# Programmable Conference Server

Henning Schulzrinne, Kundan Singh and Xiaotao Wu [*]
Department of Computer Science, Columbia University
{hgs,kns10,xiaotaow}@cs.columbia.edu

**Abstract**

Conferencing services for Internet telephony and multimedia can be enhanced by the integration of other Internet services, such as instant messaging, presence notification, directory lookups, location sensing, email and web. These services require a service programming architecture that can easily incorporate new Internet services into the existing conferencing functionalities, such as voice-enabled conference control. W3C has defined the Call Control eXtensible Markup Language (CCXML), along with its VoiceXML, for telephony call control services in a point-to-point call. However, it cannot handle other Internet service events such as presence enabled conferences. In this paper, we propose an architecture combining VoiceXML with our Language for End System Services (LESS) and the Common Gateway Interface (CGI) for multi-party conference service programming that integrates existing Internet services. VoiceXML provides the voice interface to LESS and CGI scripts. Our architecture enables many novel services such as conference setup based on participant location and presence status. We give some examples of the new services and describe our on-going implementation.

**Keywords**: multi-party conferencing; service programming; CGI; VoiceXML; CCXML; LESS

## 1 Introduction

Most existing multi-party conference servers offer only limited functionalities, such as dial-in authentication and dial-out lists. Conferences have to occur at a pre-determined time or when manually created. Conference participants have to be pre-defined and usually only call control events get handled.

In today's Internet telephony systems, many other Internet service events besides call control events, such as presence status, location changes and stock price changes, can help to trigger a conference. The creation and membership of a conference can be dynamic and programmable by end users or conference administrators. For example, a conference server invites the conference participants only when all the essential conference members are online or in close geographic proximity. A conference server can automatically start a conference among the CEO and VPs of a company when the company's stock price drops by a threshold. The new services require a dynamic programmable service creation architecture that can incorporate new Internet services into the existing conference functionalities. We are building such a programmable conference server. As far as we know, this is the only such system.

The challenges in designing such services involve (1) defining a service programming language that can integrate existing Internet services, (2) providing a voice interface for input and output, and (3) extending the existing conferencing services. We divide a conference server into two parts: a *core server* implementing the basic functions and providing an interface over which the *higher layer* can build new services. Our main focus is on the *higher layer*.

For telephony applications, W3C has defined the XML-based languages, VoiceXML [5, 11] and Call Control XML (CCXML [13]), to program user interactions and point-to-point call control respectively. The VoiceXML interpreter and the CCXML interpreter can be co-located within a conference server and work together. However, CCXML has some limitations such as being too low level to be programed by non-specialists and, more importantly, the absence of non-telephony events such as presence indication. We are developing the Language for End System Services (LESS [15]), which provides a high-level control that supports telephony, presence and user interaction for an IP telephony endpoint. In this paper, we propose an architecture combining VoiceXML with LESS and Common Gateway Interface (CGI) for multi-party conference service programming. The integration is non-trivial because LESS is asynchronous event-based where as VoiceXML is a synchronous programming language. We extend LESS to incorporate voice interaction using VoiceXML and to provide association among multiple call dialogs in a conference.

[*]All authors have contributed equally to this work

We present some examples of advanced conferencing services in Section 2. Then we describe LESS in conjunction with VoiceXML to provide these services in Section 3, using illustrations. We describe two different models for using asynchronous event-based LESS with synchronous VoiceXML for integrating user interactions with call control. The models differ in their complexity and functionality as illustrated by an example of presence-enabled conferencing in Section 4. We describe our existing implementation in Section 5. We compare and contrast LESS and CCXML in Section 6. Finally, Section 7 presents the conclusion and future work.

## 2 Programmable services

There are two types of conference events: those external to the conference server such as user location changes or presence, and those generated by the conference itself such as when a person joins or leaves. These events can trigger certain programmable actions by the conference server, e.g., "call the manager when five people are in the room". We motivate our work on the event-driven programmable conference server architecture by the following two types of services: *automated conference creation* based on presence or location information and *conference control and indication* via touch-tone keys.

