PROGRAMMING SIP SERVICES – THE SIP APIS

Pavel SEGEČ
Department of InfoComm Networks, Faculty of Management Science and Informatics,
University of Žilina, Univerzitná 8215/1, 010 26 Žilina, Slovak Republic,
tel.: +421 41 5143 323, e-mail: pavel.segec@fri.uniza.sk

ABSTRACT

The Session Initiation Protocol (SIP) is a signalling protocol developed to set up, modify and tear down multimedia sessions such as voice and video calls, game sessions, messages exchange and the like over the Internet Protocol (IP). A few of protocols have been design for it. However, the SIP seems to be the most relevant protocol with the future. That's all thanks to its manifold features. One of the most interesting is the Internet approach for programming new service. There are a couple of options that can be used for creating new SIP services. To create a service using the SIP, we should use SIP baseline protocol mechanisms. Another option is to define extensions to the baseline SIP protocol specification, defining new headers and new methods. Finally, dedicated programming tools for SIP can be used. Examples include a Call Processing Language – SIP CPL, Common Gateway Interface – SIP CGI, SIP Servlets, Java API for Integrated Networks - JAIN APIs. In this paper we present the usability of dedicated tools for creation and control of integrated services over SIP. We analyze different features of dedicated programming tools and provide their comparison.

Keywords: *SIP, services, SIP CGI, SIP CPL, SIP Servlets, JAIN, JAIN SIP API*

1. INTRODUCTION

Technologies for transporting voice over IP based networks (below IP telephony) have become one of the most important technologies, leading the process of network convergence [1]. These technologies bring a new Internet way of service creation opportunities to develop innovative and attractive services for customers. This approach allows to integrate many separated services into a new, converged service environment. Nowadays, there is a number of signalling protocols defined for such new IP multimedia service environments, however, the Session Initiation Protocol (SIP) [2] seems to be the most significant thanks to its manifold features. SIP is offering many forms for programming new integrated services. Using SIP dedicated programming tools such as Call Processing Language – SIP CPL, Common Gateway Interface – SIP CGI, SIP-servlets, Java applets, Java API for Integrated Networks - JAIN APIs, Parlay is one of them. SIP-based services can be programmed either by trusted (such as administrators), or by untrusted (such as end users) users. This model allows creation of services not only by providers of the integrated network infrastructure, but also by the third parties (TP) developers and even by the users themselves; this was not a case in the PSTN.

In this paper we focus on the SIP service creation approach to create integrated services with help of SIP dedicated programming tools.

2. SIP (SESSION INITIATION PROTOCOL)

The Session Initiation Protocol (SIP) is an IP based application protocol that has been developed by IETF [2] as a signalling protocol for multimedia communications established over packet IP network. SIP provides signalling and control functionality for a large range of multimedia communications including voice, data, images, etc. The communication between participants can be unicast or naturally multicast type. Similarly to other signalling protocols, main SIP functions include location of parties, invitation to service sessions, negotiation of session parameters. To accomplish this, SIP uses a small number of text-based signalling messages, which are exchanged between the SIP entities.

Presently the SIP was adopted as a multimedia call control protocol by the Third Generation Partnership Project (3GPP) [3] for its IP Multimedia Subsystem (IMS) and by the European Telecommunications and Standardisation Institute (ETSI) [4] for the Next Generation Network (NGN) architecture and for the fixed mobile converged networks.

2.1. SIP entities

SIP defines some logical functional entities, which all have their specific tasks and allow to establish SIP based communication infrastructure providing their users personal, terminal and service mobility. The entities can be categorised either as SIP endpoints or as SIP servers.

SIP endpoints are User agents (UA)**,** Back-to-back User Agent (B2BUA), SIP gateways**.** The UA represents the main end system, which initiates sessions between two or more UAs. The B2BUA is a SIP device that receives a SIP requests and SIP responses, reformulates them, and then sends them out as new messages. A B2BUA is usually used as a topology hiding element or as a part of signalling gateways. The SIP gateway (GW) is a special kind of the UA, placed at the border of a SIP network and a network that is working with other type of signalling protocol (e.g., H.323) or a network with completely different communication stack (SS7, DSS1).

