Implementing Endpoint Services Using the SIP Servlet Standard

Eric Cheung
AT&T Labs — Research
180 Park Avenue, Florham Park, NJ, U.S.A.
cheung@research.att.com

Abstract

Voice over IP applications at endpoint devices are typically implemented using proprietary methods and are closed and not extensible. This hinders innovation and competition. On the other hand, a number of open standards have emerged for programming network services. The SIP Servlet API is the dominant standard and is widely implemented. This paper examines the feasibility of using the SIP Servlet API for programming endpoint applications. An enterprise communication system is used as the case study. This investigation shows that this is a promising approach with advantages such as application reuse and support for application composition. A number of issues are identified that may be addressed by enhancing the current SIP servlet container implementations.

1. Introduction

The Session Initiation Protocol (SIP) [12] was initially developed for end-to-end voice or multimedia session establishment over IP networks. It was assumed that applications would reside at the endpoints, with the network only providing basic call routing functions. Examples of endpoints include software applications that run on computers equipped with headsets, microphones, and video cameras. These are often called *softphones*. Other endpoints include dedicated hardware IP telephones designed for desktop or mobile use, and may have built-in video cameras and small color displays for video communication.

As the Voice-over-IP (VoIP) industry matured, vendors and service providers realized that there was a need to support services in the network and began to define a new class of SIP elements called SIP application servers (AS). For example, the 3GPP IMS architecture is the dominant standard for mobile and wireline systems. In this architecture, multiple application servers may be invoked in the signaling path between two endpoints. As another example, there are a large number of *IP PBX* products in the market that provide sophisticated enterprise PBX functionality in a centralized server, while the soft or hardphones at the users' desks can be kept simple and less full-featured.

In concert with this development, a number of open standards have emerged to define languages and APIs for developing applications targeted for SIP AS. These standards promote portability and encourage third-party application development and software re-use. The current set of standards include Call Processing Language [10], Parlay/OSA and Parlay X [14], SIP Servlet [3], and JAIN Service Logic and Execution Environment (SLEE) [8]. Each of these standards has its strengths and weaknesses, and application developers and deployers can pick the best standard for their needs.

It is perhaps surprising that in the area of endpoint services, there is not a corresponding wide choice of standards. In most cases the endpoints provide a fixed set of features developed by the vendor using proprietary means. In a few cases, the endpoints support the deployment of third-party applications, but the programming interface is proprietary. This serves as the motivation for this investigation into whether programming models for application servers are also suitable for endpoint applications. In particular, the SIP servlet API is examined for this purpose, and an enterprise VoIP system is used as a case study.

2. Endpoint and Network Services

2.1. Where Should Applications Reside?

There have been continued debates regarding the optimal location of VoIP applications. The guiding principles in the design of the Internet since its inception in the 1980s have been the *end-to-end arguments* proposed by Saltzer, Reed and Clark [13], which argue that higher level functionalities should be implemented at the endpoints of the communications system as opposed to in the core of the network. In the Internet Engineering Task Force (IETF) that defines the SIP and related standards, some participants use the end-to-end arguments to propose that SIP applications should reside at the endpoints. According to this view, while some functions outside of the endpoints are acceptable, such as SIP registrars and proxies, more complex applications executing on SIP AS are discouraged.

This view is in stark contrast to the trend in the VoIP industry towards network-based application servers discussed above. In addition, there are certain applications that cannot be implemented at the endpoint. For example, in a residential service that the author was involved in [5], an application called *Safe Forwarding* redirects incoming calls to an alternative destination if the called endpoint is offline, perhaps due to a power outage or network failure. Clearly this application cannot be implemented at the endpoint.

