Programmable End System Services Using SIP

Xiaotao Wu and Henning Schulzrinne
{xiaotaow,hgs}@cs.columbia.edu
Department of Computer Science, Columbia University, New York

***Abstract*—**
In Internet telephony, end systems can take a much larger role in providing services than in traditional telephony systems. We analyze the importance of end system services and describe the services and the Service Logic Execution Environment (SLEE) implemented in our SIP user agent, SIPC. Since we believe that end system services differ in their requirements from network services, we define a new service creation scripting language called Endpoint Service Markup Language (ESML). Compared with other service creation languages, ESML is extensible, can be easily understood by non-programmers and contains commands and events for direct user interaction and the control of media applications.

I. INTRODUCTION

One of the key promises of Internet telephony lies in the ability of developing and deploying innovative and lucrative services rapidly and efficiently. Internet telephony services are not limited to those performed in servers operated by service providers; end systems can play a much larger role in providing services than in traditional telephone networks. In Internet telephony systems, end systems could be PCs or embedded SIP (Ethernet) phones with CPU and memory. End systems can execute programs to perform call control and other telecommunication services. For example, our SIP user agent, SIPC, can execute SIP CGI [1] programs or Call Processing Language (CPL) [2] [3] scripts to automatically handle SIP requests and responses. The Pingtel Xpressa SIP phones allow users to upload Java class files to perform services such as caller-specific ring tones.

We define end system services as the services that are performed in IP telephony end systems such as PCs, PDAs, Ethernet phones or Internet-connected appliances and network services as the services that are be performed in network servers such as SIP [4] proxy servers.

Unlike network services, end system services can directly control media applications and interact with users. This allows them to completely automate all aspects of a call. For example, only an end system service can automatically accept a call based on address information.

Enabling end system services not only provides additional user convenience, but also encourages service innovation. In general, while it is difficult for subscribers to modify network services that are owned by carriers, they can install and modify services on end systems they own.

The development of end system services depends on the underlying protocols used for call signaling. SIP promotes end system services because it allows end-to-end operations: two SIP user agents (UAs) can talk to each other directly; service related information is explicitly contained in SIP header fields, making it possible for end system to perform services; the simplicity of SIP also makes it easier to develop services in end systems since end systems often have limited computational capabilities. In Section II, we show the services and Service Logic Execution Environment (SLEE) provided in our SIP user agent, SIPC.

Since traditional service creation methods address the needs of carriers with trained personnel, we defined a new XML-based script language called Endpoint Service Markup Language (ESML) specifically for end system service creation (Section III). We then compare it with other languages.

II. END SYSTEM SERVICES AND SERVICE LOGIC EXECUTION ENVIRONMENT IN SIPC

SIPC is a SIP user agent developed at Columbia University. It can be used for Internet telephony calls, multimedia conferences, instant messaging, shared web browsing and control of network appliances.

A. End system services in SIPC

SIPC offers call control and presence-related services, emergency services, network appliance control and shared web browsing. We describe them in detail below.

1) Call control services: Call control services define how incoming and outgoing signaling messages related to a call session are handled. The call control services implemented in SIPC include many services defined in ITU-T recommendation Q.1211 [5] such as abbreviated dialing (ABD), automatic call back (ACB), call forwarding on busy/don't answer (CFC), customized ringing (CRG), originating call screening (OCS), terminating call screening (TCS), just to name a few.

2) Presence-related services: SIPC is a presence user agent (PUA) that can notify others about the user's presence state, such as online, offline or idle status, and receive notifications. With the support of CPL extensions for presence [6], SIPC can automatically set up calls when a user comes on-line.

3) Emergency services: An end system can connect to devices, for example, a fire detector or a temperature monitor, to detect emergency events and use the SIP event notification architecture for emergency notification [7]. In addition, the end system can issue device control command to handle the emergencies. In SIPC, we use SOAP messages [8] as the content of emergency notifications. When SIPC gets an emergency event, it can, for example, execute a script to flash lights or close fire doors.