SIP servers can have a role of a Registrar server, Proxy server or Redirect server. The Registrar server receives registration requests sent by the UA and makes a location binding record of a SIP address (SIP Uniform Resource Identifier (URI)) and an IP address of the UA. The Proxy server receives messages from other entities (UAs), processes them and then forwards them to an appropriate location. It is located along the way between two user

agents. Its main task is message routing and service provisioning. The Redirect server provides an initiator of the call with a list of alternative contact SIP addresses, where the call should be redirected.

The SIP architecture follows a client/server architecture model, which can be characterized as an architecture with higher implementable administrative effort. For this reason, several approaches on how to integrate Peer-2-peer (P2P) mechanisms into SIP without the need to implement SIP server entities have been developed.

SIP is a protocol, in which clients send requests which are answered by a response generated by the servers. The session between UAs consists from the sets of text coded messages. There are two types of SIP messages: Requests (or Methods) and Responses. Requests are generated by clients. Receiving requests usually starts some processing on the server side. A result is send back using some of the Response messages.

3. USAGE OF THE DEDICATED PROGRAMMING TOOLS FOR CREATION OF SIP SERVICES

Creation and implementation of service logic (or just briefly a logic) into the SIP architecture represents the main aspect of programming SIP service. In general, any arbitrary programming language can be used for this purpose. The logic is used to influence and to handle a specific SIP signalling message flow or just to react on special conditions represented by special events, triggered by receiving a specific SIP message or a SIP header value or an argument of a specific message.

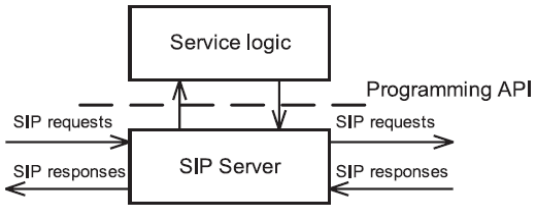

Fig. 1 The basic model of SIP service logic implementation

Service logic can be added to the both kinds of SIP entities. In the case of extending the UA, special problems can emerge that arise from specific implementation conditions (e.g. differences between end platforms, different UA implementations, issues regarding security and trustworthiness, etc.). One of the reasons that can cause problems is that the UA is usually owned by the end user, not by the service provider. On the other hand, implementing a logic into the SIP server entities means that the logic itself can control and manage servers' activities based on specific input criteria, (e.g., callee or caller address, time of day, subject of a call, etc.). The basic model of SIP service logic implementation is provided in Figure 1 [5].

The basic model supposes the extension of SIP server entities by a programmed service logic (application), where the logic is responsible for providing a service with the expected features. The logic communicates with the server through an application programming interface (API). When a specific message comes to the server (an event occurs), the SIP server passes the information to the logic. Based on the received information and potentially on the other input information of the main service received from other sources (configuration of a service, database, directory services, etc.), the logic makes a decision and instructs backward the SIP server about the actions it has to perform.

The model is relatively simple; however, there are some issues that have to be considered. Definition of the trustworthiness between a service application and SIP entities (i.e. SIP server) is an example. The ratio of trustworthiness depends on the fact who is the creator of the service. A creator of a logic (i.e. a service) can be either the owner of the infrastructure (highest trustworthiness, full access, trust user) or a third party service developer or provider (limited trustworthiness, limited access, untrusted user), or the end user itself (limited trustworthiness, limited access, untrusted user).

The logic can be placed either in the SIP server itself, or in a separate, independent system. In the latter case the role of the API interface takes over a specialized protocol – Remote Procedure Call (RPC) mechanism or distributing computing platform (CORBA, DCOM, etc.).

A model that may reuse a functionality of such a service layer (represented by programming interface) allows that a service simple uses the underlying network control and signalling infrastructure and significantly simplifies a process of development and implementation of new communication services. At the same time the model clearly stirs the old strict relations between the process of service development and a network infrastructure. This is allowed by mechanisms that use standardized service developing interfaces. The development of new communication services and applications is becoming simpler (from time, technology as well as economic points of view).

