

A Comparative Study of SIP Programming Interfaces

Laurent Burgy Laurence Caillot Charles Consel Fabien Latry Laurent Réveillère

E-mail: {burgy,caillot,consel,latry,reveillere}@labri.fr

INRIA / LaBRI

ENSEIRB - 1, Avenue du Docteur Albert Schweitzer,
Domaine universitaire - BP 99; F-33402 Talence Cedex, France
Phone: +33 5 56 84 23 21 Fax: +33 5 56 84 21 80

Abstract : The Session Initiation Protocol (SIP) is a signalling protocol for Internet Telephony, multimedia conferencing, and instant messaging. The behavior of SIP platforms can be configured thanks to various programming interfaces.

In this paper, we present a study of different existing SIP platforms. From this study, we identify key requirements that must be fulfilled by a SIP programming interface. We discuss existing solutions and propose to introduce a language approach to addressing these needs.

1. Introduction

The convergence of telecommunications and computer networks, taking the form of telephony over IP, has added a host of new functionalities to telephony services including Web resources, databases, *etc.* Making these rapidly evolving functionalities available to customers critically relies on developing a stream of new telephony services. Such a wide spectrum of functionalities enables telephony to be customized with respect to preferences, trends and expectations of ever demanding users. As a consequence, these customizations entail a proliferation of telephony services, which in turn emphasizes the need for programming services.

Fortunately, the convergence of telecommunications and computer networks has made the programming of telephony services as accessible as the programming of networking services. In practice, most telephony platforms offer solutions to enable service programming; this approach sharply contrasts with the traditional proprietary and closed telephony platforms. However, this openness comes at the expense of undermining the robustness of the underlying platform, which can compromise a basic commodity such as telephony.

To address the robustness issue, it is critical to define requirements to service programming. For such a study to be useful, we need to focus on a particular type of platforms, integrating telecommunications and computer networks. To do so, we selected platforms based on the rapidly emerging Session Initiation Protocol (SIP) [10]. This protocol is being deployed in key contexts such as Universal Mobile Telecommunications System (UMTS).

This Paper

This paper aims to identify and motivate a set of requirements needed to program services in the telephony domain. It examines how these requirements are taken into account in existing SIP platforms. Furthermore, it discusses the necessity to rigorously take into account all these requirements and suggests that to do so a language dedicated to telephony services should be introduced.

2. SIP Platforms

To offer rich functionalities, a SIP platform provides the programmer with a rich and usually complex interface. This programming interface enables access to call parameters (*e.g.*, the origin, the time, and the location), signalling operations (*e.g.*, call forwarding), and non-signalling resources (*e.g.*, Web and databases). The resulting complexity of the interface can make the rapid design, development and deployment of a service a real challenge.

The SIP community has made major advances towards improving this situation. Interestingly, most of these advances are aimed to introduce high-level programming interfaces and to design dedicated languages to further raise the level of abstraction. These trends are key aspects to identify service programming requirements.

Our study covers a variety of SIP platforms, namely, Vocal (Vovida) [3], SIP Express Router (iptel.org) [4], Live Communications Server (Microsoft) [8], and AppEngine (DynamicSoft). These SIP platforms provide a more or less complete framework for developing SIP applications. Nevertheless, their aim is to simplify and accelerate service creation by providing an interface that is well-suited for service programming.

Vocal, for Vovida Open Communication Library, is a complete open source software solution, implemented as a distributed platform. The Vocal system consists of various SIP servers and can be configured at deployment-time using an ad hoc graphical user interface. One of these servers, the Feature Server, enables service creation using the Call Processing Language (CPL) [7, 11]. CPL



is an XML-based scripting language for describing and controlling call services. A CPL script simply represents a decision tree whose nodes specify actions or decisions to take.

Sip Express Router (SER) is an open source project developed by iptel.org aimed to be a high-performance SIP platform. The core of the platform proposes a configurable SIP-compliant server. It can be statically configured thanks to a dedicated C-like language, defining the message-routing logic. Nevertheless, like the HTTP Apache server, an API offers hooks to extend the core of the platform, through modules written in C.