Blumenthal and Clark more recently [4] make a distinction between elements “in” the core of the network, versus elements “attached to” or “on” the network. In the former case, devices such as routers provide the basic data forwarding service, and a device failure can crash the network and affect all applications. Adding functions to these devices “in” the network is undesirable because they constrain application behavior and add complexity and risk. In the latter case, functions that serve a specific application may be implemented in an element that is “on” the network, for example a server computer. From the perspective of the core network, this computer and these functions represent an endpoint. If this element fails, only the specific application is affected. The impact is much more confined. Therefore, the latter case becomes a subject of application design. VoIP application servers clearly belong to the latter case.

We believe that in the architectural design of VoIP services, applications at the endpoint and in application servers on the network are both required. Building on the work in [4, 15], the considerations and tradeoffs for the two options are summarized below:

Applications in endpoints

- An endpoint can interact with its user more easily, for example with a graphical user interface.
- An endpoint has direct access to the media streams.
- Communication privacy and data security do not rely on trust relationships with the network.
- Endpoint services may be more robust with respect to network outages.
- Endpoint services can be more scalable because adding endpoints automatically adds more processing power.

Applications in servers

- Software distribution and upgrade are easier.
- If a feature must be invoked even if the user’s endpoint is offline or turned off, it must be implemented in a network server. Many coverage type features for the called party fall into this category, such as Safe Forwarding described above.
- It is often easier to provide high availability and reliable network connectivity and power to servers in the network than to the endpoint device of a user.
- More processing and storage capacity can be provided in the network, whereas they can be limited on endpoint devices due to size, power and price concerns.
- The parties in a communication may not have a trust relationship with each other. In this case services provided by a mutually trusted third party are necessary.
- If the user has multiple devices, some features on network servers may be convenient. For example a network-based address book may be accessible from a variety of endpoint devices.

2.2. Case Study of an Enterprise Communication System

We will examine a trial SIP-based enterprise communication system (ECS) that has been in use at our office. This system supports user presence, and real-time communication with voice, video, and text instant messages (IM). For the endpoints, we use the CounterPath X-Lite softphone software [7] that runs on Windows or Mac OS X operating systems. The X-Lite software supports the SIP and the SIMPLE (SIP for Instant Messaging and Presence Leveraging Extensions) protocols, and offers superior audio and video quality. It is free of charge, but closed-source.

A SIP servlet container running on a Linux server in the network provides the application execution environment for ECS, and will be referred to as the Network Application Server (NAS). For calls to the Public Switched Telephone Network (PSTN), a SIP-to-PSTN gateway is also included. Figure 1 shows the ECS system architecture.

In SIP, calls are made to a user’s public address or *address-of-record* (AOR) which must then be mapped to the user’s device. A SIP registrar and routing proxy application has been developed using the SIP servlet API and deployed on the NAS for this function. When a user starts up the softphone, it registers with the registrar and provides its contact address information. Subsequently, when another user calls the first user’s AOR, the routing proxy application on the NAS maps the AOR to the device address, and proxies the request to the softphone.

As we use this system for our day-to-day communication, a number of new features have been identified. Since the X-Lite software is closed-source and does not offer any

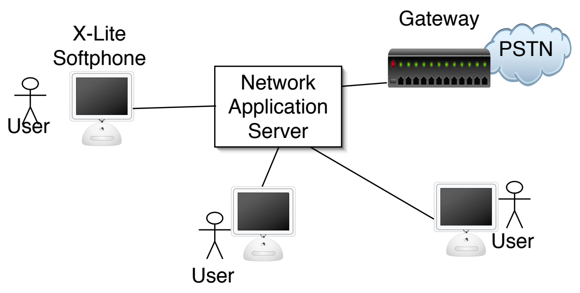

Figure 1. Architecture of the Enterprise Communication System

means to augment its functionality, it is necessary to implement these features elsewhere in the system. One approach is to implement them in the central NAS. However, we have identified a number of features that for a variety of reasons are more suited for the endpoints:

Caller ID — For convenience, each user of ECS is identified by an alphanumeric AOR instead of a phone number (e.g. `sip:cheung@att.com` instead of `+1 212 555 1212`). However, when calling out to the PSTN a phone number should be provided to the gateway so the called party will receive a Caller ID.

