SIPPING
Internet-Draft
Expires: August 15, 2004

E. Burger (Ed.)
J. Van Dyke
A. Spitzer
SnowShore Networks, Inc.
February 15, 2004

Basic Network Media Services with SIP
draft-burger-sipping-netann-08

Status of this Memo

Copyright Notice

Abstract

In SIP-based networks, there is a need to provide basic network media services. Such services include network announcements, user interaction, and conferencing services. These services are basic building blocks, from which one can construct interesting applications. In order to have interoperability between servers offering these building blocks (also known as Media Servers) and application developers, one needs to be able to locate and invoke such services in a well-defined manner.

This document describes a mechanism for providing an interoperable protocol interface between Application Servers, which provide

application services to SIP-based networks, and Media Servers, which provide the basic media processing building blocks.

Conventions used in this document

RFC2119 [1] provides the interpretations for the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" found in this document.

Table of Contents

1. Overview

In SIP-based media networks (RFC3261 [2]), there is a need to provide basic network media services. Such services include playing announcements, initiating a media mixing session (conference), and prompting and collecting information with a user.

These services are basic in nature, are few in number, and fundamentally have not changed in 25 years of enhanced telephony services. Moreover, given their elemental nature, one would not expect them to change in the future.

Multifunction media servers provide network media services to clients using server protocols such as SIP, often in conjunction with markup languages such as VoiceXML [9], KPML [10], and MSCML [11]. This document describes how to identify to a multifunction media server what sort of session the client is requesting, without modifying the SIP protocol.

Announcements are media played to the user. Announcements can be static media files, media files generated in real-time, media streams generated in real-time, multimedia objects, or combinations of the above.

Media mixing is the act of mixing different RTP streams, as described in RFC1889 [12]. Note that the service described here suffices for simple mixing of media for a basic conferencing service. This service does not address enhanced conferencing services, such as floor control, gain control, muting, subconferences, etc. MSCML [11] addresses enhanced conferencing. However, that is beyond the scope of this document. Interested readers should read conferencing-framework [13] for details on the IETF SIP conferencing framework.

Prompt and collect is where the server prompts the user for some information, as in an announcement, and then collects the user's response. This can be a one-step interaction, for example by playing an announcement, "Please enter your pass code", followed by collecting a string of digits. It can also be a more complex interaction, specified, for example, by VoiceXML [9] or MSCML [11].

2. Mechanism

In the context of SIP control of media servers, we take advantage of the fact that the standard SIP URI has a user part. Multifunction media servers do not have users. Thus we use the user address, or the left-hand-side of the URI, as a service indicator.

Note that the set of services is small, well defined, and well contained. The section The User Part (Section 6) discusses the issues with using a fixed set of user-space names.

For per-service security, the media server SHOULD use the security protocols described in RFC3261 [2].

The media server MAY issue 401 challenges for authentication. The media server SHOULD support the sips: scheme for the announcement service. The media server MUST support the sips: scheme for the dialog and conference services. The level of authentication to require for each service is a matter of local policy.

The media server, upon receiving the INVITE, notes the service indicator. Depending on the service indicator, the media server will either honor the request or return a failure response code.

The service indicator is the concatenation of the service name and an optional service instance identifier, separated by an equal sign.

Per RFC3261 [2], the service indicator is case insensitive. The service name MUST be from the set alphanumeric characters plus dash (US-ASCII %2C). The service name MUST NOT include an equal sign (US-ASCII %3D).

The service name MAY have long- and short-forms, as SIP does for headers.

A given service indicator MAY have an associated set of parameters. Such parameters MUST follow the convention set out for SIP URI parameters. That is, a semi-colon separated list of keyword=value pairs.

Certain services may have an association with a unique service instance on the media server. For example, a given media server can host multiple, separate conference sessions. To identify unique service instances, a unique identifier modifies the service name. The unique identifier MUST meet the rules for a legal user part of a SIP URI. An equal sign, US-ASCII %3D, MUST separate the service indicator from the unique identifier.

Note that since the service indicator is case insensitive, the service instance identifier is also case insensitive.

The requesting client issues a SIP INVITE to the media server, specifying the requested service and any appropriate parameters.

If the media server can perform the requested service, it does so,

following the processing steps described in the service definition document.

If the media server cannot perform the requested service or does not recognize the service indicator, it MUST respond with the response code 488 NOT ACCEPTABLE HERE. This is appropriate, as 488 refers to a problem with the user part of the URI. Moreover, 606 is not appropriate, as some other media server may be able to satisfy the request. RFC3261 [2] describes the 488 and 606 response codes.

Some services require a unique identifier. Most services automatically create a service instance upon the first INVITE with the given identifier. However, if a service requires an existing service instance, and no such service instance exists on the media server, the media server MUST respond with the response code 404 NOT FOUND. This is appropriate as the service itself exists on the media server, but the particular service instance does not. It is as if the user was not home.

3. Announcement Service

A network announcement is the delivery of a multimedia resource, such as a prompt file, to a terminal device. Note the multimedia resource may be any multimedia object that the media server supports. This service can play a single object with multiple streams, such as a video and audio prompt. However, this service cannot play multiple objects on the same SIP dialog.

There are two types of network announcements. The differentiating characteristic between the two types is whether the network fully sets up the SIP dialog before playing the announcement. The analog in the PSTN is whether answer supervision is supplied; i.e. does the announcement server answer the call prior to delivering the announcement.

Playing an announcement after call setup is straightforward. First, the requesting device issues an INVITE to the media server requesting the announcement service. The media server negotiates the SDP and responds with a 200 OK. After receiving the ACK from the requesting device, the media server plays the requested object and issues a BYE to the requesting device.

If the media server supports announcements, but it cannot find the referenced URI, it MUST respond with the 404 NOT FOUND response code.

If the media server receives an INVITE for the announcement service without a "play=" parameter, it MUST respond with the 404 NOT FOUND response code, as there is no default value for the announcement

service.

If there is an error retrieving the announcement, the media server MUST respond with a 404 NOT FOUND response code. In addition, the media server SHOULD include a Warning header with appropriate explanatory text explaining what failed.

The Request URI fully describes the announcement service through the use of the user part of the address and additional URI parameters. The user portion of the address, "annc", specifies the announcement service on the media server. The service has several associated URI parameters that control the content and delivery of the announcement. These parameters are described below:

play Specifies the resource or announcement sequence to be played.

repeat Specifies how many times the media server should repeat the announcement or sequence named by the "play=" parameter.

delay Specifies a delay interval between announcement repetitions. The delay is measured in milliseconds.

duration Specifies the maximum duration of the announcement. The media server will discontinue the announcement and end the call if the maximum duration has been reached. The duration is measured in milliseconds.

locale Specifies the language and optionally country variant of the announcement sequence named in the "play=" parameter. The language is defined as a two-letter code per ISO 639-1 [3]. The country variant is also defined as a two-letter code per ISO 3166-1 [4]. These elements are concatenated with a single under bar (%x5F) character, such as "en_CA". If only the language is specified, such as locale=en, the choice of country variant is an implementation matter. Implementations SHOULD provide the best possible match between the requested locale and the available languages in the event the media server cannot honor the locale request precisely. For example, if the request has locale=ca_FR but the media server only has fr_FR available, the media server should use the fr_FR variant. Implementations SHOULD provide a default locale to use if no language variants are available.

param[n] Provides a mechanism for passing values that are to be substituted into an announcement sequence. Up to 9 parameters ("param1=" through "param9=") may be specified. The mechanics of announcement sequences are beyond the scope of this document.

extension Provides a mechanism for extending the parameter set. If the media server receives an extension it does not understand, it MUST silently ignore the extension parameter and value.

The "play=" parameter is mandatory and MUST be present. All other parameters are OPTIONAL.

NOTE: Some encodings are not self-describing. Thus the

implementation relies on filename extension conventions for determining the media type.

Note that RFC3261 [2] implies that proxies are supposed to pass parameters through unchanged. However, be aware that non-conforming proxies may strip Request-URI parameters. That said, given the likely scenarios for the mechanisms presented in this document, this should not be an issue. Most likely, the proxy inserting the parameters is the last proxy before the media server. If the service provider deploys a proxy for load balancing or service location purposes, the service provider should ensure their choice of proxy preserves parameters.

The form of the SIP Request URI for announcements is as follows. Note that the backslash, CRLF, and spacing before the "play=" in the example is for readability purposes only.

```
sip:annc@ms2.example.net; \
  play="http://audio.example.net/allcircuitsbusy.g711"

sip:annc@ms2.example.net; \
  play="file://fileserver.example.net/geminii/yourHoroscope.wav"
```

3.1 Operation

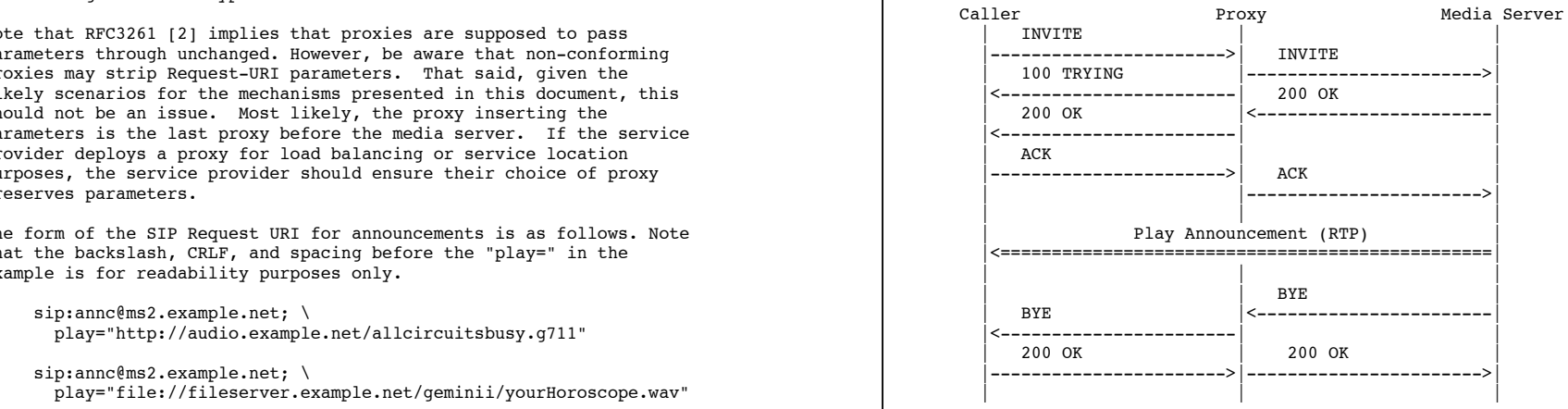
The scenarios below assume there is a SIP Proxy, application server, or media gateway controller between the caller and the media server. However, the announcement service works as described below even if the caller invokes the service directly. We chose to discuss the proxy case, as it will be the most common case.

The caller issues an INVITE to the serving SIP Proxy. The SIP Proxy determines what audio prompt to play to the caller. The proxy responds to the caller with 100 TRYING.

The proxy issues an INVITE to the media server, requesting the appropriate prompt to play coded in the play= parameter. The media server responds with 200 OK. The proxy relays the 200 OK to the caller. The caller then issues an ACK. The proxy then relays the ACK to the media server.

With the call established, the media server plays the requested prompt. When the media server completes the play of the prompt, it issues a BYE to the proxy. The proxy then issues a BYE to the caller.

3.2 Protocol Diagram



3.3 Formal Syntax

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC2234 [5].

```
ANNC-URL        = sip-ind annc-ind "@" hostport
                      annc-parameters uri-parameters

sip-ind         = "sip:" / "sips:"
annc-ind        = "annc"

annc-parameters = ";" play-param [ ";" content-param ]
                                 [ ";" delay-param]
                                 [ ";" duration-param ]
                                 [ ";" repeat-param ]
                                 [ ";" locale-param ]
                                 [ ";" variable-params ]
                                 [ ";" extension-params ]

play-param      = "play=" prompt-url

content-param   = "content-type=" MIME-type

delay-param     = "delay=" delay-value
```

```
delay-value     = 1*DIGIT

duration-param  = "duration=" duration-value

duration-value  = 1*DIGIT

repeat-param    = "repeat=" repeat-value

repeat-value    = 1*DIGIT

locale-param    = "locale=" locale-value

locale-value    = 2ALPHA [ %x5F 2ALPHA ]
                      ; ISO639-1_ISO3166-1
                      ; e.g., en, en_US, en_UK, etc.

variable-params = param-name "=" variable-value

param-name      = "param" DIGIT ; e.g., "param1"

variable-value  = 1*(ALPHA | DIGIT)

extension-params = extension-param [ ";" extension-params ]

extension-param  = token "=" token
```

"uri-parameters" is the SIP Request-URI parameter list as described in RFC3261 [2]. All parameters of the Request URI are part of the URI matching algorithm.

The MIME-type is the MIME [6] content type for the announcement, such as audio/basic, audio/G729, audio/mpeg, video/mpeg, and so on.

To date, none of the IETF audio MIME registrations have parameters. Vendor-specific registrations, such as audio/x-wav, do have parameters. However, they are not strictly needed for prompt fetching.

On the other hand, the prevalence of parameters may change in the future. In addition, existing video registrations have parameters, such as video/DV. To accommodate this, and retain compatibility with the SIP URI structure, the MIME-type parameter separator (semicolon, %3b) and value separator (equal, %d3) MUST be escaped. For example:

```
sip:annc@ms.example.net; \
    play=file://fs.example.net/clips/my-intro.dvi; \
    content-type=video/mpeg%3bencode%d3314M-25/625-50
```


The locale-value consists of a 2-letter language code as specified in ISO 639-1 [3] and a 2-letter country code specified in ISO 3166-1 [4] separated by a single under bar (%x5Fh) character.

The definition of hostport is as specified by RFC3261 [2].

The syntax of prompt-url consists of a URL scheme as specified by RFC2396 [7] or a special token indicating a provisioned announcement sequence. For example, the URL scheme MAY include any of the following.

- http/https
- ftp
- file (referencing a local, NFS (RFC3010 [14]), or AFS file)
- nfs (RFC2224 [15])

If a provisioned announcement sequence is to be played the value of prompt-url will have the following form:

```
prompt-url      = "/provisioned/" announcement-id

announcement-id = 1*(ALPHA | DIGIT)
```

Note that the scheme "/provisioned/" was chosen because of a hesitation to register a "provisioned:" URI scheme.

This document is strictly focused on the SIP interface for the announcement service and as such does not detail how announcement sequences are provisioned or defined.

Note that the media type of the object the prompt-url refers to can be most anything, including audio file formats, text file formats, or URI lists. See the Prompt and Collect Service (Section 4) section for more on this topic.

4. Prompt and Collect Service

This service is also known as a voice dialog. It establishes an aural dialog with the user.

The dialog service follows the model of the announcement service. However, the service indicator is "dialog". The dialog service takes a parameter, voicexml=, indicating the URI of the VoiceXML script to execute.

```
sip:dialog@mediaserver.example.net; \
    voicexml=http://vxmlserver.example.net/cgi-bin/script.vxml
```

A Media Server MAY accept additional SIP request URI parameters and

deliver them to the VoiceXML interpreter session as session variables.

Although not good VoiceXML programing practice, VoiceXML scripts might contain sensitive information, such as a user's pass code in a DTMF grammar. Thus the media server MUST support the https scheme for the voicexml parameter for secure fetching of scripts. Likewise, dynamic grammars often do have user identifying information. As such, the VoiceXML browser implementation on the media server MUST support https fetching of grammars and subsequent documents.

Returned information often is sensitive. For example, the information could be financial information or instructions. Thus the media server MUST support https posting of results.

4.1 Formal Syntax for Prompt and Collect Service

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC2234 [5].

```
DIALOG-URL        = sip-ind dialog-ind "@" hostport
                        dialog-parameters

sip-ind           = "sip:" / "sips:"
dialog-ind        = "dialog"

dialog-parameters = ";" dialog-param [ vxml-parameters ]
                                     [ uri-parameters ]

dialog-param      = "voicexml=" dialog-url

vxml-parameters   = vxml-param [ vxml-parameters ]

vxml-param        = ";" vxml-keyword "=" vxml-value

vxml-keyword      = token

vxml-value        = token
```

The dialog-url is the URI of the VoiceXML script. If present, other parameters get passed to the VoiceXML interpreter session with the assigned vxml-keyword vxml-value pairs. Note that all vxml-keywords MUST have values.

If there is a vxml-keyword without a corresponding vxml-value, the media server MUST reject the request with a 400 BAD REQUEST response code. In addition, the media server MUST state "Missing VXML Value" in the reason phrase.

The media server presents the parameters as environment variables in the connection object. Specifically, the parameter appears in the connection.sip tree.

If the Media Server does not support the passing of keyword-value pairs to the VoiceXML interpreter session, it MUST ignore the parameters.

"uri_parameters" is the SIP Request-URI parameter list as described in RFC3261 [2]. All parameters in the parameter list, whether they come from uri-parameters or from vxml-keyworks, are part of the URI matching algorithm.

5. Conference Service

One identifies mixing sessions through their SIP request URIs. To create a mixing session, one sends an INVITE to a request URI that represents the session. If the URI does not already exist on the media server and the requested resources are available, the media server creates a new mixing session. If there is an existing URI for the session, then the media server interprets it as a request for the new session to join the existing session. The form of the SIP request URI for conferencing is:

sip:conf=uniqueIdentifier@mediaserver.example.net

The left-hand side of the request URI is actually the username of the request in the request URI and the To header. The host portion of the URI identifies a particular media server. The "conf" user name conveys to the media server that this is a request for the mixing service. The uniqueIdentifier can be any value that is compliant with the SIP URI specification. It is the responsibility of the conference control application to ensure the identifier is unique within the scope of any potential conflict.

In the terminology of the conferencing framework conferencing-framework [13], this URI convention tells the media server that the application server is requesting it to act as a Focus. The conf-id value identifies the particular focus instance.

As a focus in the conferencing framework, the media server MUST support the ";isfocus" parameter in the Request URI. Note however, that the presence or absence of the ";isfocus" parameter has no protocol impact at the media server.

It is worth noting that the conference URI shared between the application and media servers provides enhanced security, as the SIP control interface does not have to be exposed to participants. It

also allows the assignment of a specific media server to be delayed as long as possible, thereby simplifying resource management.

One can add additional legs to the conference by INVITEing them to the above mentioned request URI. Per the matching rules of RFC3261 [2], the conf-id parameter is part of the matching string.

Conversely, one can remove legs by issuing a BYE in the corresponding dialog. The mixing session, and thus the conference-specific request URI, remains active so long as there is at least one SIP dialog associated with the given request URI.

If the Request-URI has "conf" as the user part, but does not have a conf-id parameter, the media server MUST respond with a 404 NOT FOUND.

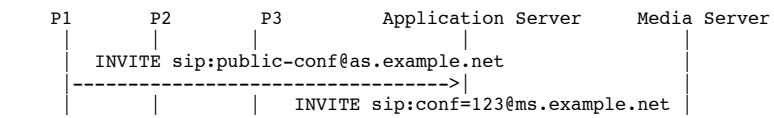
NOTE: The media server could create a unique conference instance and return the conf-id string to the UAC if there is no conf-id present. However, such an operation may have other operational issues, such as permissions and billing. Thus an application server or proxy is a better place to do such an operation. Moreover, such action would make the media server into a Conference Factory in the terminology of conference-framework [13]. That is not the appropriate behavior for a media server.

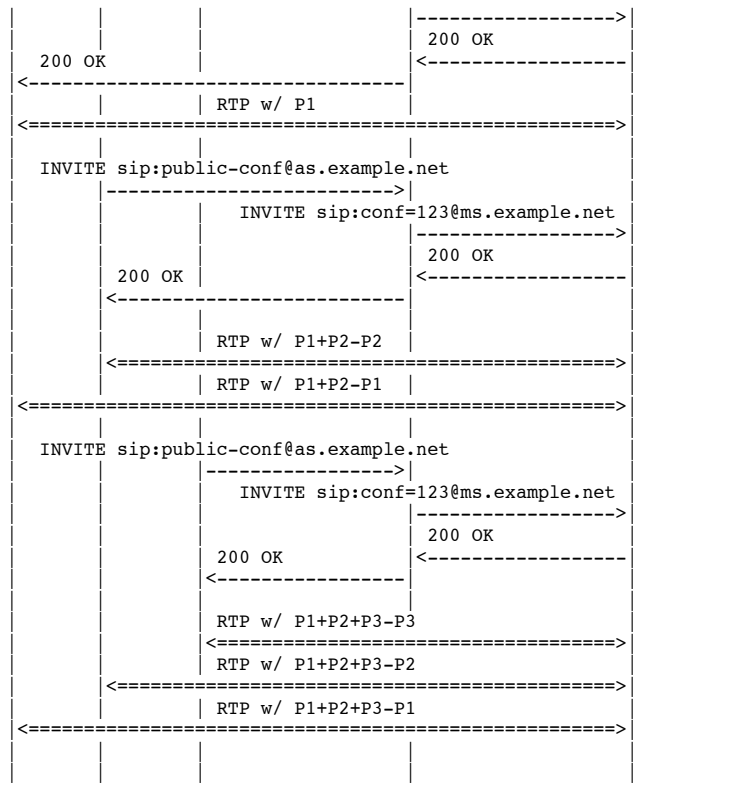
Since some conference use cases, such as business conferencing, have billing implications, the media server SHOULD authenticate the application server or proxy. At a minimum, the media server MUST implement sip: digest authentication and sips:.

5.1 Protocol Diagram

This diagram shows the establishment of a three-way conference. This section is informative. It is only one method of establishing a conference. This example shows a simple back-to-back user agent.

The conference-framework [13] describes additional parameters and behaviors of the Application Server. For example, the first INVITE from P1 to the Application Server would include the ";isfocus" parameter; the Application Server would act as a Conference Factory; and so on. However, none of that protocol machinery has an impact on the operation of the Application Server to Media Server interface, which is the focus of this protocol document.





Using the terminology of conference-framework [13], the Application Server is the Conference Factory and the Media Server is the Conference Focus.

Note that the above call flow does not show any 100 TRYING messages that would typically flow from the Application Server to the UAC's, nor does it show the ACK's from the UAC's to the Application Server or from the Application Server to the Media Server.

Each leg can drop out either under the supervision of the UAC by the UAC sending a BYE or under the supervision of the Application Server

by the Application Server issuing a BYE. In either case, the Application Server will either issue a BYE on behalf of the UAC or issue it directly to the Media Server, corresponding to the respective disconnect case.

It is left as a trivial exercise to the reader for how the Application Server can mute legs, create side conferences, and so forth.

Note that the Application Server is a server to the participants (UAC's). However, the Application Server is a client for mixing services to the Media Server.

5.2 Formal Syntax

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC2234 [5].

```
CONF-URL        = sip-ind conf-ind "=" instance-id "@" hostport
                  [ uri_parameters ]

sip-ind         = "sip:" / "sips:"

conf-ind        = "conf"

instance-id     = token
```

"uri-parameters" is the SIP Request-URI parameter list as described in RFC3261 [2]. All parameters in the parameter list are part of the URI matching algorithm.

6. The User Part

There has been considerable debate about the wisdom of using fixed user parts in a request URI. The most common objection is that the user part should be opaque and a local matter. The other objection is that using a fixed user part removes those specified user addresses from the user address space.

We will address the latter issue first. The common example is the Postmaster address defined by RFC2821 [16]. The objection is that by using the Postmaster token for something special, one removes that token for anyone. Thus, the Postmaster General of the United States, for example, cannot have the mail address Postmaster@usps.gov. One may debate whether this is a significant limitation, however.

One may point out that "annc", for example, has the potential for more conflict than Postmaster. This is true. However, one cannot

confuse the namespace at a Media Server with the namespace for an organization.

For example, let us take the case where a network offers services for "Ann Charles". She likes to use the name "annc", and thus she would like to use "sip:annc@example.net". We offer that there is ABSOLUTELY NO NAME COLLISION WHATSOEVER. Why is this so? This is so because sip:annc@example.net will resolve to the specific user at a specific device for Ann. As an example, example.net's SIP Proxy Server can resolve sip:annc@example.net to annc@anns-phone.example.net . One directs requests for the media service annc directly to the Media Server, e.g., sip:annc@ms21.ap.example.net . Moreover, by definition, Ann Charles, or anything other than the announcement service, will NEVER be directly on the Media Server. If that were not true, no phone in the world could use the user part "eburger", as eburger is a reserved user part in the SnowShore domain.

The most important thing to note about this convention is that the left-hand side of the request URI is opaque to the network. The only network elements that need to know about the convention are the Media Server and client.

Some have proposed that such naming be a pure matter of local convention. For example, the thesis of the informational RFC3087 [17] is that you can address services using a request URI. However, some have taken the examples in the document to an extreme. Namely, that the only way to address services is via arbitrary, opaque, long user parts. It is possible to provision the service names, rather than fixed names. While this can work in a closed network, where the Application Servers and Media Servers are in the same administrative domain, this does not work across domains. This is because the client of the media service has to know the local name for each service / domain pair. This is particularly onerous for situations where there is an ad hoc relationship between the application and the media service. Without a well-known relationship between service and service address, how would the client locate the service?

One very important result of using the user part as the service descriptor is that we can use all of the standard SIP machinery, without modification. For example, Media Servers with different capabilities can SIP Register their capabilities as users. For example, a mixing-only device will register the "conf" user, while a multi-purpose Media Server will register all of the users. Note that this is why the URI to play is a parameter. Doing otherwise would overburden a normal SIP proxy or redirect server. Likewise, this scheme lets us leverage the standard SIP proxy behavior of using an intelligent redirect server or proxy server to provide high-available

services. For example, two Media Servers can register with a SIP redirect server for the annc user. If one of the Media Servers fails, the registration will expire and all requests for the announcement service ("calls to the annc user") get sent to the surviving Media Server.

7. Security Considerations

Exposing network services with well-known addresses may not be desirable. The Media Server SHOULD authenticate and authorize requesting endpoints per local policy.

Some interactions in this document result in the transfer of confidential information. Moreover, many of the interactions require integrity protection. Thus the Media Server MUST implement digest authentication for the sip: scheme and MUST implement the sips: scheme. in addition, application developers are RECOMMENDED to use the security services offered by the Media Server to ensure the integrity and confidentiallity of their user's data, as appropriate.

Untrusted network elements could use the protocol described here for providing information services. Many extant billing arrangements are for completed calls. Successful call completion occurs with a 2xx result code. This can be an issue for the early media announcement service. This is one of the reasons why the early media announcement service is deprecated.

8. Contributors

Jeff Van Dyke and Andy Spitzer of SnowShore did just about all of the work developing netann, in conjunction with many application developers, media server manufacturers, and service providers, some of whom are listed in the Acknowledgements section. All I did was do the theory and write it up. That also means all of the mistakes are mine, as well.

9. Acknowledgements

We would like to thank Kevin Summers and Ravindra Kabre of Sonus Networks for their constructive comments, as well as Jonathan Rosenberg of Dynamicsoft and Tim Melanchuk of Convedia for their encouragement. In addition, the discussion at the Las Vegas Interim Workgroup Meeting in 2002 was invaluable for clearing up the issues surrounding the left-hand-side of the request URI. Christer Holmberg helped tune the language of the multimedia announcement service. Orit Levin from Radvision gave a close read on the most recent version of the draft document. Pete Danielsen from Lucent has consistently provided excellent reviews of the many of the different versions of this document.

Pascal Jalet provided the theoretical underpinning and David Rio provided the experimental evidence for why the conference identifier belongs in the user part of the request-URI.

I am particularly indebted to Alan Johnston for his review of this document and ensuring its conformance with the SIP conference control work in the IETF.

The authors would like to give a special thanks to Walter O'Connor for doing much of the initial implementation.

Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[3] ISO, "Codes for the representation of names of languages -- Part 1: Alpha-2 code", ISO 639-1, July 2002.

[4] ISO, "Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes", ISO 3166-1, October 1997.

[5] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[6] Borenstein, N. and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, September 1993.

[7] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[8] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.

Informative References

[9] World Wide Web Consortium, "Voice Extensible Markup Language (VoiceXML) Version 2.0", W3C Candidate Recommendation ,

February 2003, <http://www.w3.org/TR/2003/
CR-voicexml20-20030220/>.

[10] Burger, E. and M. Dolly, "Keypad Stimulus Protocol (KPML)", draft-ietf-sipping-kpml-02 (work in progress), February 2004.

[11] Burger, E., Van Dyke, J. and A. Spitzer, "Media Server Control Markup Language and Protocol", draft-vandyke-mscml-03 (work in progress), July 2003.

[12] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[13] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-00 (work in progress), May 2003.

[14] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M. and D. Noveck, "NFS version 4 Protocol", RFC 3010, December 2000.

[15] Callaghan, B., "NFS URL Scheme", RFC 2224, October 1997.

[16] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[17] Campbell, B. and R. Sparks, "Control of Service Context using SIP Request-URI", RFC 3087, April 2001.

[18] Handley, M., Schulzrinne, H., Schooler, E. and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.

[19] Charlton, N., Gasson, M., Gybels, G., Spanner, M. and A. van Wijk, "User Requirements for the Session Initiation Protocol (SIP) in Support of Deaf, Hard of Hearing and Speech-impaired Individuals", RFC 3351, August 2002.

Authors' Addresses

Eric Burger
SnowShore Networks, Inc.
285 Billerica Rd.
Chelmsford, MA 01824-4120
USA

EMail: e.burger@ieee.org

Jeff Van Dyke
SnowShore Networks, Inc.
285 Billerica Rd.
Chelmsford, MA 01824-4120
USA

EMail: jvandyke@snowshore.com

Andy Spitzer
SnowShore Networks, Inc.
285 Billerica Rd.
Chelmsford, MA 01824-4120
USA

EMail: woof@snowshore.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING Working Group G. Camarillo
Internet-Draft Ericsson
Expires: August 6, 2004 February 6, 2004

Ad-Hoc URI List Management in the Session Initiation Protocol (SIP)

draft-camarillo-sipping-adhoc-management-00.txt

Status of this Memo

Copyright Notice

Abstract

This document defines two mechanisms to manage ad-hoc URI lists in SIP. In the first mechanism, the user agent sends an updated version of the entire list to the server. In the second mechanism, the server provides the user agent with a URI (e.g., http) that can be used to manipulate the list using an out-of-band mechanim (e.g., XCAP). We define the Associated-List-Manipulation header field that carries a URI that allows manipulating an ad-hoc list.

Table of Contents

1. Introduction

SIP messages can carry URI lists using the "list" SIP and SIPS URI parameter defined in [3]. An application server receiving a SIP request with a URI list creates a so called ad-hoc URI list, which is valid for the duration of the service provided by the server.

Once an ad-hoc URI list is created at the server, the user agent may need to manipulate it (e.g., add URIs to the list and remove URIs from the list). Section 3 and Section 4 describe two methods to perform ad-hoc URI list management.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [1] and indicate requirement levels for compliant implementations.

3. List Substitution

A user agent MAY provide an application server with an updated version of the ad-hoc list by sending a request with a "list" parameter [3] in its Request-URI. The "list" parameter MUST contain a pointer to the updated list. (The method of this request depends on the service being delivered.) On reception of such a request, the application server MUST substitute the previous ad-hoc list with the list referenced by the "list" parameter.

