experimental results

The CPS experimental data of SIMBA both with and without Rhino is shown in Table 6. For SIMBA with Rhino, when the number of Call Attempts per Second (CAPS) increases to 120, the probability of successful calls decreases below 95% to 93%. 95% that is the benchmark for acceptable CPS in carrier grade requirements [43]. The CPS value of SIMBA with Rhino is therefore determined to be 110 where the probability of successful calls is 98.9%. For SIMBA without Rhino, the CPS value is 120 where the probability of successful calls is 96.2%. The change of successful CPS with the increasing of CAPS is shown in Figure 37, including SIMBA both with and without Rhino.

Call attempts per second (CAPS)		80	90	100	110	120	130	140
the mean value of five tests(successful calls)	SIMBA with Rhino	79.2	88.6	97.2	108.8	111.6	113	115.6
	SIMBA without Rhino	79.6	89.2	98.8	109	115.4	116.4	117
the probability of successful calls	SIMBA with Rhino	99%	98.4%	97.2%	98.9%	93%	86.9%	82.5%
	SIMBA without Rhino	99.5%	99%	98.8%	99.1%	96.2%	89.5%	84%

Table 6 CPS experimental data

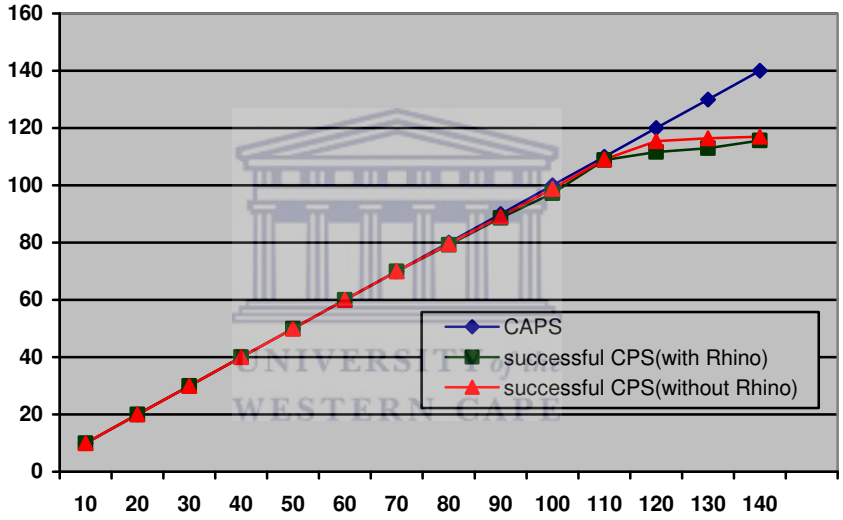

Figure 37 Successful CPS comparison between with and without Rhino

The square mark shows the change of successful CPS with the increasing of CAPS in SIMBA with Rhino. The triangle mark shows the change of successful CPS with the increasing of CAPS in SIMBA without Rhino. The CPS change of SIMBA with Rhino is close to that of SIMBA without Rhino.

6.2.3 T-test calculation

Based on the data samples of RPS and CPS in Appendix I and the t-value-calculation formulas in section 5.2.2.4, the significance level *t* of the variation of the two systems (SIMBA with and without Rhino) for RPS and CPS can be computed. To calculate the significance level for RPS, RAPS is selected from 100 to 150 because all the RPS

values of the two systems are the same as the RAPS value, when RAPS ranges from 10 to 90. Moreover, since the RPS value is determined when RAPS reaches 120, three more increases of RAPS are tested to ensure the RPS value was properly identified. As can be seen, it is no use to continue the experiment after RAPS is 150. The same explanation holds can be explained for the calculation of CPS by confining CAPS between 80 and 140.

Registration Attempts per Second (RAPS)		80	90	100	110	120	130	140	150
Significance level "t" of SIMBA with and without Rhino	RPS			0.2857	0.2676	0.5926	0.4371	0.3935	0.451
	CPS	0.8944	0.7746	0.8835	0.343	0.9435	0.8675	0.2406	

Table 7 *t-test* results of RPS and CPS

Table 7 shows that all the effective values of significance level *t* are in the range of -2.306 and 2.306. According to the *t-test* statistical theory in section 5.2.2.4 and the analysis in the proceeding paragraph, this means that there is no significant variance between SIMBA with Rhino and SIMBA without Rhino in terms of RPS and CPS when RAPS is selected discretely from 10 to 150 and CAPS from 10 to 140.