This can be achieved by adding a *Caller ID Insertion* application that uses the caller's AOR in a request to look up and insert the corresponding phone number into the request. As this application is only required when the endpoint is running, it can be implemented at the endpoint. An advantage is that if the same user accesses the service from different computers at different locations, for example at the office and at home, the application can be configured differently on the two computers to insert the office and home phone numbers as appropriate.

Security of instant messages — Although the SIMPLE protocol allows encryption of message contents, the X-Lite software does not support this feature. Therefore the content of IM conversations traverses the network and the ECS server in the clear. If the user does not trust the security of the network or the server, an *IM Encryption* application can be used to encrypt the message body before sending it to the NAS, and correspondingly decrypt incoming messages. Because of the trust issue, this application should be implemented at the endpoint.

IM archiving and timestamping The X-Lite softphone does not show the time when an IM is received, or provide automatic logging of conversations. An *IM Timestamping* application adds timestamps to incoming messages so they will be displayed as part of the IM on the softphone. An *IM Archiving* application writes IM conversation to a file. If the IM Encryption application is also required, these two applications must also reside at the endpoint to access the unencrypted messages.

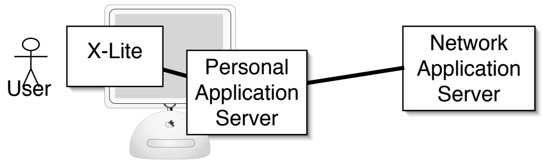

Figure 2. Personal Application Server running on endpoint

We now begin to investigate the feasibility of implementing these features on an application server that runs either on the same machine as the softphone or another co-located machine. In general, there is one such Personal Application Server (PAS) for each endpoint. This arrangement is shown in Figure 2.

3. Implementation

3.1. The SIP Servlet Standard

The SIP servlet API (SSAPI), standardized in the Java Community Process (JCP) [3], defines a container-based programming model for building SIP applications similar to the HTTP servlet API. SSAPI provides a level of abstraction above the SIP protocol stack, and offers a programming model that is already familiar to a large web application developer community. The SIP servlet container handles low-level functionality such as message formatting, retransmission, correlating messages to sessions, and protocol timeouts. The container also provides persistence of application state, and optionally can support high availability via clustering and automatic fail-over.

The SSAPI is well integrated with Java Enterprise Edition (Java EE) technologies, and in particular supports converged applications that handle both SIP and HTTP requests. Thus the SSAPI is well suited for blending Web and VoIP applications.

At the time of writing, there are at least five commercial and four open source SIP servlet containers, making SSAPI the best-supported SIP application environment. The new version 1.1 of SSAPI was released in August 2008 and already several of these implementations have been upgraded to the new version. The availability of open source containers is particularly important for the work described in this paper, as it makes deploying a container for each endpoint economically feasible.

3.2. SIP Request Routing

The SSAPI standardizes a mechanism for *application composition*. Multiple applications may be deployed on a container and invoked in an application chain to serve a call. A special component called the *application router* (AR) is responsible for application selection. When the container receives an initial SIP request (such as a INVITE request which initiates a session), it queries the AR. If the AR returns an application deployed locally, the container dispatches the request to the application. Alternatively, the AR may return an external route. In this case the container sends the request to the external destination directly without invoking any local application.

The standard does not mandate a particular application selection algorithm, only an API between the container and the AR. Deployers of SIP servlet systems may plug in different AR implementations. The Distributed Feature Composition Application Router (DFC AR) is a compliant AR implementation [6] based on the Distributed Feature Composition architecture [9]. It supports subscription to applications by addresses and precedence relationships among applications. By controlling which applications are invoked and in what order, the DFC AR helps manage feature interaction.

