Analysis and Design of IMS Services on IMS AS for Teaching Experiments

Kun Li, Baozhong Cheng, Xiaoyan Zhang, Youzheng Chen, Yi Sun and Jin Xie
School of Software Engineering
Beijing University of Posts and Telecommunications
Beijing, China
kunsland1990@gmail.com
bzcheng@bupt.edu.cn

Abstract—**IP Multimedia Subsystem (IMS) has been deployed worldwide for several years, but some people, especially beginners, are still confused about what we can do with IMS. This paper presents the design and implementation of two lab experiments to help students to learn about the IMS. We focus on the development of two IMS services on an IMS Application Server: Chat Room and Presence Service. The architecture and the detailed design of the services are presented. These two experiments have been applied to the practical training courses for undergraduate students.**

Keywords*—*IMS; SIP; SIP Servlet; Application Server; Teaching Case; Chat Room; Presence Service

I. Introduction

IP Multimedia Subsystem (IMS) is getting popular in recent years for its potential to converge the Internet, TV cable network, and telephone network. IMS networks are being deployed worldwide, there is high demand for people who understand the concepts and master the software development techniques of IMS. We began to develop an IMS software development training program. Based on the open source projects, Open IMS and Mobicents, several IMS experiments have been developed. The experiments have been used in practical training course, and are welcomed by students.

For beginners, it is usually better to learn by examples. So we introduce the following steps to make it easier to start with. First, we explain the environment setup of the experiments. Secondly, we analyzed the requirements according to a simplified version of IMS specification, and created a UseCase Model. Finally, we explain the design of the experiments using Unified Modeling Language (UML) diagrams.

II. The Environment Setup of Our Experiments

In this paper, we focus on the application layer development and try to explain to beginners how to develop a service on an IMS application server. One of the services designed uses Session Initiation Protocol (SIP) Instant Message (IM) to implement Chatroom Service, and another service was designed to implement the Presence Service. There are four types of application servers (AS) in IMS, namely SIP AS, Parlay/OSA AS, IM-SSF, and Web portals. SIP is defined by IETF. And it is the basic signaling protocol in IMS. [1] Both of our teaching experiments are designed to work on SIP AS. SIP AS provides an open plug-and-play platform for service development. The experiments help beginners to understand the basic architecture of IMS.

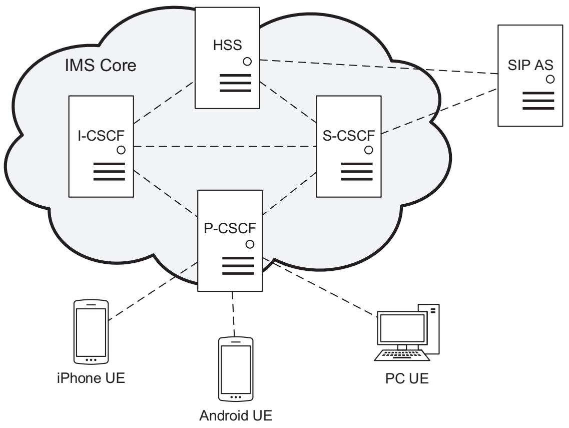


Fig. 1. The Architecture of the IMS Experimental Network

We use Open IMS Core [2] as a test bed and the enhanced JBoss as the container of AS. The Open IMS Core is an open source implementation of IMS core network, which consists of Proxy-Call Session Control Function (P-CSCF), Interrogating CSCF (I-CSCF), Serving CSCF (S-CSCF), and a lightweight Home Subscriber Server (HSS). As JBoss AS itself does not support SIP Servlet technique, we use a JBoss-based IMS application server, Mobicents SIP Servlet server (MSS), developed by an open source organization called Mobicents. In terms of development kit, we choose Eclipse in Java EE version. The IMS UE used in the experiments includes PC-based IMS UE, Android-based IMS UE, and iPhone-based IMS UE. The IMS core network is connected to the IMS UE with LAN or WLAN, as shown in the Figure 1. [3]

The IMS core is actually composed of four parts, P-CSCF server, I-CSCF server, S-CSCF server and HSS server. [4] Each part of IMS core can be deployed on an independent server and connected with each other. Also, four parts of IMS core can work together and act as one to the user or application server. The SIP application server is run on JBoss AS. [5] The Figure 2 shows the general architecture of the AS application. [6]

To keep the experiments simple, only one IMS domain is used in the design of the two experiments.