6.2.4 Performance experimental results analysis

RPS and CPS experimental results show that SIMBA without a JSLEE performs a bit better than SIMBA with a JSLEE because SIMBA with JSLEE has to do extra operations on calls. For instance, the CPS of SIMBA with a JSLEE is 110 and CPS of SIMBA without a JSLEE is 120. However, as shown in *t-test* experimental results, even if the SIMBA with JSLEE has some negative effect on performance, the performance variation of the two systems is low and acceptable. That is, the SIMBA life-cycle management capability, which is provided by a JSLEE (Rhino), does not have a significant negative influence on the performance of call processing.

6.3 Summary

In this chapter, SIMBA life-cycle management experiment date is collected. It shows that a JSLEE (Rhino) can provide life-cycle management for SIMBA, including deploying, uninstalling, activating, deactivating and online upgrading SIMBA. The performance experimental results, which include RPS and CPS, show that SIMBA life-cycle management will not cause a significant decrease on the performance of call processing when it is provided by a JSLEE. In the same way, the applications of life-cycle management to other IP communication applications can be carried out in a JSLEE as well.

Chapter 7 CONCLUSION AND FUTURE WORK

The evaluation suggests that a JSLEE is able to provide life-cycle management for SIMBA. Then the results indicate that approach of using Application Server (JSLEE) integration should be sufficiently general to provide life cycle management, and indeed other carrier grade capabilities, for other IP communication applications.. Finally, the future work of this research considers other carrier grade characteristics, such as fault tolerance and overload balance.

7.1 Conclusion

The main aim of this thesis is to provide carrier grade characteristics for IP communication applications with regard to service life-cycle management. This thesis focuses on five features of the life-cycle management: deployment, activation, deactivation, uninstallation and online upgrade of IP communication applications. This is done by using a SoftBridge application as an example of IP communication applications.

Then the work related to providing life-cycle management for IP communication applications was reviewed. We studied several projects, in which the life-cycle management was considered for IP-Telecommunication applications but only limited to their own services. Thus, it cannot be applied to general IP communication applications. Moreover, an Application Server in an NGN is able to provide the creation, execution and management environments for service life-cycle management. As an open service API for an NGN, JAIN provides a JSLEE that implements the function of an Application Server, for carrying out the life-cycle management of IP-Telecommunication applications. Parlay, on the other hand, does not explicitly address services life-cycle management. The work related to the SoftBridge was also reviewed. However, most of the related work was not explicitly concerned with service life-cycle management.

Therefore an Application Server is proposed to provide life-cycle management for IP communication applications in this thesis. Using a JSLEE as a representation of an Application Server and a SoftBridge application as an example of IP communication applications, this thesis shows how an IP communication application can be deployed, activated, deactivated, uninstalled and upgraded online in a JSLEE.

Two methodologies are applied in this thesis: exploratory prototyping in the development of a SoftBridge application and comparison in an empirical evaluation of the SoftBridge application with regard to life-cycle management.

A SoftBridge application, called SIMBA, has been built upon an open JAIN-SIP Proxy according to the service development mode of the JSLEE specification. Although the Deaf Telephony service is similar to a previous Deaf Telephony SoftBridge [26], SIMBA's SoftBridge design and implementation are unique to this thesis. The JAIN-SIP Proxy system acts as a SIP proxy, registrar and presence server and SIMBA implements bridging between text and voice on the JAIN-SIP Proxy. SIMBA enables a text-based IM client to communicate seamlessly with an IP phone, a telephone or a cellular phone. SIMBA can be packaged into deployable Jars and deployed into Rhino, which acts as a JSLEE and provides life-cycle management for SIMBA.

