Internet Engineering Task Force
Internet Draft

SIPPING WG
J. Rosenberg
dynamicsoft
H. Schulzrinne
Columbia U.

draft-rosenberg-sip-call-package-01.txt
March 1, 2002
Expires: September 2002

SIP Event Packages for Call Leg and Conference State

STATUS OF THIS MEMO

Abstract

This document defines two new event packages for the SIP Events architecture, along with two new data formats used in notifications for those packages. The first is a call-leg package, and the second is a conference package. The call-leg package allows users to subscribe to another user, an receive notifications about the changes in state of call legs that the user is involved in. The conference package allows users to subscribe to a URL that is associated with a conference. Notifications are sent about changes in the membership of this conference, changes in active speaker, and media mixing information. These general purpose packages can enable many new SIP services, including single line extension, automatic callback, unattended consultation-hold transfer, call park and pickup, and IM-

a-call.

Table of Contents

1 Introduction

The SIP Events framework [1] defines general mechanisms for subscription to, and notification of, events within SIP networks. It introduces the notion of a package, which is a specific "instantiation" of the events mechanism for a well-defined set of events. Packages have been defined for user presence [3], watcher information [4], and message waiting indicators [5], amongst others. Here, we define two new packages - one for dialogs, and the other for conferences.

The need for these packages is driven based on the fact that many applications are driven off of knowledge about the progress of dialogs and conferences. In the case of dialogs, we see many potential applications that require knowledge of dialog state:

Automatic Callback: In this basic PSTN application, user A calls user B. User B is busy. User A would like to get a callback when user B hangs up. When B hangs up, user A's phone rings. When A picks it up, they here ringing, and are being connected to B. In VoIP, this requires A to receive a notification when the dialogs at A are complete.

Presence-Enabled Conferencing: In this application, a user A wishes to set up a conference call with users B and C. Rather than scheduling it, it is to be created automatically when A, B and C are all available. To do this, the server providing the application would like to know whether A, B and C are "online", not idle, and not in a phone call. Determining whether or not A, B and C are in calls can be done in two ways. In the first, the server acts as a call stateful proxy for users A, B and C, and therefore knows their call state. This won't always be possible, however, and it introduces scalability, reliability, and operational complexities. Rather, the server would subscriber to the dialog state of those users, and receive notifications as it changes. This enables the application to be provided in a distributed way; the server need not reside in the same domain as the users.

IM Conference Alerts: In this application, a user can get an IM sent to their phone whenever someone joins a conference that the phone is involved in. The IM alerts are generated by an application separate from the conference server.

In general, defining dialog and conference state packages allows for construction of distributed applications, where the application requires information on dialog and conference state, but is not co-

resident with the end user or conference server. We think this is a very important piece of the SIP services model.

2 Dialog Event Package

This section provides the details for defining a SIP Events package, as specified by [1].

2.1 Event Package Name

The name of this event package is "dialog". This package name is carried in the Event and Allow-Events header, as defined in [1].

2.2 Event Package Parameters

This package does not define any event package parameters.

2.3 SUBSCRIBE Bodies

A SUBSCRIBE for a dialog package MAY contain a body. This body defines a filter to apply to the subscription.

A SUBSCRIBE for a dialog package MAY be sent without a body. This implies the default subscription filtering policy. The default policy is:

- Notifications are generated every time there is any change in the state of any dialogs for the user identified in the request URI of the SUBSCRIBE.

- Notifications do not normally contain full state; rather, they only indicate the state of the dialog whose state has changed. The exception is a NOTIFY sent in response to a SUBSCRIBE. These NOTIFYs contain the complete view of dialog state.

- The notifications contain the identities of the participants in the dialog, and the dialog identifiers. Additional information, such as the route set, remote target URI, CSeq numbers, SDP information, and so on, are not included normally unless explicitly requested and/or explicitly authorized.

2.4 Subscription Duration

Dialog state changes fairly quickly; once established, a typical phone call lasts a few minutes (this is different for other session types, of course). However, the interval between new calls is typically infrequent.

We do note that there are two distinct use cases for dialog state. The first is when a subscriber is interested in the state of a specific dialog (and they are authorized to find out about just the state of that dialog). In that case, when the dialog terminates, so too does the subscription. In these cases, the refresh interval can be very long, since there exists an easy alternative way to destroy subscription state. As a result, the default duration of these subscriptions is one day. The subscriber MAY request other durations.

