https://doi.org/10.26552/com.C.2008.4.19-22

Pavel Segec – Tatiana Kovacikova *

CALL PROCESSING LANGUAGE (CPL) – A TOOL FOR CREATION OF INTERNET TELEPHONY SERVICES BY THE END USER

Creation and programming new services are considered as crucial for the Internet telephony (IPT). A number of protocols have been defined for IPT, however, one of them – the Session Initiation Protocol (SIP) seems to be the most relevant thanks to its manifold features. The SIP offers many forms that can be used for programming new IPT services. One of them is to use the SIP baseline protocol mechanisms, the other – to define extensions to the baseline SIP protocol specification (defining new headers, new methods). Finally, the dedicated programming tools such as a Call Processing Language – SIP CPL, Common Gateway Interface – SIP CGI, SIP-servlets, Java applets, Java API for Integrated Networks - JAIN APIs, Parlay can be used for creation of new IPT services. In this paper we focus on one of the SIP IPT features allowing creation and control of IPT services by the end user himself – CPL (Call Processing Language).

1. Introduction

Internet telephony (IPT) is the future; the next generation of today's telephony. There is no doubt. The technology is mature enough and its implementation is rapidly growing around the world, blurring borders between Telco Providers and Internet Service Providers (ISP). On the one hand, IPT offers services very similar to services available in current Public Switched Networks (PSTN), on the other hand it offers many new features, which should be one of the key drivers of IPT. Wide opportunities for the creation of new, customer attractive services, based on the integration of telecommunication and data worlds, are undoubtedly among them. Controlling the process of service creation and programmability of services are crucial issues. Nowadays, there is a number of protocols defined for IPT, however, one of them – Session Initiation Protocol (SIP) [1], seems to be the most relevant thanks to its manifold features. The SIP is offering many forms that can be used for programming new IPT services. One of them is to use the SIP baseline protocol mechanisms, the other – to define extensions to the baseline SIP protocol specification (defining new headers, new methods). Finally, the 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 can be used for creation of new IPT services. It should be noted that 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 IPT network infrastructure, but also by third parties developers and the users themselves; this was not a case in the PSTN.

In this paper we focus on one of the SIP IPT features which allow creation and control of IPT services by the end user himself. For this purpose a special programming language or tool – Call Processing Language (CPL) [2] – was developed for SIP. The CPL is not a tool for definition of new integrated services that combine telephony services with data services (email, instant messaging, presence etc). This is not its target. The CPL is a programming tool that provides the end user with the possibility to design and implement his own services. In other words, the user may define activities that influence a processing function of a SIP IPT server during call processing. End users can create and modify their own services directly, either manually via the CPL language definition, or by means of visual Graphic User Interfaces (GUIs) tools. In the latter case, the deep knowledge of the CPL language is even not required.

2. SIP (Session Initiation Protocol)

The SIP [1] is a signalling protocol developed to set up, modify and tear down multimedia sessions over the Internet. The SIP presents the Internet approach to voice and video communication over the Internet Protocol. The SIP provides signalling and control functionality for a large range of multimedia communications. The main functions are location of parties, invitation to service sessions, and negotiation of session parameters. To accomplish this, the SIP uses a small number of text-based messages, which are exchanged between the SIP peer entities, i.e., SIP User Agents. The SIP User Agents may operate as a client or a server depending on the role in a particular call. Messages can traverse through the SIP network entities like proxy servers or redirect servers that are used for support functions such as address resolution, routing calls to other entities, etc.

The SIP is a protocol that was designed to work hand in hand with other core Internet protocols. Many functions in a SIP-based

* **Pavel Segec, Tatiana Kovacikova**
Department of InfoComm Networks, Faculty of Management Science and Informatics, University of Zilina, Slovakia,
E-mail: Pavel.Segec@fri.uniza.sk

network rely on complementary protocols, for example, a Session Description Protocol for session definition, a Domain Name Service for addressing, a Real Time Transport Protocol (RTP) for media delivery etc. The SIP itself defines the initiation of a session only. The session itself is then described in two levels. The SIP itself contains the parties' addresses and protocol processing features, but the description of the media streams exchanged between the parties of a session is defined by another protocol. The Session Description Protocol (SDP) [3] is used for it. The SDP is not a protocol in the right sense, but rather a structured, text-based media description format that can be carried in the SIP message body. The message body is transparent to the SIP, thus, any other session description can be carried (e.g., a weblink, an e-mail address etc.). From this point of view, SIP sessions are not limited to telephony calls or conferences only, but they can also include information retrieval or broadcast sessions depending on the session description.