4) Integration with other Internet services: Several SIP headers can contain URIs that are resolved by the end system. For example, a URI in the Call-Info header describes the caller or callee. A HTTP URL in a Contact header can forward a call to a web page. In addition to using SIP headers to access call-related web information, we defined a mechanism to use the SIP MESSAGE method for shared web browsing [9]. Shared web browsing allows a group of people to visit the same web sites. SIPC can interact with the local web browser to retrieve the URL currently being viewed. The visited URL is then sent to the remote party via the SIP MESSAGE method. The remote SIPC will instruct its local web browser to visit the same URL.

5) Conference control: An end system can also cooperate with network servers to perform conference control services. An ongoing project is to use the SIP event architecture and SOAP for conference floor control [10]. With SIPC, the moderator can grant or revoke floors, participants can claim the floor and watch its status.

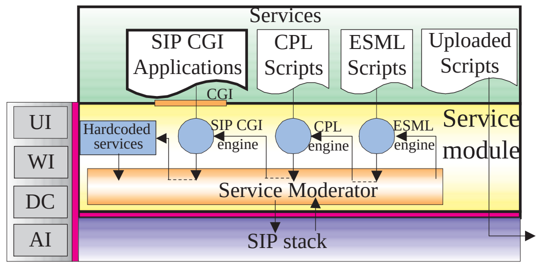
B. *The service execution environment in* SIPC

Several service interfaces and languages have been defined that allow the creation of services for SIP-based systems. These include SIP CGI, SIP servlets [11] and CPL. Because SIP servlets is built upon Java, while SIPC is implemented in C/C++ and Tcl/Tk, we only support SIP CGI and CPL. SIP CGI inherits the HTTP CGI model [12]. It uses the Common Gateway Interface to exchange information between service applications and SIP entities. For an incoming SIP message, the SIP entity invokes the service application and transfers message information to it through environment variables. The service application then instructs the SIP entity via commands written to standard output.

Since SIP CGI scripts allow general-purpose languages and imposes no restrictions on scripts, they are ill-suited for untrusted third parties. In contrast, CPL is an XML-based language that is intentionally limited in its capabilities, supporting neither loops nor variables nor recursion.

Figure 1 shows the architecture of the end system service execution environment in SIPC. The services include SIP CGI applications, CPL scripts, ESML (see Section III) scripts and the scripts uploaded to SIP servers. By using the SIP REGISTER method, a SIP UA can send service scripts to network servers such as SIP proxy servers, saving a local copy of the uploaded service definition. The service creation environment is responsible for examining all the services for potential conflicts, though the service execution engines will also check at runtime.

The service engines in SIPC do not interact with the signaling module directly, rather, they communicate with the service moderator. The service moderator is responsible for prioritizing the services and resolve any conflicts. SIPC executes services in the order ESML, CPL, sip-cgi and finally hard-coded services. We chose this order so that the script most suitable for end systems is executed first.



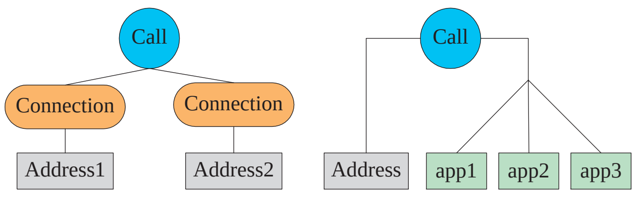
UI: User Interaction WI: Web Interaction
DC: Device Control AI: Application Interaction

Fig. 1. Architecture of end system service execution environment

III. Endpoint Service Markup Language (ESML)

A. *Motivation*

Many existing service languages are designed for on network services. The call model for end system services and network services are different. Figure 2 shows the models of a two-party call for a network service and an end system service. A network service establishes connections between multiple addresses, while end system services instruct the local application to send media to and receive media from remote addresses. The different call model implies different states, events and actions for services, thus a network service script is usually unsuitable for end systems and vice versa.



a. Network service call model b. End system service call model

Fig. 2. Call models of network services and end system services

In addition, the two kinds of services have different developers. Network services are usually implemented by experienced programmers so the functional richness of the service language is more important than its simplicity. On the other hand, end system services are usually developed by non-programmers, making simplicity a requirement.