A goal of this work is that when a user installs a PAS, the NAS should not require any modification. An endpoint and the composite of an endpoint and a PAS should appear identical to the NAS. To achieve proper routing of SIP requests among the PAS, the NAS, and the endpoints, two applications are required on the PAS to handle SIP routing: Personal Registrar (PR) and Locate Me (LOC). The DFC AR is configured with the subscription and precedence below:

Region	Address	Applications in order
Originating	User's AOR	Personal Registrar External route to NAS
Terminating	User's AOR	Locate Me

Note that the DFC AR does not select applications based on the method of the initial request (e.g. REGISTER or INVITE). Instead, when the container invokes an application, the application may indicate to the container that it is not interested in handling the request. For example, PR is only interested in handling REGISTER requests.

The following sections examine how routing is achieved under different scenarios.

3.2.1. Softphone Registration The softphone is configured to register with the PAS. When PAS receives the REGISTER request, the AR matches the originating address to the user's AOR, and selects the first originating application in the configuration, PR, to handle the REGISTER request.

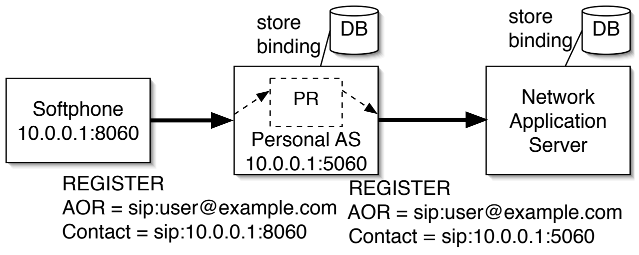
store binding
DB
store binding
DB
Softphone 10.0.0.1:8060
PR
Personal AS 10.0.0.1:5060
Network Application Server
REGISTER
AOR = sip:user@example.com
Contact = sip:10.0.0.1:8060
REGISTER
AOR = sip:user@example.com
Contact = sip:10.0.0.1:5060

Figure 3. Message flow when softphone registers with PAS

PR acts as a regular SIP registrar and stores the AOR-to-Contact binding of the softphone. However, PR also issues a REGISTER request with the user's AOR, but with the contact address of the PAS. The container queries the AR again, which returns the next item in the configuration i.e. external route to the NAS. The PAS sends the REGISTER request to the NAS. As a result, NAS has a binding for the user's AOR to the PAS, and PAS has a binding to the softphone. This call flow is illustrated in Figure 3.

3.2.2. Incoming Call to the User When another user sends an INVITE request to the AOR, the NAS sends the request to the PAS based on the stored AOR-to-Contact binding. On the PAS, because the terminating address is the user's AOR, the AR selects LOC to handle the request. LOC is a straightforward routing proxy application as defined in[12]. LOC reads the AOR-to-Contact binding information previously stored by the PR, and proxies the request. Because there are no more terminating applications, PAS sends the request to the softphone.

The SIP SUBSCRIBE, NOTIFY, and MESSAGE requests are used for presence and IM, and they are routed in a similar way.

3.2.3. Outgoing Call When the softphone places an outgoing call, it sends the INVITE request to the PAS. As in the case of registration above, the AR selects PR as the first application. However, because PR only handles REGISTER requests, the container queries the AR again. This time the AR returns an external route to the NAS, and the container sends the request to NAS. At this point, processing continues normally on the NAS.

3.2.4. Adding the feature applications With the basic SIP routing achieved by the proper configuration of the AR and the PR and LOC applications, applications that implement the features discussed in Section 2.2 may be added. Clearly, Caller ID Insertion is only required when the user is placing an outbound call. The IM Timestamping application is only required when the user receives a text message. On the other

Region	Address	Applications in order
Originating	User's AOR	Personal Registrar Caller ID Insertion IM Archiving IM Encryption External route to NAS
Terminating	User's AOR	IM Encryption IM Archiving IM Timestamping Locate Me

Table 1. DFC AR Configuration for routing and feature applications

Container	Disk Space (Mbytes)	Memory (Mbytes)
SailFin on GlassFish	137	938
Mobicents on Tomcat	17	297
Mobicents on JBoss	122	773

Table 2. Disk Space and Virtual Memory Usage of SIP Servlet Containers

hand, IM Encryption and IM Archiving must be invoked whether the user is sending or receiving a request.

