

OSIP: An Open Source SIP Architecture

Stefan Foeckel, Matthias Kranz, Jiri Kuthan, Dorgham Sisalem
GMD-Fokus
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany
{foeckel,kranz,kuthan,sisalem}@fokus.gmd.de

Abstract—In this paper, we present an overview of the architecture of OSIP. OSIP is an open source implementation of the session initiation protocol (SIP) currently discussed for IP Telephony and group communication. To allow for easy portability and wider usage this implementation was developed in JAVA. The message parsing part of OSIP was designed to be automatically generated based on the protocol specification. Finally, in order to support various communication scenarios OSIP was designed in a modular manner with open interfaces that allow the development of applications of various complexity using the same core protocol implementation.

I. INTRODUCTION

The rapid growth, increasing bandwidth and the availability of low-cost multimedia end systems has made it possible to use the Internet for multimedia applications ranging from telephony to conferencing, distance learning, media-on-demand and broadcast applications [1]. Due to the prospects of tremendous financial gains IP-based telephony is ever more attracting the attention of service providers and consumers as well. However, due to the entirely different nature of IP-based networks compared to traditional circuit-switched networks new paradigms for signaling a communication request between two end systems are needed.

Due to its simplicity, the session initiation protocol [2] designed within the IETF is ever more gaining in acceptance as the standard protocol for IP Telephony.

To provide for a portable and expandable open source implementation of SIP and hence support and simplify the efforts of other institutions and companies in their investigations and testing of SIP we started some time ago designing and implementing the protocol stack of SIP in JAVA. In addition to providing the protocol functionalities we considered in our design the aspects of portability and possible extension of the language itself as well as the applications using the protocol stack.

In Sec. II of this paper we first present a short overview of the session initiation protocol and related work. Sec. III describes the general architecture of OSIP, the current implementation status as well as the interfaces provided for

realizing different communication scenarios and applications. We conclude the paper in Sec. IV with a look at our further plans and required enhancements to OSIP.

II. BACKGROUND AND RELATED WORK

While the session initiation protocol was favored for IP Telephony by the IETF, H.323 [3] is being propagated by the ITU standardization group and is being used in various conferencing tools such as *netmeeting*¹ and some of the currently popular IP Telephony applications such as *iphone*². The ITU H.323 series of recommendations includes H.245 for control, H.225.0 for connection establishment, H.332 for large conferences, H.450 for supplementary services, H.325 for security and H.246 interoperability with circuit switched services. Already the number of defined protocols indicates the complexity and size of an H.323 compliant implementation. Besides the size, Schulzrinne and Rosenberg describe in [4] various other disadvantages of H.323 compared to SIP such as extensibility, security, scalability and supported services.

The most important SIP operation is that of inviting new participants to a call. A user first obtains an address where the user is to be called and translates this address into an IP address where a server may be found. Once the server's address is found the client can send an invitation message to the server. However, as the server which receives the message is not likely to be the host where the user to be invited is actually located we need to distinguish between different server types that a complete SIP implementation should fulfill [5]:

Proxy: A proxy server receives a request and then forwards it towards the current location of the callee -either directly to the callee or to another server, that might be better informed about the actual location of the callee.

Redirect: A redirect server receives a request and informs the caller of the next hop server. The caller then contacts the next hop server directly.

User agent server: This server resides on the host where the callee is actually located. It is capable of querying the

¹Available from: <http://www.microsoft.com/windows/netmeeting>

²Applications available under: <http://www.vocaltec.com/products/iphone5>

user about what to do with the incoming call, i.e., accept, reject or forward, and sends the response back to the caller. *Register*: To assist the end systems in locating their requested communication partner, SIP supports a further server type called register server. The register server is mainly thought to be a data base containing locations as well as user preferences as indicated by the user agents.

There are currently already different publicly available SIP implementations³.

However, most of the available implementations use hard-wired parsers, are not publicly freely available or are programmed in C or C++. To our knowledge there are no publicly available Java implementations that are shipped with source code, use a parser generator, or are completely free.

III. GENERAL ARCHITECTURE OF OSIP

Based on our previous experience with earlier versions of SIP [6] we identified the following requirements:

Portability: To ease its usage over different platforms JAVA was chosen for implementing OSIP. This is especially helpful for users who execute their signaling services at heterogeneous end-devices.