B. Requirement for an end system service language

An end system service language needs to be simple and easy to understand by non-programmers. Users would like to move services they created to different platforms, so a platform-neutral high-level language is called for. The language needs to be able to express user interactions and control media and other end system applications. It should be extensible to accommodate new services. Since it is restricted to a certain class of services, it does not have to be Turing-complete.

C. Design strategy of ESML

The development of the Internet and the rise of the eXtensible Markup Language (XML) as a language standard have prompted proposals that XML-based scripting languages be used for creating telecommunications services. Among other advantages, XML is platform, network and technology neutral, independent of underlying programming languages, and readable by machines as well as humans. For these reasons, we based ESML on XML.

ESML inherits the tree-like structure from CPL. Like CPL, it avoids the use of loops, variables and recursion to allow program inspection and the back-and-forth translation between a graphical and textual representation.

To make ESML extensible, we use packages to group ESML events and actions. Initial packages support user, media and application interaction (Section III-E).

ESML is defined as an XML schema rather than a DTD. Schemas allow the derivation of new types from existing ones, so that ESML can derive a new package from an existing package. For example, the events and actions in the ESML 'sip' package can be derived from the 'generic' package, which we will explain in detail in Section III-E.2. Also, XML schema provides pre-defined data types, such as datetime and list, making it easier to define ESML and to validate ESML scripts.

D. Elements of ESML

Figure 3 shows a simple ESML script.

The script automatically places an outgoing call and alerts the user when `xyz@foo.com` is online. The 'name' attribute in ESML tags provides a reference of the feature. Users can enable or disable features by names. The 'notification' tag represents an incoming status notification event. The 'priority' attribute in the 'notification' tag can be used to solve feature conflicts (Section III-F). The 'address-switch' and 'address' tag check

```
<esml name="online_call"
   require="generic presence ui">
  <notification status="online"
     priority="0.5">
    <address-switch field="origin">
      <address is="xyz@foo.com">
        <call />
        <alert sound="call.au"
         text="Calling xyz@foo.com" />
      </address>
    </address-switch>
  </notifying>
</esml>
```

Fig. 3. ESML service example

whether the notification is from `xyz@foo.com`. The 'call' tag performs the action placing an outgoing call to `xyz@foo.com` and the 'alert' tag alerting the user.

The example shows that ESML consists of three basic elements, namely events, switches and actions. Most of the switches are inherited from CPL, such as `address-switch`, `time-switch` and `language-switch`.

ESML is based on events to invoke features. When an event happens, the ESML engine performs the actions defined in the ESML script. Events can be caused by call signaling, user interaction, timers or other applications.

E. ESML packages

ESML groups events, switches and actions into packages. An ESML engine can support only a subset of packages; the 'require' attribute in ESML tags indicates which packages the script uses. Each package is also a namespace, avoiding naming conflicts for extensions.

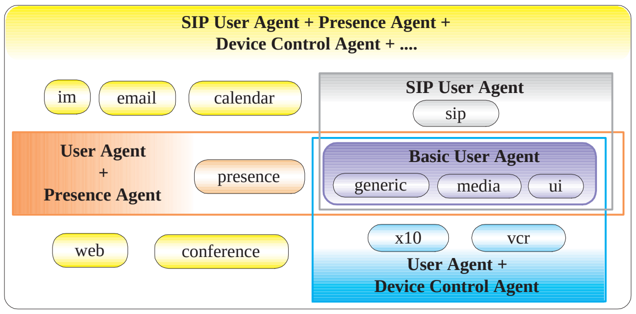


Fig. 4. ESML packages for a full-function end system

As shown in Figure 4, a simple user agent only needs to support the 'generic', 'media' and 'ui' packages. A SIP user agent adds the 'sip' package. The 'presence' package can be used for presence agent and 'im' package for instant messenger. A device control agent needs device packages. For example, with X10 CM11A computer interface, an end system can control X10 powerline devices and the 'x10' package is introduced. With Slink-e serial controller, an end system can send infrared signal out and control devices, like VCR. The 'vcr' package, containing commands such as play, record and fast forward, is then introduced for VCR control. Other packages include the 'conference', 'web', 'email' and 'calendar' packages, supporting conference control, interaction with web content, email and calendaring services, respectively.

Below, we briefly introduce the events and actions of the generic, sip and email packages.