An empirical evaluation of SIMBA is carried out. SIMBA is used as an example of a SoftBridge application and Rhino is used as the implementation of a JSLEE in the experiment. In comparison, SIMBA with and without Rhino are evaluated in a service life-cycle management experiment and a performance experiment. The service life-cycle management experimental results show that SIMBA can be deployed, uninstalled, activated, deactivated and upgraded online in Rhino through the Rhino web management interface. The performance results show that SIMBA life-cycle management capability, which is provided by Rhino, does not have a significant negative influence on the performance of call processing.

In conclusion, a JSLEE, one representation of an Application Server, is able to provide life-cycle management for an IP-based SoftBridge application, including the

deployment, activation, deactivation, uninstallation and online upgrading of the SoftBridge application.

It can be implied as well that if one of carrier grade characteristics, such as life-cycle management, can be applied to a SoftBridge application through a JSLEE, so could the other characteristics. This is reasonable because a JSLEE acts as an Application Server that supports most carrier grade characteristics, such as service life-cycle management, fault tolerance, overload balance and control, trace, alarm and so on.

Furthermore, since the carrier grade characteristics can be successfully applied to one IP communication application, such as an IP-based SoftBridge application-SIMBA, they can be applied to general IP communication applications. This is because a JSLEE is a representation of open Application Server, and it is applicable to all IP communication applications rather than some specified applications. Thus, what can be implied is that carrier grade characteristics can be provided for all IP communication applications via an Application Server.

Notably, as described above, an Application Server in an NGN is used as the key approach to implement the introduction of carrier grade characteristics to IP communication applications.

7.2 Future work

The current SIMBA only supports a basic bridging service between a Deaf user with an IM client and a hearing user with a telephone. Future work should address more service features. This thesis focuses on service life-cycle management capabilities, therefore, other carrier grade characteristics, such as fault tolerance, overload balance and call process capability, should be addressed in future work.

7.2.1 Services by SIMBA

SIMBA should cooperate with more SIP/SIMPLE products and provide more bridging service features, such as call forwarding and multi-modal conferencing.

Cooperation test with more SIP/SIMPLE products

SIMBA was successfully tested with the NIST-SIP IM client and the JAIN-SIP proxy. It should support more SIP/SIMPLE products, such as Microsoft Windows Messenger, Siemens UA, and Cisco UA. However some SIMPLE-based IM clients support session-mode messaging, therefore SIMBA should add the feature of session-mode messaging and support more IM clients.

Call forwarding feature

SIMBA can bridge between a telephone and an IM client, allowing seamless communication. If the telephone is busy, SIMBA sends a "busy" prompt message to the IM client. The feature of call forwarding allows SIMBA to forward the call to other terminals of the called user, such as cellular phone or SIP phone. If all of them are not available, the system can send an e-mail or SMS to the called user.

Conferencing feature

The conference feature allows more than two users to join in a conference using various end user equipments, such as an IM client, SIP client, PDA, telephone and cellular phone. SIMBA does not currently support this advanced feature.

7.2.2 The issue of carrier grade characteristics for SIMBA

Besides service life-cycle management, more carrier grade characteristics, such as fault tolerance, overload balance and control, should be put into SIMBA.

Fault tolerance

The current system runs as a single node. If it is down abnormally, the call in-process will be lost. Fault tolerance allows the system to run with multiple nodes in the cluster. It can detect node failures and automatically forward the request to other nodes. Therefore a call in-process can be kept although some nodes fail. Future work should consider enabling SIMBA to support fault tolerance in a JSLEE.

Overload balance and control

An overload balance feature would enable SIMBA to allocate incoming calls to different nodes according to its allocation strategies. New nodes can be dynamically

added to a running cluster. An overload control feature would allow SIMBA to reject incoming calls in the case of an overload, to process in-progress calls within acceptable operating parameters and avoid a potential failure due to an overload. Future work should consider enabling SIMBA to support overload balance and control in a JSLEE.

7.3 Summary

In this chapter, the entire thesis is first reviewed to reach a conclusion that the carrier grade characteristics can be provided for IP communication applications by using a JSLEE as an Application Server. Then the future work is described with regard to two main aspects: more services provided by SIMBA, such as call forwarding and multi-modal conferencing, and more carrier grade characteristics, such as fault tolerance, overload control and balance.