Microsoft Office Live Communications Server 2003 is a manageable and extensible instant messaging server. This platform requires the Windows Server 2003 operating system, and can only be extended using Microsoft products. Thus, it is completely integrated into the Microsoft framework, and its openness is limited by some features of this environment (*e.g.*, authentication, application dependencies). It supports SIP and SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE). This extensible platform for real-time communications includes various communication services like chat, video and audio conferencing, and data collaboration.

AppEngine is a commercial SIP platform developed by Dynamicsoft. It is a pure Java framework providing a powerful SIP Servlet API [5] for developing and deploying SIP based applications. The AppEngine streamlines the development of communication services enabling the use of existing Web-based application development tools.

Noticeably, our study is based on very different types of SIP platforms. Their features and their benefits are very diverse. Indeed, the target of these platforms range from residential/SOHO/small business to large-scale infrastructures. Furthermore, they exhibit different levels of programming interfaces; they handle programmers with different levels of trust; and, they illustrate different software architectures, enabling various degrees of extensibility.

3. Requirements for Programming Telephony Services

Because telephony is a basic commodity, it has stringent requirements [9]. This section defines these requirements, and illustrates them in the context of existing signalling platforms.

Multi-level programming. Programming telephony services can be viewed as performing two kinds of actions: signalling actions and non-signalling actions. Sig-

nalling actions include call forwarding or call responding, whereas non-signalling actions are related to databases, Web resources, *etc.* One of the requirements to program services is thus to facilitate and simplify the use of signalling actions in the common cases, while providing an extended interface to program elaborate call treatments. To address this need, Microsoft's platform splits the programming of a service into two parts, each corresponding to a specific programming language. The top part is written in a kind of configuration language (MSPL) and aims to define basic SIP message routing. The bottom part is written in a general-purpose language, namely C#; it gives full access to the state and functionalities of the platform.

In contrast to the Microsoft platform, Vocal only provides a basic programming interface, simply aimed to route SIP messages, thus giving restricted programmability.

Abstraction level. The abstraction level of the programming language is an important issue. Indeed, high-level abstractions enable the programmer to ignore the intricacies of both the platform and the underlying protocols. In doing so, a whole class of errors can be eliminated. The resulting services are easier to read, develop and maintain. Concretely, programming services in SIP Servlets or the SER language requires the understanding of the details of the underlying protocol. In comparison, Vocal, thanks to the high-level nature of CPL, requires very little expertise to program services.

Untrusted Users. An important issue for a telephony platform is to define the service programmer community. In particular, one question is whether or not untrusted users should be allowed to write and deploy services. Evidently, if untrusted users are allowed to develop services, they should be provided with a programming interface and/or a programming language that prevent a faulty or malicious service from crashing the platform. Among the platforms we studied, only Vocal enables untrusted services to be developed and deployed without sacrificing the platform robustness. This is achieved by providing untrusted users with a restricted programming language. All other platforms included in our study only offer unrestricted programming interfaces and languages.

Verifiability. The platform must be able to verify automatically that a user-defined service is well-formed and can be successfully executed by the signalling server. This needs to be verified at the time the service is submitted, since discovering a fault at run time can lead calls to being lost. Moreover, it must ensure call completion, that is, the service completely executes in a finite amount of time and eventually performs a signalling action. This property is ensured in the Vocal platform thanks to a syntactic verification performed on a service when it is

submitted. In contrast, nothing is done in the SER platform, where the interaction between external modules can compromise the termination of a service.

Performance. Because a signalling platform is usually intensively solicited, it must run on high-performance equipment to process calls at high rates. Although a telephony platform can be configured to handle standard call processing, it may not be fit to handle an arbitrary set of telephony services, whether or not written by untrusted users. Indeed, it is difficult to assess the resource consumption of telephony services, and thus how their execution might affect the rate at which calls are processed. In SER, C-modules can perform arbitrary computations and slow down the call processing rate. In contrast, the performance impact of MSPL scripts could be more easily estimated due to the restrictions of this language.