### 2.1 Automated conference creation

***Presence-enabled conference*:** For a conference that requires the presence of several essential participants, the conference server should check the presence status of these participants and start the conference only when all of them are online. This reduces the waiting time for people to join in a conference. When the first essential participant is online, the presence notification event will invoke an action to check all the other essential participants' presence status. If all the essential participants are online, the event handling action invites all the available participants to the conference and also continues to check the optional participants' presence status. Once the conference starts, the presence notification event of an optional participant will invoke an outgoing call invitation to that participant.

***Location-based conferencing*:** When a group of people meet in a conference room, the location update event can invoke an action to automatically place them in a conference. These participants can see the slides being discussed, follow the web sites being visited and are placed in a common chat room with a common conference floor. External participants can also be invited into the meeting.

The presence and location-based conferences are examples of event-driven conferencing. Use of a generic event notification system can allow extending such architecture to other event-based conferencing, such as "put the board of directors of a company in a conference when the stock price suddenly drops" or "invite all office staff into a conference for an automatic emergency announcement".

### 2.2 Conference control and notification

***Conference control via telephone*:** The conference moderator should be able to control who can speak, who should be invited to or kicked out of the conference and what media are allowed from which participant. The control can be via telephone touch-tone keys.

***Custom announcements*:** When new participants join or leave, the conference server can send announcements indicating so. It can also convert the received text-based instant message (IM) to voice and send it to the voice-only participants such as the telephone users.

If all such features are built-in into the conference server, it becomes rigid and hard to modify or create new related features. Alternatively, if the features are programmable, the conference service provider or the moderator can create new features or customize the existing ones. The *core server* provides the basic features and interface to the *higher layer* to build additional services as shown in Fig. 1. Such division of functions have been used in other related fields such as interactive voice response (IVR) systems based on VoiceXML [5], where the voice interpreter presents the voice dialogues to the telephone user, whereas the actual application logic such as voice mail access or tele-banking can be built on the back-end web server using the Common Gateway Interface (CGI [1]) or Java servlets [3].

Next, we identify the primitive functions needed in the core server, and define the programming interface to the higher layer applications. From the telephony signaling perspective the conference server acts like an endpoint receiving the calls from the participants, or making out-bound calls in the case of a dial-out conference. Besides the

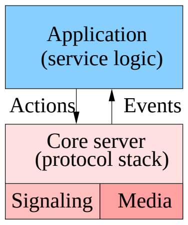

Figure 1: Division of conference server functions

basic call control, the server should be able to authenticate the participants, assign special privileges to participants or control media traffic. The media control includes audio mixing, transcoding, video forwarding, audio volume balancing and so on. Table 1 lists only the basic signaling functions needed by the core server.

Table 1: Core server signaling functions

| Name | Description |
|---|---|
| Make call | Make an outgoing call to a participant. |
| Accept | Accept an incoming call from a participant. |
| Reject | Reject an incoming call from a participant. |
| Transfer | Transfer an existing call to a new destination. |
| Authenticate | Authenticate an incoming call. |
| Join | Add a participant to the conference. |
| Unjoin | Remove an existing participant from the conference. |
| Subscribe | Subscribe to receive the presence or location information of a participant. |
| unsubscribe | Un-subscribe a previous subscription. |

The server can also receive events such as presence indication or result of an out-bound call attempt. These events need to be handled asynchronously as and when they arrive.

We describe the programming models to design and implement such interfaces in the next section.

## 3 Programming models

There can be different programming models depending on the complexity and level of details covered. Putting everything that the conference server can do in the programming interface will make the higher layer applications more complex. On the other hand, having very high level interface functions will limit the functionality of the higher layer applications.