A couple of interfaces between service logic and SIP server have been defined; some of them are derived from the interfaces that are used for the development of web services. These interfaces allow creation of services either by trusted users (SIP CGI, SIP-servlets, Java applets, JAIN APIs, Parlay), or by untrusted users (SIP CPL). Of course, there are also many proprietary APIs, however those provide lower portability of developed services. On the other hand, they often provide integrated solutions that allow better integration and usability of service components of the same company. However, some history events show that if a company producing such proprietary solutions would like to keep their competitiveness, it should put the effort on the implementation of the standardized interfaces.

4. CPL (CALL PROCESSING LANGUAGE)

The CPL (specified by the IETF in [6]) was one of first tools designated to easily support the development of IP Telephony (IPT) services. CPL is not strictly coupled with any of signalling protocols, and this is one of its main advantages. CPL is a programming tool which provides the end users with the possibilities to create their own services. Using the CPL, a user may define activities which impact call handling activities performed by SIP server during call processing.

CPL has been designed as a simple and easily extendible language, still enough flexible and powerful to provide means for flexible development of a wide range of services and its features. CPL is taking into consideration that services are developed and defined by end users, but they run on a network, in SIP providers' servers. Therefore, the CPL language has the performance and security limitations that ensure running the CPL service without the performance or security degradation of a network or a server. The main intention of the CPL is to provide the users with the ability to design, create and implement their own IP telephony services and to disallow the creation of complex, high performance and time consuming service processes. CPL was designed to be also available to semi-trusted users. CPL does not enable users to create invalid or ill-conceived service, who could (through malice or incompetence) attempt to create invalid or ill-conceived service descriptions. For this reason, the creation of program loops, starting up and calling up other processes or programs inside the CPL, are not allowed. It does not define any variables.

CPL has been based on the Extensible Markup Language (XML) that simplifies parsing of the CPL code by syntactic analyzers (i.e. parsers). It is editable, either manually, via the CPL language definition or by means of visual GUIs tools. Knowledge of the CPL language and syntax is required in the both cases, however, manual editing requires the knowledge to be much deeper. Services can be created by users even without the deep CPL knowledge. For this, some kind of easily understandable interface is required (web). Definition of the service (design and creation) made by users corresponds to the creation of the CPL script (manually or be composed through the GUI). The structure of the CPL script corresponds to its behaviour, so a syntactical correctness of the final CPL script can be easily verified by the CPL editor or the SIP server itself, even before including the script to a real service. Usage of the CPL service requires some methods of delivering the script from a user (i.e. creator of the service) to the SIP server. For this purpose, the SIP REGISTER method or the HTTP upload can be used. Other proprietary methods are available, too (for example, upload connections across IP network and placement of the script into a database).

The CPL is not a common programming language (compared to full featured, high level programming languages such as Java, C/C++, etc.), it is more similar to scripting and markup languages (HTML, SGML). It represents a light, simple but still powerful tool designated for creation of IPT services, especially by the end user himself. A CPL script, representing a SIP service, is quite straightforward and relatively easy, especially when available GUI tools are used. The CPL script can ba analyzed and verified quite easily (visually by a user, by a CPL editor, or by a CPL SIP server). The CPL script natively cannot perform performance and security risky operations. It is suitable for IPT service providers to extend their IPT service portfolio; moreover, it allows end users to develop their services.

CPL is suitable when the end user may influence call control mechanisms performed during call processing in the SIP server only. Using CPL does not enable to create more complex, feature rich services, especially those which integrate different network services and features (localization, messaging, web, mail, chat, etc.). CPL neither allows the cooperation with other programming languages. Moreover, it is relatively complicated to integrate it with other software components (e.g. web services, mail services). Finally, CPL does not provide users with the features that would enable the creation of interactive services. To overcome the shortcomings mentioned here, the extension of the CPL language would be required in future.