In another case, a subscriber is interested in the state of all call legs for a specific user. In these cases, a shorter interval makes more sense. The default is one hour for these subscriptions.

> OPEN ISSUE: We should probably have a single default subscription duration.

2.5 NOTIFY Bodies

The body of the notification contains a dialog information document. The format of this document is described in Section 3. Its MIME type is "application/dialog-info+xml". All subscribers MUST support this format, and MUST list its type in any Accept header in the SUBSCRIBE. When no Accept header is present in the SUBSCRIBE, its default value is "application/dialog-info+xml".

Other dialog information formats might be defined in the future. In that case, the subscriptions MAY indicate support for other formats. However, they MUST always support and list "application/dialog-info+xml" as an allowed format.

Of course, the notifications generated by the server MUST be in one of the formats specified in the Accept header in the SUBSCRIBE request.

2.6 Notifier Processing of SUBSCRIBE Requests

The dialog information for a user contains very sensitive information. Therefore, all subscriptions SHOULD be authenticated and then authorized before approval. Authorization policy is at the discretion of the administrator, as always. However, a few recommendations can be made.

It is RECOMMENDED that if the policy of a user is that A is allowed to call them, dialog subscriptions from user A be allowed. However, the information provided in the notifications does not contain any dialog identification information; merely an indication of whether the user in in one or more calls, or not. Specifically, they should

not be able to find out any more information than if they sent an INVITE.

It is RECOMMENDED that if a user agent registers with the address-of-record X, that this user agent authorize subscriptions that come from any entity that can authenticate itself as X. Complete information on the dialog state SHOULD be sent in this case. This authorization behavior allows a group of devices representing a single user to all become aware of each other's state. This is useful for applications such as single-line-extension.

2.7 Notifier Generation of NOTIFY Requests

Notifications are generated for the dialog package when a new dialog comes into existence at a UA, or when the state of an existing dialog changes.

For the purposes of this package, we define the states of a dialog through numeric codes. These codes are equivalent to the most recent SIP status codes sent in response to the INVITE which created the call leg. The status code "0" is reserved for the case where no response has yet been received or sent.

When a UAC initially creates an INVITE to establish a call, this causes a change to state "0". When it receives the first non-100 provisional response, the state changes to the value of that status code. Any further provisional responses cause the UA to change state to the value of that status code. When a final response is received, the state changes to the value of that response. If the response was a non-200, the dialog is considered terminated, and no further state changes are possible. Multiple 2xx responses received create additional dialogs, each with the state of that specific 2xx.

When a UAS initially receives an INVITE to establish a call, this causes a change to the state of the provisional response which was sent. Any subequent provisional responses cause a change in state to the value of that response. A final response causes a transition in state to that response code. There is no change in state when the ACK arrives. However, if no ACK is received, and the UAS destroys the call, the state changes to a value of -1.

When the call is terminated as a result of a BYE, the state changes to -1.

> OPEN ISSUE: This is kind of ugly. We could alternately define a more formal state machine.

2.8 Subscriber Processing of NOTIFY Requests

The SIP Events framework expects packages to specify how a subscriber processes NOTIFY requests in any package specific ways, and in particular, how it uses the NOTIFY requests to contruct a coherent view of the state of the subscribed resource.

Typically, the NOTIFY for the dialog package will only contain information about those dialogs whose state has changed. To construct a coherent view of the total state of all dialogs, a subscriber to the dialog package will need to combine NOTIFYs received over time. The subscriber maintains is complete dialog list in a table, indexed by the id. This ID is different from the formal dialog ID as defined in [2], which is the concatenation of the local tag, remote tag, and Call-Id. This ID is conveyed in the id attribute of the dialog element of the "application/dialog-info+xml" type. If the dialog information in a NOTIFY has a dialog listed with an ID not in the table, an entry is added to that table. The version number from the dialog element is also extracted, and placed in the table. If the dialog information in a NOTIFY has a dialog listed with an ID in the table, and the version in the NOTIFY is greater than the version stored in the table, the dialog information in the table for that dialog is updated, including the version number. If a dialog is updated such that its status is now "-1", that entry MAY be removed from the table at any time.

2.9 Handling of Forked Requests

A forked SUBSCRIBE request for dialog state can install multiple subscriptions. Subscribers to this package MUST be prepared to install subscription state for each NOTIFY generated as a result of a single SUBSCRIBE.