### 3.1 VoiceXML

VoiceXML [5] works similar to web programming model. It is designed to facilitate IVR where a VoiceXML interpreter running on an Internet host or a telephony gateway interacts with the telephone user. The interpreter fetches the initial VoiceXML page from the web server. The page contains instructions to generate prompts or receive user input via spoken audio or touch-tone keys. The input may be submitted to the next server-side script that generates another VoiceXML page for subsequent dialogue with the telephone user. VoiceXML interpreter is the web browser substitute to a telephone user.

VoiceXML is rich in user interaction but has limited call control functions, as only *call disconnect* and *transfer* can be invoked. To complement VoiceXML, usually another call control language is needed.

## 3.2 LESS

LESS [15] is an example of a call control language. We have designed it for IP telephony endpoints to perform advanced functions such as "make a call when the user is online". It is extended from the Call Processing Language (CPL [4]) and inherits its advantages of being simple, safe, extensible and easily editable by graphical clients.

Although conference servers usually reside in the network, they can perform many services that are designed for end-user-operated endpoints, such as "automatically accepting incoming calls" or "initiating outgoing calls".

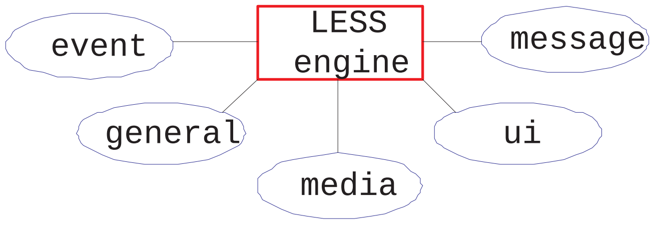

Figure 2: LESS packages

LESS uses packages for extensibility (Fig. 2). The general package contains the basic call handling functions such as accepting a call or making an outgoing call. The ui package handles user dialogs. The event package deals with the event notification, which includes presence indication. The media package performs media handling such as copying the media information from one session to another. The message package is for instant messages. The design of the LESS schema [15] allows easily adding a new packages.

The packages make LESS good for many innovative services. For example, with the event package, a conference server can check online status of all the essential participants. Once all the essential participants are online, the conference server can start the conference by inviting all the available participants to join.

## 3.3 Combining LESS with VoiceXML

In this paper we extend LESS to use existing voice-based user dialogs. LESS does not define its own method to handle voice inputs because VoiceXML already exists for this. There are two ways to combine LESS with VoiceXML. In the first approach, LESS is the main controller and uses VoiceXML for voice interaction. In the second, both VoiceXML and LESS are interpreted for basic dialogues and call control functions whereas the complexity, state management and control logic are moved to the web server scripts. We describe and compare both these methods in this section.

### 3.3.1 VoiceXML as a user interface component

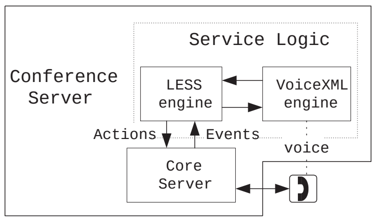

Figure 3: VoiceXML as a voice interface

In Fig. 3, all service logics exist in the LESS script, whereas the VoiceXML engine acts as the user interface component similar to the existing graphical or abstract user interface notions in LESS. An example service for this architecture is *auto-callout conference service* in which an event can trigger a conference. For example, when the first participant joins a conference, the conference server automatically calls all the other intended participants. Fig. 4 shows the outline of a LESS script for implementing this feature. The LESS module handles the main call control and invokes the VoiceXML script, `checkPin.vxml`, which in turn calls the VoiceXML interpreter to prompt the user for her personal identification number (PIN). The return value of the VoiceXML script is saved in voicexml.return for further processing in the LESS script.

```
<less>
 <CPL:incoming>
  <accept>
   <VoiceXML:form url="file:///checkPin.vxml">
    <VoiceXML:completed>
     <Conference:conference id="voicexml.return">
      <Conference:join>
       <Conference:success>
        <CPL:lookup source="participants">
         <call />
        </CPL:lookup>
       ...
```

Figure 4: LESS script for the auto-callout service

Treating VoiceXML script as a user interface component makes the LESS control logic visible to the user and easier to understand and modify the services by graphical tools. However, since LESS is designed to be simple and safe, it is intentionally limited in its capabilities. For services requiring more complex logic, we need to move the complexity to the web server scripts as described below.