The precedence relationship among the applications is very important. For example, the functions of IM Timestamping and IM Archiving require access to the clear text of the messages. Therefore, IM Timestamping and IM Archiving must be inserted between IM Encryption and the softphone. The new configuration of the DFC AR incorporating the three applications is shown in Table 1.

4. Performance and Evaluations

4.1. Endpoint Requirements

At the time of writing, there are three open source SIP servlet container implementations that comply with version 1.1 of SSAPI — SailFin on GlassFish Java EE container [2], and Mobicents SIP Servlets on either Apache Tomcat servlet container or JBoss Java EE container [1]. Table 2 summarizes the hard disk and virtual memory usage after each container is installed and started on a MacBook Pro laptop computer with 2.16 GHz Intel Core Duo processor and 2 GB RAM, running Mac OS X 10.4.

Not surprisingly, the two Java EE containers have higher system requirements. However, all three options should run successfully on modern desktop or laptop computers with-

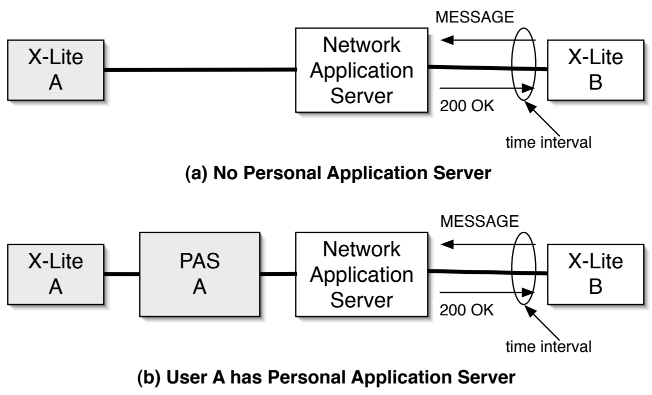
X-Lite A
Network Application Server
MESSAGE
200 OK
X-Lite B
time interval
(a) No Personal Application Server
X-Lite A
PAS A
Network Application Server
MESSAGE
200 OK
X-Lite B
time interval
(b) User A has Personal Application Server

Figure 4. Configurations Used to Measure Round Trip Delay Added by PAS

out straining the system resources. For smaller devices such as mobile phones, a smaller footprint may be required.

4.2. Signaling Delays

Because the PAS executes on each user's computer, the main consideration is not the processor utilization or maximum throughput. Compared to the centralized NAS, the PAS only has to handle a small number of SIP messages for its single user. It is more important that the addition of the PAS should not degrade the responsiveness of the overall communication system excessively.

To assess the additional signaling delay caused by the introduction of the PAS, the time interval between a SIP MESSAGE request and the corresponding response is measured at a peer softphone. In the testing configuration shown in Figure 4(a), softphone B sends a MESSAGE request, which is relayed by NAS to softphone A. Softphone A sends a response automatically, and the response is relayed by NAS back to softphone B. The time interval as measured at softphone B between the request and response is the round trip time of a SIP transaction. Similarly, the configuration shown in Figure 4(b) measures the round trip time when a PAS is introduced. Only the routing applications are deployed in PAS. Ten SIP transactions are measured to give an average value in each case. Table 3 shows the results measured on SailFin, and indicates that on average the PAS adds about 13 milliseconds of round trip time. This should have no noticeable impact on the user experience.

The round trip time is expected to increase when applications providing features are added to the PAS. However, if these applications are deployed on the NAS, similar increase is expected on the NAS. Therefore, the feature applications are excluded from these configurations.