2.10 Rate of Notifications

For reasons of congestion control, it is important that the rate of notifications not become excessive. As a result, it is RECOMMENDED that the server not generate notifications for a single subscriber at a rate faster than once every 5 seconds.

2.11 State Agents

Dialog state is ideally maintained in the user agents in which the dialog resides. Therefore, the elements that maintain the dialog are the ones best suited to handle subscriptions to it. Therefore, the usage of state agents is NOT RECOMMENDED for this package.

3 Dialog Data Format

We specify an XML-based data format to describe the state of a dialog. The MIME type for this format is "application/dialog-info+xml", consistent with the recommendations provided in RFC 3023 [6].

3.1 Structure of Dialog Information

A dialog-info document starts with a user tag that identitifies the user. Within that tag are a series of dialog tags. Each of those use attributes to identify the dialog and provide its version number. There are also attributes to provide the formal dialog identifier, using the local and remote tags, and the Call-ID. Additional attributes are present to specify the local and remote URIs. There is also an attribute that indicates whether the user initiated this dialog or not. Within the dialog tags are a single mandatory tag which contains the status, followed by a series of optional tags that contain additional information about the dialog.

The top level tag is user:

```
<!ELEMENT user (dialog*)>
<!ATTLIST user uri CDATA  #REQUIRED>
```

The mandatory uri attribute is the identifier of the user whose dialog state is being reported.

What follows is a series of dialog tags:

```
<!ELEMENT dialog (status,local-sdp?,remote-sdp?,
                  route-set?,remote-target?,local-cseq?,remote-cseq?)
<!ATTLIST dialog     id                CDATA                #REQUIRED
                     version           CDATA                #REQUIRED
                     call-id           CDATA                #IMPLIED
                     local-uri         CDATA                #IMPLIED
                     local-tag         CDATA                #IMPLIED
                     remote-uri        CDATA                #IMPLIED
                     remote-tag        CDATA                #IMPLIED
                     direction         (iniatiator|recipient) #IMPLIED>
```

The local-uri, local-tag, remote-uri, remote-tag and call-id attributes convey their corresponding components of the dialog state as defined in [2]. The direction attribute is "initiator" if the user

initiated this dialog, and "recipient" if it did not. The remote tag attribute won't be present if there is only a "half-dialog", resulting from generation of a request that can create a dialog.

For example, if a UAC sends an INVITE that looks like, in part:

```
INVITE sip:callee@foo.com SIP/2.0
From: sip:caller@bar.com;tag=123
To: sip:callee@foo.com
Call-ID: 987@1.2.3.4
```

the dialog tag sent out in a notification might looks like:

```
<dialog id="as7d900as8" version="0" call-id="987@1.2.3.4"
        local-uri="sip:caller@bar.com"
        local-tag="123" remote-uri="sip:callee@foo.com"
        direction="initiator">
```

If a 200 OK is received, which looks like, in part:

```
SIP/2.0 200 OK
From: sip:caller@bar.com;tag=123
To: sip:callee@foo.com;tag=abc
Call-ID: 987@1.2.3.4
```

The dialog is now confirmed, and the notification sent out will have a dialog tag which looks like:

```
<dialog id="as7d900as8" version="0" call-id="987@1.2.3.4"
        local-uri="sip:caller@bar.com"
        local-tag="123" remote-uri="sip:callee@foo.com"
        remote-tag="abc" direction="initiator">
```

3.2 Dialog Sub-Elements

There are many sub-elements defined for the dialog element.

3.2.1 Status

The only mandatory sub-element of dialog is status.

```
<!ELEMENT status CDATA>
<!ATTLIST status     code              CDATA               #REQUIRED>
```

The mandatory code attribute contains the status code. This is the SIP response code last sent or received for this leg in the initial INVITE that established the leg. If no response has been sent or received, the value of zero is used. If the call ends, a value of -1 is used.

The value within the status tag is a textual phrase that can be rendered to described call status. The reason phrase from the response is RECOMMENDED.

Example:

```
<status code="180">Ringing</status>
```

3.2.2 Local SDP

The local SDP tag contains the SDP used by the notifier for its end of the dialog. This tag should generally NOT be included in the notifications, unless explicitly requested by the subscriber.

```
<!ELEMENT local-sdp CDATA>
```

The SDP is included, verbatim, between the tags.