### 3.3.2 Moving complexity to web server scripts

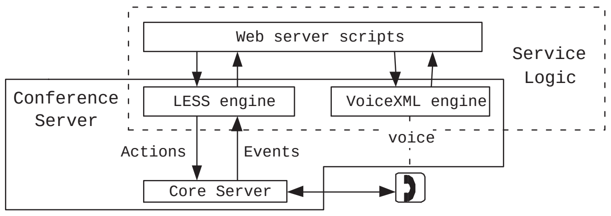

Figure 5: Moving complexity to web scripts

In Fig. 5, all service logics exist in the web server scripts such as CGI scripts or Java servlets. The VoiceXML engine serves as a voice interface to users, and the LESS engine serves as a functional module. The server-side program generates the scripts in LESS or VoiceXML which are interpreted by the appropriate interpreters running on the conference server.

Suppose the LESS module receives the incoming call indication, and needs to prompt the user to enter PIN. The LESS script can have a `<submit next="http://.../login.cgi" />` tag in the incoming call processing. This causes the `login.cgi` script to run on the web server and generate the next page. The next page is a VoiceXML page which prompts the user to enter PIN and collects the user digits. The VoiceXML page then calls `<submit next="http://... /auth.cgi" namelist="pin" />` which invokes the script `auth.cgi` with parameter "pin". This script validates whether the user PIN is correct and identifies the user. It generates the next VoiceXML page to indicate incorrect PIN or a LESS page to join this call to the pre-established conference. The control passes back and forth between the LESS and VoiceXML interpreters. All the service logic is built using the CGI scripts, whereas simple call control functions (Table 1) such as "accept the call", "transfer" or "join to a conference" are done using LESS. The CGI scripts should be well designed and verified to prevent loops.

New extensions in LESS are needed to dynamically create new events such as event=auth-complete, and new commands such as submit. It also changes the model from a *reactive* to an *active* one because invoking a LESS script can now trigger actions without external stimulus. This model does not need the program control flow constructs such as if-then-else or switch-case in LESS, but can be done in CGI. However, using CGI means that the service creation can not be done with the simple graphical tools, and requires additional programming knowledge. This is similar to the web CGI programming model.

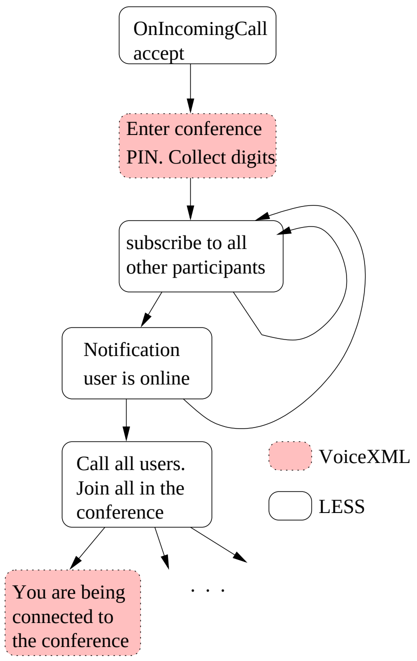

Figure 6: Presence-enabled auto-callout service

The CGI-model is shown in Fig. 6. An initial LESS script accepts the call and invokes a VoiceXML page to prompt for conference identifier. The VoiceXML interpreter collects the user input (DTMF digits) and submits them to the next CGI script. This script finds out all the needed participants for this conference from the database and generates a LESS script to subscribe for presence status of all the other participants. This LESS script is installed as a handler for notification. When it detects that an user is online, it invokes the CGI script with the parameters containing the current list of online users. The CGI script finds out if all the users are online or not. If not, then it goes back to waiting for further notifications. If all the users are online, it generates a new LESS script to place new out-bound calls to those users, and to connect all the users in the conference. It then invokes the VoiceXML pages for notifying every user that they are being connected to the conference.