5. SIP COMMON GATEWAY INTERFACE (SIP CGI)

The design of the SIP protocol was inspired by some of the IETF protocols, especially, by the Hyper Text Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP). This is why SIP is quite similar to both of them. For example, it has the same syntax, semantics, client/server nature as the HTTP. As a consequence, it is natural to adopt some of the HTTP service creation approaches and tools for SIP environments. This can certainly lead to acceleration of the SIP service creation process and usability. The Common Gateway Interface (CGI) represents one of the tools that are very common for web environments. The HTTP CGI and the SIP CGI are similar in some common issues, however, the technical specification is different, especially due to operational differences between HTTP and SIP servers (for example, SIP can be statefull, HTTP does not need to be statefull; SIP supports forking, HTTP does not; SIP supports client's registration, HTTP does not).

The SIP CGI (specified by the IETF in [7]) is not a programming language. It is rather the application interface implemented into SIP servers. When specific SIP events occurre, SIP servers are allowed to start up and communicate with an arbitrary program with implemented service logic. The main service logic application (the program) may be created in an arbitrary programming language or in a scripting language (Perl, Python etc.). The induction and execution of the SIP CGI scripts representing the SIP service is fully managed by the SIP server (if a specific event occurs, for example SIP message type, parameters, address values, etc.). SIP server executes the SIP CGI script whenever a new process inside of the operating system (OS) resources of the SIP server occurs. To process SIP events correctly and to generate the following activities, the communication between the SIP CGI script and the SIP server is needed. For the information exchange, the OS input/output standard system is used together with SIP CGI environment variables called "metavariables". The metavariables are the environment variables of an OS where the SIP server is running.

The information is put into a SIP CGI script by means of metavariables. Their values have been set up by the SIP server before the SIP CGI script is called. Using metavariables almost the whole SIP message header is provided to the CGI script (with except of some sensitive authorization information for example). The rest of the SIP message (the SIP body) is provided by the system input (stdin). The script executes required processing and then, using the system output (stdout), it informs the SIP

server about actions that are required (for example redirection, message construction, rewriting, etc). The output is done by means of SIP CGI messages. One output consists from one or more SIP CGI messages. Ones the output is generated the SIP CGI script is terminated.

In general, SIP servers implementations are statefull. That means that a SIP server has to keep the list of all running SIP transactions. The message processing during the transaction depends on previous message exchange of the transaction. The HTTP CGI does not support this feature. The statefull support in the SIP CGI is maintained by a special feature called a token (represented as the SIP CGI cookies). The token allows the CGI script to come back to the same transaction. When the CGI script is called again for the same transaction, this token is passed back to the CGI script [7].

Using SIP CGI for the service creation provides a couple of advantages [8][9]. All SIP message headers are available to the SIP CGI script. SIP CGI may generate all parts of a SIP response message, headers and the body. The SIP CGI allows reusing the written HTTP CGI codes and there are also similarities with the HTTP CGI addressing wide web developer base. The SIP CGI interface allows the execution of the SIP CGI scripts coded in an arbitrary programming language. The script allows an arbitrary operation and provides access to external resources. Generally, the SIP CGI provides unlimited service creation possibilities.

On the other side, the SIP CGI has some weak points. It is not fully platform independent; some scripts have to be compiled into executable binaries (sometimes platform dependent). The SIP CGI may be an arbitrary program, running over OS resources, where the functionality and behaviour of the script is not able to be fully verified before its execution. Moreover, each calling and initialisation of the SIP CGI script invokes a completely new process running over the OS resources. As the script may be a complex, vast program, especially if it is poorly designed, the system resources may be quickly exhausted.

Scripts have usually an access to the system resources; this can cause problems in the area of system security and integrity. Therefore, the usage of the SIP CGI requires the implementations of stronger system security protection mechanisms from a server provider side. Its adoption may rapidly restrict the CGI possibilities for service creation, especially from third party developers and users, which can harder achieve the code verification and predictability.

6. SIP SERVLET API

Similarly to the SIP CGI, a SIP Servlet API [10][11] provides another example of reusing a web technology used for developing web applications (HTTP Servlet API) for the SIP service technology. The SIP Servlet API is a Java API interface which extends the functions of a generic Java Servlet API. The HTTP Servlet API has been also derived from generic Java Servlets API. The SIP Servlet API allows SIP servers and UAs to create and implement SIP communication services by means of Java Servlets technology. The purpose of the SIP Servlet API is to provide a standardized platform for developing and delivering SIP based services. The SIP Servlet API defines interfaces, abstractions and mechanisms, which allow creation of the service logic (Fig. 2).