4. Out-of-Band Management

Section 3 describes how to send a complete URI list to an application server that substitutes the previous one. Following this approach, a user agent that wants to modify a single URI in a long URI list needs to resend the whole list.

Still, there are URI list management mechanisms, such as the XCAP usage defined in [2], that allow user agents to manipulate URI lists more efficiently. We define a new SIP header field called Associated-List-Manipulation that allows a server to provide a URI to the client to manipulate the ad-hoc list using an out-of-band mechanism. The XCAP Usage for Resource Lists MUST be supported. Other mechanisms MAY be supported.

The ABNF of the Associated-List-Manipulation header field is:


```
List-Manipulation = "Associated-List-Manipulation" HCOLON
                    absoluteURI
```

5. Examples

This section shows how to use the mechanisms described in Section 3 and Section 4 to manipulate the list of participants in an ad-hoc conference. This example illustrates the use of both mechanisms. It does not mandate how ad-hoc conference services have to be implemented.

When the ad-hoc conferencing server in this example receives an initial INVITE with a URI list, it sends out an INVITE to each URI in the list and creates an ad-hoc conference with all of them. If, at a later point, a URI is added to the list, the conference server INVITEs the new user. If a URI is removed from the list, the conference server BYEs the user.

Carol creates an ad-hoc conference on the server by sending the INVITE request shown in Figure 1. The list parameter in the Request-URI points to a MIME body that carries the list of participants.

```
INVITE sip:ad-hoc@example.com;list=cid:cn35t8jf02@example.com SIP/2.0
Via: SIP/2.0/TCP client.chicago.example.com
    ;branch=z9hG4bKhjhs8ass83
Max-Forwards: 70
To: "Ad-Hoc Conferences" <sip:ad-hoc@example.com>
From: Carol <sip:carol@chicago.example.com>;tag=32331
Call-ID: d432fa84b4c76e66710
CSeq: 1 INVITE
Contact: <sip:carol@client.chicago.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
     SUBSCRIBE, NOTIFY
Allow-Events: dialog
Accept: application/sdp, message/sipfrag,
     application/resource-lists+xml
Conten-Type: multipart/mixed;boundary="boundary1"
Content-Length: 731

--boundary1
Content-Type: application/sdp
Content-Length: 160

v=0
o=carol 2890844526 2890842807 IN IP4 chicago.example.com
```

```
s=Example Subject
c=IN IP4 192.0.0.1
t=0 0
m=audio 20000 RTP/AVP 0
m=video 20002 RTP/AVP 31

--boundary1
Content-Type: application/resource-lists+xml
Content-Length: 367
Content-ID: <cn35t8jf02@example.com>

<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <list name="ad-hoc-1">
    <entry name="1" uri="sip:bill@example.com" />
    <entry name="2" uri="sip:joe@example.com" />
    <entry name="3" uri="sip:ted@example.com" />
    <entry name="4" uri="sip:bob@example.com" />
  </list>
</resource-lists>
--boundary1--
```

Figure 1: INVITE request

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP client.chicago.example.com
     ;branch=z9hG4bKhjhs8ass83;received=192.0.2.4
To: "Ad-Hoc Conferences" <sip:ad-hoc@example.com>;tag=733413
From: Carol <sip:carol@chicago.example.com>;tag=32331
Call-ID: d432fa84b4c76e66710
CSeq: 1 INVITE
Contact: <sip:34@example.com>;isfocus
Associated-List-Manipulation: http://xcap.example.com/lists/yourlist
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
       SUBSCRIBE, NOTIFY
Allow-Events: dialog, conference
Accept: application/sdp, application/conference-info+xml,
       application/resource-lists+xml, message/sipfrag
Supported: replaces, join
Content-Type: application/sdp
Content-Length: 312

v=0
o=focus431 2890844526 2890842807 IN IP4 ms5.conf.example.com
s=Example Subject
i=Example Conference Hosted by Example.com
u=http://conf.example.com/3402934234
```



```
e=3402934234@conf-help.example.com
p=+1-888-555-1212
c=IN IP4 ms5.conf.example.com
t=0 0
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
```

Figure 2: 200 (OK) response

The conference server responds with the 200 (OK) in Figure 1, which carries the URI for the conference in its Contact header field and a URI for manipulating the URI list in its Associated-List-Manipulation header field.

5.1 List Substitution

Carol wants to remove Bill and Joe from the conference. She sends the re-INVITE in Figure 3 to the conference server with an updated URI list in a "list" parameter.

```
INVITE sip:34@example.com;isfocus;list=cid:cn35t8j@example.com SIP/2.0
Via: SIP/2.0/TCP client.chicago.example.com
    ;branch=z9hG4bKhjhs8ass83
Max-Forwards: 70
To: "Ad-Hoc Conferences" <sip:ad-hoc@example.com>
From: Carol <sip:carol@chicago.example.com>;tag=32331
Call-ID: d432fa84b4c76e66710
CSeq: 2 INVITE
Contact: <sip:carol@client.chicago.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
     SUBSCRIBE, NOTIFY
Allow-Events: dialog
Accept: application/sdp, message/sipfrag,
     application/resource-lists+xml
Conten-Type: multipart/mixed;boundary="boundary1"
Content-Length: xxx

--boundary1
Content-Type: application/sdp
Content-Length: 160

v=0
o=carol 2890844526 2890842807 IN IP4 chicago.example.com
s=Example Subject
c=IN IP4 192.0.0.1
t=0 0
m=audio 20000 RTP/AVP 0
```

```
m=video 20002 RTP/AVP 31

--boundary1
Content-Type: application/resource-lists+xml
Content-Length: xxx
Content-ID: <cn35t8j@example.com>

<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <list name="ad-hoc-1">
    <entry name="3" uri="sip:ted@example.com" />
    <entry name="4" uri="sip:bob@example.com" />
  </list>
</resource-lists>
--boundary1--
```

Figure 3: Re-INVITE

5.2 Out-of-Band Management

Now, Carol wants to add Alice to the conference. This time, she uses the http URI received in the Associated-List-Manipulation header field. She uses XCAP to add Alice's URI, so no SIP traffic is exchanged between her and the server.

6. Security Considerations

TBD.

7. IANA Considerations

This document registers the Associated-List-Manipulation SIP header field, which is described in Section 4. This header field is to be added to the header field registry under http://www.iana.org/assignments/sip-parameters.

Header Name: Associated-List-Manipulation

Compact Form: (none)

8. Acknowledgments

Adam Roach, Jonathan Rosenberg, and Orit Levin provided useful comments on this document.

Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Rosenberg, J., "An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Usage for Presence Lists", draft-ietf-simple-xcap-list-usage-01 (work in progress), October 2003.

[3] Camarillo, G., "Providing a Session Initiation Protocol (SIP) Application Server with a List of URIs", draft-camarillo-sipping-uri-list-00 (work in progress), November 2003.

Author's Address

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Gonzalo.Camarillo@ericsson.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING Working Group G. Camarillo
Internet-Draft Ericsson
Expires: August 6, 2004 February 6, 2004

Requirements and Framework for Session Initiation Protocol (SIP) Exploder Invocation
draft-camarillo-sipping-exploders-02.txt

Abstract

This document describes the need for SIP exploders and provides requirements for their invocation. Additionaly, it defines a framework which includes all the SIP extensions needed to meet these requirements.

Table of Contents

1. Introduction

Some applications require that, at a given moment, a SIP UA performs a similar transaction with a number of remote UAs. For example, an instant messaging application that needs to send a particular message (e.g., "Hello folks") to n receivers needs to send n MESSAGE requests; one to each receiver.

When the transacton that needs to be repeated consists of a large request, or the number of recipients is high, or both, the access network of the UA needs to carry a considerable amount of traffic. Completing all the transactions on a low-bandwidth access would require a long time. This is unacceptable for a number of applications.

A solution to this problem consists of introducing exploders in the network. The task of an exploder is to receive a request from a UA and send a number of similar requests to a number of destinations. Once the requests are sent, the exploder needs to inform the UA about their status. Effectively, the exploder behaves as a B2BUA.

Note that resource lists, as described in [2], already use SIP exploders for SUBSCRIBE transactions. Still, the set of destinations needs to be preconfigured using out-of-band mechanisms (e.g., XCAP).

The Advanced Instant Messaging Requirements for SIP [3] also mentions the need for exploders for MESSAGE transactions:

"REQ-GROUP-3: It MUST be possible for a user to send to an ad-hoc group, where the identities of the recipients are carried in the message itself."

The remainder of this document provides requirements to invoke exploders in an efficient manner and a framework that meets these requirements.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [1] and indicate requirement levels for compliant implementations.

3. Requirements

This section contains the requirements:

1. The invocation mechanism MUST allow the invoker to provide a list of destination URIs to the exploder. This URI list MAY consist of one or more URIs.

2. It MUST be possible to send URI list "deltas" to update the list of URIs handled by the exploder.

3. The invocation mechanism MUST NOT be request specific.

4. The invocation mechanism SHOULD NOT require more than one RTT.

5. An exploder MAY provide services beyond request explosion. That is, exploders can be modelled as application servers. For example, an exploder handling INVITE requests may behave as a conference server and perform media mixing for all the participants.

6. The interpretation of the meaning of the URI list sent by the invoker MUST be at the discretion of the application to which the list is sent.

7. It MUST be possible for the invoker to find out about the result of the operations performed by the application with the URI list. An invoker may, for instance, be interested in the status of the transactions initiated by the exploder.

8. It MUST be possible for the application that makes use of a list of URIs to convey the list of URIs to any recipients of messages created by the application from that list. OPEN ISSUE: do we really need this requirement?

9. Exploders MUST NOT perform any request explosion without authenticating the invoker.

10. The UA MUST be able to provide credentials to the exploder so that the exploder can use them to prove to the destinations that it is sending requests on behalf of the UA.

4. Framework

Although Section 3 contains specific requirements for SIP exploders, this framework is not restricted to application servers that only provide request explosion services. We also deal with application servers that provide a particular service that includes a request explosion (e.g., a conference server that INVITEs several participants which are chosen by a user agent).

We need to use several SIP extensions to meet the requirements in Section 3. We list these extensions in the following sections and explain which role they play within the framework.

4.1 Carrying URI Lists in SIP

User agents can send a list of URIs to an application server using the list SIP and SIPS URI parameter defined in (draft-camarillo-sipping-uri-list-01). The user agent adds a list parameter to the Request-URI of the SIP request sent to the application server. This parameter contains a pointer to a URI list, which can be carried in the SIP request itself or can be stored in an external server (e.g., an http URI pointing to an XCAP resource list). The way the application server interprets the URI list received in the request is service specific.

4.2 Managing Ad-Hoc URI Lists

An application server that receives a request with a URI list (or a pointer to it) creates a so called ad-hoc list, whose lifetime depends on the service provided by the server. Services that involve ad-hoc lists that are valid for a period of time need to allow user agents to modify these lists.

A user agent can manage ad-hoc lists at a server in two ways, as described in (draft-camarillo-sipping-adhoc-management-00): using SIP or using an external means (e.g., XCAP).

User agents using SIP to manage ad-hoc lists send a new SIP request with a pointer to a new list that will substitute the old list.

User agents using an external means to manage ad-hoc lists need to obtain from the server a URI that allows them to manipulate the list (e.g., an http URI pointing to an XCAP resource list). The server provides such a URI in an Associated-List-Manipulation header field in the response to the request that created the ad-hoc list.

4.3 Transaction State Information

User agents may be interested in the results of the message explosion at the application server. That is, user agents may want to know the result of the transactions that the application server initiated towards the URIs in the URI list provided by the user agent. The transaction state event package defined in (draft-camarillo-sipping-transac-package-00) provides this information to the user agent subscribing to this package.

Still, in order to subscribe to the transaction state event package,

the user agent needs a URI to subscribe to. The application server provides such a URI in an Associated-Transactions-State header field in the response to the request that triggered the new transactions, as defined in (draft-camarillo-sipping-transac-package-00).

4.4 Multiple REFER Targets

Building REFER requests with multiple REFER targets requires special considerations, as described in (draft-camarillo-sipping-multiple-refer-00). The Refer-To header field carries a pointer to a URI list, and the NOTIFIES carry transaction state information using the transaction state event package. User agents may use bodies whose disposition type is template to describe the messages to be sent by the application server.

A conferencing application is an example of an application that may use REFERs with multiple REFER targets. A user agent may send a REFER to the conferencing server so that the server BYEs a set of users.

5. Security Considerations

Requirements related to security are considered in Section 3.

TBD: this section should be expanded considerably.

6. Acknowledges

Duncan Mills and Miguel A. Garcia-Martin supported the idea of 1 to n MESSAGEs.

Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Informational References

[2] Roach, A., Rosenberg, J. and B. Campbell, "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists", draft-ietf-simple-event-list-04 (work in progress), June 2003.

[3] Rosenberg, J., "Advanced Instant Messaging Requirements for the Session Initiation Protocol (SIP)", draft-rosenberg-simple-messaging-requirements-00 (work in progress), December 2002.

Author's Address

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Gonzalo.Camarillo@ericsson.com

Intellectual Property Statement

Full Copyright Statement

HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING Working Group
Internet-Draft
Expires: August 6, 2004

G. Camarillo
Ericsson
A. Niemi
H. Khartabil
M. Isomaki
Nokia
February 6, 2004

Refering to Multiple Resources in the Session Initiation Protocol (SIP)

draft-camarillo-sipping-multiple-refer-00.txt

Status of this Memo

Copyright Notice

Abstract

This document defines extensions to the SIP REFER method so that this method can be used to refer to multiple resources. These extensions include the use of pointers to URI-lists in the Refer-To header field, the use of bodies to describe the requests to be sent by the server, and the use of a new event package to report the state of several transactions.

Table of Contents

1. Introduction

The need for exploders in SIP [2] is described in [6]. Mechanisms to invoke exploders in SIP need to meet the requirements listed there.

The SIP REFER method [4] allows a user agent to request a server to send a request to a third party. Still, a number of applications need to request a server to initiate transactions towards a set of destinations. We define several extensions to REFER so that REFER can be used to refer to multiple destinations (e.g., a user agent requesting a conferencing server to INVITE several new participants).

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [1] and indicate requirement levels for compliant implementations.

We define the following three new terms:

REFER-Issuer: the UA issuing the REFER request.

REFER-Recipient: the UA receiving the REFER request.

REFER-Target: the UA designated in the Refer-To URI.

3. Carrying Multiple Destinations

We represent the multiple REFER-Targets of a REFER using a URI list. We use the Refer-To header field to carry a pointer to that URI-list.

draft-camarillo-sipping-uri-list provides rules to carry a pointer to a URI list in a URI parameter called list. Refer-To header fields carring a pointer to a URI-list follow the same rules. That is, the Refer-To header field of REFER with multiple REFER-Targets MUST contain a URI that points to a URI list. The XCAP resource list format [5] MUST be supported; any other URI list formats MAY be supported.

The following is an example of a REFER with a Refer-To header field that points to a URI list which is carried in the message body. The option-tag "multiple-refer" in the Require header, which is defined in Section 5, ensures that the REFER-Recipient understands Refer-To header fields with pointers to URI lists.


```
REFER sip:b@atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP agenta.atlanta.example.com;branch=z9hG4bK2293
To: <sip:b@atlanta.example.com>
From: <sip:a@atlanta.example.com>;tag=193402342
Call-ID: 898234234@agenta.atlanta.example.com
CSeq: 1 REFER
Max-Forwards: 70
Require: multiple-refer
Refer-To: cid:cn35t8jf02@example.com
Contact: sip:a@atlanta.example.com
Accept: application/sdp, message/sipfrag,
     application/transaction-info+xml
Content-Type: application/resource-lists+xml
Content-Length: xxx
Content-ID: <cn35t8jf02@example.com>

<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <list name="ad-hoc-1">
    <entry name="1" uri="sip:bill@example.com" />
    <entry name="2" uri="sip:joe@example.com" />
    <entry name="3" uri="sip:ted@example.com" />
    <entry name="4" uri="sip:bob@example.com" />
  </list>
</resource-lists>
```

Figure 1

4. The Transaction State Event Package

REFER requests establish an implicit subscription to the "refer" event package, defined in [4]. The data format used in this event package is message/sipfrag, which is defined in [3]. The REFER specification says the following about the NOTIFIES triggered by the REFER's implicit subscription:

> Each NOTIFY MUST contain an Event header field with a value of refer and possibly an id parameter.
>
> Each NOTIFY MUST contain a body of type "message/sipfrag".

We keep the first statement, but relax the second statement (about the body format) so that the refer event package is aligned with any other event package. That is, the notifier can choose any format that is applicable to the event package and that appears in the Accept header field of the request that created the subscription (the REFER, in our case). The resulting normative statement is the following:

The notifications generated by the server MUST be in one of the formats specified in the Accept header field in the REFER request.

The default format, which MUST be supported by all UAs that generate REFERs with the option-tag "multiple-refer" in a Require header field, is "application/transaction-info+xml", as defined in (draft-camarillo-sipping-transac-package).

The following is an example of the body of a NOTIFY generated by the REFER-Recipient of the REFER in Figure 1.

```
<?xml version="1.0"?>
   <transaction-info xmlns="urn:ietf:params:xml:ns:transaction-info"
                version="0"
                state="full"
                entity="sip:b@atlanta.example.com">
     <transaction id="frgd870th87" r-uri="sip:bill@example.com">
       <state code="200">complete</state>
     </transaction>
     <transaction id="234f12345" r-uri="sip:joe@example.com">
       <state code="404">complete</state>
     </transaction>
     <transaction id="fghd2345" r-uri="sip:ted@example.com">
       <state code="200">complete</state>
     </transaction>
     <transaction id="12dvg2345" r-uri="sip:bob@example.com">
       <state code="200">complete</state>
     </transaction>
   </transaction-info>
```

Figure 2

5. The Multiple-Refer SIP Option-Tag

We define a new SIP option-tag for the Require and Supported header fields: multiple-refer.

A UA including the multiple-refer option-tag in a Supported header understands Refer-To header fields that point to URI lists and understands the "application/transaction-info+xml" body type.

A UA generating a REFER with a pointer to a URI-list in its Refer-To header field MUST include the multiple-refer option-tag in the Require header field of the REFER.

6. The Template Disposition-Type

When using REFER, the new request to be sent is described using URI parameters. For example, the following Refer-To header field contains the values of the Accept-Contact and Call-ID header fields of the new request.

```
Refer-To: <sip:bob@biloxi.example.net?Accept-Contact=sip:bobsdesk.
       biloxi.example.net&Call-ID%3D55432%40alicepc.atlanta.example.com>
```

REFERs with multiple REFER-Targets usually request that the REFER-Recipient sends a set of similar requests. Describing a set of similar requests by adding the same URI parameters to all the URIs in the definition of the URI list is not an efficient way to encode that information.

We define a new disposition-type: template. Bodies of this disposition-type (typically sipfrag bodies as defined in RFC 3420 [3]) provide the server with a template for the messages to be sent.

The following example shows a body whose disposition-type is template. It indicates that the requests to be sent should be MESSAGEs carrying the text "Hello world."

```
Content-Disposition: template;handling=required
Content-type: message/sipfrag
Content-Length: xxx

MESSAGE sip:whoever.invalid SIP/2.0
Content-Type: text/plain
Content-Length: 12

Hello World.
```

If any of the URIs defining the REFER-Targets has a URI parameter indicating a different value for a header field than the one indicated in the template, the exploder MUST use the value in the URI parameter.

Note that in order to include the method in a sipfrag body, it is necessary to include the Request-URI as well (the whole Request-line needs to be included as specified in RFC 3420 [3]. If the REFER request contains a single REFER-Target, the URI of the Request-Target SHOULD be placed in the Request-URI of the template body. Otherwise, it is RECOMMENDED that the Request-URI in the template body is an invalid URI.

OPEN ISSUE: we may want to define an option-tag for this disposition-type, or to include support for this disposition type in the multiple-refer option tag.

7. Example

We need to add the whole call flow.

```
REFER sip:b@atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP agenta.atlanta.example.com;branch=z9hG4bK2293
To: <sip:b@atlanta.example.com>
From: <sip:a@atlanta.example.com>;tag=193402342
Call-ID: 898234234@agenta.atlanta.example.com
CSeq: 1 REFER
Max-Forwards: 70
Require: multiple-refer
Refer-To: cid:cn35t8jf02@example.com
Contact: sip:a@atlanta.example.com
Accept: application/sdp, message/sipfrag,
     application/transaction-info+xml
Conten-Type: multipart/mixed;boundary="boundary1"
Content-Length: xxx

--boundary1
Content-Disposition: template;handing=required
Content-type: message/sipfrag
Content-Length: xxx

MESSAGE sip:whoever.invalid SIP/2.0
Content-Type: text/plain
Content-Length: 12

Hello World.

--boundary1
Content-Type: application/resource-lists+xml
Content-Length: xxx
Content-ID: <cn35t8jf02@example.com>

<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <list name="ad-hoc-1">
    <entry name="1" uri="sip:bill@example.com" />
    <entry name="2" uri="sip:joe@example.com" />
    <entry name="3" uri="sip:ted@example.com" />
    <entry name="4" uri="sip:bob@example.com" />
  </list>
```



```
</resource-lists>
--boundary1--
```

8. Security Considerations

TBD

9. IANA Considerations

TBD: we need to register the multiple-refer option-tag and the template disposition type.

Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[3] Sparks, R., "Internet Media Type message/sipfrag", RFC 3420, November 2002.

[4] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[5] Rosenberg, J., "An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Usage for Presence Lists", draft-ietf-simple-xcap-list-usage-01 (work in progress), October 2003.

Informational References

[6] Camarillo, G., "Requirements for Session Initiation Protocol (SIP) Exploder Invocation", draft-camarillo-sipping-exploders-01 (work in progress), November 2003.

Authors' Addresses

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Gonzalo.Camarillo@ericsson.com

Aki Niemi
Nokia
P.O. Box 321
NOKIA GROUP, FIN 00045
Finland

EMail: Aki.Niemi@nokia.com

Hisham Khartabil
Nokia
P.O. Box 321
NOKIA GROUP, FIN 00045
Finland

EMail: Hisham.Khartabil@nokia.com

Markus Isomaki
Nokia
Itamerenkatu 11-13
Helsinki 00180
Finland

EMail: Markus.Isomaki@nokia.com

Intellectual Property Statement

Full Copyright Statement

HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING Working Group
Internet-Draft
Expires: August 6, 2004

G. Camarillo
Ericsson
M. Garcia-Martin
Nokia
February 6, 2004

A Transaction Event Package for the Session Initiation Protocol (SIP)

draft-camarillo-sipping-transac-package-00.txt

Status of this Memo

Copyright Notice

Abstract

SIP provides a SIP Events notification framework that is extensible throught the addition of event packages. This document defines a transaction event package for the SIP Events notification, along with a data format used in notifications for this package. The transaction package allows users to subscribe to a resource in an application server and receive notifications about the changes in state of transactions the application server initiates as part of a service. Additionally, we define a new SIP Associated-Transactions-State header field that allows a server to return a subscribe URI that provides transactions notification information.

Table of Contents

1. Introduction

The SIP [5] Events framework [6] 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. Here, we define an event package for transactions.

An example of an application using this package is a MESSAGE exploder (SIP exploders are described in [8]) that sends MESSAGE requests to a set of destinations on behalf of a user. The user subscribes to the transaction state of the transactions generated by the application server as part of the service. The subscriber receives one more notifications containing the status of those transactions. This way, the user is informed of which MESSAGE requests were delivered to their destination and which ones failed.

The transaction state of the transactions generated by the application server as part of the service is identified by a URI. The user agent uses this URI to subscribe to this state. The user agent may use different mechanisms to obtain such a URI. Section 6 defines one of such mechanisms: the Associated-Transactions-State SIP header field.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [1] and indicate requirement levels for compliant implementations.

3. Definitions

We define the following terms:

Triggering transaction: A SIP transaction that triggers a set of actions in an application server. These actions usually include the generation of one or more SIP associated transactions by the application server.

Triggering request: The SIP request that is part of a triggering transaction.

Associated transaction: A SIP transaction that is generated by an application server on reception of a triggering request.

4. The Transaction Event Package

This section provides the details for defining a SIP Events package, as specified by RFC 3265 [6].

4.1 Event Package Name

The name of this event package is "transaction". This package name is carried in the Event and Allow-Events header fields, as defined in [6].

4.2 Event Package Parameters

This package does not define any event package parameters.

4.3 SUBSCRIBE Bodies

This package does not define any SUBSCRIBE bodies.

4.4 Subscription Duration

The default subscription duration for this event package is 60 seconds.

4.5 NOTIFY Bodies

In this event package, the body of the notification contains a transaction information document. This document describes the state of one or more transactions associated with the subscribed resource. All subscribers and notifiers MUST support the "application/transaction-info+xml" data format described in Section 5. The subscribe request MAY contain an Accept header field. If no such header field is present, it has a default value of "application/transaction-info+xml". If the header field is present, it MUST include "application/transaction-info+xml", and MAY include any other types capable of representing transaction state.

The notifications generated by the server MUST be in one of the formats specified in the Accept header field in the SUBSCRIBE request.

4.6 Notifier Processing of SUBSCRIBE Requests

The transaction information for a resource contains sensitive information. So, all subscriptions SHOULD be authenticated. Authorization policy is at the discretion of the administrator of the notifier.

4.7 Notifier Generation of NOTIFY Requests

The notifier MUST generate a notification containing the state of all the transactions associated with the subscribed resource as soon as any of the following actions take place: a) all the transactions complete; b) the subscription timer expires.

This behaviour guarantees that the subscriber gets at least one notification as soon as the transactions are complete or, within the subscription timer.

The notifier MAY send notifications more often (e.g., once every time the state of a transaction changes) if there are filters applied to the subscription.

4.8 Subscriber Processing of NOTIFY Requests

On reception of a valid NOTIFY request, the subscriber SHOULD immediately render the transaction status to the end-user in an implementation specific way.

4.9 Handling of Forked Requests

By their nature, the resources supported by this package are centralized. So, SUBSCRIBE requests 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 five seconds.

4.11 State Agents

Transaction state is ideally maintained in the element which generates the transactions. Consequently, the elements that generate the transactions are the ones best suited to handle subscriptions to it. The usage of state agents is NOT RECOMMENDED for this package.

5. Transaction Information Format

Transaction information is an XML document that MUST be well-formed and SHOULD be valid. Transaction information documents MUST be based on XML 1.0 and MUST be encoded using UTF-8. This specification makes use of XML namespaces for identifying Transaction information

documents. The namespace URI for elements defined by this specification is a URN [2], using the namespace identifier 'ietf' defined by RFC 2648 [3] and extended by [7]. This URN is:

urn:ietf:params:xml:ns:transaction-info

A Transaction information document begins with the root element tag "transaction-info".

5.1 Structure of the Transaction Information

A transaction information document starts with a "transaction-info" element. This element has three mandatory attributes:

version: allows the recipient of transaction information documents to properly order them. Versions start at 0, and increment by one for each new document sent to a subscriber. Versions are scoped within a subscription. Versions MUST be representable using a 32 bit integer.

state: indicates whether the document contains the "full" transaction information, or whether it contains only information on those transactions which have changed since the previous document ("partial").

entity: contains a URI that identifies the resource whose transaction information is reported in the remainder of the document.

The "transaction-info" element has a series of zero or more "transaction" sub-elements.

5.1.1 Transaction Element

The "transaction" element contains information on a particular associated transaction. It has two mandatory attributes: "id" and "r-uri".

id: provides a single string that can be used as an identifier for this transaction. The "id" is created when the request that initiates the transaction is sent and it MUST be unique amonst all the transactions at the subscribed resource.

r-uri: provides the Request-URI of the request that initiated the associated transaction.

The transaction element has a mandatory sub-element: the "state" sub-element.

5.1.1.1 State Element

The "state" element indicates the state of the transaction. Its value is an enumerated type describing one of the following two states: "pending" or "complete". It has an optional "code" attribute that contains a provisional response status code (in the pending state) or a final response code (in the complete state).

The following is an example of a state element:

```
<state code="404">complete</state>
```

5.2 Constructing Coherent State

The subscriber to the transaction information maintains a table for the list of transactions. The table contains a row for each transaction. Each row is indexed by an ID, present in the "id" attribute of the "transaction" element. The contents of each row contain the state of that transaction as conveyed in the document. The table is also associated with a version number. The version number MUST be initialized with the value of the "version" attribute from the "transaction-info" element in the first document received. Each time a new document is received, the value of the local version number, and the "version" attribute in the new document, are compared. If the value in the new document is one higher than the local version number, the local version number is increased by one, and the document is processed. If the value in the document is more than one higher than the local version number, the local version number is set to the value in the new document, and the document is processed. If the document did not contain full state, the subscriber SHOULD generate a refresh request to trigger a full state notification. If the value in the document is less than the local version, the document is discarded without processing. The "transaction-info" element contains an "entity" attribute that indicate the URI of the subscribed resource.

The processing of the transaction information document depends on whether it contains full or partial state. If it contains full state, indicated by the value of the "state" attribute in the "transaction-info" element, the contents of the table are flushed. They are repopulated from the document. A new row in the table is created for each "transaction" element. If the document contains partial state, as indicated by the value of the "state" attribute in the "transaction-info" element, the document is used to update the table. For each "transaction" element in the document, the subscriber checks to see whether a row exists for that transaction. This check

is done by comparing the ID in the "id" attribute of the "transaction" element with the ID associated with the row. If the transaction does not exist in the table, a row is added, and its state is set to the information from that "transaction" element. If the transaction does exist, its state is updated to be the information from that "transaction" element. If a row is updated or created, such that its state is now terminated, that entry MAY be removed from the table at any time.

5.3 Schema

The following is the schema for the application/transaction-info+xml type:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:transaction-info"
           xmlns:tns="urn:ietf:params:xml:ns:transaction-info"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
   <!-- This import brings in the XML language attribute xml:lang-->
   <xs:import namespace="http://www.w3.org/XML/1998/namespace"
              schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>
   <xs:element name="transaction-info">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="tns:transaction"
                        maxOccurs="unbounded"/>
            <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="version" type="xs:nonNegativeInteger"
                       use="required"/>
         <xs:attribute name="state" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="full"/>
                  <xs:enumeration value="partial"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="entity" type="xs:anyURI"
                       use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="transaction">
      <xs:complexType>
```

```
            <xs:sequence>
               <xs:element ref="tns:state"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string"
                          use="required"/>
            <xs:attribute name="r-uri" type="xs:anyURI"
                          use="required"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="state">
         <xs:complexType>
            <xs:simpleContent>
               <xs:extension base="xs:string">
                  <xs:attribute name="code" use="optional">
                     <xs:simpleType>
                        <xs:restriction base="xs:positiveInteger">
                           <xs:minInclusive value="100"/>
                           <xs:maxInclusive value="699"/>
                        </xs:restriction>
                     </xs:simpleType>
                  </xs:attribute>
               </xs:extension>
            </xs:simpleContent>
         </xs:complexType>
      </xs:element>
   </xs:schema>
```

5.4 Example

The following is an example of a application/transaction-info+xml document:

```
<?xml version="1.0"?>
   <transaction-info xmlns="urn:ietf:params:xml:ns:transaction-info"
                 version="0"
                 state="full"
                 entity="sip:explosion44@exploder12.example.com">
     <transaction id="frgd870th87" r-uri="sip:bob@example.com">
       <state code="200">complete</state>
     </transaction>
     <transaction id="234f12345" r-uri="sip:alice@example.com">
       <state code="404">complete</state>
     </transaction>
     <transaction id="fghd2345" r-uri="sip:mary@example.com">
       <state>pending</state>
```



```
     </transaction>
   </transaction-info>
```

6. The Associated-Transactions-State Header Field

User agents may need to obtain a URI that identifies a set of transactions at the application server in order to subscribe to the state of those transactions. A user agent can use different means to obtain such a URI. One of them consists of using the Associated-Transactions-State SIP header field, which we define here.