3.2.3 Remote SDP

The remote SDP tag contains the SDP used by the notifier for the other end of the dialog. This tag should generally NOT be included in the notifications, unless explicitly requested by the subscriber.

```
<!ELEMENT remote-sdp CDATA>
```

The SDP is included, verbatim, between the tags.

3.2.4 Route Set

The route-set tag contains the route set as constructed by the user agent for this dialog, as defined in RFC BBBB [2]. It is constructed from the Record-Route header field used for this dialog. This tag should generally NOT be included in the notifications, unless explicitly requested by the subscriber.

```
<!ELEMENT route-set CDATA>
```

The route set is included verbatim. It is structured as a comma separated list of URLs.

Example:

```
<route-set>sip:proxy2.example.com;lr</route-set>
```

3.2.5 Remote Target

The remote-target contains the remote-target URI as constructed by the user agent for this dialog, as defined in RFC BBBB [2]. It is constructed from the Contact header of the INVITE. This tag should generally not be included in notifications, unless explicitly requested by the subscriber.

```
<!ELEMENT remote-target CDATA>
```

The remote target URI is included verbatim between the tags.

Example:

```
<remote-target>sip:user@pc33.example.com</remote-target>
```

3.2.6 Local CSeq

The local-cseq tag contains the most recent value of the CSeq header used by the UA in an outgoing request on the dialog. This tag should generally NOT be included in the notifications, unless explicitly requested by the subscriber.

```
<!ELEMENT local-cseq CDATA>
```

The numeric value of the CSeq is included as the CDATA.

3.2.7 Remote CSeq

The remote-cseq tag contains the most recent value of the CSeq header seen by the UA in an incoming request on the dialog. This tag should generally NOT be included in the notifications, unless explicitly requested by the subscriber.

```
<!ELEMENT remote-cseq CDATA>
```

The numeric value of the CSeq is included as the CDATA.

4 Conference Event Package

The conference event package allows a user to subscribe to a conference. A conference is a collection of users that are all able to communicate with each other. Generally, when multicast is not used, a conference is associated by a set of dialogs that have their media mixed together. This is true for all of the non-multicast models in [7]. However, some of the models use topologies where there is no root to which all dialogs are connected. These topologies do not work well with the mechanism here.

This package allows a user to subscribe to a conference, identified by a SIP URI. Ideally, this SIP URI routes the SUBSCRIBE to the entity acting as the root of the topology (which is why it doesn't work well for the non-centralized topologies). The notifications contain information on the participants in the conference. The specific information conveyed is:

- o The SIP URI identifying the user.

- o The dialog state associated with that users attachment to the conference.

- Their status in the conference (active, declined, departed).

- Their status in terms of receiving media in the conference.

This section provides the details for defining a SIP Events package, as specified by [1].

4.1 Event Package Name

The name of this event package is "conference". This package name is carried in the Event and Allow-Events header, as defined in [1].

4.2 Event Package Parameters

This event package does not define any event package parameters.

4.3 SUBSCRIBE Bodies

A SUBSCRIBE for a dialog package MAY contain a body. This body defines a filter to apply to the subscription.

A SUBSCRIBE for a conference package MAY be sent without a body. This implies the default subscription filtering policy. The default policy is:

- Notifications are generated every time there is any change in the set of users participating in the conference, or a change their state (dialog state, media mixing state, etc.)

- Notifications do not normally contain full state; rather, they only indicate the state of the participant whose state has changed. The exception is a NOTIFY sent in response to a SUBSCRIBE. These NOTIFYs contain the complete view of conference state.

- For a given user, the notifications contain the identity information and status.

4.4 Subscription Duration

The default expiration time for a subscription to a conference is one hour. Of course, once the conference ends, all subscriptions to that particular conference are terminated, with a reason of "noresource" [1].

4.5 NOTIFY Bodies

The body of the notification contains a conference information

document. The format of this document is described in Section 5. Its MIME type is "application/conference-info+xml". All subscibers MUST support this format, and MUST list its type in an Accept header in the SUBSCRIBE. The default value for the Accept header when it is not present in a request is "application/conference-info+xml".

Other conference information formats might be defined in the future. In that case, the subscriptions MAY indicate support for other formats. However, they MUST always support and list "application/conference-info+xml" as an allowed format.

Of course, the notifications generated by the server MUST be in one of the formats specified in the Accept header in the SUBSCRIBE request.