Resource control. Introducing programmability in a domain such as telephony raises a number of challenges. One of these challenges relates to the cost of services that are deployed on a platform. The cost of services, expressed in terms of resource usage, serves a number of purposes including the admission control of services, the platform configuration, and the definition of billing policies.

Based on our domain analysis, we define a resource as being either an operation to perform or an entity to acquire, whose unsuccessful completion can block the execution of a service. This definition covers more than the conventional, general-purpose resources such as CPU, memory, and bandwidth. It introduces domain-specific resources that can be classified in four categories: (1) signalling actions (*e.g.*, proxy, forward, redirect), (2) non-signalling actions, carried out by the platform (*e.g.*, DNS request, SIP registration, and lookup for users location), (3) entities available on the telephony platform (*e.g.*, PSTN lines and authentication), and (4) external services (*e.g.*, Web servers, e-mail, and RPC call).

To ensure the successful execution of services, a resource control mechanism must be introduced. To do so, for a given service, resource analysis and control can be performed *statically* and/or *dynamically*. On the one hand, static analysis and control of resources has the advantage of occurring at compile time and thus not incurring any run time overhead. However the lack of execution context may lead to over-estimate the resources required by the service and does not permit the availability of the resources required by the service at invocation time to be checked.

On the other hand, dynamic analysis and control of resources can exploit the execution context, and thus check the availability of the resources required by the service. The main drawback of this strategy is that it incurs a run-time overhead, making it difficult to meet

the stringent performance constraints of the telephony domain. Also, deferring resource checks at run time may lead to disrupt the processing of a telephone call, if the service consumes an excessive amount of resources. Yet, this strategy is used by most existing SIP signalling platforms: they monitor resource allocation at run time and terminate any ill-behaved service.

The traditional static/dynamic stage separation is too coarse-grained when analyzing and controlling the resource requirements of a telephony service. Instead, we propose to perform this process over four stages [6]: (1) design and compilation, (2) platform deployment, (3) service invocation, and (4) service execution. The information needed to control the resources varies from one stage to another. In fact, a service may be rejected at any stage of the process, if information available at the stage enables the over-consumption of resources to be detected. This stepwise process is fundamental to enable ill-behaved services to be detected as early as possible.

4 Assessment

In this section, we review the requirements that must be fulfilled by a SIP platform to enable services to be created. We provide some insights resulting from the different platforms we studied. Finally, we propose a language approach to addressing all of these requirements.

Summary

Table 1 summarizes the requirements to be considered when enabling programming of telephony services. It assesses each requirement with respect to the SIP platforms covered by our study. As can be observed, no platform addresses the complete list of the requirements, although each of them takes steps towards fulfilling this list. As a continuation of our study, we have been designing and developing a programming language dedicated to the telephony domain that would fulfill all of the above mentioned requirements.

Discussion

The key feature of SIP is its programmability. Combined with its simplicity, this programmability has allowed SIP to be widely adopted as the *de-facto* standard for VoIP. To fully exploit programmability without sacrificing the safety and security of the platform, the requirements discussed earlier must be addressed.

Multi-level programming is needed to develop more evolved services than basic filtering. By offering high-level abstractions, a language is more accessible to users with limited programming experience and little domain expertise. Nevertheless this accessibility must not allow untrusted users to develop error-prone or malicious services.

	SER	Vocal	SIP Servlet	MS Live Communications Server
Multi-level Programming	Not addressed	Not addressed	Through an XML configuration file	Through MSPL
Abstraction Level	Low-level interface	High-level CPL language	High-level Java interface	High-level C# interface
Untrusted Users	Not addressed	Trusted and untrusted	Not addressed	Not addressed
Verifiability	Difficult	Easy	Difficult	Easy in MSPL Difficult in C#
Resource Control	standard resource reservation (DiffServ, IntServ, RSVP) but no advanced control			

Table 1. Requirements for programming telephony services

The safety and the security of services depend on domain-specific properties that can be identified at four different levels: (1) telephony domain, (2) SIP signalling protocol, (3) SIP signalling platform, and (4) telephony services. Checking these properties in a service is a major step towards ensuring its safe and secure execution.