BIBLIOGRAPHY

[1]. S. Abu-Hakima, R. Liscano and R. Impey. "A common multi-agent tested for diverse seamless personal information networking applications", *IEEE Communication Magazine,* Vol. 36, No. 7, 1998, pp: 68–74.

[2]. Apache Software Foundation. "Apache Ant 1.6.2 Manual", available at http://ant.apache.org/manual/index.html, 2004.

[3]. S. Arbanowski and S. V. D. Meer. "Service personalization for unified messaging systems", *IEEE International Symposium on Computers and Communication,* 1999, pp. 156–163.

[4]. S. Arbanowski, S. Steglich, I. Radusch and R. Popescu-Zeletin. "Super distributed objects: an execution environment for I-centric services", *Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems,* 2003, pp. 201–208.

[5]. M. A. Bauer, N. Coburn, D. L. Erickson, J. P. Finnigan, J. W. Hong, P-Å. Larson and J. Slonim. "An integrated architecture for distributed applications", *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering*. Vol. 1, 1993, pp.8-26.

[6]. S. Beddus, C. Bruce and S. Davis. "Opening up networks with JAIN Parlay", *IEEE Communication Magazine,* Vol. 38, No. 4, 2000, pp: 136–143.

[7]. S. Bessler, A-V. Nisanyan, K. Peterbauer, R. Pailer, J. Stadler. **"**A Service Platform for Internet-Telecom Services using SIP**",** *Proceedings of the IFIP TC6 WG6.7 Sixth International Conference on Intelligence in Networks: Telecommunication Network Intelligence,* 2000, pp. 59–72.

[8]. R. R. Bhat and R. Gupta. "JAIN protocol APIs", *IEEE Communication Magazine,* Vol. 38, Issue. 1, 2000, pp. 100–107.

[9]. E. H. Blake and W. D. Tucker. "Bridging Communication across the Digital Divide", *CTIT Proceedings of the 3rd Workshop on Social Intelligence Design,* SID 2004, Enschede, Netherlands, pp. 29–38.

[10]. I. Boyd and M. Carr. "Parlay API: Business Benefits and Technical Overview", available at http://more.btexact.com/projects/ prognet/parlaydocs/TCS0.1a4colour.pdf, 2001.

[11]. B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema and D. Gurle. "Session Initiation Protocol (SIP) Extension for Instant Messaging", *Internet Engineering Task Force (IETF), Request for Comments (RFC) 3428*, December 2002.

[12]. M. Chetty, W. D. Tucker and E. Blake. "Developing Locally Relevant Applications for Rural Areas: A South African Example", *Proceedings of SAICSIT 2004*, Cape Town, South Africa, pp. 239–234.

[13]. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. "Hypertext Transfer Protocol", *Internet Engineering Task Force (IETF), Request for Comments (RFC) 2616*, June 1999.

[14]. R. Gayraud and O. Jacques. "SIPp Reference Document v1.0", available at http://sipp.sourceforge.net/doc/reference.html, 2004.

[15]. C. Gbaguidi, J-P Hubaux, G. Pacifici and A. N. Tantawi. "Integration of Internet and telecommunication: An Architecture for Hybrid Services", *IEEE JSAC,* Vol. 17, No. 9, 1999.

[16]. M. Glaser, W. D. Tucker and D. Mashao. "Preparation of Deaf end-users and the SoftBridge for semi-automated relay trials", *Proceedings of the 7th South African Telecommunication Networks & Applications Conference*, SATNAC 2004, Stellenbosch, South Africa.

[17]. M. Glaser and W. D. Tucker. "Telecommunication bridging between Deaf and hearing users in South Africa", *Conference and Workshop on Assistive Technologies for Vision and Hearing Impairment,* CVHI 2004, Granada, Spain.

[18]. R. H. Glitho and A. Poulin. "A High Level Service Creation Environment for Parlay in a SIP Environment", *IEEE International Conference on Communication,* ICC 2002, Vol. 4, pp. 2008–2013.

[19]. M. Handley and V. Jacobson. "SDP: Session Description Protocol". *Internet Engineering Task Force (IETF), Request for Comments (RFC) 2327*, April 1998.