Alternatively, the participants can be invited to the conference as and when they become online, instead of waiting for all of them to become online. Advanced scripts can wait for the essential participants to be online, and add the optional participants as they come.

We extend LESS to identify a particular call for generating the final prompts specific for each call by defining a session tag as follows:

```
<session id="22134">
  <submit next="prompt.vxml" />
</session>
```

One important difference in the two models is that the CGI-model must be allowed only by the trusted users or administrators so that the web server scripts have restricted access, whereas *VoiceXML-as-user-interface* model is more safe and can be allowed for ordinary users of the system.

## 4 Presence-enabled conferencing

In a presence-enabled conference, the conference server detects the presence status of all the essential participants and start the conference only when all are online. We start by describing the LESS script and then propose a way to perform event aggregation, which is required to detect multiple presence status.

## 4.1 Using LESS for presence-enabled conferencing

```
<less>
 <timer dtstart="09012003T090000"
        dtend="09012003T110000">
  <CPL:location url="sip:conf-pa@foo.com">
   <EVENT:subscribe package="aggregation"
     expires="7200" content='......'>
    <EVENT:success>
     <EVENT:notification>
      <EVENT:event-switch>
       <EVENT:event package="aggregation"
                    is="match">
        <CPL:lookup source="participants">
         <CPL:success>
          <call/>
         ...
```

Figure 7: Presence-enabled conferencing script

Fig. 7 shows a LESS script for the presence-enabled conference. The script is invoked by a timer event. The start time of the timer event can be set to the scheduled conference start time. Inside the script, it first tries to subscribe to the presence status [7] of several essential participants. The content of the subscription is shown in Fig. 8 and discussed in detail in Section 4.2. If the subscription gets accepted, the LESS script engine waits for the notifications. The presence agent checks the presence status of the essential participants. Once the status matches the one requested in the subscription, the presence agent sends a notification to the conference server. The conference server checks the event and makes calls to all those participants.

## 4.2 Presence aggregation

```
<trigger>
 <all>
  <match contact="sip:tom@example.com"
   package="presence" status="open"/>
  <match contact="sip:bob@example.com"
   package="presence" status="open"/>
  <any>
   <match contact="sip:alice@foo.com"
    package="presence" status="open"/>
   <match contact="sip:steve@foo.com"
    package="presence" status="open"/>
  </any>
 </all>
</trigger>
```

Figure 8: Simple event aggregation script

It is easy for LESS to check one person's presence status and perform actions. However, it is difficult to check multiple presence status simultaneously and to trigger actions based on the combined status. This requires aggregation of all the presence status subscriptions. To keep the service logic easier to implement and understand, we suggest that the aggregation be handled in a separate presence agent. The conference server handles the service logic only on the aggregated event. For example, if a conference server wants to wait until Tom and Bob from `examples.com`, and one of Alice and Steve from `foo.com` are online to start a conference, it can simply put the script shown in Fig. 8 as the content of the subscription.

The aggregation can be more complex if more conditions such as time, callee's capabilities, and language preferences are considered. For example, if Tom serves as the moderator of a conference, a PSTN phone is not convenient for him to perform conference control functions. The presence aggregation may check the URI (Universal Resource Identifier) information of Tom's user agent. For a SIP user agent, if the `user=phone` parameter is present in Tom's contact URI, the presence agent should not consider this contact as a match for the presence event aggregation. Further details on the presence aggregation specification are outside the scope of this paper.

The presence agent performs the aggregation by de-coupling the aggregated subscription and sending the individual subscriptions to the corresponding parties.

An alternative to event aggregation is to define an aggregated presentity such as `my-group@domain`, and define rules such as "my-group is online when all of its participants are online". Rules can be simple logical AND or OR, or a more complex function of individual events. For simple cases such as a room, the entity can be defined naturally, e.g., `room460@columbia.edu`.