The Associated-Transactions-State header field can be used when an application server receives a SIP request that causes the application server to initiate a set of transactions. We refer to this SIP request as the triggering request and the set of transactions as the associated transactions.

For example, an application server provides conferencing services. When this application server receives an INVITE from a user who is the first joining a particular conference, the application server sends a MESSAGE to the rest of the participants to inform them that there is already a user in the conference. In this case, the triggering request is the INVITE, and the associated transactions are the MESSAGE transactions.

Application servers MAY include a Associated-Transactions-State header field in the responses to a triggering request. Clients can use the URI in this header field to subscribe to the state of the associated transactions.

In our example, the response to the INVITE request carries a Associated-Transactions-State header field. The client subscribes to the URI received in this header field to monitor the state of the MESSAGE transactions. This way, the user knows who of the rest of the participants receives the MESSAGE.

6.1 Syntax

The ABNF of the Associated-Transactions-State header field is:

```
Associated-Transactions-State = HCOLON transactions-state-uri
transactions-state-uri        = SIP-URI / SIPS-URI
```

OPEN ISSUE: do we want to have transactions-state-uri = addr-spec, which includes SIP-URI / SIPS-URI / absoluteURI ? Do we need non-SIP URIs?

7. Security Considerations

TBD.

8. IANA Considerations

This document registers a new MIME type, application/ transaction-info+xml, new XML namespace and a new SIP header field.

8.1 MIME Registration for application/transaction-info+xml

MIME media type name: application

MIME subtype name: transaction-info+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [4].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [4].

Security considerations: See Section 10 of RFC 3023 [4] and Section 7 of this specification.

Interoperability considerations: none.

Published specification: This document.

Applications which use this media type: This document type has been used to support SIP applications such as MESSAGE exploders.

Additional Information:

Magic Number: None

File Extension: .tin or .xml

Macintosh file type code: "TEXT"

Personal and email address for further information: Gonzalo Camarillo, <Gonzalo.Camarillo@ericsson.com>

Intended usage: COMMON

Author/Change controller: The IETF.

8.2 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:transaction-info

This section registers a new XML namespace, as per the guidelines in [7]

URI: The URI for this namespace is urn:ietf:params:xml:ns:transaction-info.

Registrant Contact: IETF, SIPPING working group,<sipping@ietf.org>, Gonzalo Camarillo, <Gonzalo.Camarillo@ericsson.com>

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
     content="text/html;charset=iso-8859-1"/>
  <title>Transaction Information Namespace</title>
</head>
<body>
  <h1>Namespace for Transaction Information</h1>
  <h2>application/transaction-info+xml</h2>
  <p>See <a href="[[[URL of published RFC]]]">RFCXXXX</a>.</p>
</body>
</html>
END
```

8.3 Schema Registration

This specification registers a schema, as per the guidelines in [7].

URI: please assign.

Registrant Contact: IETF, SIPPING working group,<sipping@ietf.org>, Gonzalo Camarillo, <Gonzalo.Camarillo@ericsson.com>

XML: The XML can be found in Section 5.3.

8.4 Associated-Transactions-State Header Field Registration

9. Acknowledgements

This document is partially based on the event package for INVITE initiated dialogs [9].

Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Moats, R., "URN Syntax", RFC 2141, May 1997.

[3] Moats, R., "A URN Namespace for IETF Documents", RFC 2648, August 1999.

[4] Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[5] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[6] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[7] Mealling, M., "The IETF XML Registry", draft-mealling-iana-xmlns-registry-05 (work in progress), June 2003.

Informational References

[8] Camarillo, G., "Requirements for Session Initiation Protocol (SIP) Exploder Invocation", draft-camarillo-sipping-exploders-01 (work in progress), November 2003.

[9] Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-03 (work in progress), October 2003.

Authors' Addresses

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Gonzalo.Camarillo@ericsson.com

Miguel A. Garcia-Martin
Nokia
P.O.Box 407
NOKIA GROUP, FIN 00045
Finland

EMail: miguel.an.garcia@nokia.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

Internet Engineering Task Force SIP WG
Internet Draft
G. Camarillo
Ericsson

draft-camarillo-sipping-transc-b2bua-01.txt
February 7, 2004
Expires: August, 2004

The Session Initiation Protocol Conference Bridge Transcoding Model

STATUS OF THIS MEMO

Abstract

This document describes how to invoke transcoding services using the conference bridge model. This way of invocation meets the requirements for SIP regarding transcoding services invocation to support deaf, hard of hearing and speech-impaired individuals.

Table of Contents

1 Introduction

The framework for transcoding with SIP [1] (draft-ietf-sipping-transc-framework) describes how two SIP UAs can discover imcompatibilities that prevent them from establishing a session (e.g., lack of support for a common codec or for a common media type). When such incompatibilities are found, the UAs need to invoke transcoding services to successfully establish the session. Using the conference bridge model is one way to perform such invocation.

In the conference bridge model for transcoding invocation, a transcoding server that provides a particular transcoding service (e.g., speech-to-text) behaves as a B2BUA between both UAs and is identified by a URI.

2 Caller's Invocation

Figure 1 shows the message flow for the caller's invocation of a transcoder T. The caller (A) sends an INVITE (1) to the transcoder (T) to establish the session A-T. The URI in the Request-URI of this INVITE contains a list parameter, as defined in [2] (draft-camarillo-sipping-uri-list-01), with a pointer to a URI list. This URI list contains a single URI: the callee's URI, as shown below:

```
INVITE sip:transcoder@example.com;list=cid:cn35t8@example.com SIP/2.0
Via: SIP/2.0/TCP client.chicago.example.com
    ;branch=z9hG4bKhjhs8ass83
Max-Forwards: 70
To: "Transcoder" <sip:transcoder@example.com>
From: Caller <sip:caller@chicago.example.com>;tag=32331
Call-ID: d432fa84b4c76e66710
CSeq: 1 INVITE
Contact: <sip:caller@client.chicago.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
     SUBSCRIBE, NOTIFY
Conten-Type: multipart/mixed;boundary="boundary1"
Content-Length: xxx

--boundary1
Content-Type: application/sdp
Content-Length: xxx

v=0
o=caller 2890844526 2890842807 IN IP4 chicago.example.com
s=Example Subject
c=IN IP4 192.0.0.1
t=0 0
```



```
m=audio 20000 RTP/AVP 0

--boundary1
Content-Type: application/resource-lists+xml
Content-Length: 367
Content-ID: <cn35t8@example.com>

<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <list name="ad-hoc-1">
    <entry name="1" uri="sip:callee@example2.com" />
  </list>
</resource-lists>
--boundary1--
```

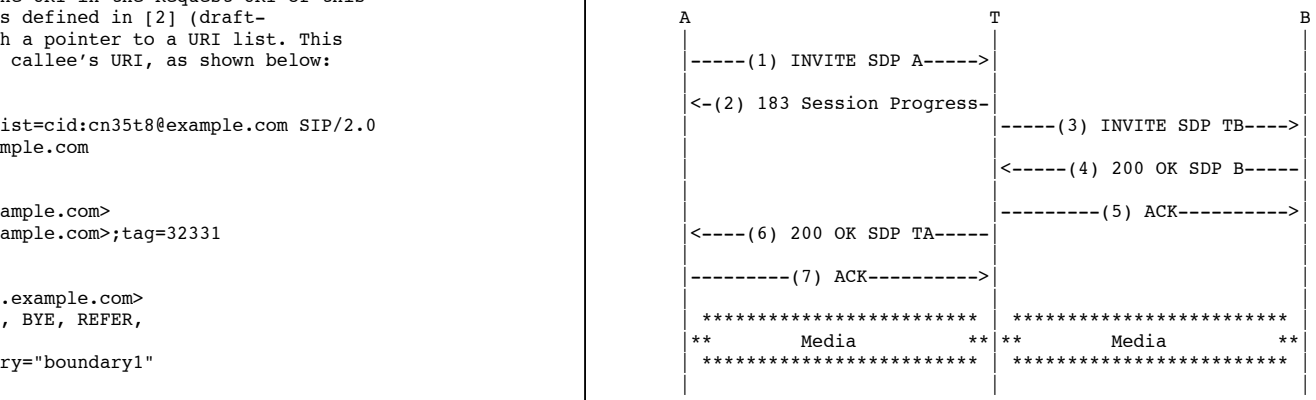


Figure 1: Successful invocation of a transcoder by the caller

On reception of the INVITE, the transcoder generates a new INVITE towards the callee. The transcoder acts as a B2BUA, so, this new INVITE (3) belongs to a different transaction than the INVITE (1) received by the transcoder.

When the transcoder receives a final response (4) from the callee, it

generates a new final response (6) for INVITE (1). This new final response (6) has the same status code as the one received in the response from the callee (4).

The advantage of this message flow is that, for both user agents, is indentical to the flow for establishing a regular session (i.e., without transcoder) between them. Additionaly, the only difference in the message contents is that the caller needs to use a list parameter in the Request-URI of the initial INVITE.

2.1 Unsuccessful Session Establishment

Figure 2 shows a similar message flow as the one in Figure 1. Nevertheless, this time the callee generates a non-2xx final response (4). Consequently, the transcoder generates a non-2xx final response (6) towards the caller as well.

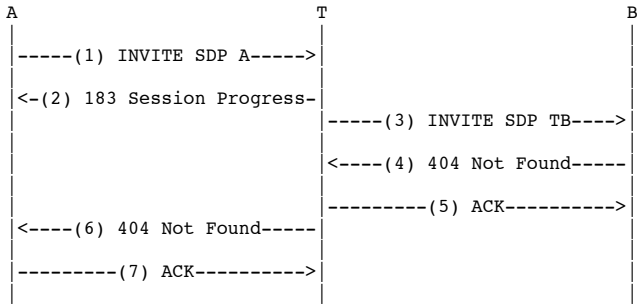


Figure 2: Unsuccessful session establishment

The problem with this flow is that the caller does not know whether the 404 (Not Found) response means that the initial INVITE (1) did not reach the transcoder or that the INVITE generated by the transcoder (4) did not reach the callee. To resolve this, it is recommended that the caller uses the reliable provisional responses [3] SIP extension.

Figure 3 shows the resulting message flow when the caller requires the use of the reliable provisional responses [3] SIP extension. The repection of the 183 (Session Progress) reliable provisional response

informs the caller that the transcoder was contacted susccessfully. So, the 404 (Not Found) response indicates that the callee could not be reached.

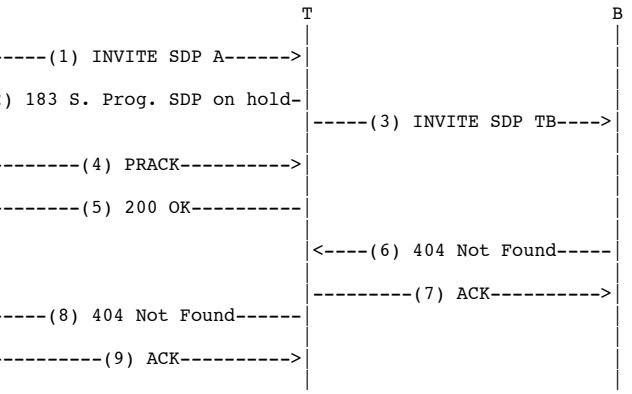


Figure 3: Invocation using reliable provisional responses

3 Callee's Invocation

If a UA receives an INVITE with an offer that is not acceptable, it can only invoke a transcoder if the caller supports the Replaces [4] extension. This support is indicated by the Supported header field in the INVITE.

If the caller (A) does not support Replaces, the callee (B) can always reject the session and attempt to establish a new session with A following the procedures in Section 2. This way, B would act as a caller and, consequently, it would follow the procedures for caller's invocation of transcoders.

Assuming that the caller (A) supports Replaces, the callee (B) follows the steps shown in Figure 4 to invoke a transcoder. The callee sends a 183 (Session Progress) response (2) to the caller. This response carries a tag in the To header field. The caller needs to receive this To tag so that this early dialog can be replaced later in (5). So, the callee SHOULD use the reliable provisional

responses [3] SIP extension. The SDP in the 183 (Session Progress) response may put the media streams on hold. If the caller did not support this extension, the callee MAY send a 200 (OK) putting the media streams on hold.

OPEN ISSUE: can we use 0.0.0.0 instead of hold here?

After returning a response with a To tag to the caller, the callee sends an INVITE (2) to the Transcoder. The URI in the Request-URI of this INVITE contains a list parameter, as defined in [2] (draft-camarillo-sipping-uri-list-01), with a pointer to a URI list. This URI list contains a single URI: the URI received in the Contact header field of the initial INVITE (1) with an escaped Replaces header field, as shown in the following example:

```
sip:caller@client.chicago.example.com?Replaces=40d432fa84b4c76e66710;
                ;from-tag=32331
                ;to-tag=12dr45
```

We recommend the use of the reliable provisional responses between the callee and the transcoder so that the callee is able to distinguish between problems with the transcoder and problems with the caller, as we described in Section 2.1.

When A receives this INVITE (5), it replaces the original dialog (1) with this new dialog. The caller sends a CANCEL (10) to cancel the original dialog (1) and receives a 487 (Request Terminated) response (11) from the callee.

4 Security Considerations

TBD.

5 Contributors

This document is the result of discussions amongst the conferencing design team. The members of this team include Eric Burger, Henning Schulzrinne and Arnoud van Wijk.

6 OPEN ISSUES

In SIP, the Route header field is used to traverse proxies, but is seems that using it for traversing B2BUAs would be stretching its semantics too much.

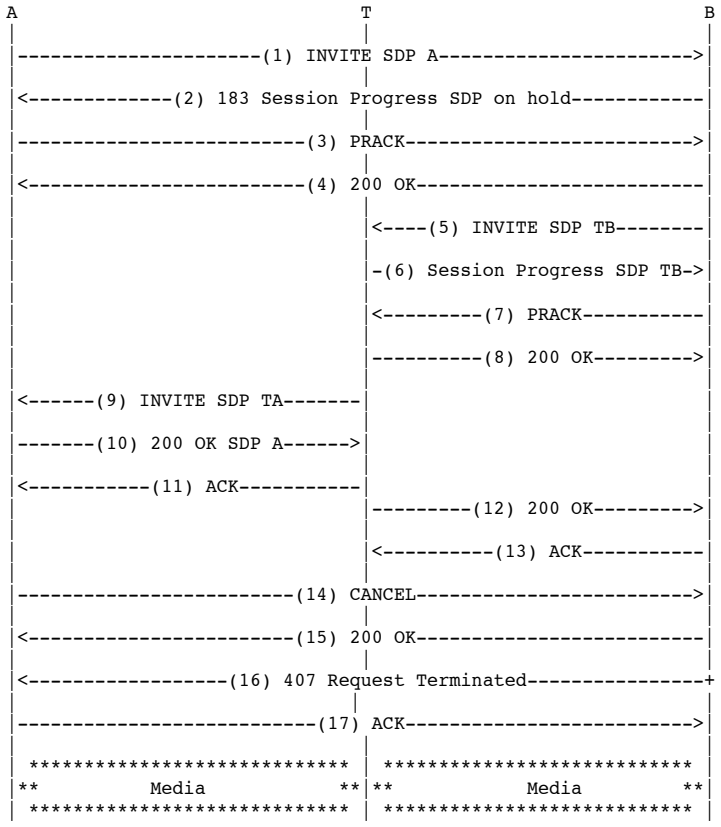


Figure 4: Callee's invocation of a transcoder

7 Authors' Addresses

Gonzalo Camarillo
Ericsson

Advanced Signalling Research Lab.
FIN-02420 Jorvas
Finland
electronic mail: Gonzalo.Camarillo@ericsson.com

8 Bibliography

[1] G. Camarillo, "Framework for transcoding with the session initiation protocol," Internet Draft draft-camarillo-sipping-transc-framework-00, Internet Engineering Task Force, Aug. 2003. Work in progress.

[2] G. Camarillo, "Providing a session initiation protocol (SIP) application server with a list of URIs," Internet Draft draft-camarillo-sipping-uri-list-00, Internet Engineering Task Force, Nov. 2003. Work in progress.

[3] J. Rosenberg and H. Schulzrinne, "Reliability of provisional responses in session initiation protocol (SIP)," RFC 3262, Internet Engineering Task Force, June 2002.

[4] B. Biggs, R. W. Dean, and R. Mahy, "The session inititation protocol (SIP) Engineering Task Force, Aug. 2003. Work in progress.

SIPPING Working Group G. Camarillo
Internet-Draft Ericsson
Expires: August 6, 2004 A. Roach
dynamicsoft
February 6, 2004

Providing a Session Initiation Protocol (SIP) Application Server with a List of URIs

draft-camarillo-sipping-uri-list-01.txt

Status of this Memo

Copyright Notice

Abstract

This document describes how a user agent can provide an application server with a list of URIs. The way the application server uses the URIs in the list is service specific.

Table of Contents

1. Introduction

The need for exploders in SIP is described in [7]. Mechanisms to invoke exploders in SIP need to meet the requirements listed there.

UAs need to have a means to provide application servers with a set of URIs for certain services. For example, a UA creating a conference needs to provide the conference server with the participants. The same way, a UA requesting presence information from a set of users needs to provide the resource list server with the URIs of the users that belong to the list.

These lists are typically configured using out-of-band methods. For instance, a UA can use XCAP [6] to create a list of URIs and to associate this list with a SIP URI. It can, then, send a SIP request (an INVITE or a SUBSCRIBE in our previous examples) to that SIP URI.

Still, there is a need to create lists of URIs in an ad-hoc way and send them directly in a SIP message. We define a SIP and SIPS URI paramerer called "list", which carries a URI. This URI is a pointer to a URI list.

A UA creating a SIP request that needs to carry a URI list proceeds this way. It places the URI list (e.g., an XCAP resource list [4]) in a body part, and then, it adds a "list" parameter to the Request-URI. This "list" parameter contains a Content-ID URL [2] that points to the body part that carries the URI list.

Alternatively, the URI in the "list" parameter can point to an external URI list (e.g., an http URI). In this case, the URI list would not be carried in the SIP request.

The way the application server interprets the URI list received in the request is service specific.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [1] and indicate requirement levels for compliant implementations.

3. URI Parameter vs. Header Field

We have chosen to transport the pointer to the URI list in a URI parameter rather than in a header field because, this way, the Request-URI fully indentifies the service being invoked and all the

recipients of the service. Using a header field instead would imply that the Request-URI did not carry the list of the reciptiens.

Network administrators should note that they need to configure proxies to route correctly Request-URIs that contain a "list" parameter and are addressed to their domain.

4. The SIP and SIPS URI List Parameter

We define the "list" parameter for SIP and SIPS URIs. It MUST contain a URI that points to a URI list. The XCAP resource list format [4] MUST be supported; any other URI list formats MAY be supported. The ABNF of the "list" parameter is:

```
list-param = "list=" absoluteURI
```

The following is an example of a SIP URI with a list parameter pointing to a body part using a Content-ID URL [2]:

sip:group@example.com;list=cid:cn35t8jf02@example.com

The following is an example of a SIP URI with a list parameter pointing to an external URI:

sip:group@example.com;list=http://xcap.example.com/lists/mylist.xml

5. Ad-Hoc List' Life Time

An application server that receives a request with a URI list (or a pointer to it) creates a so called ad-hoc list, whose lifetime depends on the service provided by the server.

Ad-Hoc lists created by requests that do not establish a dialog usually expire immediately. Ad-Hoc lists created by requests that establish a dialog usually expire when the dialog terminates.

6. The Content-ID SIP Header Field

The Content-ID MIME header field is defined in RFC 2045 [5]. We define here the same header field to be used in SIP messages. Its ABNF is:

```
Content-ID = "Content-ID" HCOLON msg-id
```

RFC 2822 [3] defines msg-id in Section 3.6.4.

The Content-ID value is used to uniquely identify a body or a body part. The Content-ID header field MAY appear in any SIP request or

response that contains a body.

7. Examples

This section shows how to use the list parameter to create an ad-hoc conference and to subscribe to the presence information to a set of users. These examples illustrate the usage of the "list" parameter. They do not mandate how the previously mentioned services have to be implemented.

7.1 Ad-Hoc Conference

Carol creates an ad-hoc conference by sending the INVITE request shown in Figure 1. The list parameter in the Request-URI points to a MIME body that carries the list of participants.

```
INVITE sip:ad-hoc@example.com;list=cid:cn35t8jf02@example.com SIP/2.0
Via: SIP/2.0/TCP client.chicago.example.com
    ;branch=z9hG4bKhjhs8ass83
Max-Forwards: 70
To: "Ad-Hoc Conferences" <sip:ad-hoc@example.com>
From: Carol <sip:carol@chicago.example.com>;tag=32331
Call-ID: d432fa84b4c76e66710
CSeq: 1 INVITE
Contact: <sip:carol@client.chicago.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
     SUBSCRIBE, NOTIFY
Allow-Events: dialog
Accept: application/sdp, message/sipfrag,
     application/resource-lists+xml
Conten-Type: multipart/mixed;boundary="boundary1"
Content-Length: 731

--boundary1
Content-Type: application/sdp
Content-Length: 160

v=0
o=carol 2890844526 2890842807 IN IP4 chicago.example.com
s=Example Subject
c=IN IP4 192.0.0.1
t=0 0
m=audio 20000 RTP/AVP 0
m=video 20002 RTP/AVP 31

--boundary1
Content-Type: application/resource-lists+xml
```



```
Content-Length: 367
Content-ID: <cn35t8jf02@example.com>

<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <list name="ad-hoc-1">
    <entry name="1" uri="sip:bill@example.com" />
    <entry name="2" uri="sip:joe@example.com" />
    <entry name="3" uri="sip:ted@example.com" />
    <entry name="4" uri="sip:bob@example.com" />
  </list>
</resource-lists>
--boundary1--
```

Figure 1: INVITE request

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP client.chicago.example.com
     ;branch=z9hG4bKhjhs8ass83;received=192.0.2.4
To: "Ad-Hoc Conferences" <sip:ad-hoc@example.com>;tag=733413
From: Carol <sip:carol@chicago.example.com>;tag=32331
Call-ID: d432fa84b4c76e66710
CSeq: 1 INVITE
Contact: <sip:3402934234@example.com>;isfocus
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
      SUBSCRIBE, NOTIFY
Allow-Events: dialog, conference
Accept: application/sdp, application/conference-info+xml,
      application/resource-lists+xml, message/sipfrag
Supported: replaces, join
Content-Type: application/sdp
Content-Length: 312

v=0
o=focus431 2890844526 2890842807 IN IP4 ms5.conf.example.com
s=Example Subject
i=Example Conference Hosted by Example.com
u=http://conf.example.com/3402934234
e=3402934234@conf-help.example.com
p=+1-888-555-1212
c=IN IP4 ms5.conf.example.com
t=0 0
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
```

Figure 2: 200 (OK) response

The conference server responds with a 200 (OK) that carries the URI for the conference in its Contact header field. If the UA wants to obtain information about the status of the conference, for instance, it will SUBSCRIBE to the conference package using this URI.

7.2 Presence List

Carol subscribes to the presence information of four of her friends using the list parameter.

```
SUBSCRIBE sip:ad-hoc@example.com;list=cid:cn35t8jf02@example.com SIP/2.0
Via: SIP/2.0/TCP client.chicago.example.com
    ;branch=z9hG4bKhjhs8ass83
Max-Forwards: 70
To: "Ad-Hoc Presence List" <sip:ad-hoc@example.com>
From: Carol <sip:carol@chicago.example.com>;tag=32331
Call-ID: d432fa84b4c76e66710
CSeq: 1 INVITE
Contact: <sip:carol@client.chicago.example.com>
Require: eventlist
Event: presence
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
     SUBSCRIBE, NOTIFY
Allow-Events: presence
Accept: application/sdp, message/sipfrag,
     application/resource-lists+xml, application/rlmi+xml
Content-Type: application/resource-lists+xml
Content-Length: 367
Content-ID: <cn35t8jf02@example.com>

<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <list name="ad-hoc-1">
    <entry name="1" uri="sip:bill@example.com" />
    <entry name="2" uri="sip:joe@example.com" />
    <entry name="3" uri="sip:ted@example.com" />
    <entry name="4" uri="sip:bob@example.com" />
  </list>
</resource-lists>
```

Figure 3: SUBSCRIBE request

8. Security Considerations

TBD.

9. IANA Considerations

This document registers the "list" SIP and SIPS URI parameter, which is described in Section 4. This parameter is to be added to the SIP and SIPS URI parameter registry under http://www.iana.org/ TBD.

This document registers the Content-ID SIP header field, which is described in Section 6. This header field is to be added to the header field registry under http://www.iana.org/assignments/ sip-parameters.

Header Name: Content-ID

Compact Form: (none)

Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Levinson, E., "Content-ID and Message-ID Uniform Resource Locators", RFC 2392, August 1998.

[3] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[4] Rosenberg, J., "An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Usage for Presence Lists", draft-ietf-simple-xcap-list-usage-01 (work in progress), October 2003.

Informational References

[5] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[6] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", draft-ietf-simple-xcap-01 (work in progress), October 2003.

[7] Camarillo, G., "Requirements for Session Initiation Protocol (SIP) Exploder Invocation", draft-camarillo-sipping-exploders-01 (work in progress), November 2003.

Authors' Addresses

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Gonzalo.Camarillo@ericsson.com

Adam Roach
dynamicsoft
5100 Tennyson Pkwy
Suite 1200
Plano, TX 75024
US

EMail: adam@dynamicsoft.com

Intellectual Property Statement

Full Copyright Statement

HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

AAA Working Group
Internet Draft
draft-hiller-uri-list-index-00.txt
Category: Standards Track

Tom Hiller
Lucent Technologies
Adam Roach
Dean Willis
dynamicsoft
March 2004

SIP URI List Index

Status of this Memo

Abstract

This draft extends the schema of the resource list specified in draft-ietf-simple-xcap-list-usage-01 by defining an index attribute (membercode). It also defines two MIME types that refer to subsets of a resource list. These MIME types can be used to identify subsets of a resource list for use with SIP requests.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [2].

Table of Contents

1. Problem Statement

Some 3G wireless physical layers are extremely lightweight to the point of being fragile. For example, cdma2000 has a mechanism called "short data burst" that provides a low latency IP over PPP access for mobile stations, albeit it does so with severe message length limitations. In the case of cdma2000, "low latency" means on the order tens of milliseconds, and "severe" means less than a hundred bytes including PPP. The length limitation arises on the basis of radio engineering considerations. In addition to the length limitations, the absolute amount of bandwidth over such physical channels is highly limited. Constraints such as these represent non-trivial problems for the transport of SIP requests such as the INVITE. Despite these physical layer mechanisms being so lightweight, various service providers would like to use them if at all possible to transport SIP requests.

IP header compression and SIGCOMP compression provide help reducing the number of bytes in a SIP request that need to be transported. The EXPLODE method [EXPLODE] provides help reducing the number of SIP requests that need to be transported. However, even with all of this help, there is still yet another problem to overcome: The EXLPODE method's SIP URI List [URI-LIST] is, in fact, an arbitrary length list of arbitrary URI elements, and therefore, may be too lengthy for highly constrained physical layers, such as the cdma2000 short data burst.

Based on the foregoing, it would be helpful to have a highly compact means to convey a URI List in SIP request bodies.

2. General Solution

2.1. Summary

This draft adds an attribute called "membercode" to elements of the resource list schema of [RF] and defines two MIME [MIME-1] types to convey (represent) a URI List [URI-LIST] in the body of a SIP request. The MIME types are based on the identity of the user's resource list along with indices (the membercodes) that have been previously stored in a user's resource list. Both MIME types require that the server hosting the list assign membercodes to all URIs of the user's Resource List entries. The MIME type conveys identity of

the resource list and the membercodes associated with the URIs on a URI List. The MIME instance replaces the actual URI elements, thereby saving many bytes.

The membercode is a non-negative integer that is unique within a given resource list. The maximum value (size) of the membercode should be on the order of the number of lists and list entries of the resource list.

2.2. Membercode Attribute Management

The document draft-ietf-simple-xcap-list-usage-01 [RL] states the requirements on XCAP for a client to manipulate the resource list.

An XCAP client does not include the membercode attribute when it creates or modifies a resource list element, as the membercode is optional. Instead the server creates a unique membercode when the resource is created. The server leaves the membercode unchanged when a list or list element is modified. The addition of an index to the resource list is transparent to XCAP. Having the presence list server assigns membercodes to resource list elements avoids conflicts and/or race conditions that could arise due to multiple users creating or modifying resource list elements.

Users of the resource list may subscribe for updates to receive presence notifications [RL-NOTIFY] that carry the assigned membercodes. Also, users may access the resource list directly via XCAP to learn the membercode attributes created. Otherwise, a means to synchronize the membercodes in user devices must be provided by means outside the scope of this document.

In order for a users learn the values of membercode attributes via presence notifications [RL-NOTIFY] the user has to subscribe for notifications and the resource list's "subscribe" flag MUST be set).

2.3. MIME Types

The first MIME, application/resource-lists-indices, is a list of the membercodes of the elements of a URI list.

The second MIME, application/resource-lists-bitmap, is a bit map of the membercodes of the elements of the URI list. In the latter case, a bit set at location 'x' in the bit map corresponds to a membercode of value '2**x".

MIME bodies may be further compressed with procedures that are part of a general SIGCOMP [SIGCOMP] "program".

3. Definition of Membercode Attribute for Resource List

As explained above, this draft adds an attribute called "membercode" to elements of the resource list schema of [RF]. , The resulting schema is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:resource-lists"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:ietf:params:xml:ns>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="list" type="listType" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  <xs:complexType name="listType">
    <xs:sequence maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="list" minOccurs="0"
             maxOccurs="unbounded">
          <xs:complexType>
            <xs:complexContent>
              <xs:extension base="listType"/>
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="external" type="xs:anyURI" minOccurs="0"
          maxOccurs="unbounded"/>
        <xs:element name="entry" type="entryType" minOccurs="0"
          maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax"
          minOccurs="0"  maxOccurs="unbounded"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="uri" type="xs:anyURI" use="optional"/>
    <xs:attribute name="subscribeable" type="xs:boolean"
                  use="optional"/>
    <xs:anyAttribute namespace="##other"/>
    <xs: attribute name="membercode"
                  type="unique positiveInteger" use="optional" />
  </xs:complexType>
  <xs:complexType name="entryType">
    <xs:sequence>
      <xs:element name="display-name" type="display-nameType"/>
      <xs:any namespace="##other" processContents="lax"
         minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="uri" type="xs:anyURI" use="optional"/>
    <xs: attribute name="membercode"
               type="unique positiveInteger" use="optional" />
  </xs:complexType>
  <xs:simpleType name="display-nameType">
```

```
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:schema>
```

4. IANA Considerations

The document draft-camarillo-uri-list-00.txt defines a "list" parameter for SIP and SIPS URIs that points to an XCAP resource list. This document defines two MIME types to which the list parameter may point and is consistent with [MIME-2] and [MIME-4].

4.1. Index List

The MIME Content-Type is "application/resource-lists-indices" and is a list of membercodes separated by white space. The presence of an membercode in the list means that the associated URI is to be included on the URI list. The MIME type includes the resource list URI.

The URI and membercode are encoded as is encoded in UTF-8. The membercode attributes, which are numbers, are coded as hex digits. The URI and member codes are separated by a white space. The exact efficiency of the encoding of membercodes is less important because a SIGCOMP program can compress these digits, which are represented as characters, to binary numbers.

The ABNF [ABNF] for this MIME type is as follows.