Extensibility: Due to the continuing changes and developments in the area of IP Telephony, the specifications of SIP is facing continuous changes and enhancements. To reduce the implementation overhead required for taking such developments into account, the message parsing part in OSIP was generated using a compiler-compiler approach based on the abstract specifications of the protocol.

Flexibility: As described in Sec. II a SIP entity can be used to fulfill different tasks such as acting as an end user agent or as a proxy server. While the actual functionalities of those entities differ, the actions of parsing the protocol messages and handling their transport is generic to all of them. Hence, to allow the development of SIP agents with different functionalities, the parsing and networking parts of the SIP stack were separated by a clear interface from the message handling and application parts. This allows for more flexible upgrading of either the generic parts or the message handling instances.

A major goal of the OSIP implementation was to simplify the task of building different kinds of SIP servers such as user agents, proxies and redirect servers. This is to be realized by using a BNF generated parser and combining reusable building blocks.

³Listings of SIP implementations being available or under construction may be found at: <http://www.cs.columbia.edu/~hgs/sip/implementations.html> <http://www.fokus.gmd.de/research/cc/glone/projects/ipt>

As Fig. 1 depicts, the OSIP stack consists of three layers. The first layer constitutes the "core" SIP stack. It contains a number of service-primitives common to all SIP servers including network handling, parsing of SIP messages, management of SIP transactions, retransmission of UDP packets and locating SIP servers. The second layer consists of SIP server modules implementing different specific functions based on the server personality using SIP requests and responses. Finally, the third layer represents the application which is responsible for processing the SIP message body and handling or initiating the actual telephony communication.

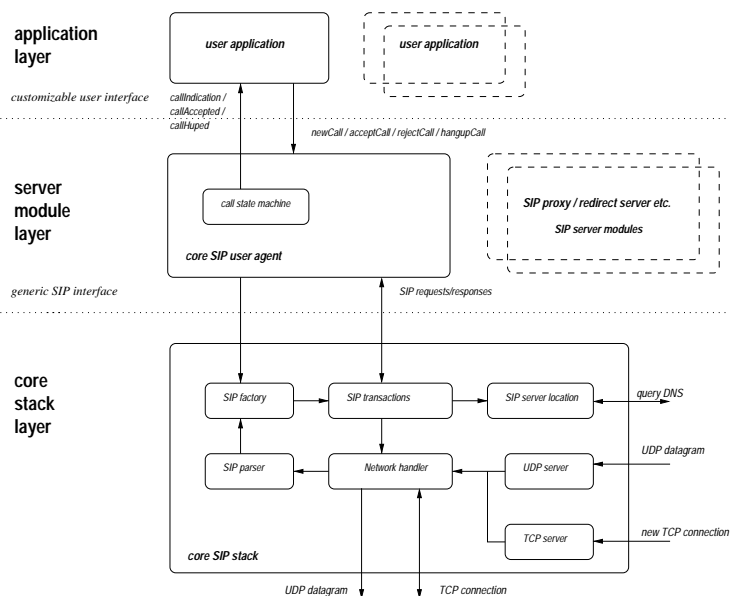


Fig. 1. OSIP stack overview

A. Extensible SIP Parser

The aim of the parser is to process the abstract incoming data stream, perform a lexical analysis and a semantic analysis according to the grammar specified in [2]. The output is an intermediate data structure, which from then on will be the only object for manipulation through a well defined interface.

The goal was to develop a rule based one-pass parser, which would be easy to maintain particularly with regard to new header elements or more changed grammar. Therefore we decided to use a tool called *mparse*⁴, a Java-based compiler-compiler that generates JAVA code out of BNF specifications. Its input language is an extension of Java, offering capabilities such as inheritance, method overloading, encapsulation and more. The parser production spec-

⁴*mparse* is a successor of Sun Microsystems' JavaCC and is available under www.metamata.com

ifications are written within methods and the definition of tokens looks similar to static initializers providing a very clear representation.

After receiving an incoming request or response the parser processes the provided stream and generates an instance of a class called SipMessage. The Sip Message object provides methods for accessing all header fields, the message body or parts of them if necessary. The parser also detects all unknown header fields and puts them in a list, which can be evaluated by the application. While the message body is recognized, it is not processed as this should be done by the application layer. Currently, all header fields defined in [2] are recognized. If any malicious headers or header field contents are detected, the parser will tolerate them as far as possible, but set a failure flag for each so that subsequent processing entities can evaluate them appropriately.