### 4.3 Presence aggregation with location-based services

The presence aggregation can further interact with location-based services. Suppose Tom, Bob and Alice would like to have a conference. Their user agents can know their physical locations by infrared location sensors, Bluetooth beacons or DHCP options [8]. The user agents will publish the location information to their presence agents. The presence agents will notify the conference server which subscribes to their physical location information [6]. The conference server discovers that Tom and Bob are in the same building, whereas Alice is away. With Service Location Protocol (SLP [2]), the server may find that "room 460" is close to both Tom and Bob and has good communication capabilities. The conference server sends instant messages to Tom and Bob asking them to go to that room. When the server discovers that both Tom and Bob are in that room, it starts a conference, and invites the devices in "room 460", such as the room speakers and video projectors, to the conference. It also invites Alice to the same conference. By this way, Tom and Bob can talk to each other face-to-face, and tele-conference with Alice.

## 5 Implementation

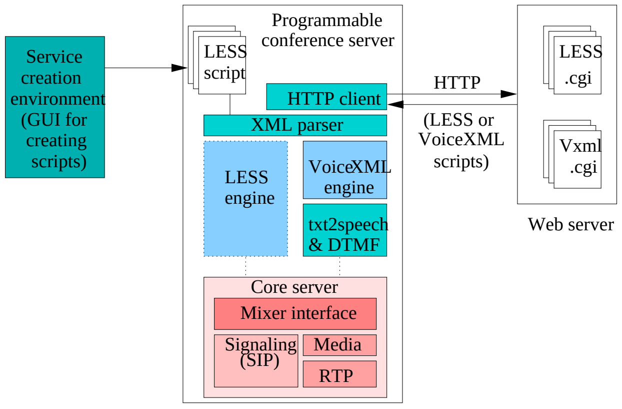

Figure 9: Design of our programmable conference server

The design of our programmable conference server is shown in Fig. 9. We have already implemented a SIP-based centralized conference server [10] and a VoiceXML interpreter [11] in our Columbia InterNet Extensible Multimedia Architecture [9, 12], which is an infrastructure for enterprise multimedia collaboration. We also have LESS support in our IP telephony user agent [14]. We have implemented a CPL engine for our SIP proxy server. Since LESS is based on CPL, the existing CPL engine can be extended to a LESS engine and be integrated with our conference server.

The modules and interfaces that are still incomplete are shown with dotted lines in Fig. 9. The architecture is designed to support both the ways of combining LESS with VoiceXML. The core server provides the SIP signaling stack, Real-time Transport Protocol (RTP) stack, media codecs and conference mixing interface to the higher layer. The service logic consists of the LESS and VoiceXML engines. Text-to-speech and DTMF is used by the VoiceXML engine. We use external XML parser to parse the scripts, and HTTP client to fetch the scripts from remote server. The service creation environment (SCE) provides a Tcl/Tk-based GUI to create LESS scripts that are uploaded to the conference service logic module.

The back end server side application logic is implemented using CGI scripts written in the Tool Command Language (Tcl). The LESS or VoiceXML engine can use the internal HTTP client to fetch the dynamically generated LESS or VoiceXML scripts from the web server. We have written a Tcl library to facilitate generation of VoiceXML pages so that the script is easy to modify and understand. We plan to extend this to support LESS as well. An example script fragment is shown in Fig. 10.

```
voicexml {
 vxml_form {
  vxml_field name=pin {
   vxml_prompt {
    puts "Enter your PIN."
   }
  }
  vxml_block {
   vxml_submit next=auth.cgi namelist=pin
  }
 }
}
```

Figure 10: Tcl code fragment to generate a VoiceXML page

## 6 Related work

CCXML [13] is a W3C standard. It is designed to provide telephony call control support for dialog systems, making it suitable for only a subset of the Internet end systems such as voice-only endpoints. The conference servers in Internet telephony systems can potentially handle the non-telephony events, such as presence indication, and perform actions in addition to the telephony calls, such as instant message, shared web browsing and desktop sharing. LESS is designed for IP telephony end systems and is able to handle the non-telephony functions. For example, LESS may instruct a SIP user agent to send instant messages, or based on presence information to start a conference. Unlike LESS, CCXML does not contain the functions for other Internet services.