1) GENERIC *package:* The generic package covers most call control services, such as accepting, forwarding, rejecting, and terminating calls and placing outgoing calls. It can handle events for incoming and outgoing calls, call termination and timers.

2) SIP *package:* The sip package defines the SIP-specific call control actions and events. Most of the actions and events in the sip package extend the generic package with richer attributes. Figure 5 shows the schema fragment of the 'incoming' event of the sip package. It extends the 'incoming' event of generic package with the additional SIP attribute 'priority'.

```
<xs:schema targetNamespace="esml:sip"
            xmlns:sip="esml:sip"
            xmlns:generic="esml:generic">
  ..........
 <xs:complexType name="IncomingType">
  <xs:complexContent>
   <xs:extension
       base="generic:IncomingType">
    <xs:attribute name="priority"
                  type="PriorityType"/>
    ..........
    </xs:extension>
   </xs:complexContent>
 </xs:complexType>
 ..........
```

Fig. 5. 'incoming' event in sip package

3) EMAIL *package:* The email services in ESML are like the services provided by procmail [13], but in a higher level abstraction. The email package contains an event called 'incoming' indicating an arriving email. With using the commands from generic and im packages, the event handler can send an instant message or make a call to notify the user for the arriving email. The email package also contains an action 'send' for sending email.

F. Handling feature interaction in ESML

The priority attribute in the `event` tag helps to resolve conflicts when multiple scripts subscribe to the same event. The script with highest priority gets executed first. If the actions defined in the higher-priority script cannot be performed (e.g., the switches cannot be matched), the ESML engine will continue to execute the lower-priority scripts.

G. ESML service creation

Users can use a text editor to create ESML service scripts directly. However, it is more efficient to break the service creation process into two stages. In the first stage, the service template in created, shown in Figure 6, the service.esml (template). The service template is also written in ESML, but with some conventions for the configurable values. For example, `<address is="${var}">` means the address is configurable. `${var}` should be replaced by a user input value. The second stage is to configure the service template and generate the ESML scripts. Users can use graphical editor to configure the template, or the template can be translated to HTML page by eXtensible Stylesheet Language (XSL) and XSL Transformations (XSLT). When performing translation, with using `xsl:if` tag, the XSLT checks the value of each attribute to see whether it's configurable. For configurable attributes, the XSLT uses HTML `input` tag in the HTML file so that users can input values. The CGI program, translate.cgi, then translates the HTML file to a user configured ESML script.

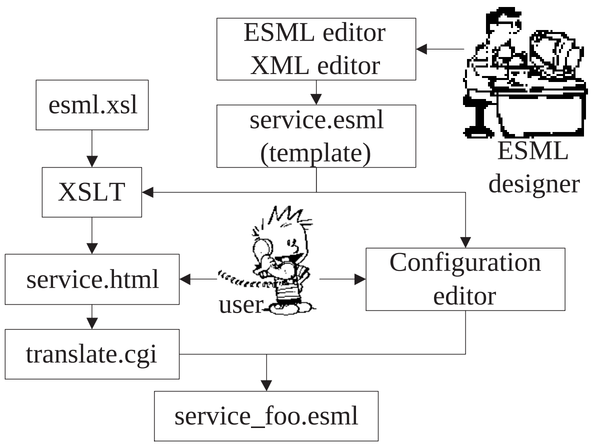


Fig. 6. Create ESML services

H. Comparison with the other script languages

Several XML-based call control languages have been proposed, such as the Call Policy Markup Language (CPML) [14], Telephony Markup Language (TML) [15], CallXML [16], Call Processing Language (CPL), Service Creation Markup Language (SCML) [17], and Call Control eXtensible Markup Language (CCXML) [18]. Space constraints prevent a complete survey of all these languages. We observe that the more mature and interesting of these proposals are CPL, being standardized by the Internet Engineering Task Force (IETF), SCML, being developed by JAIN forum, and CCXML, being developed by the Voice Browser working group in the World Wide Web Consortium (W3C). Our examination of CPL, SCML and CCXML concludes that none of the three approaches provides enough support for end system services.

