A High-Level, Open-Ended Architecture For SIP-based Services

Laurent Burgy, Charles Consel, Fabien Latry, Nicolas Palix, Laurent Réveillère

A High-Level, Open-Ended Architecture For SIP-based Services

Laurent Burgy Charles Consel Fabien Latry Nicolas Palix Laurent Réveillère

E-mail : {burgy,consel,latry,palix,reveillere}@labri.fr
INRIA / LaBRI / ENSEIRB
Department of Telecommunications - 1, Avenue du Docteur Albert Schweitzer,
Domaine universitaire - BP 99; F-33402 Talence Cedex, France

1 Introduction

Now that Internet Telephony can interact with systems such as databases, e-mail facilities and Web services, it can offer a host of new functionalities. However, developing enriched, real-size services is quite a challenge considering the requirements that must be fulfilled by the service developer. Such developer must (1) have an extensive knowledge on network protocols and distributed systems; (2) be familiar with often large and complex platform APIs (*e.g.*, JAIN [6]); and (3) fully understand the signaling protocol (*e.g.*, SIP [5]) to develop services that do not compromise the processing of the calls, nor the platform.

All these areas of expertise are required by most existing platforms. They offer unrestricted APIs and support mainstream programming languages such as C, C# and Java. They provide little abstraction, and thus rely on the programmer to manage the intricacies of the underlying technologies (protocols, network layers, and signaling). Other platforms enable service creation through a scripting language, such as CPL [3, 4] and LESS [7], that offers a restricted expressiveness and mostly targets the creation of individual user services.

We present a high-level architecture of an Application Server for SIP-based services. Our architecture abstracts over the intricacies of the underlying technologies and facilitates both the development and the management of services.

By revolving around an Application Server, our approach allows a uniform and coherent basis of telephony services to be offered to the platform users, regardless of the heterogeneity of their end systems.

2 Requirements

Because availability of services critically relies on an Application Server (AS), it must fulfill stringent requirements. This section lists these requirements and discusses potential problems if they are not fulfilled.

Service Management. An important issue for a telephony platform is to provide a powerful mechanism to deploy and manage services.

Feature Interactions. Many services can be deployed in an Application Server and associated to the same user or group of users. A key issue is to determine which services to trigger and in what order, when a SIP request is received.

Abstraction level. Service creation usually requires a detailed understanding of the underlying platform and protocols to overcome the intricacies and pitfalls of such development. Insufficiently abstract API results in services that are hard to develop, understand and maintain.

State Management. The implementation of a service is typically split into various processing components. If a state is required during the life-cycle of the service, it must be explicitly handled by the service developer making programming laborious and error prone.

Open-ended. The ability of Internet telephony services to interact with systems such as databases is a major factor to the success of ToIP. This ability critically relies on the open-ended nature of the AS, permitting new functionalities to be interfaced to the services.

3 Application Server

To address the needs of telephony services in a SIP platform, we have developed a new Application Server architecture. The overall view of our architecture and its main components are displayed in Figure 1. Let us present each of these components in turn.

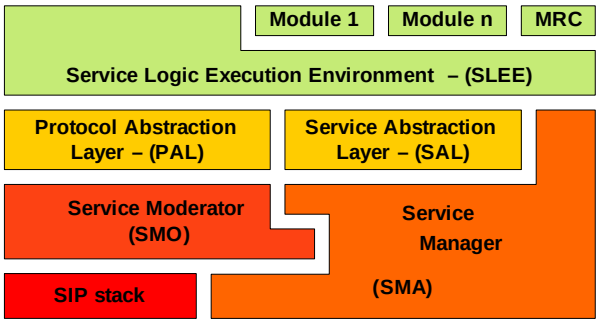

Figure 1. AS architecture

Service MAnager (SMA). The SMA provides an interface to control the life-cycle of services. Services are stored in a local repository in their executable form. Once deployed, a service can be associated to a specific user or group of users. Additionally, the SMA analyses services and estimates their resource usage. This information is used as an input of an admission control mechanism to new services.