The states and events for CCXML is in a lower level abstraction than those for LESS and CPL. For example, in CCXML, the call event is represented as sub-events such as `call.CALL_CONNECTED`, `call.CALL_ACTIVE`, `connection.CONNECTION_ALERTING`. Such signaling-derived events are too low-level to be programmed by non-technical users.

## 7 Conclusions and future work

Advanced conference services such as presence and location-based conferences can be implemented by separating the core server functions from the service logic. The service logic can be programmed using call control language such as CCXML and LESS, and voice dialog language such as VoiceXML. This allows various event-based services as well as voice-based conference control and indications.

In this paper, we propose to use VoiceXML with our LESS to perform event-driven conference services. We describe and compare several programming models for conference services. The VoiceXML as a user interface component in LESS model is easy to program and understand, and is sufficient for simple applications. On the other hand, moving the service logic to the web server script model is good for complex services. We further describe the on-going implementation of a programmable SIP conference server in our CINEMA collaboration framework. In addition, we provide a user friendly service creation interface so that users can create, modify or customize their conference services easily. We are extending it to be web based where the web server is co-located with the conference server so the created services can be stored in the conference server directly.

## 8 Acknowledgments

The research described here is supported by a grant from SIPquest, Inc.

## References

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

[2] E. Guttman, C. E. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC 2608, Internet Engineering Task Force, June 1999.

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

[4] J. Lennox, X. Wu, and H. Schulzrinne. CPL: a language for user control of Internet telephony services. Internet draft, Internet Engineering Task Force, Aug. 2003. Work in progress.

[5] S. McGlashan, D. Burnett, J. Carter, S. Tryphonas, J. Ferrans, A. Hunt, B. Lucas, and B. Porter. Voice extensible markup language (voicexml) version 2.0. Technical report, World Wide Web Consortium (W3C), Feb. 2003. http://www.w3.org/TR/voicexml20/.

[6] J. Peterson. A presence-based GEOPRIV location object format. Internet Draft draft-ietf-geopriv-pidf-lo-01, Internet Engineering Task Force, Feb. 2004. Work in progress.

[7] A. B. Roach. Session initiation protocol (sip)-specific event notification. RFC 3265, Internet Engineering Task Force, June 2002.

[8] H. Schulzrinne. DHCP option for civil location. Internet draft, Internet Engineering Task Force, July 2003. Work in progress.

[9] K. Singh, W. Jiang, J. Lennox, S. Narayanan, and H. Schulzrinne. CINEMA: columbia internet extensible multimedia architecture. technical report CUCS-011-02, Department of Computer Science, Columbia University, New York, New York, May 2002.

[10] K. Singh, G. Nair, and H. Schulzrinne. Centralized conferencing using SIP. In *Internet Telephony Workshop*, New York, Apr. 2001.

[11] K. Singh, A. Nambi, and H. Schulzrinne. Integrating VoiceXML with SIP services. In *Conference Record of the International Conference on Communications (ICC)*, May 2003.

[12] K. Singh, X. Wu, J. Lennox, and H. Schulzrinne. Comprehensive multi-platform collaboration. In *SPIE Conference on Multimedia Computing and Networking (MMCN 2004)*, Santa Clara, CA, Jan 2004.

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

[14] X. Wu. Columbia university sip user agent (sipc). http://www.cs.columbia.edu/IRT/sipc.

[15] X. Wu and H. Schulzrinne. Programmable end system services using SIP. In *Conference Record of the International Conference on Communications (ICC)*, May 2003.