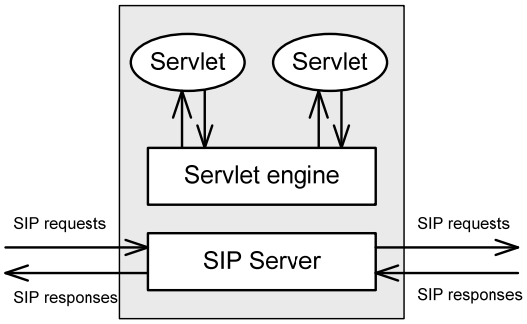

Fig. 2 Basic SIP Servlet model

The SIP servlet is a Java application component which is managed by a SIP server (through its servlet container) and which performs SIP signalling. The service logic is provided as a servlet application consisting from one or more SIP servlets running a specific service function. On a SIP server, more services are usually realized. It is a SIP servlet (also called server engine) container that takes a decision on which applications (servlet) to invoke and in which order. The process is controlled by an occurrence of a specific SIP event (type of message, some header field value and so on). The container also manages the servlets lifecycle. The SIP servlet container is a part of an application server that can be built into a SIP server, or installed as an add-on component [11]. Once the SIP servlet is invoked, it interacts with SIP UAs by exchanging request and response messages through the servlet container over defined network listen points. The servlet is initialised as a process of the Java Virtual Machine (JVM). Until its lifecycle is finished (by a servlet container), servlets functions are used each time the service is called. Communications between the SIP servlet and the SIP server (its servlet container) is done by means of Java objects. Java objects are used for modelling SIP messages. Using objects enables that all parts of received SIP messages are provided to the SIP servlet. The Servlet application then performs all actions required to achieve a service behaviour and instructs the SIP servlet container to perform the next actions (for example, proxying message, fork message, finish transaction, etc.). The SIP servlet may behave as an UAC (generation of SIP requests) or as a SIP Server (serving SIP messages). The SIP servlet itself does not support statefull behaviour, this feature is realized at a servlet container level. The Servlet container supports both types of SIP processing, stateless as well as statefull.

The SIP servlet container is a part of a SIP server and provides networking functions of the SIP server to SIP servlets. The SIP servlet container manages network listen points (the combination of the transport protocol, the port number and the IP address).It receives and transmits the SIP traffic through them. In this way the SIP servlet may communicate through a network using the SIP. Servlet container maintains a lifecycle of SIP servlets. Based on servlet mapping, the container decides which servlet will be invoked and in which order. The rules are specified by the deployment descriptor (DTD). It is the SIP servlet

container which provides servlet functionality to the SIP server. Like that the server is able to maintain services created by the servlet technology.

SIP servlets technology was developed as an alternative to the SIP CGI technology. The main target was to avoid of SIP CGI limitations, the platform dependability of the SIP CGI scripts and a weak performance scalability of the SIP CGI. These restrictions were removed by an exclusive usage of Java development environment to create SIP Servlet API (and then services). Thus, the platform independence of SIP servlet services has been achieved. Using Java brings other advantages coming from the Java platform itself, i.e. the security, flexibility and many existing APIs (JDBC for database connectivity, JMF for media processing, JavaMail, JNDI for directory processing and so on).

From the performance scalability point of view, the SIP servlet accesses the system resources more effectively than the SIP CGI. The first service call initiates and starts SIP servlets. They remain active and their functions may be reused whenever the service is called again. The Servlet remains active until its lifecycle is finished by the container. This is unlike to the SIP CGI, in which each service call starts a completely new process.