[20]. S, Ian. *Software engineering*, 6th edition, Addison Wesley Professional, 2000.

[21]. IBM Inc. "IBM WebSphere Application Server Overview", available at ftp://ftp.software.ibm.com/software/webserver/appserv/v5/g325-2047-003.pdf, 2004.

[22]. IBM Inc. "The History of Notes and Domino", available at http://www-10.lotus.com//ldd/today.nsf/0db661345413ad1d852567ba006bb090/bc684f3a96b4efd185256b9c0064ae9c/$FILE/ND-history.pdf, 2003.

[23]. Jabber Software Foundation. “Jabber User Guide", available at http://www.jabber.org/user/userguide/index.html, 2003.

[24]. J. Keijzer, D. Tait and R. Goedman. “JAIN: a new approach to services in communication networks”, *IEEE Communication Magazine,* Vol. 38, No. 1, 2000, pp. 94–99.

[25]. K-H Lee; K-O Lee; K-C Park; J-O Lee; Y-H Bang. “Architecture to be deployed on strategies of next-generation networks”, *IEEE International Conference on Communication,* Vol. 2, 2003, Anchorage, USA, pp. 819–822.

[26]. J. Lewis, W. Tucker, E. Blake. “SoftBridge: A Multimodal Instant Messaging Bridging System”, *Proceedings of the 6th South African Telecommunication Networks & Applications Conference*, SATNAC 2003, George, South Africa.

[27]. S. B. Lim and D Ferry. “JAIN SLEE 1.0 Specification, final release“, available http://compose.labri.fr/documentation/sip/Documentation/Papers/Programming_SIP/Paper_Publication_and_Draft/Java_SIP/jain_slee-1_0-fr-spec.pdf, 2004.

[28]. S. B. Lim, D. Ferry, P. O’Doherty and D. Page. “JAIN SLEE tutorial”, available at http://java.sun.com/products/jain/JAIN-SLEE-Tutorial.pdf, 2002.

[29]. R. Liscano, R. Impey, P. Gordon and S. Abu-Hakima. “A system for the seamless integration of personal messages using agents developed on a lotus notes platform”, *Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research*, Toronto, Canada, pp. 24–32.

[30]. M. Lu and S. Cheng. “SoftSwitch Technology”, *ZhongXing Telecom Technology*, Vol. 41, 2002, pp. 29–31.

[31]. S. V. D. Meer, S. Arbanowski and T. Magedanz. “An approach for a 4th generation messaging system”, *The Fourth International Symposium on Autonomous Decentralized Systems Integration of Heterogeneous Systems,* 1999, pp. 158–167.

[32]. S. V. D. Meer and S. Arbanowski. “From unified messaging towards I-centric services for the virtual home environment”, *IEEE Intelligent Network Workshop,* 2001, pp. 152–160.

[33]. Ministry of Information Industry P.R China, ZHONGXING Inc, BISC Inc, HUAWEI Inc. “Technical Requirements for Application Server Based on Softswitch”, *Communication Industry Standard of P.R.China*, 2003.

[34]. A-J Moerdijk and L. Klostermann. “Opening the networks with Parlay/OSA: standards and aspects behind the APIs”, *IEEE Network,* Vol. 17, No. 3, 2003, pp. 58–64.

[35]. A. Niemi, Ed. "Session Initiation Protocol (SIP) Extension for Presence Publication", *Internet Engineering Task Force (IETF), draft 01*, June 2003.

[36]. NIST (National Institute of standards And Technology). "A JAIN-SIP Instant Messaging Client for the People!", available at http://snad.ncsl.nist.gov/proj/iptel/src/nist-sip/jain-sip-presence proxy/docs/gov/nist/sip/instantmessaging/readme.html, 2003.

[37]. NIST (National Institute of standards And Technology). "A JAIN-SIP Proxy for the People!", available at http://snad.ncsl.nist.gov/proj/iptel/src/nist-sip/jain-sip-presence proxy/docs/gov/nist/sip/proxy/readme.html, 2003.

[38]. Open Cloud Inc. "A SLEE for all Seasons", available at http://www.opencloud.com/slee/downloads/asfas.pdf, 2003.