The overall performance of a signalling platform can be significantly degraded by a resource-greedy service. Thus, operations that restrain the consumption of resources must be introduced. However, considering the complexity of a resource control mechanism, it should not be directly exposed to the end user.

Existing signalling platforms rely on various programming paradigms. In fact, each platform offers its own paradigm. Consequently, a service targeted to a specific platform can not be easily ported to another one. Reusing an existing service on a different platform requires completely rewriting the code to suit the new target.

To overcome the above mentioned limitations, we propose to introduce a domain-specific language. The design and implementation of this language is based on a thorough domain analysis of telephony services.

Our language is portable in that it can be targeted to different platforms. Its dedicated nature is represented by domain-specific abstractions and notations that ease service programming. A service written in our language is compiled into the native programming interface of the target platform (*e.g.*, CPL, SER scripting language, SIP-Servlet, MSPL/C#).

5. Conclusion and Future Work

The convergence of telephony and computer networks have brought a host of new functionalities to the domain of telecommunications. However, this evolution should lead telephony services to be developed by an increasing number of non-expert programmers. To make this evolution worse, the existing techniques to program server extensions are rather low-level and/or unrestricted.

These shortcomings make the programming of telephony services an error-prone process, jeopardizing the robustness of the platform.

In this context, a safe and high-level language is needed to enable the programmer to concentrate on the service logic without dealing with the low-level details, while implementing robust services. Such a language must fulfill requirements inherent to the telephony domain. Finally, it is designed independently of a given platform so as to enable services to be portable across different platforms.

Based on the domain analysis of the telephony and the study of a variety of existing signalling platforms, we have identified the key requirements for a language dedicated to this application area. We are now designing this language. It should provide the programmer with high-level constructs that abstract over low-level details of a SIP platform and underlying protocols. Thanks to its domain-specific nature, it should have semantic restrictions and include domain-specific extensions. These features should enable specific analyses to ensure critical safety and security properties at compile time [1, 2].

Acknowledgment

This work has been partly supported by the *Conseil Régional d'Aquitaine* under contract 20030204003A and Microsoft Research Ltd under contract 2004-364.

References

- [1] C. Consel. *Domain-Specific Program Generation; International Seminar, Dagstuhl Castle*, chapter From A Program Family To A Domain-Specific Language, pages 19–29. Number 3016 in Lecture Notes in Computer Science, State-of-the-Art Survey. Springer-Verlag, 2004.
- [2] C. Consel and L. Réveillère. *Domain-Specific Program Generation; International Seminar, Dagstuhl Castle*, chapter A DSL Paradigm for Domains of Services: A Study of Communication Services, pages 165–179. Num-

ber 3016 in Lecture Notes in Computer Science, State-of-the-Art Survey. Springer-Verlag, 2004.

- [3] L. Dang, C. Jennings, and D. G. Kelly. *Practical VoIP Using Vocal*. O'Reilly, July 2002.
- [4] iptel.org. *SER Developer's guide*, Sept. 2003.
- [5] A. Kristensen. SIP Servlet API 1.0 Specification. Java Specification Request 116, Java Community ProcessSM Program, Feb. 2003.
- [6] F. Latty. Étude de la Qualité de Service dans un environnement distribué : Application au domaine de la Téléphonie sur IP. Master's thesis, University of Bordeaux, June 2004.
- [7] J. Lennox. *Services for Internet Telephony*. PhD thesis, Columbia University, Jan. 2004.
- [8] Microsoft. *Live Communications Server 2003 : Reference and Deployment Guide*, Oct. 2003.
- [9] J. Rosenberg, J. Lennox, and H. Schulzrinne. Programming internet telephony services. *IEEE Internet Computing Magazine*, Mar. 1999.
- [10] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP : Session initiation protocol. Request for Comments 3261, The Internet Engineering Task Force, 1895 Preston White Drive, Suite 100, Reston, VA 20191-5434 – USA, June 2002.
- [11] H. Schulzrinne and J. Lennox. Call processing language framework and requirements. Request for Comments 2824, The Internet Engineering Task Force, May 2000.