```
resource-lists-indices = (resource-list-URI SP *(membercode SP))
     resource-list-URI = SIP-URI
                       ; this is an SIP URI to the resource list
                       ; see [SIP]
           membercode = *HEXDIG
                       ; the member code is on the list if the
                       ; associated URI is on the URI list
```

Information per [MIME-4] is as follows:

MIME media type name: application

MIME subtype name: resource-lists-indices

Mandatory parameters: none

Optional parameters: none

Encoding considerations: UTF-8

Security considerations: See the security section of this specification

Interoperability considerations: none.

Published specification: This document.

Applications which use this media type: SIP Requests with an EXPLODE method based URI list.

Additional Information:

Magic Number: None

File Extension: tbd

Macintosh file type code: tbd

Personal and email address for further information: Tom Hiller, tomhiller@lucent.com

Intended usage: COMMON

Author/Change controller: The IETF

4.2. Bit Map MIME

The MIME Content-Type is an "application/resource-lists-bitmap", and is a binary string whose individual bit positions correspond to the values of membercodes. A bit set in the bit map means the URI associated with the membercode whose value matches that bit position is on the URI list. The MIME type includes the resource list URI.

If the bit map has fewer bits than the maximum value of the membercode, then URIs corresponding to "missing" bit positions are not included in the URI list. If the bit map has bit positions that do not correspond to membercodes or more bits than the maximum value possible of the membercode, then the "extra" bits MUST be ignored.

The URI and bitmap are encoded as is encoded in UTF-8. The bit flags of the membercode are coded as four bits to a hex digit. Any bits in hex digit for which membercodes do not exist are set to zero, which occurs if the number of bits in the bit map isn't a multiple of four. The bit map positions correspond to the power of two in the resulting hex number. Therefore, in string of hex digits, the most significant bit of the most significant hex digit represents the highest value membercode of the resource list.

The bit map MIME type's ABNF is as follows:

```
 resource-lists-bitmap = (resource-list-URI  *membercode-hex)
     resource-list-URI = SIP-URI
                       ; this is a SIP URI to the resource list
                       ; see [SIP]
```

```
membercode-hex = HEXDIG
                 ; a bit position M of the membercode-hex N
                 ; is set if a URI on the URI list
                 ; has a membercode of value 2**(4*N+M)
                 ; where N starts at 1 (so the first character
                 ; is M=1) and M is value of 2**M in the hex
                 ; character (so the least bit is 2**0).
```

Information per [MIME-4] is as follows:

MIME media type name: application

MIME subtype name: resource-lists-bitmap

Mandatory parameters: none

Optional parameters: none

Encoding considerations: UTF-8

Security considerations: See the security section of this specification

Interoperability considerations: none.

Published specification: This document.

Applications which use this media type: SIP Requests with an EXPLODE method based URI list.

Additional Information:

Magic Number: None

File Extension: tbd

Macintosh file type code: tbd

Personal and email address for further information: Tom Hiller, tomhiller@lucent.com

Intended usage: COMMON

Author/Change controller: The IETF

5. Security Considerations

The index proposed herein is a way to access a user on the resource list, which is used to invite people to calls, etc. However, the security of the index is no more nor less important than any other

data already contained on the list, and therefore, this document does not imply additional security concerns or considerations.

6. References

6.1. Normative

[ABNF] Crocker, "Augmented BNF for Syntax Specifications: ABNF", RFC2234, November 1997

[EXPLODE] Camarillo, Session Initiation Protocol (SIP) Exploder Invocation, draft-camarillo-sipping-exploders-solution-00.txt, November 22, 2003

[IANA] Narten, Alvestrand, "Guidelines for Writing an IANA C Considerations Section in RFCs", BCP 26, RFC 2434, October 1998

[MIME-1] Freed, Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC2045, November 1996

[MIME-2] Freed, Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types" RFC2046, November 1996

[MIME-4] Freed et al, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", RFC2048, November 1996

[RL] Rosenberg, "draft-ietf-simple-xcap-list-usage-01", October 27, 2003

[RL-NOTIFY] Roach, Rosenberg, Campbell, A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists, draft-ietf-simple-event-list-04, June 13, 2003

[SIGCOMP] Price et al, Signaling Compression (SigComp), RFC3320, January 2003

[SIP] Rosenberg et al, "The Session Initiation Protocol", RFC3261, June 2002

[URI-LIST] Camarillo, Roach, Providing a Session Initiation Protocol (SIP) Application Server with a List of URIs, November 22, 2003

7. Acknowledgements

Chris Bennet of Togabi suggested the use of the MIME type that conveys a list of indices.

8. Authors' Addresses

Questions about this memo can be directed to:

Tom Hiller
Lucent Technologies
1960 Lucent Lane
Naperville, IL 60566
USA
Phone: +1 630-979-7673
E-mail: tomhiller@lucent.com

Dean Willis
dynamicsoft Inc.
5100 Tennyson Parkway
Suite 1200
Plano, TX 75028
US
Phone: +1 972 473 5455
E-mail: dean.willis@softarmor.com
URI: http://www.dynamicsoft.com/

Adam Roach
dynamicsoft
5100 Tennyson Pkwy
Suite 1200
Plano, TX 75024
USA
E-mail: adam@dynamicsoft.com

Intellectual Property Statement

Full Copyright Statement

Session Initiation Proposal Investigation Working Group
Internet-Draft
Expires: April 18, 2004

V. Hilt
Bell Labs/Lucent Technologies
G. Camarillo
Ericsson
J. Rosenberg
dynamicsoft

October 19, 2003

A Framework for Session-Independent Intermediary Session Policies in SIP

draft-hilt-sipping-session-indep-policy-00

Status of this Memo

Copyright Notice

Abstract

Domains may have policies in place that impact the way sessions are established by user agents. Some of these policies are independent of a specific session and need to be considered for a certain period of time. It is therefore desirable to convey these policies to user agents once they become active instead of requesting them for every session. In this document, we propose a framework that enables user agents to subscribe to session policies.

Table of Contents

1. Introduction

Some domains have policies in place, which influence the sessions a user sets up. These policies are often established to support the network infrastructure, enable user agents to use services or to prevent user agents from overly burdening the network. For example, a SIP user agent might be located in a domain, that is connected to the public Internet via a Network Address Translator (NAT). This domain might have a policy in place that requires the user agent to contact a TURN [11] relay before setting up a session. Information about this policy is essential for a user agent to successfully set up a session.

In another example, SIP is used in a wireless network. The network provider has limited resources for media traffic. During periods of high activity, the provider would like to restrict codec usage on the network to lower rate codecs. In existing approaches, this is frequently accomplished by having the proxies examine the SDP [2] in the body and remove the higher rate codecs or reject the call and require the UA to start over with a different set of codecs. Having information about the current policy would enable user agents to initiate a session with an acceptable codec.

In a third example, a domain has established policies regarding the type of user agents that can use their network. For example, a domain could require that user agents using its network use a particular protocol (e.g., SIP) with a set of extensions (e.g., preconditions must be used). A user agent needs to know the exact policy of a domain in order to be able to use the right configuration to send and receive traffic in that domain.

In yet a fourth example, a user has subscribed to a network-based call recording service to record calls to certain destinations. The recording server acts as a media intermediary which needs to be included in the media path. Knowing the address of the recording server enables the user agent to route its traffic through the recording server if desired.

Some domains enforce certain session policies. For example, if the policy of a domain disallows the use of a particular codec, access routers will discard packets that transport media encoded with that codec. Unfortunately, enforcement mechanisms do not usually inform the user about what is happening. They silently keep the user from doing anything against the policy. It is therefore important for the user agent to know about session policies and the consequences of not accepting them.

Session policies may be specific to a certain session and may change

from session to session. Such policies can be set up using the framework for session-specific policies [3]. Other session policies remain stable for a longer period of time, typically in the range of hours or days. In principle, these policies could also be set up on a session-to-session basis. However, establishing the same policies over and over again is expensive, causing the continuous transmission of the same information during session setup, and possibly adding to session setup latencies. It is therefore desirable, to enable user agents to obtain the policies relevant for them and to inform the user agents about changes in these policies.

Our solution for supporting session-independent session policies is to introduce a framework that allows user agents to subscribe to session policies. This framework satisfies the requirements listed in [13].

This document is structured as follows: Section 3 introduces a framework that enables user agents to subscribe to session policies. Section 4 provides the necessary details to define an event package for the SUBSCRIBE/NOTIFY framework. Section 5 discusses the creation of policy packages for this framework and Section 6 describes a basic policy package. Section 7 discusses Security and Section 8 IANA considerations. Section 9 talks about open issues.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, [1] and indicate requirement levels for compliant implementations.

3. Framework for Session-Independent Policies

The conveying session-independent policies to a user agent is most beneficial for network providers, that are frequently involved in the sessions established by a user agent. Typically, these are the following two providers:

- The Home Domain Provider is responsible for providing SIP service to a SIP user. Typically, this is the domain present in the URI in the address-of-record of a registration. The home domain provider will usually maintain user preferences and subscriptions to services. Thus, it may want to provide session-independent policies, that are needed for services, a user has subscribed to.

- The Access Network Provider is responsible for providing IP service to a SIP agent. This may be the same provider as the home domain provider or it may be a different provider in case the user roams in a foreign network or obtains SIP services and IP connectivity from different operators. Access Network Providers often provide session-independent policies, which generally impact the traffic in their networks.

A solution for session-independent policies should enable a user agent to detect the policy servers in the domains of these two providers and retrieve the relevant policies. User agents should also be enabled to retrieve policies from different policy servers.

This framework allows user agents to subscribe to session policies based on the SIP SUBSCRIBE/NOTIFY mechanism [10]. This framework does not define the structure or semantics of policies. Session-independent policies need to be defined in policy packages. An basic policy package is discussed in Section 6.

3.1 Policy Server Discovery

In the first step, a UA needs to discover the relevant policy servers.

The UA MUST attempt to discover the policy servers of the home domains of all local users. In order to do so, the UA constructs a SIP URI for each registered address of record by taking the host component of the address of record and adding "sessionpolicy" as userinfo component to this address. For example, if an address of record is sip:bob@example.com, the UA would generate the URI sip:sessionpolicy@example.com. It uses this URI to subscribe to the policies of the home domain policy server of user Bob. Using "sessionpolicy" as the userinfo component enables proxies to route the request to a policy server.

The UA MUST also attempt to discover the policy servers of the access network provider. The UA MUST execute the discovery procedures in the order as they appear in the following sections. It SHOULD support all procedures but it MUST support the first (outbound proxy).

1. If the UA is configured with an outbound proxy for the current access domain (e.g. by using the procedures defined in [9] or via manual configuration) the UA MUST construct a URI by adding "sessionpolicy" as userinfo component to the address of the outbound proxy. For example, if the address of an outbound proxy is sip.example.com:6060, the UA would create the URI sip:sessionpolicy@sip.example.com:6060.

2. If no outbound proxy is configured or the subscription to the above URI fails, the UA SHOULD construct a fully qualified host name by adding "sessionpolicy" to the local domain name if defined (in analogy to the DNS lookup procedure defined in [9]). For example, a UA would create the host name sessionpolicy.example.com if the local domain is example.com. The UA SHOULD then try a DNS A record lookup on this domain name. If the name resolves in a DNS record, it should create a URI by adding "sessionpolicy" as userinfo component. In the above example, the UA would create the URI sip:sessionpolicy@policy.example.com.

3. If the above hostname could not be resolved via DNS, the UA SHOULD use multicast to subscribe to policies.

TBD: define this multicast address.

In addition to the discovered URIs, the UA may also have manually configured URIs to policy servers.

3.2 Subscribing to Session-Independent Policies

A UA sends SUBSCRIBE requests to the discovered policy server URIs. A UA compliant with this specification MUST support content indirection [8] including support for content indirection with multiple URIs. In addition, it MUST support the "application/basic-session-policy+xml" data format of the basic policy package described in Section 6. It MAY support the data formats of other policy packages. The UA SHOULD insert an Accept header into the SUBSCRIBE request. If an Accept header is created, its MUST include the MIME types "multipart/mixed", "message/external-body" and "application/basic-session-policy+xml". The UA MUST include the event package name "session-policy" in the Event header.

When creating the SUBSCRIBE request, the UA MUST populate the To

header field with the SIP URI of the policy server it wants to subscribe to. The UA populates the From header field with the address of record of the user for whom it subscribes. The UA MUST send a separate SUBSCRIBE message for each address of records to each policy server. The only exception are the home policy servers. This enables policy servers to learn about all users a UA has registered and allows them to provide different policies for each user. A home policy server will typically not be able to provide policies for foreign users. Therefore, a UA SHOULD only subscribe the associated address of record to a home policy server.

The UA SHOULD subscribe to all discovered home domain and all manually configured policy servers. It SHOULD subscribe to access network policy servers until the first successful response is received.

3.3 Accepting or Rejecting Session-Independent Policies

Once a UA has received a session policy from a policy server, it can decide whether it wants to accept or reject these policies. The UA does not need to inform the policy server or proxies about its decision.

The UA applies the accepted policies to new sessions it is creating. For example, if the policy lists the audio codecs allowed in a wireless network, the UA includes only those audio codecs in the session description offers and answers it creates. The way policies are applied differs from policy package to policy package and must be defined in the policy packet specification. The UA does not explicitly indicate its use of policies in an INVITE or UPDATE message. A proxy might be able to determine the acceptance of a session-independent policy by examining the actions of the UA (e.g. contacting a TURN relay) or the information it exposes in Media-Interface headers for session-specific policies [3]. The proxy may have mechanisms in place to enforce its policies.

A provider may have session-independent and session-specific policies in place, which influence the same aspect of a session (e.g. the codecs allowed). Since session-specific policies are requested during the establishment or modification of a session, they are applied after session-independent policies and may override them.

A UA, which has received an updated set of session-independent policies, MAY apply them to existing sessions for example by issuing a re-INVITE request.

4. Event Package Definition

This section provides the details needed to specify an event package as defined in Section 4.4 of RFC 3265 [10].

4.1 Event Package Name

The name of this package is "session-policy". As specified in RFC 3265 [10], this value appears in the Event header field present in SUBSCRIBE and NOTIFY requests.

```
Event: session-policy
```

4.2 Event Package Parameters

No package specific Event header field parameters are defined for this event package.

4.3 SUBSCRIBE Bodies

A SUBSCRIBE for session policy events MAY contain a body. This body would serve the purpose of filtering the subscription. The definition of such a body is outside the scope of this specification.

A SUBSCRIBE for the session policy package MAY be sent without a body. This implies that the default session policy filtering policy has been requested. The default policy is that notifications are generated every time there is any change in the policies for the user.

4.4 Subscription Duration

The default expiration of subscriptions to session policy state is one hour (3600 seconds).

4.5 NOTIFY Bodies

This event package uses content indirection to convey policy information. As such, the body of a NOTIFY message contains one or more URIs, which point to a policy object on a policy object server (typically a HTTP server). This saves bandwidth, allows for larger policy objects and enables a policy server to insert all current policies in a NOTIFY instead of tracking the policies that are actually updated or new to a UA. The UA can then decide, which policy objects it needs to retrieve.

All subscribers and notifiers MUST support the MIME types "multipart/

mixed" and "message/external-body" [8]. In addition, they MUST support the "application/basic-session-policy+xml" data format of the basic policy package described in Section 6. Subscribers MAY support the data formats of other policy packages. If the notifier receives a SUBSCRIBE request without an Accept header field, it MUST use the default value of "multipart/mixed, message/external-body, application/basic-session-policy+xml".

The data format of policy objects is not specified within this framework and must be defined in a policy package. A basic policy package for this framework is described in Section 6. A policy package may be based on XCAP [12]. XCAP enables fine grained access to XML-based configuration documents on a HTTP server.

4.6 Notifier Processing of SUBSCRIBE Requests

Session policy state can be sensitive information. Therefore, all subscriptions to it SHOULD be authenticated and authorized before approval. Authentication MAY be performed using any of the techniques available through SIP, including digest, S/MIME, TLS or other transport specific mechanisms. It is RECOMMENDED that a user be allowed to subscribe to their own session policy.

If the notifier determines that it can't provide any policy object to the subscriber now and in the near future, it SHOULD return a 480 "Temporarily Unavailable" response. This response SHOULD contain a Retry-After header indicating the time at which the subscriber should re-try the subscription. If the notifier determines that a policy object might become available in the near future, it SHOULD instead accept the subscription and create empty NOTIFY messages until the policy object is available.

If the policy server requires the use of a certain policy package and detects that the UA has not indicated support for the respective document format in the Accept header, it MAY reject a SUBSCRIBE request with a 406 "Not Acceptable" response. This way, a policy server can inform the user agent that is has session-independent policies place, which are mandatory but not understood. For example, a domain could restricts the use of certain audio codecs using a policy document format that is not understood by a user agent. By returning a 406 response to the SUBSCRIBE, the user agent would be informed that something might go wrong when establishing a session. Policy documents may contain attributes that define the required status for each policy on a fine grained basis.

4.7 Notifier Generation of NOTIFY Requests

Notifications SHOULD be generated whenever a change in one or more

session policy objects relevant to the subscribed user occurs.

A notifier will typically want to provide multiple policy objects to a user. For example, the notifier could have policy objects describing the general policies of a domain and user-specific policy objects, which describe policies only relevant for a particular user. It may also have policy objects that were created by the network provider and policy objects created by the user. The notifier MUST insert the URIs of all relevant policy objects into the NOTIFY message, even if only some of the policy objects have changed. In that sense, a notifier always provides complete state information to the subscriber. Each URI MUST have an associated Content-ID entity header, which MUST change every time the referred policy object changes. This enables subscribers to determine if they have the latest version of the policy object without downloading and comparing the objects.

The notifier MUST ensure that all URIs, it is inserting into a NOTIFY body, point to policy objects that are actually accessible on the policy object server. This is in particular important if the policy server creates policy objects on the fly. For example, a new policy object might be generated when a new user requests policies for the first time. A policy server MUST NOT delay the transmission of a NOTIFY if policy object is not yet available on the policy object server. Instead, it SHOULD not include the respective URI in the NOTIFY body and create an additional NOTIFY as soon as the new policy object is available.

If no policy object is available at the time a NOTIFY is created, the notifier SHOULD create an empty NOTIFY message which does not contain a body. This ensures that the subscription is established and the notifier can convey policy objects to the subscriber as soon as they become available.

4.8 Subscriber Processing of NOTIFY Requests

After receiving a NOTIFY, the subscriber MUST determine if any of the included URIs are pointing to a policy object, that is new or has been update since it was last downloaded. The subscriber SHOULD retrieve new or updated policy objects as soon as possible.

Session policies are typically created and maintained by network providers. They provide certain information to or request a certain behavior from user agents. The provider generated policies MUST NOT be changed by a user. However, a user MAY cerate personal session policies and store them on a policy server. A user may of course modify these policies. A policy object server MUST enforce access permissions to policy objects accordingly.

4.9 Handling of Forked Request

A subscriber establishes multiple subscription with different policy servers (home domain, access network domain, etc.). Similarly, a subscriber MAY establish multiple subscriptions on forked SUBSCRIBE requests. The NOTIFY messages created by the notifiers can be processed individually and do not need to be merged.

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

State agents have no role in the handling of this event package.

5. Policy Packages

This section describes aspects that need to be considered when packages for session-independent policies are defined.

5.1 Policy Object Format

This section MUST be present in a policy package.

A Policy Object (PO) is used to convey policies to the subscriber. Each package MUST specify or cite detailed specifications for the syntax and semantics associated with such a Policy Object.

The PO MUST have a version attribute that allows the recipient of POs to properly order them.

A PO MAY have an expires attribute that defines the time, at which the policy object expires. If this attribute is not present, a policy object expires at the time the subscription, through which it was received, is terminated. A policy object also expires when a policy object with a higher version number becomes available at the same URI.

A PO can have different scopes. It could be applicable to all sessions established by the UA or just to a subset of them. The scope of a PO MUST be defined in a policy package, by either specifying a default value or defining a scope attribute that can be populated by policy server when creating the PO. Possible scopes are:

- Sessions for a certain address of record (i.e. sessions created for a certain local user). This is useful if an end device supports multiple identities and, for example, only a subset of them has subscribed to a service requiring policies.
- Sessions to a certain remote URI. For example, a policy for NAT traversal might only apply to sessions to or from external addresses.
- Outgoing/incoming sessions only. A policy may apply only to sessions initiated by the local/the remote UA.
- A certain media stream. This enables the specification of policies on a stream-by-stream basis. For example, a policy for audio codec selection only applies to audio streams.
- Media streams in the incoming or outgoing direction. This enables independent policies for the media streams in each direction.

A PO SHOULD contain a consequences attribute. The consequences attribute is used by the policy server to indicate what the consequences of rejecting the policy are. The this attribute MUST also enable a UA to determine if the acceptance of a policy is mandatory or optional. If the consequences attribute is not present in a PO, the default value is used. The default value is optional if not defined otherwise in the policy package.

The PO MAY contain a signature attribute allowing the UA to verify the identity of the domain, which has requested policies, and the integrity of those policies.

A policy package MUST describe exactly how a UA is supposed to apply the policy contained in an PO. In particular, the policy package MUST describe how the information in the PO influences the session description a UA uses to establish sessions and if additional steps need to be taken either when accepting the policy or when setting up or modifying a session. This process MUST enable a UA to determine the consequences of accepting the policy before actually executing the necessary steps.

A PO MAY contain an attribute that explains the reasoning behind the session policy. The end device may present this text string to a human when querying whether the requested policies should be accepted or not.

5.2 XCAP Considerations

The developer of a policy package might find it helpful to specify the policy package based on XCAP [12]. In this case, a policy package defines an XCAP application usage specification.

6. Basic Session Policy Package

This section defines a basic policy package, that must be understood by all user agents.

Policy object is an XML document that MUST be well-formed and SHOULD be valid. Session policy documents MUST be based on XML 1.0 and MUST be encoded using UTF-8. This specification makes use of XML namespaces for identifying session policy documents. The namespace URI for elements defined by this specification is a URN [5], using the namespace identifier 'ietf' defined by RFC 2648 [6] and extended by [4]. This URN is:

urn:ietf:params:xml:ns:sessionpolicy

A session policy document begins with the root element tag "sessionpolicy".

6.1 Policy Object Format

A session policy document starts with a sessionpolicy element. This element has three mandatory attributes:

version: This attribute allows the recipient of session policy information documents to properly order them. Versions start at 0, and increment by one for each new document sent to a subscriber. Versions are scoped within a subscription. Versions MUST be representable using a 32 bit integer.

domain: This attribute contains the domain the policy belongs to.

entity: This attribute contains a URI that identifies the user whose media policy information is reported in the remainder of the document.

The sessionpolicy element has a series of sessionpolicy sub-elements: zero or one protocols element and zero or one media element.

6.1.1 Protocols Element

The protocols element contains a series of protocol sub-elements. Each protocol sub-element contains the policy related to the usage of a particular protocol.

The protocol element has a single mandatory attribute, name. The name attribute identifies a protocol the policy of each protocol element is referring to. The protocol element has a series of sub-elements: methods, option-tags, feature-tags, and bodies.

6.1.1.1 Methods Element

The methods element contains a default-policy attribute and method elements. The default-policy attribute contains the policy for methods that are not listed as method elements. A method element has two attributes: name and policy. The name attribute identifies a method, and the policy attribute contains the policy for that method (allowed or disallowed).

6.1.1.2 Option-tags Element

The option-tags element contains a default-policy attribute and option-tag elements. The default-policy attribute contains the policy for option-tags that are not listed as option-tag elements. An option-tag element has two attributes: name and policy. The name attribute identifies a method, and the policy attribute contains the policy for that method (mandatory, allowed, or disallowed).

6.1.1.3 Feature-tags Element

The feature-tags element contains a default-policy attribute and feature-tag elements. The default-policy attribute contains the policy for feature-tags that are not listed as feature-tag elements. An feature-tag element has two attributes: name and policy. The name attribute identifies a method, and the policy attribute contains the policy for that method (allowed, or disallowed).

6.1.1.4 Bodies Element

The bodies element contains a default-policy attribute, a default-encryption attribute and body-disposition elements. The default-policy attribute contains the policy for body dispositions that are not listed as body-disposition elements. The default-encryption attribute contains the encryption policy for body dispositions that are not listed as body-disposition elements.

A body-disposition element can have a number of attributes: name, policy, default-policy, and encryption. The name attribute identifies a body-disposition, and the policy attribute contains the policy for that body-disposition (allowed, or disallowed). The default-policy attribute contains the policy for body formats that are not listed as body-format elements. The encryption attribute indicates whether or not encryption is allowed for a particular body disposition.

A body-disposition element contains body-format elements. A body-format element can have a two attributes: name and policy. The name attribute identifies a body-format, and the policy attribute contains the policy for that body-format (allowed or disallowed).

6.1.1.5 Extensibility

Other elements from different namespaces MAY be present within a protocol element for the purposes of extensibility; elements or attributes from unknown namespaces MUST be ignored.

6.1.1.6 Example of a Protocol Element

```
<protocols>
  <protocol name="SIP">
    <methods default-policy="allowed">
       <method name="MESSAGE" policy="disallowed"/>
    </methods>
    <option-tags default-policy="disallowed">
       <option-tag name="100rel" policy="mandatory"/>
       <option-tag name="preconditions" policy="allowed"/>
    </option-tags>
    <feature-tags default-policy="disallowed">
       <feature-tag name="video" policy="allowed"/>
    </feature-tags>
    <bodies default-policy="allowed" default-encryption="allowed">
       <body-disposition name="session" policy="allowed"
                         encryption="disallowed" default-policy="disallowed">
          <body-format name="application/sdp" policy="allowed"/>
       </body-disposition>
    </bodies>
  </protocol>
</protocols>
```

6.1.2 Media Element

The media element contains the policy related to the characteristics of media streams of different types. It has three attributes: maxbandwidth, maxnostreams, and default-policy. They contain the maximum bandwidth the user can count on, the maximum number of media streams that the user is allowed to established at the same time, and the default policy (allowed or disallowed) for stream types that are not listed as stream elements.

The media element contains a series of stream elements.

6.1.2.1 Stream Element

A stream element can have a number of attributes: type, policy, maxbandwidth, and maxnostreams. The type attribute identifies a media type, and the policy attribute contains the policy for that media

type (allowed or disallowed).

The stream element has a number of optional sub-element: the codecs element, the transports element and the directions element.

6.1.2.1.1 Codecs Element

The codecs element contains a default-policy attribute and codec elements. The default-policy attribute contains the policy for codecs that are not listed as codec elements. A codec element can have two attributes: name and policy. The name attribute identifies a codec, and the policy attribute contains the policy for that codec (allowed, or disallowed).

6.1.2.1.2 Transports Element

The transports element contains a default-policy attribute and transport elements. The default-policy attribute contains the policy for transports that are not listed as transport elements. A transport element can have two attributes: name and policy. The name attribute identifies a transport, and the policy attribute contains the policy for that transport (allowed, or disallowed).

6.1.2.1.3 Directions Element

The directions element contains a default-policy attribute and direction elements. The default-policy attribute contains the policy for directions that are not listed as direction elements. A direction element can have two attributes: name and policy. The name attribute identifies a direction (sendrecv, sendonly, recvonly), and the policy attribute contains the policy for that direction (allowed, or disallowed).

6.1.2.1.4 Extensibility

Other elements from different namespaces MAY be present within a stream element for the purposes of extensibility; elements or attributes from unknown namespaces MUST be ignored.

6.1.2.2 Example of a Media Element

```
<media maxnostreams="4" default-policy="disallowed">
   <stream type="audio" policy="allowed">
        <codecs default-policy="allowed">
            <codec name="PCMU" policy="disallowed"/>
            <codec name="PCMA" policy="disallowed"/>
        </codecs>
```

```
          <transports default-policy="disallowed">
              <transport name="RTP/AVP" policy="allowed"/>
          </transports>
          <directions default-policy="disallowed">
              <direction name="sendonly" policy="allowed"/>
          </directions>
      </stream>
   </media>
```

6.2 Schema

The following is the schema for the application/session-policy+xml type:

```
<?xml version="1.0" encoding="UTF-8"?>
TBD
```

6.3 Example

The following is is an example of an application/session-policy+xml document:

```
<?xml version="1.0" encoding="UTF-8"?>
<sessionpolicy xmlns="urn:ietf:params:xml:ns:sessionpolicy"
               version="0"
               domain="example.com"
               entity="sip:alice@example.com">
 <protocols>
  <protocol name="SIP">
    <methods default-policy="allowed"/>
    <option-tags default-policy="allowed"/>
    <feature-tags default-policy="allowed"/>
    <bodies default-policy="allowed" default-encryption="allowed"/>
  </protocol>
 </protocols>
 <media default-policy="allowed"/>
</sessionpolicy>
```


7. Security Considerations

Session policy information can be sensitive information. The protocol used to distribute it SHOULD ensure privacy, message integrity and authentication. Furthermore, the protocol SHOULD provide access controls which restrict who can see who else's session policy information.

8. IANA Considerations

This document registers a new MIME type, application/session-policy+xml, and registers a new XML namespace.

8.1 MIME Registration for application/session-policy+xml

MIME media type name: application

MIME subtype name: session-policy+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [7].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [7].

Security considerations: See Section 10 of RFC 3023 [7] and Section 7 of this specification.

Interoperability considerations: none.

Published specification: This document.

Applications which use this media type: This document type has been used to download the session policy of a domain to SIP user agents.

Additional Information:

Magic Number: None

File Extension: .wif or .xml

Macintosh file type code: "TEXT"

Personal and email address for further information: Gonzalo Camarillo, <Gonzalo.Camarillo@ericsson.com>

Intended usage: COMMON

Author/Change controller: The IETF.

8.2 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:sessionpolicy

This section registers a new XML namespace, as per the guidelines in

[4]

URI: The URI for this namespace is urn:ietf:params:xml:ns:sessionpolicy.

Registrant Contact: IETF, SIPPING working group,<sipping@ietf.org>, Gonzalo Camarillo, <Gonzalo.Camarillo@ericsson.com>

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
     content="text/html;charset=iso-8859-1"/>
  <title>Session Policy Namespace</title>
</head>
<body>
  <h1>Namespace for Session Policy Information</h1>
  <h2>application/session-policy+xml</h2>
  <p>See <a href="[[[URL of published RFC]]]">RFCXXXX</a>.</p>
</body>
</html>
END
```

9. Open Issues

The following issues are still open:

- Policy server discovery. Is automatic discovery of home domain and access network domain policy server desirable? It seems that these are the two domains that will most likely want to provide policies. Automatic discovery of policy servers in both domains would make deployment of policies easier and manual configuration of multiple policy servers does not seem to be attractive. However, it complicates user agents and the current procedure overloads the user name sessionpolicies.