[39]. Open Cloud Inc. "Rhino Software Development Kit User Guide", available at http://www.opencloud.com/scripts/download.py?file=RhinoSDK-1.3.0-beta5-userguide.pdf.gz, 2004.

[40]. J. Rosenberg. "A Presence Event Package for the Session Initiation Protocol (SIP)", *Internet Engineering Task Force (IETF), draft 10*, July 2003.

[41]. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston , J. Peterson, R. Sparks, M. Handley and E. Schooler. "SIP: Session Initiation Protocol", *Internet Engineering Task Force (IETF), Request for Comments (RFC) 3261*, June 2002.

[42]. H. Schulzrinne, GMD Fokus, S. Casner, R. Frederick and V. Jacobson. "RTP: A Transport Protocol for Real-Time Applications", *Internet Engineering Task Force (IETF), Request for Comments (RFC) 3550*, July 2003.

[43]. H. Schulzrinne, N. Narayanan, J. Lennox and M. Doyle. "SIPstone - Benchmarking SIP Server Performance", available at http://www.sipstone.org/files/sipstone_0402.pdf, 2002.

[44]. J. Siegel. "OMG overview: CORBA and the OMA in enterprise computing", *ACM Communication*, Vol. 41, Issue. 10, 1998, pp: 37–43.

[45]. S. Steglich and R. Popescu-Zeletin. "Towards I-centric user interaction", *IEEE International Conference on Multimedia and Expo*, 2001, pp. 22–25.

[46]. Sun Microsystems. "JAIN Overview", available at http://developers.sun.com/dev/evangcentral/presentations/JAIN_overview.pdf, 2003.

[47]. Sun Microsystems. “The JAIN APIs: Integrated Network APIs for the Java Platform”, available at http:// www.sipcenter.com/sip.nsf/ html/WEBB5YN5GE/$FILE/WP2002.pdf, 2002.

[48]. Sun Microsystems. “Java Media Framework API (JMF) 1.0 Programmers Guide”, available at http://java.sun.com/products/java-media/jmf/1.0/guide/index.html, 2004.

[49]. Sun Microsystems. “Java 2 Platform, Standard Edition (J2SE) Version 1.4.2 API Specification”, available at http://java.sun.com/j2se/1.4.2/docs/api/index.html, 2003.

[50]. Tata Infotech Ltd. “Usage Manual: SIP Residential Gateway”, available at http://www.vovida.org/applicaions/downloads/siprg/usage_manual.html, 2002.

[51]. The PostgreSQL Global Development Group. “PostgreSQL 7.3.4 Reference Manual”, available at http://www.sql.org/sql-database/postgresql/manual/reference.html, 2003.

[52]. M. F. Triola. *Elementary Statistics*, 6th Edition, Addison-Wesley, 1994.

[53]. A. H. Thomas, S. Dalton, S. Brown, B. Holm, T. Loton, M. Kunnumpurath, S. Allamaraju, J. Bell and S. Li. *Professional Java Servlets 2.3*, Wrox Press, 2002.

[54]. W. D. Tucker, E. H. Blake and G. Marsden. “Open User Interconnect and Quality of Communication”, *Proceedings of the 7th South African Telecommunication Networks & Applications Conference*, SATNAC 2004, Stellenbosch, South Africa.

[55]. W. Walker, P. Lamere and P. Kwok. “FreeTTS 1.2beta2 API Documentation”, available at http://freetts.sourceforge.net/javadoc/index.html, 2004.

[56]. N. H. Weiderman, A. N. Habermann, M. W. Borger and M. H. Klein. “A methodology for evaluating environments ”, *Proceedings of the second ACM SIGSOFT/SIGPLAN software engineering symposium on Practical software development environments*, Vol. 22, 1987, pp: 199–207.

[57]. W. Wu and F. C. Yang. “Service Support Environment in NGN”, *Telecommunication Technology*, Vol. 1, 2002, pp. 18–21.