New services can integrate multiple types of communications, such as telephony, web, mail, instant messaging, etc. To support the expectations there are different types of SIP Servlet container implementations. For example, a special type of converged SIP container that enables the deployment of applications that use SIP, HTTP Servlet API and other Java EE components like EJBs, webservices, messaging, etc. may be used. There are three different methods of SIP Servlet implementations [11]:

- Standalone SIP Servlet container, which provides SIP interface and hosts SIP Servlets only as the applications.
- SIP Servlets and HTTP Servlets container, in which SIP and HTTP servlets share the same context. This implementation is suitable for the creation of integrated web and SIP services.
- SIP and Java EE Convergence container. This implementation facilitates the use of SIP Servlet technology in conjunction with a more complex Java EE deployment model.

Thanks to Java, the SIP Servlet technology allows creation of SIP services that use the same principles that are used for the web services development or development of other Java applications. Thus, the SIP Servlets may become very interesting platform for a wide community of Java web developers.

SIP Servlets technology is mainly focused on the area of enterprise application servers and JEE developing platform. Here, it can bring some merits already known and used in development of converged applications and services.

Even though the technology offers many interesting features, it is very complex and wholly based on Java. This can be objectively considered as weak points [9].

7. JAVA API FOR INTEGRATED NETWORKS (JAIN)

JAIN is the technology, which provides a set of standardized, integrated programming interfaces based on Java. It is used for the rapid development of Java based next generation communications products and services. JAIN provides telecommunications service developers with suitable developing environments for the creation of services which are independent from underlying networking technologies. JAIN concept addresses network convergence (services run over IP and telecommunication networks) as well as the service portability (write once, run anywhere). With respect to the controlled and secured direct access to network resources and devices, it allows to create services to anyone, either trusted or untrusted developing sites and users [12] [13].

To achieve universality, JAIN strictly separates the layer of service creation from the layer which is dependent on network and transport infrastructure. For this purpose the JAIN architecture specifies two layers and interfaces:

1. **Protocol layer:** The layer contents standardized JAVA APIs for the IP, switched circuit networks, wired and wireless signalling protocols. The APIs are generally classified into two groups: the SS7 APIs (TCAP, ISUP, INAP, MAP protocol support) and the IP APIs (MGCP, MEGACO, H.323 and SIP protocol support). The APIs provides the same interface into different industry signalling protocols and so they provide a high level of portability.
2. **Application layer:** The layer provides APIs for creating and providing communications services. The interfaces cover several areas such as service creation (JSCE – JAIN Service Creation Environments), session control (JCC - Java Call Control), service execution (JSLEE – JAIN Service Logic Execution Environment) and security of service providers towards untrusted services (JSPA – JAIN Service Provider APIs). The application layers provide a unified call model which can be in general reused for all protocol interfaces.

The JAIN technology is based on a technology of Java beans components, in which individual components may be added, removed, extended, shared and redistributed within a system or systems. Such approach allows a simplified and flexible addition, update and removal of new services and its features. JAIN APIs are intended to be used for developing new components and their integration into services and applications. A service may be created directly by the use of an arbitrary protocol, JCC or JSPA APIs or with the reuse of different Java development platforms such as JDBC, JNDI and others.

JAIN technology was formerly intended as a technology for trusted sites service development. After implementing security and authentication rules processed by PARLAY and taken into JAIN (JSPA), it has been enabled as a tool for service development by third party service developers. Third party services can run outside of a provider network, but they completely depend on services of the operator's network.

7.1. JAIN SIP API

The JAIN SIP API [14] is a low level, standardized Java interface which provides access to the SIP protocol, its dialog and transaction mechanisms. The JAIN SIP API allows creation of SIP applications (UA, proxy and etc.) or SIP communication services providing them the interfaces and objects needed for sending and receiving SIP messages by means of native SIP environments implemented into SIP clients and servers. The JAIN SIP enables transaction stateless, transaction statefull and dialog statefull control of the SIP protocol. Services can be developed directly using the JAIN SIP API (the APIs is used as an independent standalone interface to the SIP functionalities) or using higher layers of the JAIN technology (JCC, JSPA). The JAIN SIP API can be used in following ways:

- As the JSE development platform. This allows the creation of independent UAs, SIP server entities, applications and services.
- As a base implementation of the SIP protocol for the SIP servlet container allowing service creation and development inside of the SIP Servlet environment as described above.
- As a base implementation of the SIP protocol for Enterprise JavaBeans (EJB) environments.