- XCAP. Would it make sense to require the use of XCAP for policy packages? XCAP already provides a number of functionalities that are most likely needed in a policy package. Requiring XCAP restricts the number of protocols that need to be supported by a client. Otherwise, a client that supports multiple packages might need to implement a lot of different protocols and document formats. Would requiring HTTP and XML do the job?

- Mandatory and optional policies. What is the best way to indicate if the acceptance of a policy is required or optional? Does it make sense to define this case-by-case or is a general required/ optional definition sufficient for a policy package?

References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[3] Hilt, V. and J. Rosenberg, "A Framework for Session-Specific Intermediary Session Policies in SIP", September 2003.

[4] Mealling, M., "The IETF XML Registry", draft-mealling-iana-xmlns-registry-05 (work in progress), June 2003.

[5] Moats, R., "URN Syntax", RFC 2141, May 1997.

[6] Moats, R., "A URN Namespace for IETF Documents", RFC 2648, August 1999.

[7] Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[8] Olson, S., "A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages", draft-ietf-sip-content-indirect-mech-03 (work in progress), June 2003.

[9] Petrie, D., "A Framework for SIP User Agent Configuration", draft-ietf-sipping-config-framework-00 (work in progress), March 2003.

[10] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[11] Rosenberg, J., "Traversal Using Relay NAT (TURN)", draft-rosenberg-midcom-turn-01 (work in progress), March 2003.

[12] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", draft-rosenberg-simple-xcap-00 (work in progress), May 2003.

[13] Rosenberg, J., "Requirements for Session Policy for the Session Initiation Protocol (SIP)", draft-ietf-sipping-session-policy-req-00 (work in progress), June 2003.

[14] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A.,

Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[15] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A. and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, August 2002.

Authors' Addresses

Volker Hilt
Bell Labs/Lucent Technologies
101 Crawfords Corner Rd
Holmdel, NJ 07733
USA

EMail: volkerh@bell-labs.com

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Gonzalo.Camarillo@ericsson.com

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Avenue
East Hanover, NJ 07936
USA

EMail: jdrosen@dynamicsoft.com

Appendix A. Acknowledgements

Many thanks to Markus Hofmann for his contributions to this draft.

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

Session Initiation Proposal V. Hilt
Investigation Working Group Bell Labs/Lucent Technologies
Internet-Draft J. Rosenberg
Expires: April 18, 2004 dynamicsoft
October 19, 2003

A Framework for Session-Specific Intermediary Session Policies in SIP
draft-hilt-sipping-session-spec-policy-00

Status of this Memo

Copyright Notice

Abstract

Proxy servers play a central role as an intermediary in the establishment of sessions in the Session Initiation Protocol (SIP). In that role, they define and impact policies on call routing, rendezvous, and other call features. However, there is currently no standard means by which network elements can have any influence on session policies, such as the codecs that are to be used. In this document, we propose a complete and standards-based framework that enables intermediaries to request the use of policies in a SIP session.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [9] was designed to support establishment and maintenance of end-to-end sessions. Proxy servers provide call routing, authentication and authorization, mobility, and other signaling services that are independent of the session. Effectively, proxies provide signaling policy enforcement. However, numerous scenarios have arisen which require the involvement of proxies in some aspect of the session policy. One scenario is in the traversal of a firewall or NAT. The midcom group has defined a framework for control of firewalls and NATs (generically, middleboxes) [10]. In this model, a midcom agent, typically a proxy server, interacts with the middlebox to open and close media pinholes, obtain NAT bindings, and so on. In this role as a midcom agent, the proxy will need to examine and possibly modify the session description in the body of the SIP message. This modification is to achieve a specific policy objective: to force the media to route through an intermediary.

In another application, SIP is used in a wireless network. The network provider has limited resources for media traffic. During periods of high activity, the provider would like to restrict codec usage on the network to lower rate codecs. In existing approaches, this is frequently accomplished by having the proxies examine the SDP [4] in the body and remove the higher rate codecs or reject the call and require the UA to start over with a different set of codecs.

In yet a third application, SIP is used in a network that has gateways which support a single codec type (say, G.729). When communicating with a partner network that uses gateways with a different codec (say, G.723), the network modifies the SDP to route the session through a converter that changes the G.729 to G.723.

The desire to impact aspects of the session occurs in domains where the administrator of a SIP domain is also the owner and administrator of an IP network over which it is known that the sessions will traverse. This includes enterprises, Internet access providers, and in some cases, backbone providers. Typical session policies established in such domains influence NAT/firewall traversal or control bandwidth usage by selecting low-rate codecs. The desire to impact aspects of a session also occurs in domains that provide services to a user. Typical session policies enable the inclusion of a media intermediary in the media path such as a transcoding gateway or a call recording server.

In general, session policies enable a domain to impact aspects of a session description, ask a UA to perform extra steps when establishing a session (e.g. contact a NAT/firewall) or request the

exposure of details about session that is being set up or modified to a proxy. Since SIP is the protocol by which the details of these sessions are negotiated, it is natural for providers to wish to impose their session policies through some kind of SIP means. To date, this has been accomplished through SDP editing, a process where proxies dig into the bodies of SIP messages, and modify them in order to impose their policies. However, this SIP editing technique has many drawbacks as discussed in [7].

Our solution is to introduce a framework that allows intermediary elements to request session policies from user agents when a session is being set up or modified. It enables proxies to examine aspects of the session description currently being used and to dynamically create the policies that apply to this session. Policies, that are independent of a certain session description and may affect multiple sessions, can be requested using the framework described in [2]. This framework satisfies the requirements listed in [7].

This document is structured as follows: Section 3 introduces a framework for requesting session-specific policies during the establishment or modification of a session. Section 4 discusses the creation of policy packages for this framework and Section 5 describes an example policy package for selecting codecs. Section 6 discusses Security and Section 7 IANA considerations. Section 8 describes the syntax of SIP extensions defined in this document. Section 9 talks about open issues.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [3] and indicate requirement levels for compliant implementations.

3. Framework for Session-Specific Policies

This framework for session-specific policies enables proxy servers to request session policies from UAs during the creation or modification of a session. The syntax and semantics of a specific session policy is not part of this framework and needs to be defined in a separate session policy package (see Section 4). An example can be found in Section 5.

The basic operation of the framework consists of two steps: first, UAs expose aspects of their session description to proxies in Media Interface Objects (MIOs). For example, a UA can create an MIO describing the IP addresses and ports of each media stream and another MIO describing the set of codecs it supports. Having this information in an MIO frees proxies from the burden of finding and understanding session descriptions in message bodies. In the second step, the proxies request session policies for a session by creating Media Filter Objects (MFOs). An MFO contains a set of rules, the UA is requested to execute on a certain media aspect. For example, an MFO could contain a list of codecs allowed in a session. Each proxy on the way can request policies independently. MIOs and MFOs are only useful in conjunction with a session description and must travel in the same SIP message (e.g. in a INVITE request and a 200 OK response).

Policies are set up independently for media streams in both directions. An INVITE request with an SDP offer can establish the policies for media streams sent from UAS to UAC whereas the corresponding INVITE response establishes policies for media streams sent from UAC to UAS. It is important to note that, the policies for each direction can be completely independent of each other. For example, the media streams from UAS to UAC could be directed through a firewall whereas the media streams from UAC to UAS could be sent directly. This differs from session descriptions, where the answer is always based on the contents of the offer.

Summing up, the following steps are executed to request policies for media streams in one direction:

1. The receiver of a media stream creates MIOs and inserts them into the SIP messages, that carries the corresponding session description.

2. Proxies inspect those MIOs insert MFOs.

3. The sender of the media stream receives the SIP message, examines the session description and the MFOs and decides whether it wants to accept or reject the the requested policies. It applies the

accepted policies.

4. The accepted policies are conveyed back to the receiver of a media stream.

3.1 Requesting Policies using Request/Response/ACK

Proxies can request policies in INVITE and UPDATE [6] transactions, in which the session description offer is carried in the request and an answer is carried in the response. The basic message flow is depicted in Figure 1.

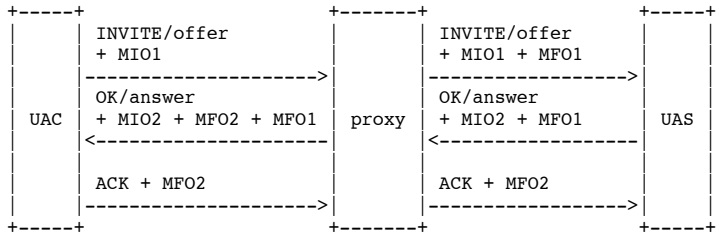


Figure 1

3.1.1 Constructing the INVITE/UPDATE Request

The UAC composes an INVITE or UPDATE request as usual. In addition to the session description, it creates MIOs for those aspects of the session, it wishes to permit the network to examine. For example, if the UAC wants to allow the network to examine the media codecs, it would insert MIOs representing these codecs. The UAC SHOULD expose as much information as possible in MIOs.

Since the MIOs are meant to be inspected by proxies, and since they are provided to enable a SIP feature (proxy insertion of session policy), the MIOs are carried as SIP headers (see Section 3.3).

A UAC that supports this framework MUST insert a SIP Supported header with the option tag "policy".

3.1.2 Proxy Processing of Requests

As the request traverses proxies, the proxies can insert Media Filter Objects (MFOs). MFOs contain the policies, the proxy wants to request. A proxy can generate MFOs in response to information contained in a specific MIO in the request. These MFOs represent "diffs" that the proxy wants to apply to the MIO. For example, if an MIO contains an IP address and port for receiving an audio stream, a proxy can insert an MFO which changes that address and port to that of a media intermediary. A proxy may inspect MFOs that have been inserted by previous proxies to determine, which policies are already requested. However, MFOs created by a proxy MUST represent the differences to the original MIO. A proxy can also generate MFOs independent of the MIOs contained in the request. Such an MFO describes a general policy applicable to the current session. For example, an MFO could contain a list of audio codecs that are allowed in the current session.

The session description contained in an INVITE/UPDATE request describes media streams transmitted from UAS to UAC. Consequently, MFOs inserted into an INVITE/UPDATE request MUST contain policies for media streams transmitted in this direction.

The proxy does not modify the MIO - that is fundamental. By specifying the requested modifications in MFOs rather than directly modifying MIOs and the session description, we enable an explicit consent and knowledge model. The UAs can know exactly, which policies where requested against the session.

A proxy MAY only insert MFOs (or other policy related headers) into the INVITE/UPDATE request, if the UAC has indicated its support for policies by including a Supported header with the value "policy" into the request. If no such Supported header was present and the proxy insists on the use of policies, it MAY return a 421 (Extension Required) response. However, this behavior is NOT RECOMMENDED as it generally breaks interoperability.

A proxy MAY insert a Require header with the option tag "policy" if it wants to make sure that the request fails in case the UAS does not support session policies. A proxy MUST insert a policy Require header if it has marked some policies as required in the MFO (see Section 3.4). However, not all session policies will be mandatory. Policies could be optional, in which case none of the inserted MFOs would contain a required policy and a policy Require header would not be inserted.

If an MIO contained in the request is not acceptable to the proxy, it MAY insert an MFO indicating the failure or it MAY reject the request

by returning a 488 (Not Acceptable Here) response. This enables a proxy to inform the UAC that the information in the MIO is not acceptable under the current policies or that information required by the current policy was not exposed in an MIO. For example, a proxy, which wants to route a media stream through a firewall, would not accept MIOs containing no information about the transport address. The failure MFO SHOULD explain the reason, why the MIO was not acceptable. Similarly, the 488 response SHOULD include a Warning header field value explaining why the request was rejected. The proxy SHOULD copy the MFOs that caused the problems from the request into the 488 response. This allows the UAC to know exactly why the request has failed and if it can attempt to retry with different MIOs.

[TBD: define warning codes and texts.]

To achieve backwards compatibility with devices that do not support policies, the proxy MUST NOT return a 488 response to requests that do not include a Supported header with the value "policy". However, a proxy SHOULD reject requests if the UAC has indicated its support for policies and knows how to correct the problem and re-try the request. Rejecting a request is a quick way for the proxy to inform a policy-enabled UAC about policy related problems. It prevents that the request is forwarded to the UAS, which would then reject it because of an included failure MFO. Returning a 488 response can't be used to enforce a policy. Such an enforcement would not be effective since it can be circumvented by a UAC, for example by creating fake MIOs.

In addition to adding an MFO, a proxy MAY generate an MFO-Reason header. This header contains the domain name of the proxy and explains the reasoning behind the session policy. The end device may present this text string to a human when querying whether the requested policies should be accepted or not.

[TBD: define the format to this header.]

A proxy that supports forking of requests, MAY generate a different set of MFOs for each target the request is sent to.

3.1.3 Processing Requests and Generating Responses

When the INVITE/UPDATE request reaches the UAS, the UAS will know exactly what the UAC indicated in MIOs, and which policies have been requested by intermediate domains. The UAS decides if it wants to accept some or all of these policies. If it decides to reject a policy that is marked as required or if the message contains a failure MFO, the UAS MUST reject the request with a 488 (Not Acceptable Here) response. This response SHOULD include a Warning

header field value explaining, why the policies were not acceptable and a copy of the declined MFOs or the failure MFO.

If all required (and possibly some optional) policies are acceptable to the UAS, it will eventually generate a response which contains a session description answer. If both user agents support reliable provisional responses [8], it is RECOMMENDED that the UAS returns the answer in a reliable provisional response. Using a reliable provisional response has the advantage that the UAC has the chance to reject policies before the session is established.

The UAS then inserts its own set of MIOs for its side of the session into the response. It MUST copy all MFOs it has accepted (required and optional) from the request into the response. The copied MFOs are purely informational, for the benefit of the proxy and the UAC. They inform proxies which policies have been accepted. They also ensure that proxies cannot establish policies without having the UAC become aware of them. The copied MFOs are end-to-end, and not meant for modification by proxies. They MAY be protected by end-to-end security mechanisms.

A UAS MUST NOT apply the procedures defined in this specification to INVITE/UPDATE requests, that don't contain a Supported header with value "policy". If a UAS applies this specification, it MUST insert a Require header with the value "policy" into the response it creates. A Supported header with the value "policy" MUST be included in every response to an INVITE/UPDATE request.

3.1.4 Proxy Processing of Responses

If the response contains a Require header with the value "policy", the proxy knows that the UAC and the UAS support the use of session policies and that it may apply this extension. The proxy can determine which policies have been accepted by the UAS by examining the list of MFOs, the UAS has copied into the response.

The proxy can insert MFOs containing policies for media streams transmitted from UAC to UAS into the response to an INVITE request. These MFOs are created and formatted identically to those inserted into the request. If the MIOs contained in the response are not acceptable to a proxy, it may insert a failure MFO.

A proxy could also insert MFOs into the response to an UPDATE request. However, these MFOs would not be copied back to the UAS since UACs do not PRACK or ACK UPDATE responses. Thus, the proxy would not be informed which policies have been accepted and the UAS would not become aware of these policies. Such a behavior violates the requirement that both UAs need to know the set of policies

requested along the call path and that the proxy needs to be informed about accepted policies. It is therefore NOT RECOMMENDED.

[Open issue: would it make sense to send an additional message from UAC to UAS or to get rid of sending MFOs back. See Section 9.]

3.1.5 Processing Responses and Generating ACKs

After receiving a 1xx or 2xx response, the UAC examines if a Requires header with the value "policy" is present and if the response contains MFOs. If so, it can either reject or accept the policies. If it accepts all policies marked required, the UAC MUST copy the MFOs, that were accepted, into the PRACK or ACK. These MFOs are informational to the proxy and the UAS. They may be protected by end-to-end integrity mechanisms. Due to forking of requests in proxies, the UAC may receive multiple responses from different UASs for one request, which may contain different policies. If the response did not contain a policy Requires header, the UAC must ignore all policy related information in the response (e.g. MFOs).

If the UAC decides to reject some of the required policies or if the response contained a failure MFO, the UAC should terminate the dialog associated with this response. If the UAS has responded with a 2xx response, the UAC must send an ACK and then terminate the dialog with a BYE. If the UAS has responded with a reliable provisional response, the UAC can terminate the dialog without fully establishing it by generating a CANCEL (after sending a PRACK, of course). The UAC does not copy the MFOs from the request into the PRACK or ACK. Instead, the declined MFOs SHOULD be copied into the BYE or CANCEL requests together with a Reason header [5] explaining why the policies were rejected.

[TBD: need to define reason code, phrases etc.]

If the UAC receives a 488 response and the reason explains that existing or missing MIOs caused the rejection, the UAC MAY try to correct the problem (e.g. by adding an additional MIO) and re-send the request.

3.1.6 Processing ACKs

If the MFOs contained in a PRACK or ACK message are not acceptable to the UAS, it may decline them by terminating the dialog with a CANCEL or BYE. The CANCEL or BYE SHOULD contain a copy of the declined MFOs and a Reason header [5] explaining why these policies were rejected.

3.1.7 Applying Policies

If both UAs have accepted the policies, they MUST apply them to the media streams they generate. This may involve, for example, sending media to an intermediary indicated in an MFO. Since the user agents know about the full set of intermediaries, they have many options in the event of a failure (detected through an ICMP error, for example). The endpoint can try to send the media to the next intermediary on the path. Or, if the MFO specifies the intermediaries as a FQDN instead of an IP address, the endpoint can attempt to use DNS to find an alternative, and begin routing media through that.

3.2 Requesting Policies using Response/ACK

Proxies may also request policies in INVITE transactions, which carry a session description offer in the response and an answer in the following ACK request. The basic message flow is depicted in Figure 2.

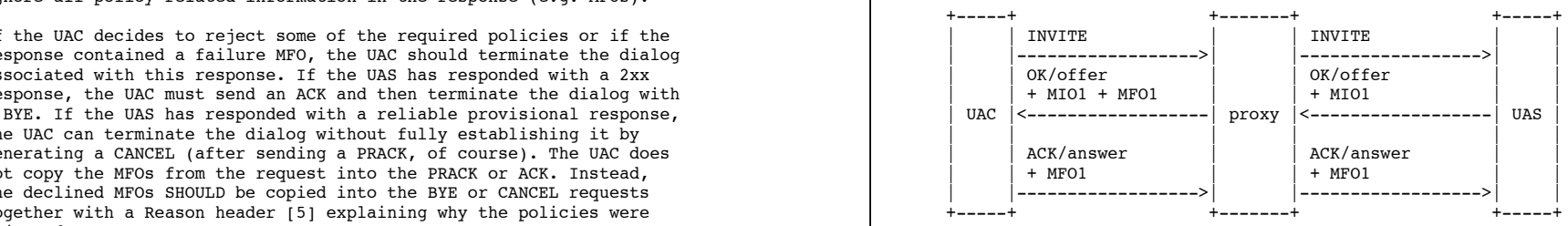


Figure 2

3.2.1 Creating the INVITE Response

The UAS creates the response as usual. It applies this extension to the response, if the request contains a Supported header with the value "policy". The UAS MUST insert a Require header with the value "policy" and SHOULD insert all MIOs it can create for its side of the session description. A Supported header with the value "policy" MUST be included in every response to an INVITE/UPDATE request.

It is RECOMMENDED that the UAS generates a reliable provisional response [8] if supported by both UAs.

3.2.2 Proxy Processing Responses

The proxy MAY add MFOs to responses that contain a Requires header with the value "policy". If an MIO contained in the response is not acceptable for the proxy, it MAY insert a failure MFO.

3.2.3 Processing Responses and Generating ACKs

The UAC may or may not accept the policies contained in the response. If it accepts all required policies, it MUST copy the accepted MFOs into the PRACK or ACK. It may protect these MFOs with end-to-end integrity mechanisms. If it declines at least one of the required policies or if the response contained a failure MFO, the UAC does not copy these MFOs into the PRACK or ACK and SHOULD terminate the dialog associated with this response.

3.2.4 Proxy Processing of ACKs/PRACKs

The proxy could insert MFOs into the PRACK or ACK. However, these MFOs would not be copied back to the UAC, which would violate the requirement that both UAs and the proxy should know the set of policies used in a session. This behavior is therefore NOT RECOMMENDED.

3.2.5 Processing ACKs/PRACKs

If the MFOs contained in a PRACK or ACK message are not acceptable to the UAS, it may decline them by terminating the dialog.

3.3 "Media-Interface" header usage

The Media-Interface header value contains Media Interface Objects (MIOs) created by a UA. The structure and semantics of MIOs needs to be defined in a policy package. However, the following general rules apply to Media-Interface header values:

The Media-Interface header value MUST consist of the policy package name, under which the MIO was created.

The Media-Interface header MAY contain a signature parameter which enables proxies to verify the identity of the UA and the integrity of the MIOs.

A UA creates a separate Media-Interface header value for each policy package it supports. A policy package MAY require the creation of multiple Media-Interface headers with different MIOs. The UAC SHOULD create MIOs for all policy packages it supports. MIOs SHOULD contain as much information about the session as possible.

In the following example, the UA supports the packages foo and bar. It exposes data1 and data2 for package foo and data3 for package bar in MIOs.

Media-Interface: foo;foo_param1=data1;foo_param2=data2, bar;bar_param=data3

3.4 "Media-Filter" header usage

Media-Filter headers serve as a container for Media Filter Objects (MFOs). Each MFO is contained in a separate Media-Filter header value. Media-Filter header values implement a stack, which enables each proxy on the way to "push" its MFOs on top of the set of existing MFOs. The Media-Filter headers implement one single stack, which contains the MFOs for all packages. If a proxy wants to insert an MFO, it inserts the respective Media-Filter header value before the topmost Media-Filter header value.

A UA, which receives a SIP message containing MFOs, processes them one after another and removes the processed element from the stack.

The structure and semantics of MFOs needs to be defined in a policy package. However, the following general rules apply to Media-Filter header values:

The Media-Filter header value MUST consist of the policy package name, under which the MFO was created.

The following general parameters are defined for Media-Filter headers. They provide basic information about the MFO to UAs even if they don’t support the policy package used.

- domain. The domain parameter carries the identity of the domain, which requested the policy. It MUST be present in each MFO.

- cns (consequences). The cns parameter is be used by the proxy to indicate the consequences of rejecting the policy to the UA. This parameter also enables a UA to determine if the acceptance of a policy is mandatory for establishing the session or not. The cns parameter contains a consequences code, which has a "required" and an "optional" range. An MFO SHOULD contain a consequences code. An MFO is optional if the cns parameter is not present.

- signature. A MFO MAY contain a signature, generated by the domain that inserted the MFO. This allows the endpoints to verify the identities of the domains, which have requested session policy, and the integrity of those policies.

[TBD: define consequence codes.]

A failure MFO is a special MFO, which indicates that the session is not acceptable to the proxy. A failure MFO is an MFO with consequence code 999. Additional package specific parameters MAY be present in a failure MFOs.

[TBD: define reason codes and texts for failure MFOs.]

In the following example, the proxy in domain example1.com has requested policies for package foo and the proxy in domain example2.com has requested policies for the packages foo and bar.

Media-Filter: foo;domain=example2.com;cns=100;foo_param=data1, bar;domain=example2.com;cns=300;bar_param=data1, foo;domain=example1.com;foo_param=data2,

3.5 "Reverse-Media-Filter" header usage

The Reverse-Media-Filter header is used to convey the MFOs, a UA has accepted, back to the peer UA. A Reverse-Media-Filter header contains a copy of the accepted MFOs and has the same structure as the Media-Filter header.

4. Session-Specific Policy Packages

This section describes aspects that need to be considered when session-specific policy packages are defined.

4.1 Media Interface Object

This section MUST be present in a policy package. It defines the structure of Media Interface Objects used within this package.

A policy package MUST describe the semantics of an MIO. It MUST describe how proxies are supposed to interpret the information contained in an MIO.

4.2 Media Filter Object

This section MUST be present in a policy package. It defines the structure of Media Filter Objects used within this package.

Media Filter Objects (MFOs) may define the differences to an existing MIO. However, it is very important that MFOs don't just define a diff to an MIO, in the Unix sense. This is because it is important that the endpoints understand the semantics of a requested policy, not just the syntactical change that is needed to affect that policy. A MFO may also define a general policy which is independent of an MIO.

A policy package MUST describe exactly how a UA is supposed to apply the policy contained in an MFO. In particular, the policy package MUST describe how the information in the MFO is applied to the session description and if additional steps need to be taken when accepting the policy. This process MUST enable a UA to determine the consequences of accepting the policy before actually executing the necessary steps.

5. Example Policy Package: Network-based Codec Selection

5.1 Session-Specific Codec Selection

This policy package enables a proxy to influence the codecs that are used within a session. The UAs are enabled to expose the codecs they support in MIOs. The MFOs created by the proxy contain the list of codecs allowed in the domain. The package is currently defined based on session descriptions in SDP [4] format. However, it is not restricted to SDP and can be used with other session description formats respectively.

The name of this package is "codec". This package name is carried in the Media-Interface, the Media-Filter and the Reverse-Media-Filter header as defined in this specification.

5.1.1 Media Interface Object

A codec MIO describes the codecs that are supported by the UA creating the MIO.

This policy package defines a media type parameter for codec MIOs (in addition to the general parameters for MIOs).

The parameter name consists of the media type, for which this MIO provides a policy. If used with a SDP session description, it MUST have the same value as the media name attribute in the media description (m=) of the corresponding SDP announcement. Typical values are "audio", "video", "application" and "data".

The value of this parameter consists of a media stream id and one or more codec formats. The media stream id provides an identifier for a media stream. It MUST have a value that is unique within the scope of the session description. The media stream id MUST be present in each codec MIO and it MUST NOT be zero. The codec format describes the codecs allowed for this media type. The format of the value is specific to each media type and has the same structure as the SDP rtpmap parameter. A UA SHOULD list all codecs is has listed for the media stream in the corresponding session description. All elements of the parameter value are concatenated with a "+" symbol.

An example for a Media-Interface header containing a codec MIO is

```
Media-Interface: codec;audio=7736ai+pcmu/8000/1+pcma/8000/1+
eg711u/8000/1;video=hha9s8sd0+h261/90000
```

This header specifies two media streams, an audio and a video stream. The available audio codecs are pcmu, pcma, and eg711u. The only video

codec supported is h261.

A proxy would create the following SDP announcement template from this MIO:

```
m=audio <port1> RTP/AVP 0 8 <no1>
a=rtpmap:0 pcmu/8000/1
a=rtpmap:8 pcma/8000/1
a=rtpmap:<no1> eg711u/8000/1
m=video <port2> RTP/AVP 31
a=rtpmap:31 h261/90000
```

5.1.2 Media Filter Object

A codec MFO describes the list of codecs that are allowed under this session policy.

In addition to the general header parameters, this policy package defines a media type parameter, which is structured exactly as the media type parameter in codec MIOs. The semantics of this parameter is as follows:

The media stream id MUST refer to a media stream contained in an MIO or contain the value zero. If the media stream id refers to a media stream in an MIO, the codec policy applies only to the referred media stream. If the media stream id is zero, the policy apply to all streams of the respective media type. A proxy MAY insert multiple media type parameters with different media stream id's for the same media type, if it wants to define different policies for different streams of the same type.

The media format element MUST list all codecs that are allowed under the current policy. It MAY contain codecs that are not listed in a respective MIO.

[TBD: Define consequence codes.]

An example for a Media-Filter header containing a codec MFO is

```
Media-Filter: codec;domain=example1.com;
audio=0+pcmu/8000/1+eg711u/8000/1,
codec;domain=example2.com;cns=100;
audio=0+eg711u/8000/1;video=0
```

This header contains two MFOs, one inserted by proxy example1.com and one by example2.com. The policy of domain example1.com is that the set of allowed audio codecs is limited to pcmu and eg711u.

Consequences for UAs rejecting this policy are not defined, which also indicates that this policy is optional. Domain example1.com has no policy for video codecs. The policy of domain example2.com is that only audio codec eg711u and no video can be used. Consequence of rejecting this policy is code 100, which indicates that the policy is mandatory. All policies apply to audio and video streams in general and are not bound to a stream listed in the MIO.

A UA would create the following SDP filter from these MFOs:

```
m=audio <port1> RTP/AVP <no1>
a=rtpmap:<no1> eg711u/8000/1
m=video <port2> RTP/AVP
```

A UA, that accepts this policy, removes all audio and video codecs that are not listed in the SDP filter.

6. Security Considerations

[TBD.]

7. IANA Considerations

[TBD.]

8. Syntax

This section describes the syntax extensions required for session policies.

8.1 Header Fields

This table expands on tables 2 and 3 in SIP [9] and on table 1 and table 2 in Reliability of Provisional Responses in SIP [8].

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	PRACK
Media-Interface		r	o	-	-	o	-	-	o
Media-Filter		a	o	-	-	o	-	-	o
Reverse-Media-Filter	r		-	-	-	o	-	-	-
Reverse-Media-Filter			o	-	-	-	-	-	o

9. Open Issues

The following issues are still open:

- Three way vs. two way. The current draft proposes a three way exchange to set up policies. The third way is purely informative and is needed to satisfy the following two requirements (see [7]): 1) both UAs need to know about all policies (REQ-CON-1 and REQ-CON-2) and 2) the proxy needs to know which policies have been accepted (REQ-CON-5 and REQ-CON-6). However, the third way is troublesome with empty INVITEs and UPDATEs. One option would be to use an extra message (SUBSCRIBE/NOTIFY, INFO,...) on the third way. Another option would be to get rid of the above requirements and the third way. In that case, we would switch to the "one-party consent" model (used in OPES [1]) since only one UA (the sender of a media stream) would know about and agree to policies. Need investigate, if the "one-party consent" model is applicable to SIP sessions.

- Preconditions. Preconditions could be used to prevent "ghost rings" in case the UAC declines policies. This needs further investigation.

References

[1] Barbir, A., "An Architecture for Open Pluggable Edge Services (OPES)", draft-ietf-opes-architecture-04 (work in progress), December 2002.

[2] Hilt, V. and J. Rosenberg, "A Framework for Session-Independent Intermediary Session Policies in SIP", September 2003.

[3] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[4] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[5] Oran, D., Schulzrinne, H. and G. Camarillo, "The Reason Header Field for the Session Initiation Protocol", draft-ietf-sip-reason-01 (work in progress), May 2002.

[6] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

[7] Rosenberg, J., "Requirements for Session Policy for the Session Initiation Protocol (SIP)", draft-ietf-sipping-session-policy-req-00 (work in progress), June 2003.

[8] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.

[9] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[10] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A. and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, August 2002.

Authors' Addresses

Volker Hilt
Bell Labs/Lucent Technologies
101 Crawfords Corner Rd
Holmdel, NJ 07733
USA

EMail: volkerh@bell-labs.com

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Avenue
East Hanover, NJ 07936
USA

EMail: jdrosen@dynamicsoft.com

Appendix A. Acknowledgements

Many thanks to Markus Hofmann for his contributions to this draft.