Appendix I Sample data of RPS and CPS

RAPS (Registration attempts per second)		10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
SIMBA with Rhino (successful registrations)	1	10	20	30	40	50	60	70	80	90	98	110	111	106	135	130
	2	10	20	30	40	50	60	70	80	90	100	103	120	130	127	120
	3	10	20	30	40	50	60	70	80	90	100	109	110	120	126	138
	4	10	20	30	40	50	60	70	80	90	95	99	108	110	120	110
	5	10	20	30	40	50	60	70	80	90	100	110	112	125	118	144
SIMBA without Rhino (successful registrations)	1	10	20	30	40	50	60	70	80	90	100	110	120	120	140	140
	2	10	20	30	40	50	60	70	80	90	100	100	120	130	140	150
	3	10	20	30	40	50	60	70	80	90	95	105	110	129	125	120
	4	10	20	30	40	50	60	70	80	90	100	110	103	115	115	110
	5	10	20	30	40	50	60	70	80	90	100	110	120	110	118	144

Table 8 Sample data of RPS

CAPS (call attempts per second)		10	20	30	40	50	60	70	80	90	100	110	120	130	140
SIMBA with Rhino (successful calls)	1	10	20	30	40	50	60	70	80	88	100	109	117	115	118
	2	10	20	30	40	50	60	70	78	87	99	109	117	112	105
	3	10	20	30	40	50	60	70	79	90	94	108	115	116	117
	4	10	20	30	40	50	60	70	79	88	93	108	105	113	114
	5	10	20	30	40	50	60	70	80	90	100	110	104	109	124
SIMBA without Rhino (successful calls)	1	10	20	30	40	50	60	70	80	90	100	109	117	130	118
	2	10	20	30	40	50	60	70	80	88	99	110	120	112	133
	3	10	20	30	40	50	60	70	79	90	95	108	115	117	110
	4	10	20	30	40	50	60	70	79	88	100	108	105	108	104
	5	10	20	30	40	50	60	70	80	90	100	110	120	115	120