Among CPL, SCML and CCXML, CPL is the only fully specified language. It is designed to run on a server where users may not be allowed to execute arbitrary programs, as it has no variables, loops, or ability to run external programs. These make it a suitable language for non-programmers. It can proxy, reject or redirect calls. However, CPL cannot originate a call, an important service for end systems. In addition, CPL cannot be activated through non-call events, such as timers.

CCXML is designed to provide telephony call control support for dialog systems, such as VoiceXML system, making it suitable for only a subset of end systems. The states and events for CCXML is in a lower level abstraction than those for ESML and CPL. For example, for ESML call event, in CCXML, the event is represented as call.CALL_CONNECTED, call.CALL_ACTIVE, connection.CONNECTION_ALERTING etc. For the users that are not familiar with communication systems, they cannot understand these events.

At the moment, SCML is still a work in progress. SCML is developed by the JAIN forum. It is closely tied to the JAIN Java Call Control (JCC) API. and defined using an XML Schema that is derived from JCC. The object model of JCC [19] is the same as the network service call model in Figure 2.a. It focuses on how to build connections between addresses, not on how to instruct local applications. Compared to ESML, SCML has not defined how to extend the language for different end system applications.

IV. Conclusion and future work

In this paper, we have discussed the importance of end system services and described the services and service architecture in our SIP user agent, SIPC. We have developed SIP CGI and CPL as service interfaces in SIPC, and are designing a new scripting language, ESML, specifically for end system services. Compared with the other existing script languages, ESML is based on a call model suited for end system services and offers simplicity, safety, extensibility and interaction with users, media applications and other end system applications. We plan to investigate how end system services are going to interact with existing network services and how to handle feature interactions between the end system and network services.

References

[1] J. Lennox, H. Schulzrinne, and J. Rosenberg, "Common gateway interface for SIP," RFC 3050, Internet Engineering Task Force, Jan. 2001.

[2] J. Lennox and H. Schulzrinne, "Call processing language framework and requirements," RFC 2824, Internet Engineering Task Force, May 2000.

[3] J. Lennox and H. Schulzrinne, "CPL: A language for user control of internet telephony services," Internet Draft, Internet Engineering Task Force, Nov. 2001. Work in progress.

[4] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: session initiation protocol," RFC 3261, Internet Engineering Task Force, June 2002.

[5] International Telecommunication Union, "General recommendations on telephone switching and signaling – intelligent network: Introduction to intelligent network capability set 1," Recommendation Q.1211, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1993.

[6] X. Wu, H. Schulzrinne, J. Lennox, and J. Rosenberg, "CPL extensions for presence," internet draft, Internet Engineering Task Force, June 2001. Work in progress.

[7] H. Schulzrinne and K. Arabshian, "Providing emergency services in internet telephony," *IEEE Internet Computing*, vol. 6, pp. 39–47, May 2002.

[8] W3C, "Simple object access protocol (soap) 1.1." http://www.w3.org/TR/SOAP.

[9] X. Wu and H. Schulzrinne, "Use SIP MESSAGE method for shared web browsing," Internet Draft, Internet Engineering Task Force, Nov. 2001. Work in progress.

[10] X. Wu *et al.*, "Use SIP and SOAP for conference floor control," Internet Draft, Internet Engineering Task Force, Apr. 2002. Work in progress.

[11] JAIN, "Sip servlet api." http://jcp.org/jsr/detail/116.jsp.

[12] "Common gateway interface." http://hoohoo.ncsa.uiuc.edu/cgi/interface.html.

[13] "procmail mail processing suits." http://www.procmail.org.

[14] "Call policy markup language (cpml)." http://xml.coverpages.org/cpml.html.

[15] "Telephony markup language (tml)." http://xml.coverpages.org/tml.html.

[16] "Callxml." http://community.voxeo.com/docs/cxml/index.jsp.

[17] J. Bakker and R. Jain, "Next generation service creation using xml scripting languages," in *Conference Record of the International Conference on Communications (ICC)*, (New York City, New York), April 2002.

[18] W3C, "Voice browser call control: Ccxml version 1.0." http://www.w3.org/TR/ccxml.

[19] R. Jain, F. M. Anjum, P. Missier, and S. Shastry, "Java call control, coordination, and transactions," *IEEE Communications Magazine*, vol. 38, Jan. 2000.