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING
Internet-Draft
Expires: August 16, 2004

J. Rosenberg
dynamicsoft

February 16, 2004

A Framework for Application Interaction in the Session Initiation Protocol (SIP)

draft-ietf-sipping-app-interaction-framework-01

Status of this Memo

Copyright Notice

Abstract

This document describes a framework for the interaction between users and Session Initiation Protocol (SIP) based applications. By interacting with applications, users can guide the way in which they operate. The focus of this framework is stimulus signaling, which allows a user agent to interact with an application without knowledge of the semantics of that application. Stimulus signaling can occur to a user interface running locally with the client, or to a remote user interface, through media streams. Stimulus signaling encompasses a wide range of mechanisms, ranging from clicking on hyperlinks, to pressing buttons, to traditional Dual Tone Multi Frequency (DTMF) input. In all cases, stimulus signaling is supported through the use of markup languages, which play a key role in this framework.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [1] provides the ability for users to initiate, manage, and terminate communications sessions. Frequently, these sessions will involve a SIP application. A SIP application is defined as a program running on a SIP-based element (such as a proxy or user agent) that provides some value-added function to a user or system administrator. Examples of SIP applications include pre-paid calling card calls, conferencing, and presence-based [10] call routing.

In order for most applications to properly function, they need input from the user to guide their operation. As an example, a pre-paid calling card application requires the user to input their calling card number, their PIN code, and the destination number they wish to reach. The process by which a user provides input to an application is called "application interaction".

Application interaction can be either functional or stimulus. Functional interaction requires the user agent to understand the semantics of the application, whereas stimulus interaction does not. Stimulus signaling allows for applications to be built without requiring modifications to the client. Stimulus interaction is the subject of this framework. The framework provides a model for how users interact with applications through user interfaces, and how user interfaces and applications can be distributed throughout a network. This model is then used to describe how applications can instantiate and manage user interfaces.

2. Definitions

SIP Application: A SIP application is defined as a program running on a SIP-based element (such as a proxy or user agent) that provides some value-added function to a user or system administrator. Examples of SIP applications include pre-paid calling card calls, conferencing, and presence-based [10] call routing.

Application Interaction: The process by which a user provides input to an application.

Real-Time Application Interaction: Application interaction that takes place while an application instance is executing. For example, when a user enters their PIN number into a pre-paid calling card application, this is real-time application interaction.

Non-Real Time Application Interaction: Application interaction that takes place asynchronously with the execution of the application. Generally, non-real time application interaction is accomplished through provisioning.

Functional Application Interaction: Application interaction is functional when the user device has an understanding of the semantics of the interaction with the application.

Stimulus Application Interaction: Application interaction is considered to be stimulus when the user device has no understanding of the semantics of the interaction with the application.

User Interface (UI): The user interface provides the user with context in order to make decisions about what they want. The user enters information into the user interface. The user interface interprets the information, and passes it to the application.

User Interface Component: A piece of user interface which operates independently of other pieces of the user interface. For example, a user might have two separate web interfaces to a pre-paid calling card application - one for hanging up and making another call, and another for entering the username and PIN.

User Device: The software or hardware system that the user directly interacts with in order to communicate with the application. An example of a user device is a telephone. Another example is a PC with a web browser.

User Input: The "raw" information passed from a user to a user interface. Examples of user input include a spoken word or a click on a hyperlink.

Client-Local User Interface: A user interface which is co-resident with the user device.

Client Remote User Interface: A user interface which executes remotely from the user device. In this case, a standardized interface is needed between the user device and the user interface. Typically, this is done through media sessions - audio, video, or application sharing.

Media Interaction: A means of separating a user and a user interface by connecting them with media streams.

Interactive Voice Response (IVR): An IVR is a type of user interface that allows users to speak commands to the application, and hear responses to those commands prompting for more information.

Prompt-and-Collect: The basic primitive of an IVR user interface. The user is presented with a voice option, and the user speaks their choice.

Barge-In: In an IVR user interface, a user is prompted to enter some information. With some prompts, the user may enter the requested information before the prompt completes. In that case, the prompt ceases. The act of entering the information before completion of the prompt is referred to as barge-in.

Focus: A user interface component has focus when user input is provided fed to it, as opposed to any other user interface components. This is not to be confused with the term focus within the SIP conferencing framework, which refers to the center user agent in a conference [12].

Focus Determination: The process by which the user device determines which user interface component will receive the user input.

Focusless User Interface: A user interface which has no ability to perform focus determination. An example of a focusless user interface is a keypad on a telephone.

Presentation Capable UI: A user interface which can prompt the user with input, collect results, and then prompt the user with new information based on those results.

Presentation Free UI: A user interface which cannot prompt the user with information.

Feature Interaction: A class of problems which result when multiple applications or application components are trying to provide services to a user at the same time.

Inter-Application Feature Interaction: Feature interactions that occur between applications.

DTMF: Dual-Tone Multi-Frequency. DTMF refer to a class of tones generated by circuit switched telephony devices when the user presses a key on the keypad. As a result, DTMF and keypad input are often used synonymously, when in fact one of them (DTMF) is merely a means of conveying the other (the keypad input) to a client-remote user interface (the switch, for example).

Application Instance: A single execution path of a SIP application.

Originating Application: A SIP application which acts as a UAC, calling the user.

Terminating Application: A SIP application which acts as a UAS, answering a call generated by a user. IVR applications are terminating applications.

Intermediary Application: A SIP application which is neither the caller or callee, but rather, a third party involved in a call.

3. A Model for Application Interaction

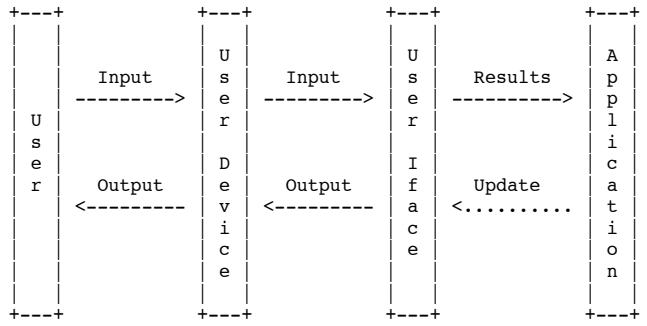


Figure 1: Model for Real-Time Interactions

Figure 1 presents a general model for how users interact with applications. Generally, users interact with a user interface through a user device. A user device can be a telephone, or it can be a PC with a web browser. Its role is to pass the user input from the user, to the user interface. The user interface provides the user with context in order to make decisions about what they want. The user enters information into the user interface. The user interface interprets the information, and passes it to the application. The application may be able to modify the user interface based on this information. Whether or not this is possible depends on the type of user interface.

User interfaces are fundamentally about rendering and interpretation. Rendering refers to the way in which the user is provided context. This can be through hyperlinks, images, sounds, videos, text, and so on. Interpretation refers to the way in which the user interface takes the "raw" data provided by the user, and returns the result to the application in a meaningful format, abstracted from the particulars of the user interface. As an example, consider a pre-paid calling card application. The user interface worries about details such as what prompt the user is provided, whether the voice is male or female, and so on. It is concerned with recognizing the speech that the user provides, in order to obtain the desired information. In this case, the desired information is the calling card number, the PIN code, and the destination number. The application needs that data, and it doesn't matter to the application whether it was collected using a male prompt or a female one.

User interfaces generally have real-time requirements towards the user. That is, when a user interacts with the user interface, the user interface needs to react quickly, and that change needs to be propagated to the user right away. However, the interface between the user interface and the application need not be that fast. Faster is better, but the user interface itself can frequently compensate for long latencies there. In the case of a pre-paid calling card application, when the user is prompted to enter their PIN, the prompt should generally stop immediately once the first digit of the PIN is entered. This is referred to as barge-in. After the user-interface collects the rest of the PIN, it can tell the user to "please wait while processing". The PIN can then be gradually transmitted to the application. In this example, the user interface has compensated for a slow UI to application interface by asking the user to wait.

The separation between user interface and application is absolutely fundamental to the entire framework provided in this document. Its importance cannot be overstated.

With this basic model, we can begin to taxonomize the types of systems that can be built.

3.1 Functional vs. Stimulus

The first way to taxonomize the system is to consider the interface between the UI and the application. There are two fundamentally different models for this interface. In a functional interface, the user interface has detailed knowledge about the application, and is, in fact, specific to the application. The interface between the two components is through a functional protocol, capable of representing the semantics which can be exposed through the user interface. Because the user interface has knowledge of the application, it can be optimally designed for that application. As a result, functional user interfaces are almost always the most user friendly, the fastest, the and the most responsive. However, in order to allow interoperability between user devices and applications, the details of the functional protocols need to be specified in standards. This slows down innovation and limits the scope of applications that can be built.

An alternative is a stimulus interface. In a stimulus interface, the user interface is generic, totally ignorant of the details of the application. Indeed, the application may pass instructions to the user interface describing how it should operate. The user interface translates user input into "stimulus" - which are data understood only by the application, and not by the user interface. Because they are generic, and because they require communications with the application in order to change the way in which they render

information to the user, stimulus user interfaces are usually slower, less user friendly, and less responsive than a functional counterpart. However, they allow for substantial innovation in applications, since no standardization activity is needed to build a new application, as long as it can interact with the user within the confines of the user interface mechanism. The web is an example of a stimulus user interface to applications.

In SIP systems, functional interfaces are provided by extending the SIP protocol to provide the needed functionality. For example, the SIP caller preferences specification [13] provides a functional interface that allows a user to request applications to route the call to specific types of user agents. Functional interfaces are important, but are not the subject of this framework. The primary goal of this framework is to address the role of stimulus interfaces to SIP applications.

3.2 Real-Time vs. Non-Real Time

Application interaction systems can also be real-time or non-real-time. Non-real interaction allows the user to enter information about application operation in asynchronously with its invocation. Frequently, this is done through provisioning systems. As an example, a user can set up the forwarding number for a call-forward on no-answer application using a web page. Real-time interaction requires the user to interact with the application at the time of its invocation.

3.3 Client-Local vs. Client-Remote

Another axis in the taxonomization is whether the user interface is co-resident with the user device (which we refer to as a client-local user interface), or the user interface runs in a host separated from the client (which we refer to as a client-remote user interface). In a client-remote user interface, there exists some kind of protocol between the client device and the UI that allows the client to interact with the user interface over a network.

The most important way to separate the UI and the client device is through media interaction. In media interaction, the interface between the user and the user interface is through media - audio, video, messaging, and so on. This is the classic mode of operation for VoiceXML [3], where the user interface (also referred to as the voice browser) runs on a platform in the network. Users communicate with the voice browser through the telephone network (or using a SIP session). The voice browser interacts with the application using HTTP to convey the information collected from the user.

In the case of a client-local user interface, the user interface runs co-located with the user device. The interface between them is through the software that interprets the users input and passes them to the user interface. The classic example of this is the web. In the web, the user interface is a web browser, and the interface is defined by the HTML document that it's rendering. The user interacts directly with the user interface running in the browser. The results of that user interface are sent to the application (running on the web server) using HTTP.

It is important to note that whether or not the user interface is local, or remote (in the case of media interaction), is not a property of the modality of the interface, but rather a property of the system. As an example, it is possible for a web-based user interface to be provided with a client-remote user interface. In such a scenario, video and application sharing media sessions can be used between the user and the user interface. The user interface, still guided by HTML, now runs "in the network", remote from the client. Similarly, a VoiceXML document can be interpreted locally by a client device, with no media streams at all. Indeed, the VoiceXML document can be rendered using text, rather than media, with no impact on the interface between the user interface and the application.

It is also important to note that systems can be hybrid. In a hybrid user interface, some aspects of it (usually those associated with a particular modality) run locally, and others run remotely.

3.4 Presentation Capable vs. Presentation Free

A user interface can be capable of presenting information to the user (a presentation capable UI), or it can be capable only of collecting user input (a presentation free UI). These are very different types of user interfaces. A presentation capable UI can provide the user with feedback after every input, providing the context for collecting the next input. As a result, presentation capable user interfaces require an update to the information provided to the user after each input. The web is a classic example of this. After every input (i.e., a click), the browser provides the input to the application and fetches the next page to render. In a presentation free user interface, this is not the case. Since the user is not provided with feedback, these user interfaces tend to merely collect information as its entered, and pass it to the application.

Another difference is that a presentation-free user interface cannot support the concept of a focus. As a result, if multiple applications wish to gather input from the user, there is no way for the user to select which application the input is destined for. The input provided to applications through presentation-free user interfaces is

more of a broadcast or notification operation, as a result.

3.5 Interaction Scenarios on Telephones

This same model can apply to a telephone. In a traditional telephone, the user interface consists of a 12-key keypad, a speaker, and a microphone. Indeed, from here forward, the term "telephone" is used to represent any device that meets, at a minimum, the characteristics described in the previous sentence. Circuit-switched telephony applications are almost universally client-remote user interfaces. In the Public Switched Telephone Network (PSTN), there is usually a circuit interface between the user and the user interface. The user input from the keypad is conveyed used Dual-Tone Multi-Frequency (DTMF), and the microphone input as Pulse Code Modulated (PCM) encoded voice.

In an IP-based system, there is more variability in how the system can be instantiated. Both client-remote and client-local user interfaces to a telephone can be provided.

In this framework, a PSTN gateway can be considered a "user proxy". It is a proxy for the user because it can provide, to a user interface on an IP network, input taken from a user on a circuit switched telephone. The gateway may be able to run a client-local user interface, just as an IP telephone might.

3.5.1 Client Remote

The most obvious instantiation is the "classic" circuit-switched telephony model. In that model, the user interface runs remotely from the client. The interface between the user and the user interface is through media, set up by SIP and carried over the Real Time Transport Protocol (RTP) [14]. The microphone input can be carried using any suitable voice encoding algorithm. The keypad input can be conveyed in one of two ways. The first is to convert the keypad input to DTMF, and then convey that DTMF using a suitance encoding algorithm for it (such as PCMU). An alternative, and generally the preferred approach, is to transmit the keypad input using RFC 2833 [15], which provides an encoding mechanism for carrying keypad input within RTP.

In this classic model, the user interface would run on a server in the IP network. It would perform speech recognition and DTMF recognition to derive the user intent, feed them through the user interface, and provide the result to an application.

3.5.2 Client Local

An alternative model is for the entire user interface to reside on

the telephone. The user interface can be a VoiceXML browser, running speech recognition on the microphone input, and feeding the keypad input directly into the script. As discussed above, the VoiceXML script could be rendered using text instead of voice, if the telephone had a textual display.

3.5.3 Flip-Flop

A middle-ground approach is to flip back and forth between a client-local and client-remote user interface. Many voice applications are of the type which listen to the media stream and wait for some specific trigger that kicks off a more complex user interaction. The long pound in a pre-paid calling card application is one example. Another example is a conference recording application, where the user can press a key at some point in the call to begin recording. When the key is pressed, the user hears a whisper to inform them that recording has started.

The ideal way to support such an application is to install a client-local user interface component that waits for the trigger to kick off the real interaction. Once the trigger is received, the application connects the user to a client-remote user interface that can play announements, collect more information, and so on.

The benefit of flip-flopping between a client-local and client-remote user interface is cost. The client-local user interface will eliminate the need to send media streams into the network just to wait for the user to press the pound key on the keypad.

The Keypad Markup Language (KPML) was designed to support exactly this kind of need [6]. It models the keypad on a phone, and allows an application to be informed when any sequence of keys have been pressed. However, KPML has no presentation component. Since user interfaces generally require a response to user input, the presentation will need to be done using a client-remote user interface that gets instantiated as a result of the trigger.

It is tempting to use a hybrid model, where a prompt-and-collect application is implemented by using a client-remote user interface that plays the prompts, and a client-local user interface, described by KPML, that collects digits. However, this only complicates the application. Firstly, the keypad input will be sent to both the media stream and the KPML user interface. This requires the application to sort out which user inputs are duplicates, a process that is very complicated. Secondly, the primary benefit of KPML is to avoid having a media stream towards a user interface. However, there is already a media stream for the prompting, so there is no real savings.

4. Framework Overview

In this framework, we use the term "SIP application" to refer to a broad set of functionality. A SIP application is a program running on a SIP-based element (such as a proxy or user agent) that provides some value-added function to a user or system administrator. SIP applications can execute on behalf of a caller, a called party, or a multitude of users at once.

Each application has a number of instances that are executing at any given time. An instance represents a single execution path for an application. Each instance has a well defined lifecycle. It is established as a result of some event. That event can be a SIP event, such as the reception of a SIP INVITE request, or it can be a non-SIP event, such as a web form post or even a timer. Application instances also have a specific end time. Some instances have a lifetime that is coupled with a SIP transaction or dialog. For example, a proxy application might begin when an INVITE arrives, and terminate when the call is answered. Other applications have a lifetime that spans multiple dialogs or transactions. For example, a conferencing application instance may exist so long as there are any dialogs connected to it. When the last dialog terminates, the application instance terminates. Other applications have a liftime that is completely decoupled from SIP events.

It is fundamental to the framework described here that multiple application instances may interact with a user during a single SIP transaction or dialog. Each instance may be for the same application, or different applications. Each of the applications may be completely independent, in that they may be owned by different providers, and may not be aware of each others existence. Similarly, there may be application instances interacting with the caller, and instances interacting with the callee, both within the same transaction or dialog.

The first step in the interaction with the user is to instantiate one of more user interface components for the application instance. A user interface component is a single piece of the user interface that is defined by a logical flow that is not synchronously coupled with any other component. In other words, each component runs more or less independently.

A user interface component can be instantiated in one of the user agents in a dialog (for a client-local user interface), or within a network element (for a client-remote user interface). If a client-local user interface is to be used, the application needs to determine whether or not the user agent is capable of supporting a client-local user interface, and in what format. In this framework,

all client-local user interface components are described by a markup language. A markup language describes a logical flow of presentation of information to the user, collection of information from the user, and transmission of that information to an application. Examples of markup languages include HTML, WML, VoiceXML, and the Keypad Markup Language (KPML) [6].

Unlike an application instance, which has very flexible lifetimes, a user interface component has a very fixed lifetime. A user interface component is always associated with a dialog. The user interface component can be created at any point after the dialog (or early dialog) is created. However, the user interface component terminates when the dialog terminates. The user interface component can be terminated earlier by the user agent, and possibly by the application, but its lifetime never exceeds that of its associated dialog.

There are two ways to create a client local interface component. For interface components that are presentation capable, the application sends a REFER [5] request to the user agent. The Refer-To header field contains an HTTP URI that points to the markup for the user interface. For interface components that are presentation free (such as those defined by KPML), the application sends a SUBSCRIBE request to the user agent. The body of the SUBSCRIBE request contains a filter, which, in this case, is the markup that defines when information is to be sent to the application in a NOTIFY.

If a user interface component is to be instantiated in the network, there is no need to determine the capabilities of the device on which the user interface is instantiated. Presumably, it is on a device on which the application knows a UI can be created. However, the application does need to connect the user device to the user interface. This will require manipulation of media streams in order to establish that connection.

The interface between the user interface component and the application depends on the type of user interface. For presentation capable user interfaces, such as those described by HTML and VoiceXML, HTTP form POST operations are used. For presentation free user interfaces, a SIP NOTIFY is used. The differing needs and capabilities of these two user interfaces, as described in Section 3.4, is what drives the different choices for the interactions. Since presentation capable user interfaces require an update to the presentation every time user data is entered, they are a good match for HTTP. Since presentation free user interfaces merely transmit user input to the application, a NOTIFY is more appropriate.

Indeed, for presentation free user interfaces, there are two

different modalities of operation. The first is called "one shot". In the one-shot role, the markup waits for a user to enter some information, and when they do, reports this event to the application. The application then does something, and the markup is no longer used. In the other modality, called "monitor", the markup stays permanently resident, and reports information back to an application until termination of the associated dialog.

5. Application Behavior

The behavior of an application within this framework depends on whether it seeks to use a client-local or client-remote user interface.

5.1 Client Local Interfaces

One key component of this framework is support for client local user interfaces.

5.1.1 Discovering Capabilities

A client local user interface can only be instantiated on a user agent if the user agent supports that type of user interface component. Support for client local user interface components is declared by both the UAC and a UAS in its Accept, Allow, Contact and Allow-Event header fields of dialog-initiating requests and responses. If the Allow header field indicates support for the SIP SUBSCRIBE method, and the Allow-Event header field indicates support for the kpml package [6], and the Contact header field indicates that its URI is a GRUU [9] it means that the UA can instantiate presentation free user interface components. In this case, the application MAY push presentation free user interface components according to the rules of Section 5.1.2. The specific markup languages that can be supported are indicated in the Accept header field.

If the Allow header field indicates support for the SIP REFER method, and the Contact header field contains UA capabilities [4] that indicate support for the HTTP URI scheme, it means that the UA supports presentation capable user interface components. In this case, the application MAY push presentation capable user interface components to the client according to the rules of Section 5.1.2. The specific markups that are supported are indicated in the Accept header field.

5.1.2 Pushing an Initial Interface Component

Generally, we anticipate that interface components will need to be created at various different points in a SIP session. Clearly, they will need to be pushed during session setup, or after the session is established. A user interface component is always associated with a specific dialog, however.

An application MUST NOT attempt to push a user interface component to a user agent until it has determined that the user agent has the neccesary capabilities and a dialog has been created. In the case of

a UAC, this means that an application MUST NOT push a user interface component for an INVITE initiated dialog until the application has seen a 200 OK followed by an ACK. For SUBSCRIBE initiated dialogs, it MUST NOT push a user interface component until the application has seen a 200 OK to the NOTIFY request. For a user interface component on a UAS, the application MUST NOT push a user interface component for an INVITE initiated dialog until it has seen a 200 OK from the UAS. For a SUBSCRIBE initiated dialog, it MUST NOT push a user interface component until it has seen a NOTIFY request from the notifier.

To create a presentation capable UI component on the UA, the application sends a REFER request to the UA. This REFER MUST be sent to the Globally Routable UA URI (GRUU) [9] advertised by that UA in the Contact header field of the dialog initiating request or response sent by that UA. Note that this REFER request creates a separate dialog between the application and the UA. The Refer-To header field of the REFER request MUST contain an HTTP URI that references the markup document to be fetched.

OPEN ISSUE: The refer needs to provide a context to the UA, and in particular, identify the specific dialog that this component is associated with. There is no obvious candidate for this when REFER is used. The former proposal, of using a grid, cannot work because of forking.

To create a presentation free user interface component, the application sends a SUBSCRIBE request to the UA. The SUBSCRIBE MUST be sent to the GRUU advertised by the UA. This SUBSCRIBE request creates a separate dialog. The SUBSCRIBE request MUST use the KPML [6] event package. The Event header field MUST contain parameters which identify the particular dialog that the interface component is being instantiated against. The body of the SUBSCRIBE request contains the markup document that defines the conditions under which the application wishes to be notified of user input.

In both cases, the REFER or SUBSCRIBE request SHOULD include a display name in the From header field which identifies the name of the application. For example, a prepaid calling card might include a From header field which looks like:

From: "Prepaid Calling Card" <sip:prepaid@example.com>

To authenticate themselves, it is RECOMMENDED that applications use the SIP identity mechanism [7] in the REFER or SUBSCRIBE requests they generate. This mechanism has the benefit that the signature is over an authenticated identity body [8], which includes the From

header field. As such, the client can obtain cryptographic assurances about the service provider (the domain in the From header field) along with the name of the application.

5.1.3 Updating an Interface Component

Once a user interface component has been created on a client, it can be updated. The means for updating it depends on the type of UI component.

Presentation capable UI components are updated using techniques already in place for those markups. In particular, user input will cause an HTTP POST operation to push the user input to the application. The result of the POST operation is a new markup that the UI is supposed to use. This allows the UI to updated in response to user action. Some markups, such as HTML, provide the ability to force a refresh after a certain period of time, so that the UI can be updated without user input. Those mechanisms can be used here as well. However, there is no support for an asynchronous push of an updated UI component from the appliciation to the user agent. A new REFER request to the same GRUU would create a new UI component rather than updating any components already in place.

For presentation free UI, the story is different. The application MAY update the filter at any time by generating a SUBSCRIBE refresh with the new filter. The UA will immediately begin using this new filter.

5.1.4 Terminating an Interface Component

User interface components have a well defined lifetime. They are created when the component is first pushed to the client. User interface components are always associated with the SIP dialog on which they were pushed. As such, their lifetime is bound by the lifetime of the dialog. When the dialog ends, so does the interface component.

However, there are some cases where the application would like to terminate the user interface component before its natural termination point. For presentation capable user interfaces, this is not possible. For presentation free user interfaces, the application MAY terminate the component by sending a SUBSCRIBE with Expires equal to zero. This terminates the subscription, which removes the UI component.

A client can remove a UI component at any time. For presentation aware UI, this is analagous to the user dismissing the web form window. There is no mechanism provided for reporting this kind of event to the application. The applicatio MUST be prepared to time

out, and never receive input from a user. For presentation free user interfaces, the UA can explicitly terminate the subscription. This will result in the generation of a NOTIFY with a Subscription-State header field equal to "terminated".

5.2 Client Remote Interfaces

As an alternative to, or in conjunction with client local user interfaces, an application can make use of client remote user interfaces. These user interfaces can execute co-resident with the application itself (in which case no standardized interfaces between the UI and the application need to be used), or it can run separately. This framework assumes that the user interface runs on a host that has a sufficient trust relationship with the application. As such, the means for instantiating the user interface is not considered here.

The primary issue is to connect the user device to the remote user interface. Doing so requires the manipulation of media streams between the client and the user interface. Such manipulation can only be done by user agents. There are two types of user agent applications within this framework - originating/terminating applications, and intermediary applications.

5.2.1 Originating and Terminating Applications

Originating and terminating applications are applications which are themselves the originator or the final recipient of a SIP invitation. They are "pure" user agent applications - not back-to-back user agents. The classic example of such an application is an interactive voice response (IVR) application, which is typically a terminating application. Its a terminating application because the user explicitly calls it; i.e., it is the actual called party. An example of an originating application is a wakeup call application, which calls a user at a specified time in order to wake them up.

Because originating and terminating applications are a natural termination point of the dialog, manipulation of the media session by the application is trivial. Traditional SIP techniques for adding and removing media streams, modifying codecs, and changing the address of the recipient of the media streams, can be applied. Similarly, the application can directly authenticate itself to the user through S/MIME, since it is the peer UA in the dialog.

5.2.2 Intermediary Applications

Intermediary applications are, at the same time, more common than originating/terminating applications, and more complex. Intermediary

applications are applications that are neither the actual caller or called party. Rather, they represent a "third party" that wishes to interact with the user. The classic example is the ubiquitous pre-paid calling card application.

In order for the intermediary application to add a client remote user interface, it needs to manipulate the media streams of the user agent to terminate on that user interface. This also introduces a fundamental feature interaction issue. Since the intermediary application is not an actual participant in the call, how does the user interact with the intermediary application, and its actual peer in the dialog, at the same time? This is discussed in more detail in Section 7.

6. User Agent Behavior

6.1 Advertising Capabilities

In order to participate in applications that make use of stimulus interfaces, a user agent needs to advertise its interaction capabilities.

If a user agent supports presentation capable user interfaces, it MUST support the REFER method. It MUST include, in all dialog initiating requests and responses, an Allow header field that includes the REFER method. Furthermore, the UA MUST support the SIP user agent capabilities specification [4]. The UA MUST be capable of being REFER'd to an HTTP URI. It MUST include, in the Contact header field of its dialog initiating requests and responses, a "schemes" Contact header field parameter include the http URI scheme. The UA MUST include, in all dialog initiating requests and responses, an Accept header field listing all of those markups supported by the UA. It is RECOMMENDED that all user agents that support presentation capable user interfaces support HTML.

If a user agent supports presentation free user interfaces, it MUST support the SUBSCRIBE [2] method. It MUST support the KPML [6] event package. It MUST include, in all dialog initiating requests and responses, an Allow header field that includes the SUBSCRIBE method. It MUST include, in all dialog initiating requests and responses, an Allow-Events header field that lists the KPML event package. The UA MUST include, in all dialog initiating requests and responses, an Accept header field listing those event filters it supports. At a minimum, a UA MUST support the "application/kpml+xml" MIME type.

For either presentation free or presentation capable user interfaces, the user agent MUST support the GRUU [9] specification. The Contact header field in all dialog initiating requests and responses MUST contain a GRUU. The UA MUST include a Supported header field which contains the gruu option tag.

Because these headers are examined by proxies which may be executing applications, a UA that wishes to support client local user interfaces should not encrypt them.

6.2 Receiving User Interface Components

Once the UA has created a dialog (in either the early or confirmed states), it MUST be prepared to receive a SUBSCRIBE or REFER request against its GRUU. If the UA receives such a request prior to the establishment of a dialog, the UA MUST reject the request.

A user agent SHOULD attempt to authenticate the sender of the request. The sender will generally be an application, and therefore the user agent is unlikely to ever have a shared secret with it, making digest authentication useless. However, the REFER or SUBSCRIBE request should have a SIP authenticated identity body [8] that conveys the identity of the application [7]. If such a body is not present, and no alternative means of identification (such as P-Asserted-ID [11]) is present, the user agent MAY reject the request with a 403 response.

Next, the user agent authorizes the application. An application is authorized to instantiate a user interface component if the application was resident within an element on the path of the dialog initiating request. An application proves to the user agent that it was on the path by presenting it with the dialog identifiers in the SUBSCRIBE or REFER request. In the case of SUBSCRIBE, those identifiers are present in the Event header field [6]. [[EDITORS NOTE: Fill in here once we know how this is done for REFER.]]

Because of the dialog identifiers serve as a tool for authorization, a user agent compliant to this framework MUST use dialog identifiers that are cryptographically random, with at least 128 bits of randomness. It is recommended that this randomness be split between the Call-ID and From header field tag in the case of a UAC.

Furthermore, to ensure that only applications resident in on-path elements can instantiate a user interface component, a user agent compliant to this specification SHOULD use the sips URI scheme for all dialogs it initiates. This will guarantee secure links between all of the elements on the signaling path.

If an application does not present a valid dialog identifier in its REFER or SUBSCRIBE request, the user agent MUST reject the request with a 403 response. A user agent MAY apply any other policies in addition to (but not instead of) the ones specified here in order to authorize the creation of the user interface component. One such mechanism would be to prompt the user, informing them of the identity of the application. If an authorization policy requires user interaction, the user agent SHOULD respond to the SUBSCRIBE or REFER request with a 202. In the case of SUBSCRIBE, if authorization is not granted, the user agent SHOULD generate a NOTIFY to terminate the subscription. In the case of REFER, the user agent MUST NOT act upon the URI in the Refer-To header field until user authorization was obtained.

If a REFER request to an HTTP URI was authorized, the UA executes the URI and fetches the content to be rendered to the user. This instantiates a presentation capable user interface component. If a

SUBSCRIBE was authorized, a presentation free user interface component was instantiated.

6.3 Mapping User Input to User Interface Components

Once the user interface components are instantiated, the user agent must direct user input to the appropriate component. In the case of presentation capable user interfaces, this process is known as focus selection. It is done by means that are specific to the user interface on the device. In the case of a PC, for example, the window manager would allow the user to select the appropriate user interface component that their input is directed to.

For presentation free user interfaces, the situation is more complicated. In some cases, the device may support a mechanism that allows the user to select a "line", and thus the associated dialog. Any user input on the keypad while this line is selected are fed to the user interface components associated with that dialog.

> TODO: Need to consider the case where the user interface is co-resident with the UAC, but the user device is separated from the UAC, and occurs through some other protocol, and the user interface and application are semi-trusted. Classic case is when the UAC is a PSTN gateway.