The JAIN SIP API architecture has the following basic entities: JAIN SIP Stack, JAIN SIP Listener and JAIN SIP Provider [8] [12]. SIP Listener communicates with SIP Provider by the use of SIP events and SIP messages (Fig. 3).

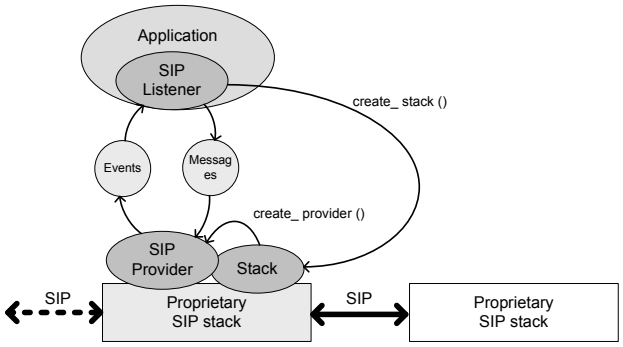

Fig. 3 JAIN SIP API architecture

The JAIN SIP Stack entity enables creation and management of the SIP Provider entities. The SIP Stack is associated with an IP address, a port number and other configuration parameters (outbound proxy, stack name, router path, extension methods, and retransmission filter). The JAIN SIP Provider entity manages the SIP message exchange with a native SIP protocol layer. The entity is specific for each proprietary corporate implementation and therefore the communications between the SIP Provider and the SIP Proprietary Stack follow the specific proprietary implementation rules. On the other side, against the SIP Listener entities, the SIP Provider provides a generic API through which the SIP Listener may access to the services and the functionalities of the proprietary corporate SIP stack implementation (for sending SIP request and SIP response messages statefully). SIP provider's task is to map a generic API to the proprietary SIP stack implementation. The JAIN SIP Listener entity (application) uses the services provided by a SIP Provider entity. These two entities communicate by means of SIP Messages and SIP Events. In order to be able to communicate, SIP Listener has to be registered with the SIP Provider (to receive SIP Events). The SIP Listener entity has to be portable and able to register with different SIP Providers implementations. The SIP Listener can be registered only with one SIP Provider entity. Once the entity does not wish to receive SIP Events, it has to unregister. There is one SIP Listener per SIP Stack only. The SIP Listener provides the interface to the application. The applications receive messages from the stack as the SIP Events via the SIP Listener. Processing SIP request's can be either statefull (managed transaction) or stateless (dependent on application logic). JAIN SIP Events encapsulate SIP messages (SIP requests and SIP responses) received from the network by the SIP Provider. Then they are passed to the SIP Listener (one way). JAIN SIP Messages are used for communication in the opposite direction, from the SIP Listener to the SIP Provider.

JAIN SIP defines different classes with respective responsibilities. They allow creating SIP requests, SIP responses and SIP headers. JAIN SIP also allows a simple JAIN SIP extension for new messages and headers that have not been yet defined or supported by the present JAIN SIP specification.

JAIN SIP API is another standardized Java interface allowing creation of SIP communication services and applications. Similarly to SIP Servlets API, it reuses a Java platform. Like the SIP Servlets, the JAIN SIP API allows the reuse of other Java programming interfaces, such as JDBC, JNID, etc. The JAIN SIP API is usually considered as a lower level API, which does not hide the SIP protocol implementation complexity and provides higher level of access and control of SIP protocol features. This is why creation of the JAIN SIP service has more requirements on a developer of a service in sense of more detailed JAIN SIP knowledge and understanding the SIP protocol mechanisms. This feature can be understood as an advantage but also as a disadvantage.

As a low level API, the JAIN SIP can be used to create a basic SIP Servlet container implementation allowing operation of a container over different proprietary SIP stack implementations. Inside the JAIN SIP, API mechanisms supporting the SIP transaction and SIP dialog model management are implemented.

When service creation is realized directly over JAIN SIP API, trusted users are authorised to use it only. Service development for untrusted users is allowed only when JSPA API is used. Therefore, the all JAIN stack has to be implemented.