B. The Core Stack Layer

The core stack layer consists of the following main modules:

UDP server: In case UDP was used for transporting SIP messages, the UDP server receives incoming SIP messages on a pre-defined port and forwards them to the network handler module

TCP server: In case TCP was used for transporting SIP messages, the TCP server is responsible for accepting new TCP connections on a pre-defined port, creating a handler object for each new connection and passing them to the network handling module.

Network handler: The handler dispatches incoming UDP packets and manages TCP connections. It creates abstract streams and passes their content to the SIP parser. Additionally, it sends outgoing SIP messages, performs retransmission when using UDP, alerts the assigned SIP transaction objects on timeouts and handles network errors.

SIP factory: The SIP factory provides static methods to create and initialize SIP stack objects and to check their validity. Additionally, in the case of local generated requests it creates new SIP transaction objects.

SIP transactions: This module validates message headers and assigns SIP request/response objects to a SIP transaction. Further, it administers SIP transaction objects and calls the registered server module for further processing of SIP messages. Finally, it passes outgoing messages to the network handler according to the requested transport method. In the case of UDP communication it additionally configures timers for retransmitting lost packets.

SIP server location: The SIP server location module is responsible for locating SIP servers on outgoing requests. Currently, this is only realized by simple DNS address

record queries. However, we are in the process of adding more advanced methods such as DNS SVR record search.

C. Server Module Layer

This layer performs the actual handling of SIP requests and responses, maintains the call state and informs the application about the arrival and processing status of SIP requests.

In contrast to the generic nature of the core stack layer, the server module layer is customized to handle server specific functionalities like proxying or redirection.

The communication with the core SIP layer is achieved by passing SIP requests and responses objects. The server module has the ability to access all parts of a SIP message through these objects. Depending on its main function and internal state the server module decides on the actions to take. For example it can alert a user application, create response-objects to answer a SIP request or create new requests by using the factory methods provided by the core stack.

At this stage only a user agent module is available. This agent is invoked by the core stack in order to process SIP request/response objects. It maintains a state machine, that creates provisional and final responses and calls the registered user application to indicate invitations or call-changes. It provides methods to the user application to create client requests.

D. Application Layer

The application layer is finally responsible for processing the body of the SIP messages, initializing the appropriate media agents for handling the actual multimedia communication and possibly providing an interface to the user. Currently, a compliant SDP parser class [7] is provided for processing the media description in the SIP body message.

To initiate the communication between a user application and the SIP server modules, the application has to register with the SIP server modules using a common interface. Additionally, in order for the server modules to inform the application about the arrival and processing status of SIP requests, the application layer needs to provide for an interface that can be used by the server modules.

IV. SUMMARY AND FUTURE WORK

OSIP is an extensible implementation of the session initiation protocol based on a modular design, rule-based parser and implemented in the JAVA programming language. The current version already supports user agent functionalities and the recognition of all fields supported in the standard [2].

We are currently in the process of enhancing the server module layer with proxy and register functionalities and are considering the aspects of security and authentication required for IP Telephony. Further, the interfaces provided between the layers are being refined to allow for easy binding with call services. Additional features like DNS server record queries will be added.

To allow for high quality communication, we are also studying the aspects and approaches for integrating QoS provisioning concepts with IP Telephony signaling.

To test the efficiency of a JAVA-based implementation compared to C or C++ implementations various tests need to be conducted and evaluated.

REFERENCES

- [1] Henning Schulzrinne, "Re-engineering the telephone system," in *Proc. of IEEE Singapore International Conference on Networks (SICON)*, Singapore, Apr. 1997.
- [2] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments (Proposed Standard) 2543, Internet Engineering Task Force, Mar. 1999.
- [3] International Telecommunication Union, "Packet based multimedia communication systems," Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.
- [4] Jonathan Rosenberg and Henning Schulzrinne, "A comparison of SIP and H.323 for internet telephony," in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Cambridge, England, July 1998.
- [5] Henning Schulzrinne and Jonathan Rosenberg, "Signaling for internet telephony," Technical Report CUCS-005-98, Columbia University, New York, New York, Feb. 1998.
- [6] Dorgham Sisalem and Henning Schulzrinne, "The multimedia internet terminal," *Journal of Telecommunication Systems*, vol. 9, no. 3, pp. 423-444, Sept. 1998.
- [7] M. Handley and V. Jacobson, "SDP: session description protocol," Request for Comments (Proposed Standard) 2327, Internet Engineering Task Force, Apr. 1998.