6.4 Receiving Updates to User Interface Components

For presentation capable user interfaces, updates to the user interface occur in ways specific to that user interface component. In the case of HTML, for example, the document can tell the client to fetch a new document periodically. However, this framework does not provide any additional machinery to asynchronously push a new user interface component to the client.

For presentation free user interfaces, an application can push an update to a component by sending a SUBSCRIBE refresh with a new filter. The user agent will process these according to the rules of the event package.

6.5 Terminating a User Interface Component

Termination of a presentation capable user interface component is a trivial procedure. The user agent merely dismisses the window (or equivalent). The fact that the component is dismissed is not communicated to the application. As such, it is purely a local matter.

In the case of a presentation free user interface, if the user wishes to cease interacting with the application, it SHOULD generate a NOTIFY request with a Subscription-State equal to "terminated" and a reason of "rejected". This tells the application that the component has been removed, and that it should not attempt to re-subscribe.

7. Inter-Application Feature Interaction

The inter-application feature interaction problem is inherent to stimulus signaling. Whenever there are multiple applications, there are multiple user interfaces. When the user provides an input, to which user interface is the input destined? That question is the essence of the inter-application feature interaction problem.

Inter-application feature interaction is not an easy problem to resolve. For now, we consider separately the issues for client-local and client-remote user interface components.

7.1 Client Local UI

When the user interface itself resides locally on the client device, the feature interaction problem is actually much simpler. The end device knows explicitly about each application, and therefore can present the user with each one separately. When the user provides input, the client device can determine to which user interface the input is destined. The user interface to which input is destined is referred to as the application in focus, and the means by which the focused application is selected is called focus determination.

Generally speaking, focus determination is purely a local operation. In the PC universe, focus determination is provided by window managers. Each application does not know about focus, it merely receives the user input that has been targeted to it when its in focus. This basic concept applies to SIP-based applications as well.

Focus determination will frequently be trivial, depending on the user interface type. Consider a user that makes a call from a PC. The call passes through a pre-paid calling card application, and a call recording application. Both of these wish to interact with the user. Both push an HTML-based user interface to the user. On the PC, each user interface would appear as a separate window. The user interacts with the call recording application by selecting its window, and with the pre-paid calling card application by selecting its window. Focus determination is literally provided by the PC window manager. It is clear to which application the user input is targeted.

As another example, consider the same two applications, but on a "smart phone" that has a set of buttons, and next to each button, an LCD display that can provide the user with an option. This user interface can be represented using the Wireless Markup Language (WML).

The phone would allocate some number of buttons to each application. The prepaid calling card would get one button for its "hangup"

command, and the recording application would get one for its "start/stop" command. The user can easily determine which application to interact with by pressing the appropriate button. Pressing a button determines focus and provides user input, both at the same time.

Unfortunately, not all devices will have these advanced displays. A PSTN gateway, or a basic IP telephone, may only have a 12-key keypad. The user interfaces for these devices are provided through the Keypad Markup Language (KPML). Considering once again the feature interaction case above, the pre-paid calling card application and the call recording application would both pass a KPML document to the device. When the user presses a button on the keypad, to which document does the input apply? The user interface does not allow the user to select. A user interface where the user cannot provide focus is called a focusless user interface. This is quite a hard problem to solve. This framework does not make any explicit normative recommendation, but concludes that the best option is to send the input to both user interfaces unless the markup in one interface has indicated that it should be suppressed from others. This is a sensible choice by analogy - its exactly what the existing circuit switched telephone network will do. It is an explicit non-goal to provide a better mechanism for feature interaction resolution than the PSTN on devices which have the same user interface as they do on the PSTN. Devices with better displays, such as PCs or screen phones, can benefit from the capabilities of this framework, allowing the user to determine which application they are interacting with.

Indeed, when a user provides input on a focusless device, the input must be passed to all client local user interfaces, AND all client remote user interfaces, unless the markup tells the UI to suppress the media. In the case of KPML, key events are passed to remote user interfaces by encoding them in RFC 2833 [15]. Of course, since a client cannot determine if a media stream terminates in a remote user interface or not, these key events are passed in all audio media streams unless the "Q" digit is used to suppress.

7.2 Client-Remote UI

When the user interfaces run remotely, the determination of focus can be much, much harder. There are many architectures that can be deployed to handle the interaction. None are ideal. However, all are beyond the scope of this specification.

8. Intra Application Feature Interaction

An application can instantiate a multiplicity of user interface components. For example, a single application can instantiate two separate HTML components and one WML component. Furthermore, an application can instantiate both client local and client remote user interfaces.

The feature interaction issues between these components within the same application are less severe. If an application has multiple client user interface components, their interaction is resolved identically to the inter-application case - through focus determination. However, the problems in focusless user interfaces (such as a keypad) generally won't exist, since the application can generate user interfaces which do not overlap in their usage of an input.

The real issue is that the optimal user experience frequently requires some kind of coupling between the differing user interface components. This is a classic problem in multi-modal user interfaces, such as those described by Speech Application Language Tags (SALT). As an example, consider a user interface where a user can either press a labeled button to make a selection, or listen to a prompt, and speak the desired selection. Ideally, when the user presses the button, the prompt should cease immediately, since both of them were targeted at collecting the same information in parallel. Such interactions are best handled by markups which natively support such interactions, such as SALT, and thus require no explicit support from this framework.

9. Example Call Flow

This section shows the operation of a call recording application. This application allows a user to record the media in their call by clicking on a button in a web form. The application uses a presentation capable user interface component that is pushed to the caller.

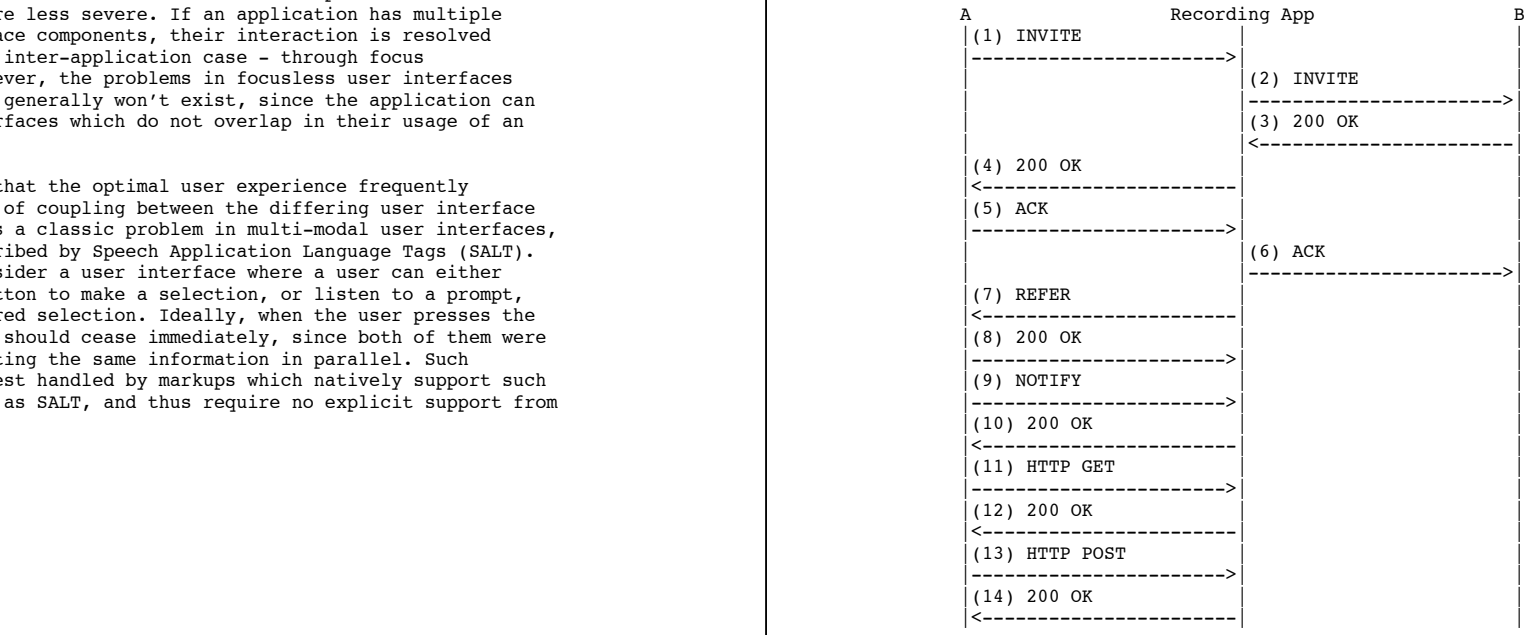


Figure 3

First, the caller, A, sends an INVITE to setup a call (message 1). Since the caller supports the framework, and can handle presentation capable user interface components, it includes the Supported header field indicating the GRUU is understood, Allow indicating that REFER is understood, and a Contact header field that includes the "schemes" header field parameter.

```
INVITE sip:B@example.com SIP/2.0
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>
Call-ID: faif9a@host.example.com
CSeq: 1 INVITE
Supported: gruu
Allow: INVITE, OPTIONS, BYE, CANCEL, ACK, REFER
Contact: <sip:bad998asd8asd0000a@example.com>;schemes="http,sip"
Content-Length: ...
Content-Type: application/sdp

--SDP not shown--
```

The proxy acts as a recording server, and forwards the INVITE to the called party (message 2):

```
INVITE sip:B@pc.example.com SIP/2.0
Record-Route: <sip:app.example.com;lr>
Via: SIP/2.0/UDP app.example.com;branch=z9hG4bK97sh
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>
Call-ID: faif9a@host.example.com
CSeq: 1 INVITE
Supported: gruu
Allow: INVITE, OPTIONS, BYE, CANCEL, ACK, REFER
Contact: <sip:bad998asd8asd0000a@example.com>;schemes="http,sip"
Content-Length: ...
Content-Type: application/sdp

--SDP not shown--
```

B accepts the call with a 200 OK (message 3). It does not support the framework, and so the various header fields are not present.

```
SIP/2.0 200 OK
Record-Route: <sip:app.example.com;lr>
Via: SIP/2.0/UDP app.example.com;branch=z9hG4bK97sh
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>;tag=7777
Call-ID: faif9a@host.example.com
CSeq: 1 INVITE
Contact: <sip:B@pc.example.com>
Content-Length: ...
```



```
Content-Type: application/sdp

--SDP not shown--
```

This 200 OK is passed back to the caller (message 4):

```
SIP/2.0 200 OK
Record-Route: <sip:app.example.com;lr>
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>;tag=7777
Call-ID: faif9a@host.example.com
CSeq: 1 INVITE
Contact: <sip:B@pc.example.com>
Content-Length: ...
Content-Type: application/sdp

--SDP not shown--
```

The caller generates an ACK (message 5).

```
ACK sip:B@pc.example.com
Route: <sip:app.example.com;lr>
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz9
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>;tag=7777
Call-ID: faif9a@host.example.com
CSeq: 1 ACK
```

The ACK is forwarded to the called party (message 6).

```
ACK sip:B@pc.example.com
Via: SIP/2.0/UDP app.example.com;branch=z9hG4bKh7s
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9zz9
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>;tag=7777
Call-ID: faif9a@host.example.com
CSeq: 1 ACK
```

Now, the application decides to push a user interface component to user A. So, it sends it a REFER request (message 7):

```
REFER sip:bad998asd8asd0000a@example.com SIP/2.0
Refer-To: http://app.example.com/script.pl
Via: SIP/2.0/UDP app.example.com;branch=z9hG4bK9zh6
From: Recorder Application <sip:app.example.com>;tag=jhgf
To: Caller <sip:A@example.com>
Call-ID: 66676776767@app.example.com
CSeq: 1 REFER
Event: refer
Contact: <sip:sip:app.example.com>
```

The REFER is answered by a 200 OK (message 8).

```
SIP/2.0 200 OK
Refer-To: http://app.example.com/script.pl
Via: SIP/2.0/UDP app.example.com;branch=z9hG4bK9zh6
From: Recorder Application <sip:app.example.com>;tag=jhgf
To: Caller <sip:A@example.com>;tag=pqoew
Call-ID: 66676776767@app.example.com
Supported: gruu
Allow: INVITE, OPTIONS, BYE, CANCEL, ACK, REFER
Contact: <sip:bad998asd8asd0000a@example.com>;schemes="http,sip"
CSeq: 1 REFER
```

User A sends a NOTIFY (message 9):

```
NOTIFY sip:app.example.com SIP/2.0
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9320394238995
To: Recorder Application <sip:app.example.com>;tag=jhgf
From: Caller <sip:A@example.com>;tag=pqoew
Call-ID: 66676776767@app.example.com
CSeq: 1 NOTIFY
Max-Forwards: 70
Event: refer;id=93809824
Subscription-State: active;expires=3600
Contact: <sip:bad998asd8asd0000a@example.com>;schemes="http,sip"
Content-Type: message/sipfrag;version=2.0
Content-Length: 20

SIP/2.0 100 Trying
```

And the recording server responds with a 200 OK (message 10)

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bK9320394238995
To: Recorder Application <sip:app.example.com>;tag=jhgf
```



```
From: Caller <sip:A@example.com>;tag=pqoew
Call-ID: 66676776767@app.example.com
CSeq: 1 NOTIFY
```

The caller, A, authorizes the application. It then acts on the Refer-To URI, fetching the script from app.example.com (message 11). The response, message 12, contains a web application that the user can click on to enable recording. When the user clicks on the link (message 13), the results are posted to the server, and an updated display is provided (message 14).

10. Security Considerations

There are many security considerations associated with this framework. It allows applications in the network to instantiate user interface components on a client device. Such instantiations need to be from authenticated applications, and also need to be authorized to place a UI into the client. Indeed, the stronger requirement is authorization. It is not so important to know that name of the provider of the application, but rather, that the provider is authorized to instantiate components.

Generally, an application should be considered authorized if it was an application that was legitimately part of the call setup path. With this definition, authorization can be enforced using the sips URI scheme when the call is initiated.

11. Contributors

This document was produced as a result of discussions amongst the application interaction design team. All members of this team contributed significantly to the ideas embodied in this document. The members of this team were:

Eric Burger
Cullen Jennings
Robert Fairlie-Cuninghame

Normative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[3] McGlashan, S., Lucas, B., Porter, B., Rehor, K., Burnett, D., Carter, J., Ferrans, J. and A. Hunt, "Voice Extensible Markup Language (VoiceXML) Version 2.0", W3C CR CR-voicexml20-20030220, February 2003.

[4] Rosenberg, J., "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", draft-ietf-sip-callee-caps-03 (work in progress), January 2004.

[5] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[6] Burger, E., "Keypad Stimulus Protocol (KPML)", draft-ietf-sipping-kpml-02 (work in progress), February 2004.

[7] Peterson, J., "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-01 (work in progress), March 2003.

[8] Peterson, J., "SIP Authenticated Identity Body (AIB) Format", draft-ietf-sip-authid-body-02 (work in progress), July 2003.

[9] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-00 (work in progress), January 2004.

Informative References

[10] Day, M., Rosenberg, J. and H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000.

[11] Jennings, C., Peterson, J. and M. Watson, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", RFC 3325, November 2002.

[12] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-01 (work in progress), October 2003.

[13] Rosenberg, J., Schulzrinne, H. and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", draft-ietf-sip-callerprefs-10 (work in progress), October 2003.

[14] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

[15] Schulzrinne, H. and S. Petrack, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals", RFC 2833, May 2000.

Author's Address

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
EMail: jdrosen@dynamicsoft.com
URI: http://www.jdrosen.net

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING Working Group
Internet-Draft
Expires: August 15, 2004

A. Johnston
MCI
O. Levin
Microsoft
February 15, 2004

Session Initiation Protocol Call Control - Conferencing for User Agents
draft-ietf-sipping-cc-conferencing-03

Status of this Memo

Copyright Notice

Abstract

This document defines conferencing call control features for the Session Initiation Protocol (SIP). This document builds on the Conferencing Requirements and Framework documents to define how a tightly coupled SIP conference works. The approach is explored from different user agent (UA) types perspective: conference-unaware, conference-aware and focus UAs. The use of URIs in conferencing, OPTIONS for capabilities discovery, and call control using REFER are covered in detail with example call flow diagrams.

Table of Contents

1. Introduction

This document uses the concepts and definitions from the high level requirements [10] and the SIP conferencing framework [11] documents.

The approach described in this document implements key functions in the conferencing framework using SIP primitives only. This allows for conducting simple conferences with defined functionalities using SIP mechanisms and conventions. Many other advanced functions can be implemented using additional means but they are not in the scope of this document.

This document presents the basic call control (dial-in and dial-out) conferencing building blocks from the UA perspective. Possible applications include ad-hoc conferences and scheduled conferences.

Note that a single conference can bridge participants having different capabilities and who potentially have joined the conference by different means (i.e. dial-in, dial-out, scheduled, and ad-hoc).

The call control and dialog manipulation approach is based on the multiparty framework [12] document. That document defines the basic approach of service design adopted for SIP which includes:

- Definition of primitives, not services
- Signaling model independent
- Invoker oriented
- Primitives make full use of URIs
- Include authentication, authorization, logging, etc. policies
- Define graceful fallback to baseline SIP.

The use of opaque URIs and the ability to communicate call control context information within a URI (as opposed to service-related header fields), as discussed in RFC 3087 [13], is fundamental to this approach.

2. Usage of the 'isfocus' Feature Parameter

2.1 General

The main design guidelines for the development of SIP extensions and conventions for conferencing are to define the minimum number of extensions and to have seamless backwards compatibility with conference-unaware SIP UAs. The minimal requirement for SIP is being able to express that a dialog is a part of a certain conference referenced to by a URI. As a result of these extensions, it is possible to do the following using SIP:

- Create a conference
- Join a conference
- Invite a user to a conference
- Expel a user by third party
- Discover if a URI is a conference URI

The approach taken is to use the feature parameter "isfocus" to express that a SIP dialog belongs to a conference. The use of feature parameters in Contact header fields to describe the characteristics and capabilities of a UA is described in the User Agent Capabilities [7] document which includes the definition of the "isfocus" feature parameter.

2.2 Session Establishment

In session establishment, a focus MUST include the "isfocus" feature parameter in the Contact header field unless the focus wishes to hide the fact that it is a focus. To a participant, the feature parameter will be associated with the remote target URI of the dialog. It is an indication to a conference-aware UA that the resulting dialog belongs to a conference identified by the URI in the Contact header field and that the call control conventions defined in this document can be applied.

The Conference URI MUST meet the requirements to be a GRUU (Globally Routable User Agent URI) as detailed in [9]

2.3 OPTIONS

Currently the only met requirement is: given an opaque URI, being able to recognize whether it belongs to a certain conference (i.e. meaning that it is a conference URI) or not. As with any other OPTIONS request, it can be done either inside an active dialog or outside a dialog. A focus MUST include the "isfocus" feature parameter in a 200 OK response to an OPTIONS unless the focus wishes to hide the fact that it is a focus.

3. SIP User Agent Conferencing Capability Types

From a conferencing perspective, the framework document outlines a number of possible different SIP components such as conference-unaware participant, conference-aware participant, and focus.

This document applies the concepts above to the SIP call control part of the conferencing components. It defines normative behavior of the SIP UAs in various conferencing situations (referred later as "scenarios").

3.1 Focus UA

A focus, as defined in the framework, hosts a SIP conference and maintains a SIP signaling relationship with each participant in the conference. A focus contains a conference-aware user agent that supports the conferencing call control conventions as defined in this document.

A focus SHOULD support the conference package [5] and indicate so in Allow-Events header fields in requests and responses. A focus MAY include information about the conference in SDP message bodies sent.

A focus SHOULD support the Replaces [8] header field.

A user agent with focus capabilities could be implemented in end user equipment and would be used for the creation of ad-hoc conferences.

A dedicated conferencing server, whose primary task is to simultaneously host conferences of arbitrary type and size, may allocate and publish a conference factory URI (as defined in the next section) for creating an arbitrary number of ad-hoc conferences (and subsequently their focuses) using SIP call control means.

3.2 Conference Factory URI

According to the framework, there are many ways in which a conference can be created. These are open to the conferencing server implementation policy and include non-automated means (such as IVR), SIP, and a conference policy control protocol.

In order to automatically create an arbitrary number of ad-hoc conferences (and subsequently their focuses) using SIP call control means, a globally routable Conference Factory URI can be allocated and published.

A successful attempt to establish a call to this URI would result in the automatic creation a new conference and its focus. As a result, note that the Conference Factory URI and the newly created focus URI MAY resolve to different physical devices.

A scenario showing the use of the conference factory URI is shown in Section 4.5.

3.3 Conference-Unaware UA

The simplest user agent can participate in a conference ignoring all SIP conferencing-related information. The simplest user agent is able

to dial into a conference and to be invited to a conference. Any conferencing information is optionally conveyed to/from it using non-SIP means. Such a user agent would not usually host a conference (at least, not using SIP explicitly). A conference-unaware UA needs only to support RFC 3261 [2]. Call flows for conference-unaware UAs are not shown in general in this document as they would be identical to those in the SIP call flows [15] document.

3.4 Conference-Aware UA

A conference-aware user agent supports SIP conferencing call control conventions defined in this document as a conference participant, in addition to support of RFC 3261.

A conference-aware UA MUST recognize the "isfocus" feature parameter. A conference-aware UA SHOULD support REFER [3], SIP events [4], and the conferencing package [5].

A conference-aware UA SHOULD subscribe to the conference package if the "isfocus" parameter is in the remote target URI of a dialog and if the conference package is listed by a focus in an Allow-Events header field.

A conference-aware UA MAY render to the user any information about the conference obtained from the SIP header fields and SDP fields from the focus.

4. SIP Conferencing Primitives

The SIP conferencing call control flows presented in this section are the call control building blocks for various SIP tight conferencing applications as described in the conferencing requirements [10] and framework [11] documents. The major design goal is that the same SIP conferencing primitives would be used by user agents having different conferencing capabilities and comprising different applications.

4.1 Joining a Conference using the Conference URI - Dial In

In this section a user knows the conference URI and "dials in" to join this conference.

If the UA is the first participant of the conference to dial in, it is likely that this INVITE will create the focus and hence the conference. However, the conference URI must have been reserved prior to its use.

If the conference is up and running already, the dialing-in participant is joined to the conference by its focus.

To join an existing specific conference a UA SHOULD send an INVITE with the Request-URI set to the conference URI. The focus MUST include the "isfocus" feature parameter in the Contact header field of the 200 OK response to the INVITE.

The SUBSCRIBE should be sent outside the INVITE-initiated dialog.

Participants leaving the conference send a BYE to the focus. If they have a current subscription to the conference package, they also must send a SUBSCRIBE with an Expires:0 header field to terminate both dialogs.

An example call flow for joining a conference is shown in Figure 1.

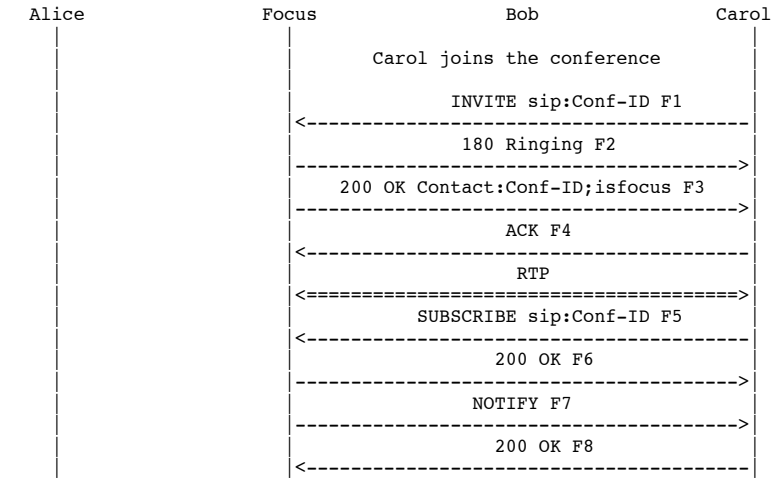


Figure 1. A Participant Joins a Conference using the Conference URI.

```
 F1   INVITE sip:3402934234@example.com SIP/2.0
      Via: SIP/2.0/UDP client.chicago.example.com
       ;branch=z9hG4bKhjhs8ass83
      Max-Forwards: 70
      To: <sip:3402934234@example.com>
      From: Carol <sip:carol@chicago.example.com>;tag=32331
      Call-ID: d432fa84b4c76e66710
      CSeq: 45 INVITE
```



```
      Contact: <sip:carol@client.chicago.example.com>
      Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
       SUBSCRIBE, NOTIFY
      Allow-Events: dialog
      Accept: application/sdp, message/sipfrag
      Supported: replaces
      Content-Type: application/sdp
      Content-Length: 274

      (SDP not shown)
```

```
 F3   SIP/2.0 200 OK
      Via: SIP/2.0/UDP client.chicago.example.com
       ;branch=z9hG4bKhjhs8ass83;received=192.0.2.4
      To: <sip:3402934234@example.com>;tag=733413
      From: Carol <sip:carol@chicago.example.com>;tag=32331
      Call-ID: d432fa84b4c76e66710
      CSeq: 45 INVITE
      Contact: <sip:3402934234@example.com>;isfocus
      Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
       SUBSCRIBE, NOTIFY
      Allow-Events: dialog, conference
      Accept: application/sdp, application/conference-info+xml,
       message/sipfrag
      Supported: replaces, join, gruu
      Content-Type: application/sdp
      Content-Length: 274

      v=0
      o=focus431 2890844526 2890842807 IN IP4 ms5.conf.example.com
      s=Example Subject
      i=Example Conference Hosted by Example.com
      u=http://conf.example.com/3402934234
      e=3402934234@conf-help.example.com
      p=+1-888-2934234
      c=IN IP4 ms5.conf.example.com
      t=0 0
      m=audio 49170 RTP/AVP 0
      m=video 51372 RTP/AVP 31
```

```
 F5   SUBSCRIBE sip:3402934234@example.com SIP/2.0
      Via: SIP/2.0/UDP client.chicago.example.com
       ;branch=z9hG4bKdf334
      Max-Forwards: 70
      To: <sip:3402934234@example.com>
      From: Carol <sip:carol@chicago.example.com>;tag=43524545
```

```
        Call-ID: k3l43id034kshereree
        CSeq: 22 SUBSCRIBE
        Contact: <sip:carol@chicago.example.com>
        Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
         SUBSCRIBE, NOTIFY
        Event: conference
        Accept: application/sdp, message/sipfrag
        Supported: replaces
        Content-Length: 0


   F7   NOTIFY sip:carol@chicago.example.com SIP/2.0
        Via: SIP/2.0/UDP ms5.conf.example.com;branch=z9hG4bK3343d1
        Max-Forwards: 70
        To: Carol <sip:carol@chicago.example.com>;tag=43524545
        From: <sip:3402934234@example.com>;tag=a3343df32
        Call-ID: k3l43id034kshereree
        CSeq: 34321 NOTIFY
        Contact: <sip:3402934234@example.com>;isfocus
        Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
         SUBSCRIBE, NOTIFY
        Event: conference
        Accept: application/sdp, message/sipfrag
        Subscription-State: active;expires=3600
        Supported: replaces, join, gruu
        Content-Type: application/conference-info+xml
        Content-Length: ...

        <conference-info version="0" state="full"
                     entity="sip:3402934234@example.com">
          <user uri="sip:carol@chicago.example.com"
                                 display-name="Carol">
            <status>connected</status>
            <joining-mode>dialed-in</joining-mode>
            <media-stream media-type="audio">
               <proto>RTP/AVP</proto>
               <ssrc>583398</ssrc>
            </media-stream>
            <media-stream media-type="video">
               <proto>RTP/AVP</proto>
               <ssrc>345212</ssrc>
            </media-stream>
          </user>
          <conf-uri>tel:+18882934234</conf-uri>
        </conference-info>
```


4.2 Adding a Participant by the Focus - Dial Out

To directly add a participant to a conference, a focus SHOULD send an INVITE to the participant containing a Contact header field with the conference URI and the "isfocus" feature parameter.

Note that a conference-unaware UA would simply ignore the conferencing information and treat the session (from a SIP perspective) as a point to point session.

An example call flow is shown in Figure 2. It is assumed that Alice is already a participant of the conference. The focus invites Carol to the conference by sending an INVITE. After the session is established, Carol subscribes to the conference URI. It is important to note that there is no dependency on Carol's SUBSCRIBE (F5) and the NOTIFY to Alice (F9) - they occur asynchronously and independently.

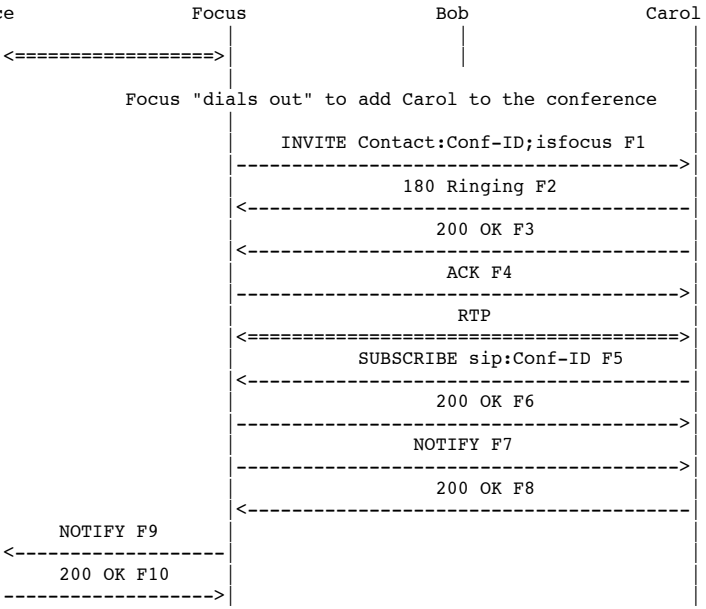


Figure 2. A Focus "dials out" to Add a Participant to the Conference.

F7
```
NOTIFY sip:carol@chicago.example.com SIP/2.0
Via: SIP/2.0/UDP ms5.conf.example.com;branch=z9hG4bK3343d1
Max-Forwards: 70
To: Carol <sip:carol@chicago.example.com>;tag=43524545
From: <sip:3402934234@example.com>;tag=a3343df32
Call-ID: k3l43id034kserereee
CSeq: 34321 NOTIFY
Contact: <sip:3402934234@example.com>;isfocus
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
 SUBSCRIBE, NOTIFY
Event: conference
Accept: application/sdp, message/sipfrag
Subscription-State: active;expires=3600
Supported: replaces, gruu
Content-Type: application/conference-info+xml
Content-Length: ...

<conference-info version="0" state="full"
             entity="sip:3402934234@example.com">
  <user uri="sip:alice@atlanta.example.com"
                            display-name="Alice">
    <status>connected</status>
    <joining-mode>dialed-in</joining-mode>
    <media-stream media-type="audio">
       <proto>RTP/AVP</proto>
       <ssrc>674231</ssrc>
    </media-stream>
    <media-stream media-type="video">
       <proto>RTP/AVP</proto>
       <ssrc>213563</ssrc>
    </media-stream>
  </user>
  <user uri="sip:carol@chicago.example.com"
                            display-name="Carol">
    <status>connected</status>
    <joining-mode>dialed-out</joining-mode>
    <media-stream media-type="audio">
       <proto>RTP/AVP</proto>
       <ssrc>583398</ssrc>
    </media-stream>
    <media-stream media-type="video">
       <proto>RTP/AVP</proto>
       <ssrc>345212</ssrc>
    </media-stream>
  </user>
  <conf-uri>tel:+18882934234</conf-uri>
</conference-info>
```


F9
```
NOTIFY sip:alice@atlanta.example.com SIP/2.0
Via: SIP/2.0/UDP ms5.conf.example.com;branch=z9hG4bK3432
Max-Forwards: 70
To: Alice <sip:alice@atlanta.example.com>;tag=43524545
From: <sip:3402934234@example.com>;tag=a3343df32
Call-ID: 8820450524545
CSeq: 998 NOTIFY
Contact: <sip:3402934234@example.com>;isfocus
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
 SUBSCRIBE, NOTIFY
Event: conference
Accept: application/sdp, message/sipfrag
Subscription-State: active;expires=2450
Supported: replaces, gruu
Content-Type: application/conference-info+xml
Content-Length: ...