Configuration	Average time interval (ms)
(a) No PAS	132
(b) One PAS	145
Difference	13

Table 3. Measured Time Interval Between Request and Response at Peer Softphone

4.3. Handling Network Topology

Endpoints often have to handle different networking configurations. In a home or remote office network there are likely to be Network Address Translation (NAT), firewalls, and Virtual Private Network (VPN) involved. Network topology discovery mechanisms such as STUN [11] are required for the SIP signaling messages to be transmitted successfully. For example, on a machine behind NAT and thus has a local IP address, X-Lite uses STUN to discover its externally routeable IP address. This external IP address is then used to populate the Contact address in outgoing SIP requests, so that responses can be routed back.

In contrast, SIP servlet containers are targeted for network servers. Neither SailFin nor Mobicents support similar mechanisms. If a SIP servlet container executes on a machine behind NAT, it will put its local IP address in outgoing requests and result in failures. Therefore, enhancements to implement these mechanisms are required to support such networking environments.

5. Conclusion and Future Work

This paper shows that the same open standard programming model, the SIP Servlet API usually used for developing network applications, can be employed successfully to program endpoint applications. The use of the same programming model for both cases affords additional re-use of software, development tools, and programmer resources. It also allows the flexibility of migrating the same application between network and endpoint in different contexts.

An important advantage of this approach is that it supports composing multiple applications, possibly from different sources, to form a complex service at the endpoint.

All three open source container implementations examined in this paper can be installed and run successfully on the same computer that executes the softphone. However, a number of issues have been identified for future work — smaller embedded versions of containers will be required for mobile phones and other small devices. Support for network topology discovery will also be required.

Due to space limitations, this paper does not discuss interactions between the users and applications, for example to alert the users of new events or to allow the users to control call handling. As indicated earlier, the SSAPI supports convergence with HTTP and other protocols, and is well suited for blending VoIP and other applications. Therefore, it is an interesting area of study to see how users may interact with SIP servlet applications at the endpoints.

6. Acknowledgements

The author continues to benefit from a long and rewarding collaboration with my colleagues Greg Bond, Hal Purdy, Tom Smith, Venkita Subramonian, and Pamela Zave.

References

[1] Mobicents SIP Servlets. http://www.mobicents.org/.
[2] SailFin project. https://sailfin.dev.java.net/.
[3] BEA. SIP servlet API version 1.1, 2008. Java Community Process JSR 289. http://jcp.org/en/jsr/detail?id=289.
[4] M. S. Blumenthal and D. D. Clark. Rethinking the design of the Internet: The end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, August 2001.
[5] G. W. Bond, E. Cheung, H. H. Goguen, K. J. Hanson, D. Henderson, G. M. Karam, K. H. Purdy, T. M. Smith, P. Zave, and J. C. Ramming. Experience with component-based development of a telecommunication service. In *Proceedings of the Eighth International SIGSOFT Symposium on Component-Based Software Engineering*, pages 289–305. Springer-Verlag LNCS 3489, 2005.
[6] E. Cheung and K. H. Purdy. An application router for SIP servlet application composition. In *IEEE International Conference on Communications*, pages 1802–1806, 2008.
[7] CounterPath. X-Lite. http://www.counterpath.net/.
[8] D. Ferry. JAIN SLEE (JSLEE) 1.1 specification, 2008. Java Community Process JSR 240. http://jcp.org/en/jsr/detail?id=240.
[9] M. Jackson and P. Zave. Distributed Feature Composition: A virtual architecture for telecommunications services. *IEEE Transactions on Software Engineering*, XXIV(10):831–847, October 1998.
[10] J. Lennox, X. Wu, and H. Schulzrinne. Call processing language (CPL): A language for user control of internet telephony services, October 2004. IETF RFC 3880.
[11] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session traversal utilities for NAT (STUN), 2008. IETF RFC 5389.
[12] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session initiation protocol, June 2002. IETF RFC 3261.
[13] J. Saltzer, D. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.
[14] The Parlay Group. Homepage. http://www.parlay.org/.
[15] X. Wu and H. Schulzrinne. Where should services reside in internet telephony systems? In *Proceedings of IP Telecom Services Workshop (IPTS) 2000*, pages 35–40, 2000.