Table 9 Sample data of CPS

Appendix II Technical Specification

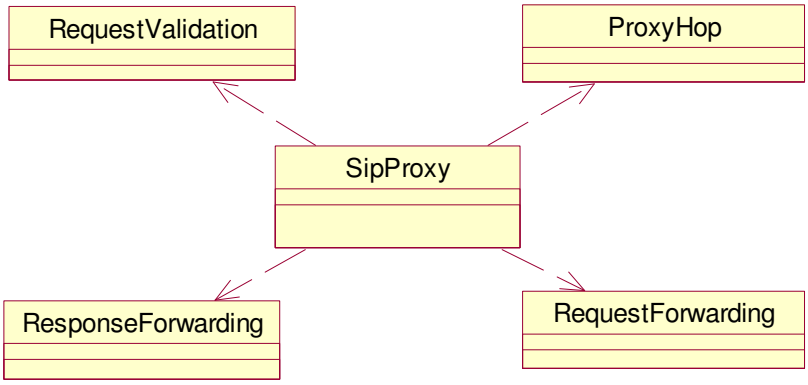

Figure 38 Class diagram of the Proxy class package

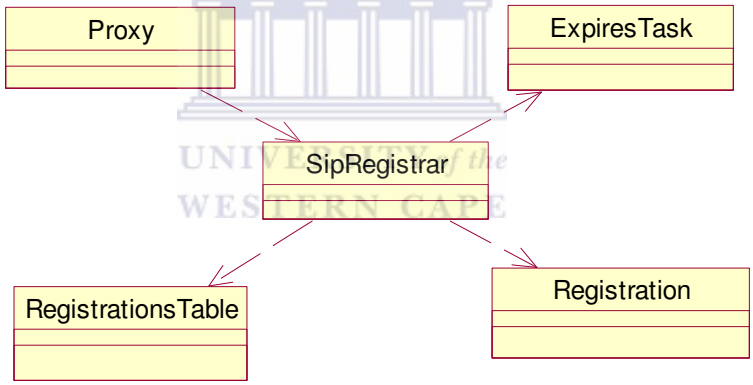

Figure 39 Class diagram of the Registrar class package

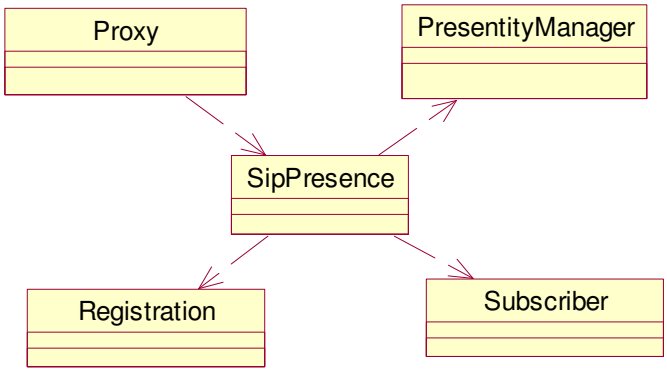

Figure 40 Class diagram of the Presence server class package

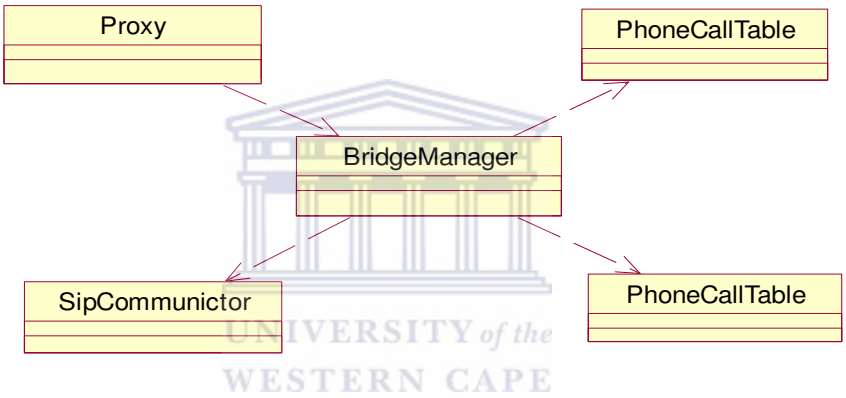

Figure 41 Class diagram of the Bridging class package

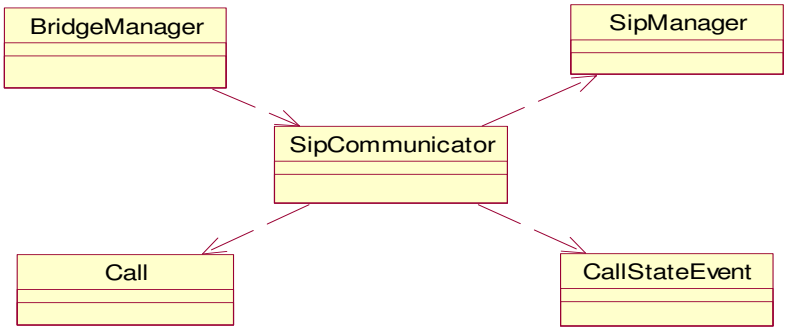

Figure 42 Class diagram of the Communicator class package

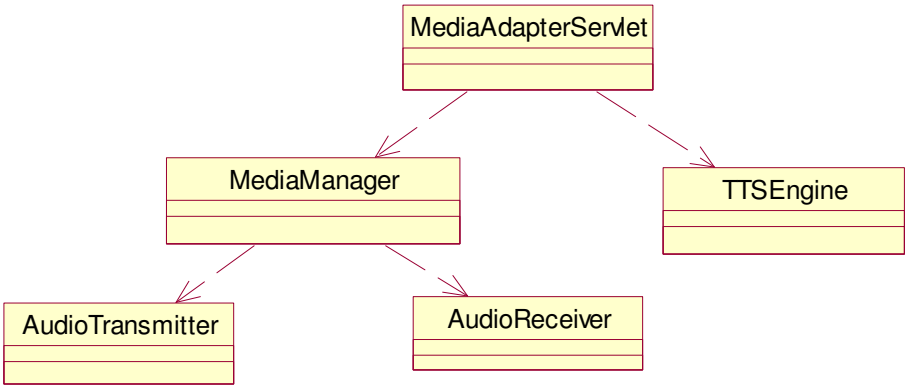

Figure 43 Class diagram in Media Adapter Server class package