3. SIP services created by dedicated programming tools

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

Service logic can be added into each of the SIP entities (User Agent – UA, Proxy, Registrar and Redirect). In the case of extending UA, in opposite to SIP server entities, special kinds of problems are emerging that arise from specific implementation conditions (e.g. differences of end platforms, differences of UAs, issues regarding security and trustworthiness, etc.). The reason is that the UA is usually owned by the end user, not by the service provider. By implementing a logic into the SIP server entities, the logic controls and manages servers' activities based on specific input criteria, (e.g. callee or caller address, time of day, subject of a call and etc.). The logic may also instruct a server to route signalling, to create a new SIP request or a response message, or to add new headers. The basic model of implementing SIP service logic is provided in Fig. 1 [4].

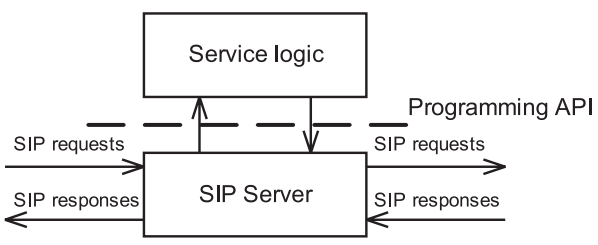

Fig. 1 The basic model of SIP service logic implementation.

The basic model supposes the extension of SIP server entities by a programmed service logic (application), where the logic is responsible for providing the main service with expected features. The logic and the server communicate through an application programming interface (API). In the moment when a specific message comes to the server (a specific event occurs), the SIP server passes the information to the logic. 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.), makes a decision and instructs backward the SIP server about the actions it has to perform.

The model mentioned above is relatively simple; however, there are some issues that should be considered. One of them is the definition of the factor of safety or trustworthiness between a service application (i.e. service logic) and SIP entities (i.e. SIP server). The ratio of trustworthiness depends on the fact - who the creator of the service is. As mentioned earlier, a creator of a logic (i.e. a service) in the IPT can be either the owner of the infrastructure or a communication IPT service (highest trustworthiness, full access, trust user), or a third IPT service developer and provider (limited trustworthiness, limited access, untrusted user), or the end user (limited trustworthiness, limited access, untrusted user).

From the logic location point of view, the logic can be placed either in the SIP server itself, or in a separate, independent system. In the latter case some issues related to communications of those two independent systems have to be solved. In this case the role of the API interface takes over some specialized protocols – Remote Procedure Call (RPC) mechanisms or distributing computing platforms (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).

Development, portability and extendibility of new services is provided by standardized forms of APIs for IPT. Nowadays, a couple interfaces are defined, a part of them is derived from the interfaces that are used for development of web services. Each of them uses its own approach for service development. These interfaces allow creation of services to trusted users (SIP CGI, SIP-servlets, Java applets, JAIN APIs, Parlay) as well as to untrusted users (SIP CPL). Of course, there are also many proprietary APIs, however those APIs provide smaller 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 cannot avoid the demand to implement such standardized interfaces. .

4. CPL (Call Processing Language)

The CPL (specified by the IETF in [2]) was one of first tools designated to support easily the development of IPT services. One of its main advantage is a fact that it is not strictly coupled with any of signalling protocols. The CPL is a programming tool which provides the end users with possibilities to create their own services, which is something unprecedented in old telephony systems. Using the CPL, a user may define activities which influence call handling activities performed by SIP IPT server during call processing.

The CPL is 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. The CPL is taking into consideration that CPL 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 on providers' servers 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. The CPL was designed to enable service providers to feel comfortable while making it available to semi-trusted users who could (through malice or incompetence) attempt to create invalid or ill-conceived service descriptions. For this reason, inside the CPL, the creation of program loops, starting up and calling up other processes or programs are not allowed. The CPL has neither defined any variables. 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, XML, and SGML).

The CPL has been based on the Extensible Markup Language (XML) that simplifies parsing of the CPL code by syntactic analyzers (i.e. parsers). The CPL is easily editable, either manually, via the CPL language definition (knowledge of CPL language and syntax is required) or through visual GUIs tools. In a latter case the deep knowledge of the CPL language is not required. It should be noted that the process of service creation is also available to users without the CPL knowledge. Definition of the service (design and creation) made by users corresponds to the creation of the CPL script (written manually or composed through the GUI). The structure of the CPL script corresponds relatively precisely to its behavior, so a syntactical rightness of the final CPL script can be easily verified by the CPL editor or the SIP server itself, even sooner than the script is put on 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 should be used. Other proprietary methods are available, too (upload connections across IP network and placing script into a database).