Service Abstraction Layer (SAL). When a service is activated by the SMA for a specific user, a platform event, named `deploy`, is raised. This event can be handled by the service, which may have defined a specific treatment for this stage. Similarly, when a service is de-activated, the `undeploy` event is raised and the corresponding handler invoked, if it is defined by the service. Although these two events are generated by the SAL, both SIP requests and platform events can be uniformly handled by the programmer.

The SAL also provides the programmer with support to manage the state used during the service life-cycle.

Service MOderator (SMO). When a SIP request is received, the SMO triggers the appropriate services. To do so, it evaluates a set of triggering criteria that produces an ordered list of services. When a service has been executed, the SMO decides which subsequent service should be executed.

Protocol Abstraction Layer (PAL). Some SIP requests are ambiguous and require some interpretation. For example, the `INVITE` request either initiates a dialog or, if used in the context of an existing dialog, re-configures the dialog. The aim of the PAL is to classify requests according to the stage at which they occur in the life-cycle of a dialog: *initial* (creation), *medial* (confirmation and modification), and *final* (termination). We call this notion of a dialog a *session*. For each kind of request, the PAL raises a specific event. For example, a SIP `INVITE` request is refined into a `REINVITE` event, when it refers to an existing session. The PAL introduces three kinds of sessions to organize various requests: the registration (`REGISTER`), the event (`SUBSCRIBE`) and the dialog (`INVITE`). Platform events are raised by the PAL to enable the service to react to a timeout session termination

Like the SAL, the PAL also provides operations to attach a state to a session and to manage this state during the life-cycle of the session.

Service Logic Execution Environment (SLEE). The SLEE reacts to events from the PAL and the SAL. It provides the service with a high-level and unified view of events. The SLEE also organizes sessions exposed by the PAL and the SAL as a hierarchy, providing a way to easily manipulate states.

Modules. Modules allow to extend the AS functionalities, as is done by other platforms such as Apache [1]. In the case of SIP-based services, every Internet services can potentially become a resource and be needed in a telephony service.

Modules are used to access a large spectrum of resources from local services (e.g., accounting information) to Internet services (*e.g.*, databases).

4 Conclusion and Future Work

In this paper, we have described an Application Server for SIP-based telephony services. The requirements to create services and to manage them have been used to design this Application Server.

At ENSEIRB, an engineering school to which the authors are affiliated, a VoIP infrastructure, TelIP, has been deployed based on the SIP Express Router (SER) [2].

To enable customization for ever demanding users, a dedicated AS, based on the JAIN SIP stack, was added in the TelIP infrastructure. The AS activation only required a few lines of configuration in SER.

The AS has been extended with a Multimedia Resource Controller (MRC) module to drive a Multimedia Resource Processor (MRP), which allows management of RTP sessions. This MRC allows the deployment of a hotline service that requires an MRP to play audio messages. Our implementation of the MRP currently supports playing audio messages and bridging of two RTP sessions... Furthermore, this MRP can be used to implement voice-mail server or virtual switchboard for example.

We are now investigating a more elaborate SMO based on the analysis of services to handle complex feature interactions. Finally, we are studying models of capabilities to ease service management for an administrator, and to control user resource consumption such as PSTN lines.

References

[1] Apache HTTP server project. http://www.apache.org/.

[2] iptel.org. *SER Developer's guide*, Sep. 2003.

[3] J. Lennox and H. Schulzrinne. Call processing language framework and requirements. RFC 2824, May 2000.

[4] J. Lennox and H. Schulzrinne. CPL: A language for user control of internet telephony services. IETF, IPTEL WG, Nov. 2000.

[5] Rosenberg, J. et al. SIP: Session Initiation Protocol, Jun. 2002.

[6] Sun Microsystems. The JAIN SIP API specification v1.1. Technical report, Sun Microsystems, June 2003.

[7] X. Wu and H. Schulzrinne. Programmable end system services using SIP. In Proceedings of the *International Conference on Communications* 2002.