4.6 Notifier Processing of SUBSCRIBE Requests

The conference information contains very sensitive information. Therefore, all subscriptions SHOULD be authenticated and then authorized before approval. Authorization policy is at the discretion of the administrator, as always. However, a few recommendations can be made.

It is RECOMMENDED that all users in the conference be allowed to subscribe to the conference.

4.7 Notifier Generation of NOTIFY Requests

Notifications SHOULD be generated for the conference whenever a new participant joins, a participant leaves, and a dial-out attempt succeeds or fails. Notifications MAY be generated for the conference whenever the media mixing status of a user changes.

4.8 Subscriber Processing of NOTIFY Requests

The SIP Events framework expects packages to specify how a subscriber processes NOTIFY requests in any package specific ways, and in particular, how it uses the NOTIFY requests to contruct a coherent view of the state of the subscribed resource.

Typically, the NOTIFY for the conference package will only contain information about those users whose state has changed. To construct a coherent view of the total state of the entire conference, a subscriber to the conference package will need to combine NOTIFYs received over time. The subscriber maintains is complete user list in a table, indexed by the id in the dialog element. If the dialog information in a NOTIFY has a dialog listed with an ID not in the table, an entry is added to that table. The version number from the

dialog element is also extracted, and placed in the table. If the dialog information in a NOTIFY has a dialog listed with an ID in the table, and the version in the NOTIFY is greater than the version stored in the table, the dialog information in the table for that dialog is updated, including the version number. If a dialog is updated such that its status is now "-1", that entry MAY be removed from the table at any time.

4.9 Handling of Forked Requests

By their nature, the conferences supported by this package are centralized. Therefore, SUBSCRIBE requests for a conference should not generally fork. Users of this package MUST NOT install more than a single subscription as a result of a single SUBSCRIBE request.

4.10 Rate of Notifications

For reasons of congestion control, it is important that the rate of notifications not become excessive. As a result, it is RECOMMENDED that the server not generate notifications for a single subscriber at a rate faster than once every 5 seconds.

4.11 State Agents

Conference state is ideally maintained in the element in which the conference resides. Therefore, the elements that maintain the conference are the ones best suited to handle subscriptions to it. Therefore, the usage of state agents is NOT RECOMMENDED for this package.

5 Conference Data Format

The conference data format is an XML document of MIME type "application/conference-info+xml", consistent with the recommendations provided in RFC 3023 [6].

5.1 Structute of the Format

The conference data format has the top level tag of conference. It consists of a set of sub-tags of type user, which contain information on the users in the conference. Each user tag contains the identity of the user, their dialog information, their status in the conference, and their media reception information.

The top level tag is conference:

```
<!ELEMENT conference (user*)>
```

```
<!ATTLIST conference uri CDATA  #REQUIRED>
```

The mandatory uri attribute contains the URI used to join the conference call (and to subscribe to its state).

What follows are a series of user tags:

```
<!ELEMENT user (status,dialog,media-status?)>
<!ATTLIST user   uri     CDATA   #REQUIRED
                 name    CDATA   #IMPLIED>
```

The uri attribute contains the URI for the user. This is a logical identifier, not a machine specific one (i.e., its taken from the To/From, not the Contact). The name is a textual name for rendering to a human. It is ususally taken from the display name.

5.2 User Sub-Elements

The sub-elements of the user tag are status, dialog aned media-status.

Status contains the status of the user in the conference.

```
<!ELEMENT status>
<!ATTLIST status
     value   (active|departed|booted|failed) "active" >
```

The statuses have the following meaning:

active: The user is in an active dialog with the conference host.

departed: The user sent a BYE, thus leaving the conference.

booted: The user was sent a BYE by the conference host, booting them out of the conference.

failed: The conference host is a dialout conference server, and its attempt to contact the specific user resulted in a non-200 class final response.

The dialog element is the same one from the dialog package above.

The media-status attribute is a series of media streams. Each media stream is associated with a media type, a sending status, and a receiving status.

```
<!ELEMENT media-status (media-stream*)>
<!ELEMENT media-stream>
<!ATTLIST media-stream
     type   (audio|video|message|application) #REQUIRED
     send-status (received-by-all|muted) "received-by-all"
     recv-status (receiving-all|anchor-only) "receiving-all">
```

If the send-status is "received-by-all", it means that the media for that stream that is being generated by the user is being mixed by the server and sent to all recipients. "muted" means that no one is receiving their media. If the receive-status is "receiving-all" it means that the user is hearing all other participants. If it is "anchor-only", the user is hearing media from just a single participant.