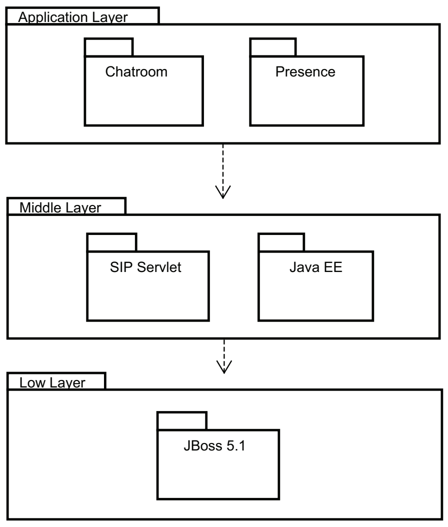
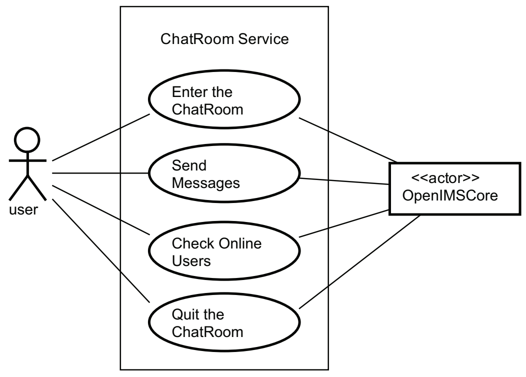
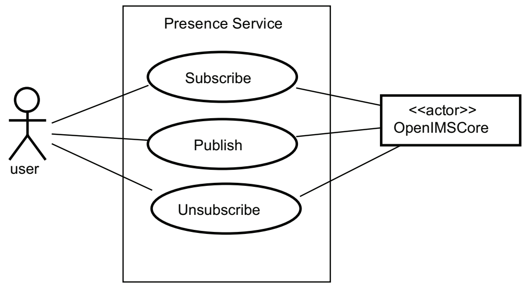


Fig. 2. Architecture of the AS Application



(a) UseCase Diagram of ChatRoom Service



(b) Use Case Diagram of Presence Service

Fig. 3. The UseCase Diagram of ChatRoom Service and Presence Service

III. The Requirement of Our Experiments

In this sections, we will discuss the requirement of our experiments, *ChatRoom Service* and *Presence Service*.

The Figure 3(a) shows the UseCase Diagram of *ChatRoom Service*. In this case, users join a ChatRoom and send messages to the ChatRoom Server. The server forwards messages to the other users who are also in the ChatRoom. Furthermore, users can check the online user list or quit the ChatRoom.

The Figure 3(b) shows the UseCase Diagram of *Presence Service*. In this case, a user first subscribes the status of other users and get confirmation from the *Presence Service*. Once

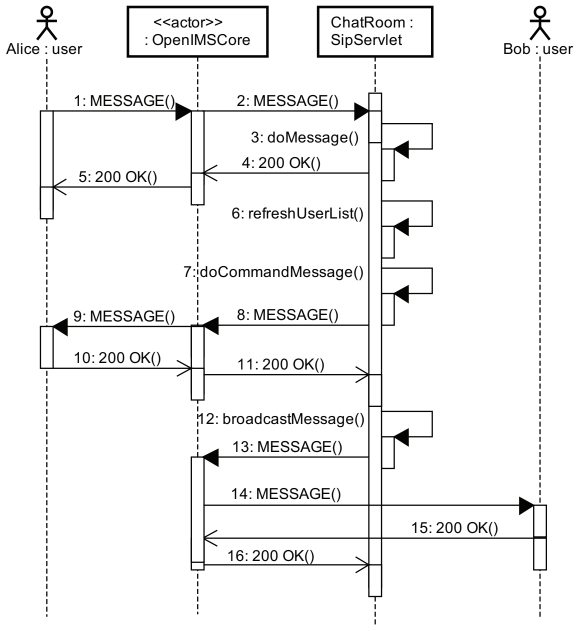


Fig. 4. Sequence Diagram of ChatRoom Service

the subscription relationship is established, the status published by one user will be received by all its subscribers. In addition, a user can unsubscribe the *Presence Service* if needed.