A CPL script contains service ancillary information of the script and the information describing call processing actions performed during a call processing [2]. Ancillary information is the information that is necessary for a server to correctly process a script, but that does not directly describe any operations or decisions. Currently, no ancillary information is defined by the basic specification, but the special section is reserved for the future use by extensions. Information describing call processing actions is structured as a tree that describes the operations and decisions required to be performed by a SIP server in a case of a call set-up event. Call processing actions are divided in two types, top-level actions and subactions. Top-level actions are the actions that are triggered by signalling events that arise in the SIP server. Two top-level actions are nowadays defined by [2]. The first one is the incoming action, which is performed when a SIP INVITE message arrives on the server and whose destination address is the owner of the CPL script. The second one is, the outgoing action, which is performed when a SIP INVITE message arrives to that server whose originator is the owner of the CPL script. Subactions are the actions which can be called from other actions. CPL forbids subactions from being called recursively.

Call processing actions are abstractly described as a collection of nodes, each of which describes operations or decisions that should be performed or a choice that should be made. A node may have several parameters, which specify the precise behavior of the node. A node usually has outputs which depend on the result of the condition or the action of a node. Nodes are arranged in a directed acyclic graph, starting in a single root node. Outputs of the nodes are connected to additional nodes. When a CPL script is invoked and running the action or condition described by the node is performed, based on the result, the server follows one of the node's outputs and that action or condition is performed. This process continues until a node with no specified outputs is reached. Because the graph is acyclic, this will occur after a bounded and predictable number of nodes are visited.

Call processing actions consist of a hierarchical tree of nodes and its outputs which are described by XML tag. The CPL specification defines four categories of CPL nodes. Switches which represent choices a CPL script can make (decision made on address, time, string and priority). *Location modifiers* which control sets of location information. *Signalling operations* which cause events in the signalling protocol (e.g. proxy, reditect, route, reject). Last category is *non-signalling operations* that trigger behavior which does not effect the underlying signalling protocol (e.g. reading e-mail).

5. CPL service examples

Call Redirect Unconditional

This CPL script uploaded by some SIP user will unconditionally redirect all his call on the new user, identified with SIP address

sip: palo@sip.uniza.sk. Into the log file named “test” the following message will be put “Call redirect on: palo@sip.uniza.sk”

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE cpl SYSTEM "file:/home/cpl-01.dtd">
<cpl>
    <incoming>
        <log comment=" Call redirect on:
palo@sip.uniza.sk" name="test">
            <location url="sip: palo@sip.uniza.sk">
                <redirect />
            </location>
        </log>
</incoming>
</cpl>
```

Fig. 2 Call Redirect Unconditional

Call screening

The following CPL script represents a SIP service where all call from anonymous callers, i.e. those who do not provide their name (Fig. 3).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE cpl SYSTEM "file:/home/cpl-02.dtd">
<cpl>
    <incoming>
        <address-switch field="origin" subfield="user">
            <address is="anonymous">
                <reject status="reject" reason="I don't
                accept anonymous calls" />
            </address>
        </address-switch>
    </incoming>
</cpl>
```

Fig. 3 Call screening

6. Conclusions

The CPL 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 IPT service, is quite straightforward and relatively easy, especially with the use of available GUI tools. The CPL script is simple enough to analyze and verify its correctness (visually by a user, by a CPL editor, by a CPL SIP server). The CPL script natively cannot perform performance and security risky operations. On the one hand, it is suitable for IPT service providers to extend their IPT service portfolio; on the other hand, it allows end users to develop their services. These are the main CPL advantages.

In order to analyze CPL weaknesses, it is important to mention that the CPL is suitable for service creation, where the end user may influence only call control mechanisms performed during call processing in the SIP server. Using CPL is not able to create more complex, feature rich services, especially those which integrate different network services and features (localization, messaging, web, mail, chat, etc.). The CPL neither allows the cooperation with other programming languages. Apart from it, it is relatively complicated to integrate other software components (e.g. web services, mail services) into the CPL. The CPL does not provide users with the features that could enable creation of interactive services. In future, these limitations can be remedied by extending the CPL.

References

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

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

[3] HANDLEY, J.: SDP: *Session Description Protocol,* RFC 2327, April, 1998

[4] ROSENBERG, J., LENNOX, J., SCHULZRINNE, H.: *Programming Internet Telephony Services,* IEEE network magazine, June 1999