5.3 Example

The following is an example conference information document:

```
<conference>
  <user uri="sip:jdrosen@dynamicsoft.com" name="Jonathan Rosenberg">
    <status value="active"/>
    <dialog id="as7d900as8" version="0" call-id="987@1.2.3.4"
        local-uri="conference3@example.com"
        local-tag="123" remote-uri="sip:jdrosen@dynamicsoft.com"
        remote-tag="abc" direction="recipient"/>
    <media-status>
      <media-stream type="audio"/>
    </media-status>
  </user>
  <user uri="sip:hgs@cs.columbia.edu" name="Henning Schulzrinne">
    <status value="active"/>
    <dialog id="as7d900as8" version="0" call-id="654@8.8.7.7"
        local-uri="conference3@example.com"
        local-tag="xyz" remote-uri="sip:hgs@cs.columbia.edu"
        remote-tag="efg" direction="recipient"/>
  </user>
</conference>
```

This document describes a conference with two users, both of which are active.

6 Relationship to User Presence

The SIP events package for user presence [3] has a close relationship with these two event packages. It is fundamental to the presence model that the information used to obtain user presence is constructed from any number of different input sources. Examples of such sources include SIP REGISTER requests and uploads of presence documents. These two packages can be considered another mechanism that allows a presence agent to determine the presence state of the user. Specifically, a user presence server can act as a subscriber for the dialog and conference packages to obtain additional information that can be used to construct a presence document.

7 Open Issues and To-Dos

- There is a strong relationship between the dialog event package, and the notifications used by the REFER specification [8]. Should these be unified, so that a REFER basically implies a subscription to the dialog state created by that REFER?.

- Reuse of dialogs for conference and dialog subscriptions needs to be discussed. It has an implication for the dialog state package. Now, the session may be terminated, but the dialog remains.

- Need to add IANA considerations

- Should we split this into two documents, or even four? Probably two.

8 Security Considerations

Subscriptions to dialog state and conference state can reveal very sensitive information. For this reason, the document recommends authentication and authorization, and provides guidelines on sensible authorization policies.

Since the data in notifications is sensitive as well, end-to-end SIP encryption mechanisms using S/MIME SHOULD be used to protect it.

9 IANA Considerations

TODO.

10 Acknowledgements

The authors would like to thank Dan Petrie for his comments.

11 Changes since -00

- Alignment with bis and sip-events
- Added direction attribute to dialog format
- Removed To-Join and To-Replace header, along with joining and replacing URIs from the various formats.
- Conference data format reuses dialog formats
- Added media mixing information to conference format
- Removal of example services (will go into service examples specification)
- Removal of floor control from conference package; rather, place it into a separate event package, as was done in

12 Authors Addresses

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Avenue
First Floor
East Hanover, NJ 07936
email: jdrosen@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: schulzrinne@cs.columbia.edu

13 Normative References

[1] A. Roach et al. , "SIP-specific event notification," Internet Draft, Internet Engineering Task Force, Feb. 2002. Work in progress.

[2] J. Rosenberg, H. Schulzrinne, et al. , "SIP: Session initiation

protocol," Internet Draft, Internet Engineering Task Force, Feb. 2002. Work in progress.

14 Informative References

[3] J. Rosenberg, "SIP extensions for presence," Internet Draft, Internet Engineering Task Force, Nov. 2001. Work in progress.

[4] J. Rosenberg, "A SIP event sub-package for watcher information," Internet Draft, Internet Engineering Task Force, July 2001. Work in progress.

[5] R. Mahy and I. Slain, "SIP event package for message waiting indication," Internet Draft, Internet Engineering Task Force, Nov. 2001. Work in progress.

[6] M. Murata, S. S. Laurent, and D. Kohn, "XML media types," Request for Comments 3023, Internet Engineering Task Force, Jan. 2001.

[7] J. Rosenberg and H. Schulzrinne, "Models for multi party conferencing in SIP," Internet Draft, Internet Engineering Task Force, Nov. 2001. Work in progress.

[8] R. Sparks, "The refer method," Internet Draft, Internet Engineering Task Force, Oct. 2001. Work in progress.