IV. Design and Implementation

When it comes to the design of software model, the class *ChatRoom* plays a role of the logic of *ChatRoom Service* and the class *PresenceServer* of the *Presence Service*. Both are extended from the class *SipServlet*. The class *Session* is designed for maintaining SIP Session status and the class *Presentity* is for maintaining status which users published.

When users connect to the IMS with UE like iPhone, Android or PC, the *Presence Service* will automatically trigger if he/she enabled the *Presence Service*. Then the class PrenseceServer running on the AS will process the SIP request of PUBLISH and SUBSCRIBE with function *doPublish* and function *doSubscribe* respectively.

When a user entered the ChatRoom by calling the ChatRoom SIP address, the class *ChatRoom* on the AS will bind the SIP address both of ChatRoom and the user. Then the *ChatRoom* will forward the SIP MESSAGE which is sent by users with method *broadcastMessage* to the members in the exact ChatRoom. In addition, the *ChatRoom* will do some ”command-like” SIP MESSAGEs with method *doCommandMessage*, using ”/who” for checking the online users and ”/quit” for quitting the ChatRoom.

The *ChatRoom* is a service developed for a group of users to exchange text messages. The Sequence Diagram of this service is shown in Figure 4.

In terms of *ChatRoom Service*, a user can join the ChatRoom, and send message to the server. Then the server will invoke method *doMessage* (shown in Figure 5), which is the Application Programming Interface (API) provided by MSS for responding the user's request.

```
// handle the SIP MESSAGE sent by user agent
protected void doMessage(SipServletRequest req){
    // send 200 OK message back to the user agent
    req.createResponse(SipServletResponse.SC_OK).send;
    // get the sip address of user agent
    String from = req.getFrom().getURI().toString();
    //get the type of message content
    String contentType = req.GetHeader("Content-Type");
    // get the message content
    Object message = req.getContent();
    ...
}
```

Fig. 5. Important Code of Handling SIP MESSAGE Request

```
// Broadcast messages
...
SipFactory factory = (SipFactory)getServletContext()
    .getAttribute("javax.servlet.sip.SipFactory");
SipApplicationSession session = factory.createApplicationSession();
// set the type of MESSAGE and the sip addresses of the server & the user
SipServletRequest request =
    factory.createRquest(session, "MESSAGE", serverAddress, userSIPAddress);
// set message content and type
request.setContent(message, contentType);
request.send(); // send message
```

Fig. 6. Important Code of Broadcasting Messages

The first action taken by the server is to send 200 OK Response back to the user agent. Then the server will add the corresponding user to the ”user list” if he or she joined the ChatRoom for the first time. Then, based on the content of the message, the server will take different steps. For ”command message” like ”/who”, which act as a request to check the names of all the users in the ChatRoom currently, the server will return a corresponding message to the sender. For other messages, the server will broadcast the message to the rest in the ChatRoom. Some important code programmed here are shown in Figure 6.

The *Presence Service* implemented here is a simplified implementation of the Presence Service defined by 3GPP [4]. For example, for any subscription messages sent to the presence server, a simplification was made in our implementation that a subscription is not authorized by the person being subscribed. [7] All the subscription will be authorized unconditionally by the presence server [8]. By using the presence service, a user can watch the status and communication willingness of people he/she is interested in, and publish his or her own status and communication willingness to other people. [9]

First of all, one should subscribe the presence status of someone by sending a SUBSCRIBE sip message. The Sequence Diagram of SUBSCRIBE service is shown in Figure 7.