JAIN SIP technology offers a great portability of applications and services between different JAIN SIP products. This feature, however, will be probably used more when more SIP JAIN product are available.

8. CONCLUSIONS

SIP protocol is one of the most important protocols for IP telephony. One reason of his successes is the way how it changes the way the telephony services will be

developed and provided. In this contribution we focused on technologies and APIs usable for creation of the SIP IP telephony services. Each technology is shortly described and features analysis is provided.

Described technologies are widely expanded and used, but there is also an another group of tools and APIs with influence on IP telephony like SIP Lite, SIP for J2ME, Parlay, SOAP which will grow on its importance and with which we should count.

ACKNOWLEDGMENTS

This work was supported by the Agency of the Slovak Ministry of Education for the Structural Funds of the EU, under project Itms:26220120007.

REFERENCES

[1] KLIMO, M. et al.: Teória IP telefónie (Theory of IP telephony), University of Žilina, 2009. p. 379, AH 25, 08, VH 25,74, ISBN 978-80-8070-915-0.

[2] ROSENBERG, J. – SCHULZRINNE, H. – CAMARILLO, G. – JOHNSTON, A. – PETERSON, J. – SPARKS, R. – HANDLEY, M. – SCHOOLER, E.: SIP: Session Initiation Protocol, RFC 3261, July 2002.

[3] http://www.3gpp.org

[4] http://portal.etsi.org/Portal_Common/home.asp

[5] ROSENBERG, J. –LENNOX, J. – SCHULZRINNE, H.: Programming Internet Telephony Services, IEEE Network Magazine, June 1999.

[6] LENNOX, J. – SCHULZRINNE, H.: Call Processing Language (CPL): A Language for User Control of Internet Telephony Services, RFC 3880, October 2004.

[7] LENNOX, J.– SCHULZRINNE, H.– ROSENBERG, J.: Common Gateway Interface for SIP, RFC 3050, January 2001.

[8] DI CAPRIO, G. – BERTIN, E. – GOIX, L. W. – IANCU, B. – KOVÁČIKOVÁ, T. – SEGEČ, P. – KUTHAN, J. – LETOUZEY, M. – MORALIS, A. – NIEMINNEN, P. – POIRIER, CH.– PROGGOURIS, N. – SISALEM, D. – VAARNS, E. – VALI, D.: Next-Gen open Service Solutions over IP (N-GOSSIP) - Service Development Through SIP, EURESCOM P1111, Heidelberg, Germany, July 2002.

[9] ANDREETTO, A. et al.: Next Generation Networks: the service offering standpoint, EURESCOM, Heidelberg, Germany, June 2001.

[10] KRISTENSEN, A.: SIP Servlet API Version 1.0, February 2003.

[11] JSR 289 Expert Group, JSR-000289 SIP Servlet 1.1 Final Release, 2008.

[12] JEPSEN, T. C. – ANJUM, F. – RAJ BHAT, R. – JAIN, R. – TAIT, D.: Java in Telecommunications: Solutions for Next Generation Network, Willey Computer Publishing, New York, August 2001, ISBN: 0-471-49826-2.

[13] MUELLER, M.: APIs and Protocols for Convergent Network Services, McGraw-Hill, New York, 2002, ISBN 0-07-138880-X.

[14] JSR-000032, JAIN SIP API Specification, Maintenance Release, 2006, http://jcp.org/aboutJava/communityprocess/mrel/jsr032/index.html

Received August 24, 2010, accepted November 5, 2010

BIOGRAPHY

Pavel Segeč was born on 20.7.1973. He received his diploma in the Information and Management Systems at the University of Transport and Communications (UTC) Žilina in 1996 and PhD degree in transport and communication technology at the University of Žilina in 2005. His thesis title was "Methodology of communication services design over SIP protocol". He is working as a teacher at the Department of InfoComm Networks of ŽU. In 2002 he received the Jozef Murgaš award for year 2001 as the appreciation of his research work and in 2003 he was the team member of the research group that got the Werner von Siemens Excellence Award 2003 for the research in the field of IP telephony.