To respond the SUBSCRIBE messages, the server will invoke method *doSubscribe* (shown in Figure 8). When someone, we take Alice for example, SUBSCRIBEs others, like Bob, the server will not forward the message to Bob. Instead, the server will keep the session and send 200 OK SIP Response which means Bob has accepted the subscription.

The most important thing in *Presence Service*, which is quite different from *ChatRoom Service*, is the SIP Session, which is created by the watcher like Alice when she sends a SUBSCRIBE message to Bob, identifies a unique watcher-to-presentity relationship. And as Figure 9 depicts, it will be

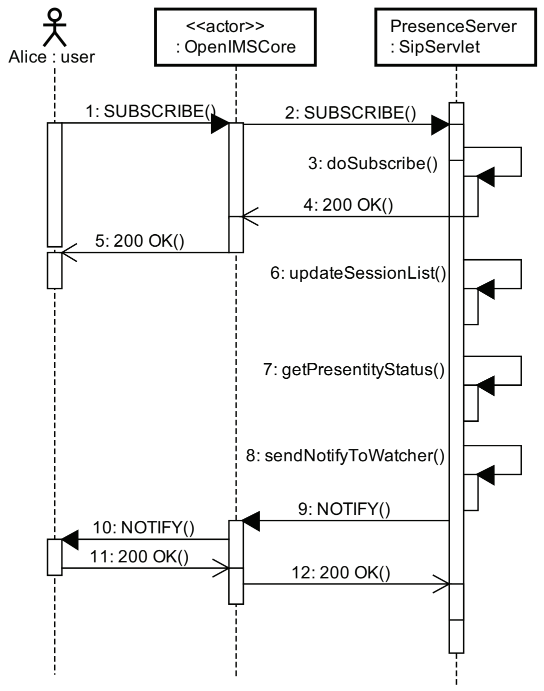


Fig. 7. Sequence Diagram of SUBSCRIBE in Simplified Presence Service

```
protected void doSubscribe(SipServletRequest req){
    // accept the subscription
    SipServletResponce resp =
        req.createResponce(SipServletResponce.SC_OK);
    ...
    resp.send();
    ...
}
```

Fig. 8. Important Code of Subscription Process

```
// save session from the SUBSCRIBE request for doPublish method
SipSession session = req.getSession();
...
// NOTIFY message
// create a NOTIFY sip message
SipServletRequest notify = session.createRequest("NOTIFY");
...
// send NOTIFY message to the the session owner
notify.send();
...
```

Fig. 9. Important Code of Notification Process

used to create a NOTIFY message to Alice when Bob sends a PUBLISH message to change his status. So the server will save the session immediately to the ”session list” when it received a SUBSCRIBE message. The following step of the server is to check Bob’s status, using the Alice-to-Bob session, which is quite different from Bob-to-Alice session, to create a NOTIFY message to Alice.

The Sequence Diagram of PUBLISH service is shown in Figure 10. Once Alice subscribed Bob, the presence server will forward Bob’s each PUBLISH message to Alice in NOTIFY message. To handle this, the server will invoke method *doPublish*. But before it sends the NOTIFY message, the server should finish another two tasks, refreshing Bob’s status (shown in Figure 11) and retrieving Bob’s ”watcher session list”, which means obtaining anyone-to-Bob session from the ”session list”.

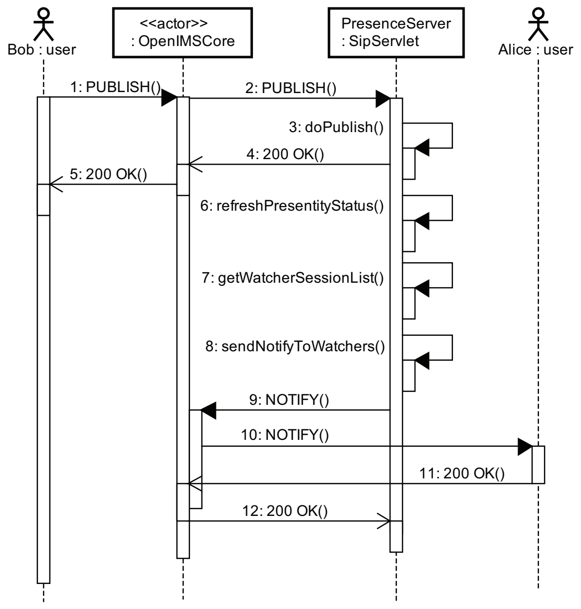


Fig. 10. Sequence Diagram of PUBLISH in Simplified Presence Service

```
protected void doPublish(SipServletRequest req){
    ...
    // retrieve the status of the one who publishes his or her status
    Object content = req.getContent();
    // retrieve watchers' sessions from instance of the class Session
    // hidden for saving space
    ...
    // notify each watcher with these sessions
    ...
    notify.setContent(content, "application/pidf+xml");
    ...
    notify.send();
}
```

Fig. 11. Important Code of Publication Process

With these sessions, the server will respectively notify Bob's watchers.

V. Teaching Cases

We have designed two teaching cases–Chat Room Service and Presence Service–to inspire the study interest of the students in IMS and value-added service in IMS.

In the teaching cases, students are assigned to two different groups, one group focuses on the aspect of IMS AS (AS Group) which is depicted above, the other one focuses on the aspect of IMS Client (Client Group) which is not described in this paper in details.

In the AS Group, students are told to go through the experiments by three steps. The first step is to design a ChatRoom server on the AS with a test client named UCT IMS Client. In this step, students learn about the basic knowledge of IMS. The second step is to design a presence server specified in this paper (in *Design and Implementation* Section) and test with Client Group. In this step, students learn about the knowledge of value-added service in IMS. The last step is to design a standard presence server specified by 3GPP and test it with Client Group.

VI. Conclusion

In this paper, we designed and implemented two teaching experiments for IMS beginners. Both experiments are based on SIP Servlet API, and can be deployed in an IMS AS. Some UML diagrams are used to explain the requirement and design of the two experiments.

The two experiments have been used for undergraduate students in a practical training course. The response from the students and teachers is positive. The experiments help the students to grasp the abstract concepts of IMS, and deepen their understanding of the IMS services.

Acknowledgement

Funded by Beijing Education Commission, it took several months to finish the design, implementation, and testing of these experiments. A lot of people have participated in the development process. We would like to thanks all the teachers and students who made contributions to this project, especially Tailun Liang, and Nian Pan.

References

[1] 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, RFC 3261, Jun. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt

[2] FOKUS. (2013, Nov.) Openimscore.org — the open source ims core project. [Online]. Available: http://www.openimscore.org/

[3] P. Subramanian and B. PG, "Convergence of java ee and sip in ims as," in *IP Multimedia Subsystem Architecture and Applications, 2007 International Conference on*, 2007, pp. 1–5.

[4] 3GPP, "Ip multimedia subsystem (ims); stage 2," 3rd Generation Partnership Project (3GPP), TS 23.228, Nov. 2013. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/23228.htm

[5] A. B. Roach, "Session initiation protocol (sip)-specific event notification," Internet Engineering Task Force, RFC 3265, Jun. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3265.txt

[6] J. Deruelle. (2013, Nov.) sipservlets - leading html5 webrtc compliant sip/ims application server. [Online]. Available: http://code.google.com/p/sipservlets/wiki/Welcome

[7] C. Chi, R. Hao, D. Wang, and Z. Cao, "Ims presence server: Traffic analysis and performance modelling," in *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*, 2008, pp. 63–72.

[8] Y. Yang, J. Luo, J. Peng, and J. Huang, "Research and implementation on presence service of ims," in *Broadband Network and Multimedia Technology (IC-BNMT), 2010 3rd IEEE International Conference on*, 2010, pp. 803–806.

[9] M. El Maarabani, A. Adala, I. Hwang, and A. Cavalli, "Interoperability testing of presence service on ims platform," in *Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops, 2009. TridentCom 2009. 5th International Conference on*, 2009, pp. 1–6.