<conference-info version="0" state="partial"
               entity="sip:3402934234@example.com">
  <user uri="sip:carol@chicago.example.com"
                              display-name="Carol">
    <status>connected</status>
    <joining-mode>dialed-out</joining-mode>
    <media-stream media-type="audio">
       <proto>RTP/AVP</proto>
       <ssrc>583398</ssrc>
    </media-stream>
    <media-stream media-type="video">
       <proto>RTP/AVP</proto>
       <ssrc>345212</ssrc>
    </media-stream>
  </user>
  <conf-uri>tel:+18882934234</conf-uri>
</conference-info>
```

4.3 Manually Creating a Conference by Dialing into a Conferencing Application

In this section, a user sends an INVITE to a conference server application. The application (such as an IVR system or a web page) is implemented because the system requires additional input from the user before it is able to create a conference. After a normal dialog is established, additional information is received and the conference together with its focus are created. At this point the conference server MUST re-INVITE the user with the conference URI in Contact

with the "isfocus" feature parameter.

Alternatively, the additional information MAY be provided by the user during an early dialog established. This could be accomplished by a 183 Session Progress response sent by the conferencing application. After the conference is created, the conference URI MUST then be returned in a Contact in the 200 OK.

An example call flow is shown in Figure 3. In this example, Alice uses a conference application which is triggered when Alice sends an INVITE to the conference application. In this example, Conf-App is used to represent the conference application URI. Alice's conference-aware UA learns of the existence of the conference from the "isfocus" feature parameter and subscribes to the conference package to receive notifications of the conference state.

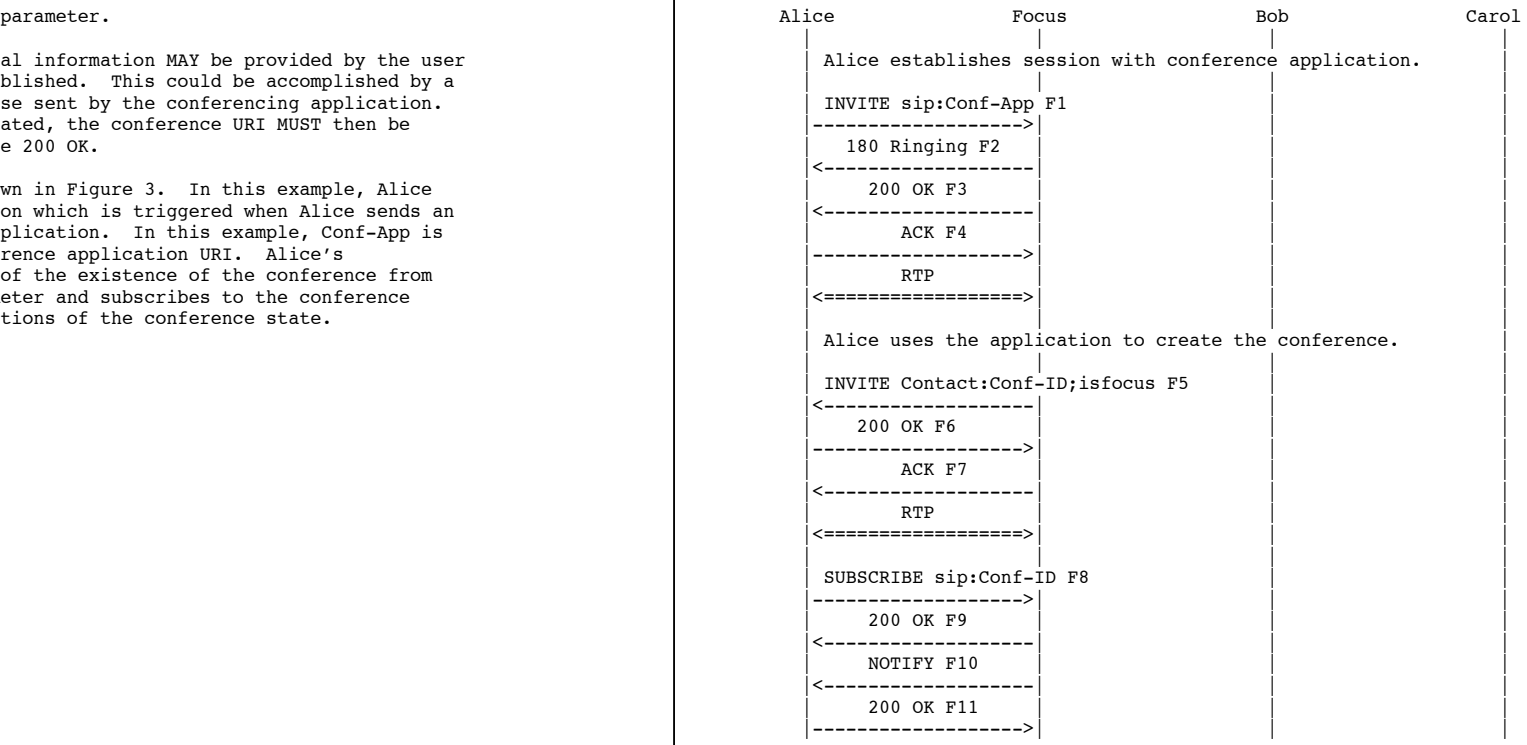


Figure 3. A Participant Creates a Conference using an Application.

4.4 Creating a Conference by a Conference-Unaware UA

It is a requirement that a user (human) be able to use a conference-unaware UA to create and add participants to a conference.

A user (human) would choose a conference URI according to system rules and insert it into the Request-URI of the INVITE. This same URI is echoed by a focus adhering to certain addressing conventions (discussed below) in the Contact header by the focus. Additional participants could be added by non-SIP means (publication of the

chosen conference URI using web pages, email, IM, etc.). Alternatively, the conference-unaware UA could then add other participants to the conference using SIP call control by establishing a session with them, then transferring [17] them to the conference URI. Note that in this scenario only the user (human) is aware of the conferencing application, and the conference-unaware UA only need support RFC 3261 and optionally call transfer.

Making this work does impose certain addressing conventions on a system. As a service/implementation choice, a system could allow the creator of the conference to choose the user portion of the conference URI. However, this requires the URI format to be agreed upon between a user and the system.

For example, a service provider might reserve the domain conf.example.com for all conference URIs. Any URI in the domain of conf.example.com would resolve to the focus. The focus could be configured to interpret an unknown user part in the conf.example.com domain as a request for a conference to be created with the conference URI as the Request-URI. For example, an INVITE sent with a Request-URI of sip:k32934208ds72@conf.example.com could be routed to the focus that would then create the conference. This conference URI should be registered by the newly created focus to become routable as a conference URI within the conf.example.com domain. The returned Contact would look as follows:

Contact: <sip:k32934208ds72@conf.example.com>;isfocus

Note, however, that this approach relies on conventions adopted between the user (human) and the focus. Also, the approach is not robust against collisions in the conference names. If a second user wishing to create a new conference happened to choose the same user part as an existing conference, the result would be that the second user would be added into the existing conference instead of creating a new one.

As a result, methods of conference creation in which the conference URI is an opaque URI generated by the focus are preferred.

An example call flow is shown in Figure 4. The participant Alice creates the conference URI (using some convention agreed to with the focus domain) and sends an INVITE to that URI which creates the focus. The focus creates the conference and returns the same conference URI in the 200 OK answer to the INVITE (which is ignored by the conference-unaware UA).

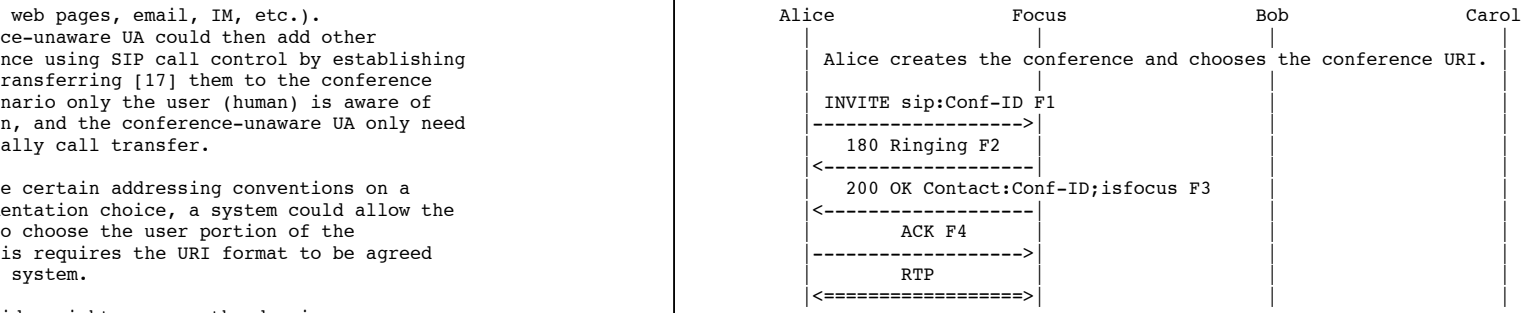


Figure 4. A Conferencing Unaware Participant Creates a Conference

4.5 Creating a Conference using Ad-Hoc SIP Methods

This section addresses creating a conference by using ad-hoc SIP means. The conference factory URI (as defined in Section 2.4) is used to automatically create the conference in this example.

The benefit of this approach is that the conference URI need not be known to the user - instead it is created by a focus and used by the participants' UAs. The main difference between this scenario and Section 4.3 is that no user intervention (IVR, web page form, etc.) is required to create the conference.

The SIP URI of the conference factory can be provisioned in the UA (as in a "create new conference" button on a SIP phone) or can be discovered using other means.

A SIP entity (such as conferencing server) can distinguish this INVITE request as a request to create a new ad-hoc conference from a request to join an existing conference by the Request-URI.

Assuming that all security and policy requirements have been met, a new conference will be created with the Contact URI returned in the 200 OK being the conference URI. The Contact header field MUST contain the "isfocus" feature parameter to indicate that this URI is for a conference.

An example call flow is shown in Figure 5. Note that Conf-Factory is shorthand for the conference factory URI and Conf-ID Is short for the conference URI. In this flow, Alice has a conference-aware UA and creates a conference by sending an INVITE to the conference factory

URI. The conference factory application creates the conference and redirects using a 302 Moved Temporarily response to the focus. Note that with proxy recursion, Alice may never see the redirect but may just receive the responses from the focus starting with message F5. Once the media session is established, Alice subscribes to the conference URI obtained through the Contact in the 200 OK response from the focus.

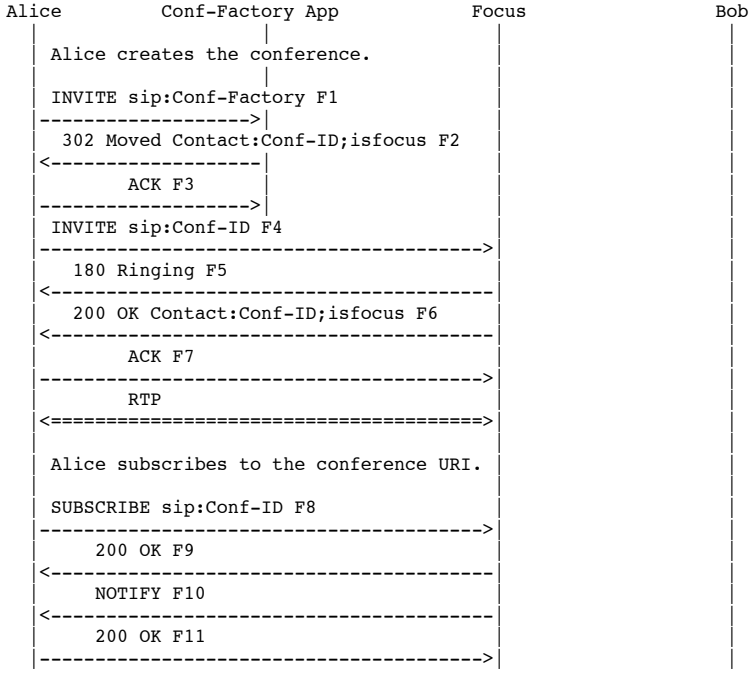


Figure 5. Creation of a Conference using SIP Ad-Hoc Methods.

4.6 Requesting the Focus Add a New Resource to a Conference

A SIP conference URI can be used to inject different kinds of information into the conference. Examples include new participants, new real-time media sources, new IM messages, and pointers to passive

information references (such as HTTP URIs).

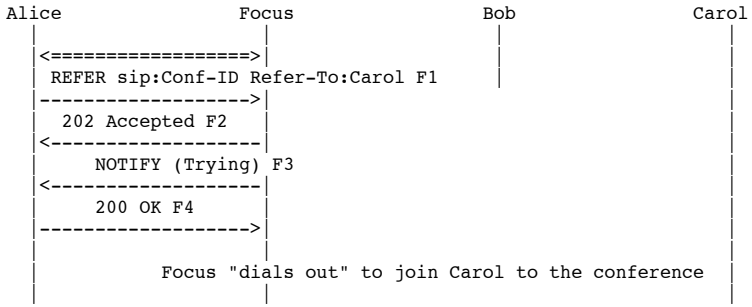
To request the focus add a new information resource to the specified conference, any SIP UA can send a REFER to the conference URI with a Refer-To containing the URI of the new resource. Since this REFER is sent to the conference URI and not the conference factory URI, the semantics to the focus are to bring the resource into the conference and make it visible to the conference participants. The resultant focus procedures are dependant both on the nature of the new resource (as expressed by its URI) and the own focus policies regarding IM, central vs. distributed real time media processing, etc.

The scenario for adding a new UA participant is important to support because it works even if the new participant does not support REFER and transfer call control - only the requesting participant and the focus need to support the REFER and transfer call control.

Upon receipt of the REFER containing a Refer-To header with a SIP URI, the focus SHOULD send an INVITE to the new participant identified by the Refer-To SIP URI containing a Contact header field with the conference URI and the "isfocus" feature parameter.

A conference-unaware UA would simply ignore the conferencing information and treat the session (from a SIP perspective) as a point to point session.

An example call flow is shown in Figure 6. It is assumed that Alice is already a participant of the conference. Alice sends a REFER to the conference URI. The focus invites Carol to the conference by sending an INVITE. After the session is established, Carol subscribes to the conference URI. It is important to note that there is no dependency on Carol's SUBSCRIBE (F11) and the NOTIFY to Alice (F15) - they occur asynchronously and independently.



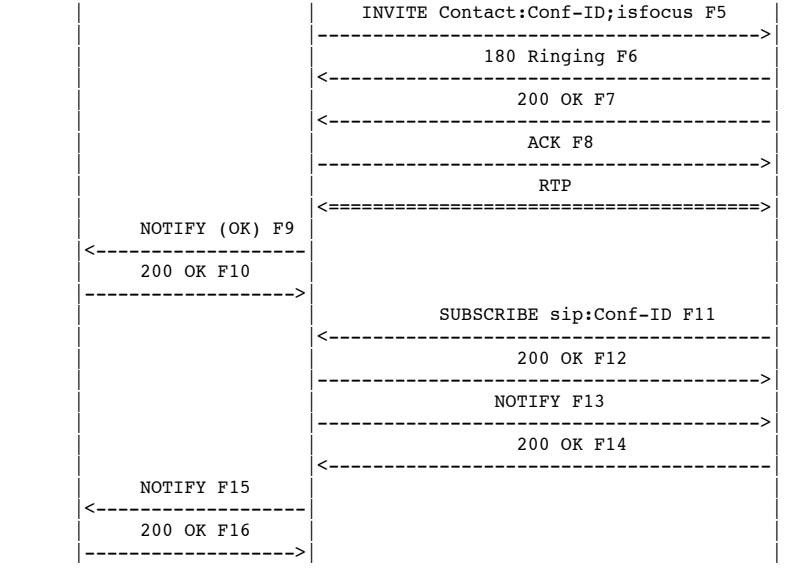


Figure 6. Participant Requests Focus add a Participant to the Conference.

```
 F1   REFER sip:3402934234@example.com SIP/2.0
      Via: SIP/2.0/UDP client.atlanta.example.com;branch=z9hG4bKg45344
      Max-Forwards: 70
      To: <sip:3402934234@example.com>
      From: Alice <sip:alice@atlanta.example.com>;tag=5534562
      Call-ID: 849392fklgl43
      CSeq: 476 REFER
      Contact: <sip:alice@alice.example.com>
      Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
       SUBSCRIBE, NOTIFY
      Accept: application/sdp, message/sipfrag
      Refer-To: <sip:carol@chicago.example.com>
      Supported: replaces
      Content-Length: 0
```


4.7 Adding a 3rd Party Using Conference URI

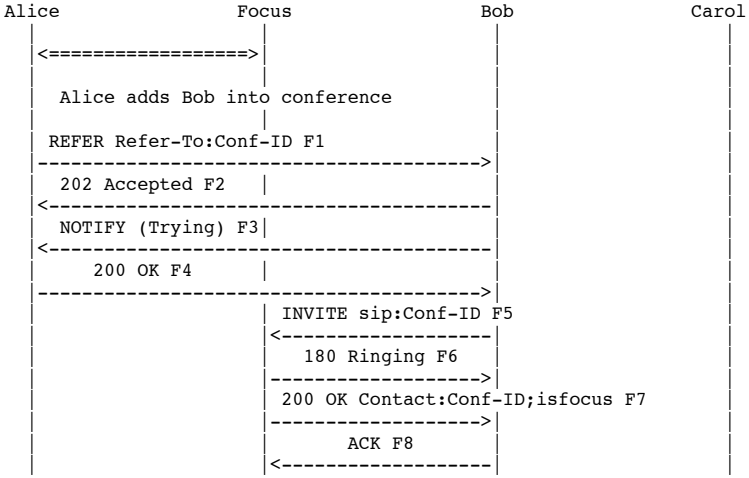
A participant wishing to add a new participant will request this participant to send an INVITE to the conference URI. This can be done using a non-SIP means (such as passing or publishing the conference URI in an email, IM, or web page). If a non-SIP means is used, then the flow and requirements are identical to Section 4.1.

The SIP mechanism to do this utilizes the REFER method.

A UA wishing to add a new participant SHOULD send a REFER request to the participant with a Refer-To header containing the conference URI.

The requirements are then identical to the "dial in" case of Section 4.1. The inviting participant MAY receive notification through the REFER action that the new participant has been added in addition to the notification received through the conference package.

An example is shown in Figure 7. In this call flow, it is assumed that Alice is already a participant of the conference. Alice sends Bob an "out of band" REFER - that is, a REFER outside of an established dialog. Should Bob reject the REFER, Alice might try sending an INVITE to Bob to establish a session first, then send a REFER within the dialog, effectively transferring Bob into the conference [17].



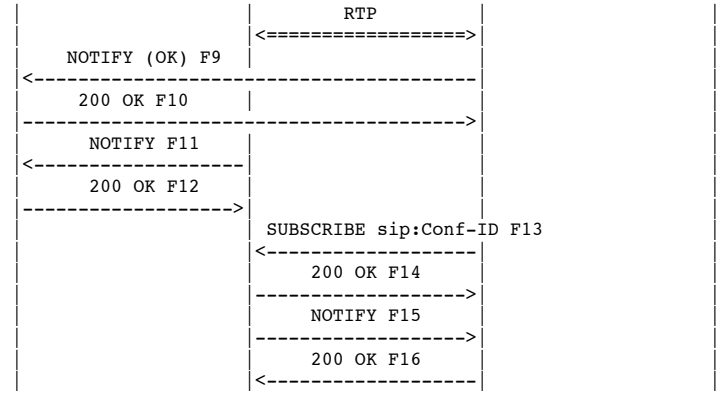


Figure 7. Adding a Participant to an Existing Conference.

4.8 Adding a 3rd Party Using a Dialog Identifier

Under some circumstances, a participant wanting to join a conference may only know a dialog identifier of one of the legs of the conference. The information may have been learned using the dialog package [18] or some non-SIP means to retrieve this information from a conference participant.

A UA can request to be added to a conference by sending a request to the focus containing a Join [6] header field containing a dialog ID of one leg of the conference (a dialog between a participant and the focus).

There are other scenarios in which a UA can use the Join header for certain conferencing call control scenarios. See [6] for further examples and details.

An example is shown in Figure 8. It is assumed that Alice is a participant of the conference. The dialog identifier between Alice and the focus is abbreviated as A-F and is known by Bob. Bob requests to be added to the conference by sending an INVITE message F1 to the focus containing a Join header which contains the dialog identifier A-F. Bob is added into the conference by the focus.

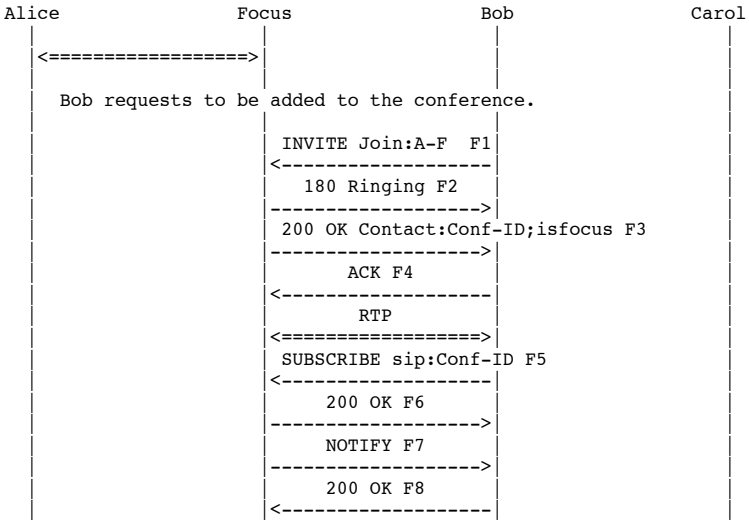


Figure 8. Adding a Participant to an Existing Conference using Join.

```
F1   INVITE sip:3402934234@example.com SIP/2.0
     Via: SIP/2.0/UDP client.biloxi.com;branch=z9hG4bKh3832
     Max-Forwards: 70
     To: <sip:3402934234@example.com>
     From: Bob <sip:bob@biloxi.example.com>;tag=32411
     Call-ID: d432fa84b4c76e66710
     CSeq: 8 INVITE
     Contact: <sip:bob@client.biloxi.example.com>
     Join: 3434034-293553453;to-tag=fdj3l34;from-tag=12f331
     Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
      SUBSCRIBE, NOTIFY
     Allow-Events: dialog
     Accept: application/sdp, message/sipfrag
     Supported: replaces, join
     Content-Type: application/sdp
     Content-Length: 274

     (SDP not shown)
```

4.9 Changing User Agents within a Conference

A participant in a conference may want to change the user agent with which they participate in the conference. While this could be done by simply sending a BYE from one user agent to leave the conference and an INVITE from the other user agent to rejoin. However, the SIP Replaces [6] primitive is perfectly suited to this operation.

An example is shown in Figure 9. It is assumed that Alice is a participant of the conference using user agent #1. The dialog identifier between Alice's user agent #1 and the focus is abbreviated as A-F. Alice switches to user agent #2 and sends an INVITE message F1 to the focus containing a Replaces header which contains the dialog identifier A-F. Note that this dialog identifier could be learned through some non-SIP mechanism, or by use of SUBSCRIBE/NOTIFY and the dialog event package [18]. Alice's user agent #2 is added into the conference by the focus. The focus sends a BYE to user agent #1. User agent #1 then automatically terminates the subscription by sending a SUBSCRIBE with Expires:0 to terminate the subscription. Note that as the participant list (roster) has not changed during this scenario, no NOTIFYs are sent by the focus to subscribers to the participant list.

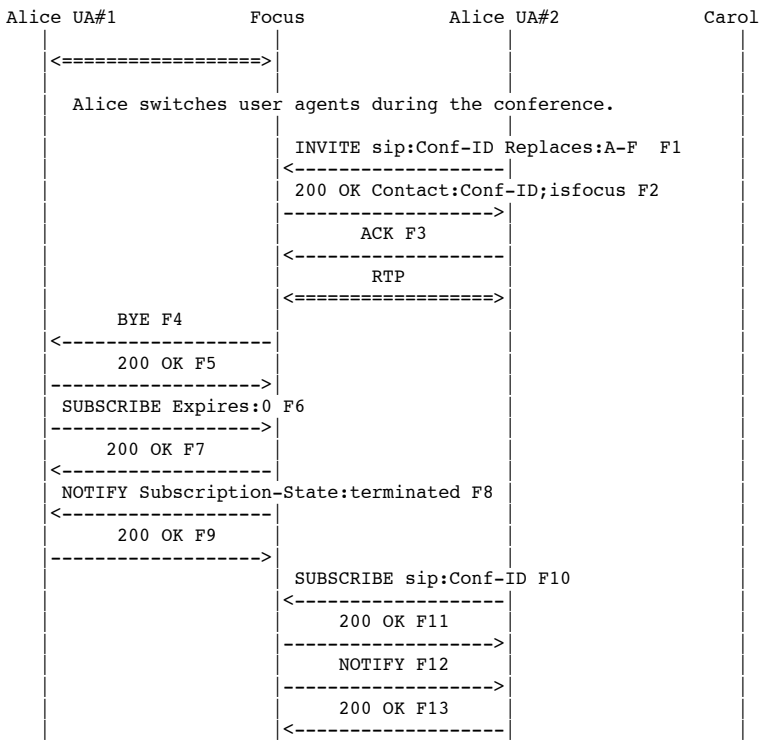


Figure 9. Adding a Participant to an Existing Conference using Join.

4.10 Bringing a Point-to-Point Dialog into a Conference

A focus is capable of bringing an existing point-to-point dialog with another UA to a conference that the focus hosts. The focus would do it by sending re-INVITE changing the Contact URI to the conference URI with the "isfocus" feature parameter. By doing this, the focus signals to the UA that it becomes a participant of the conference, specified in the Contact header.

Currently, there is no way for a UA, being in an active

point-to-point call with a focus, to express by SIP call control means a request to bridge its dialog with a specific conference or to create a new conference and include the dialog in this conference. Instead, a new dialog will need to be created. Even if the UA discovers that the other side has focus capabilities, the UA needs to close the old session and to establish a new session/dialog with the focus.

4.11 Requesting the Focus Remove a Participant from a Conference

To request the focus remove a participant from the specified conference, a properly authorized SIP UA (typically the conference owner) can send a REFER to the conference URI with a Refer-To containing the URI of the participant and with the method set to BYE. The requestor does not need to know the dialog information about the dialog between the focus and the participant who will be removed - the focus knows this information and fills it when it generates the BYE request.

An example call flow is shown in Figure 10. It is assumed that Alice and Carol are already participants of the conference and that Alice is authorized to remove members from the conference. Alice sends a REFER to the conference URI with a Refer-To header containing a URI of the form &ltsip:carol@chicago.example.com&method=BYE>.

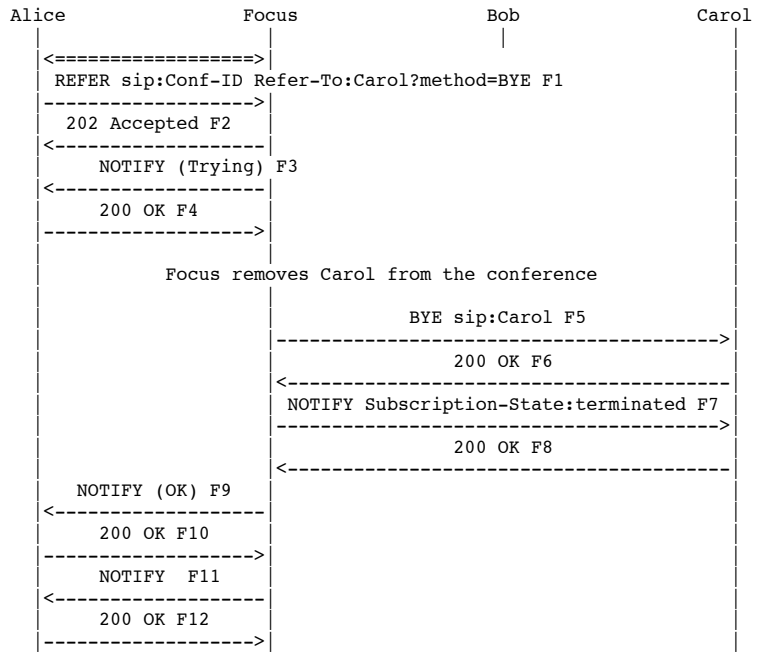


Figure 10. Participant Requests Focus Remove a Participant from the Conference.

```
 F1   REFER sip:3402934234@example.com SIP/2.0
      Via: SIP/2.0/UDP client.atlanta.example.com;branch=z9hG4bKg45344
      Max-Forwards: 70
      To: <sip:3402934234@example.com>
      From: Alice <sip:alice@atlanta.example.com>;tag=5534562
      Call-ID: 849392fklgl43
      CSeq: 476 REFER
      Contact: <sip:alice@alice.example.com>
      Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
       SUBSCRIBE, NOTIFY
      Accept: application/sdp, message/sipfrag
      Refer-To: <sip:carol@chicago.example.com;method=BYE>
      Supported: replaces
      Content-Length: 0



F5    BYE sip:carol@client.chicago.example.com SIP/2.0
      Via: SIP/2.0/UDP ms5.conf.example.com;branch=z9hG4bK343gf4
      Max-Forwards: 70
      From: <sip:3402934234@example.com>;tag=5393k2312
      To: Carol <sip:carol@chicago.example.com>;tag=32331
      Call-ID: d432fa84b4c76e66710
      CSeq: 78654 BYE
      Content-Length: 0
```

4.12 Discovery of Conferencing Capabilities using OPTIONS

A UA MAY send an OPTIONS request to discover if an opaque URI is a conference URI (resolves to a focus). In addition, the reply to the OPTIONS request can also indicate support for various SIP call control extensions used in this document.

Note that the Allow, Accept, Allow-Events, and Supported header fields should be present in an INVITE from a focus or a 200 OK answer from the focus to an INVITE as a part of a normal dialog establishment process.

An example is shown in Figure 11 where Alice sends an OPTIONS to a URI which resolves to a focus.

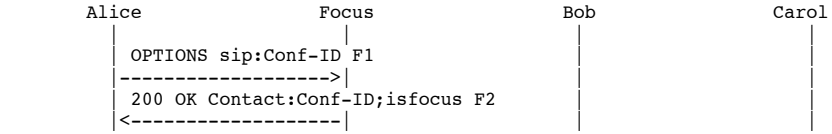


Figure 11. Participant Queries Capabilities of URI which resolves to a Focus.

Following is an example message detail of message F2 in Figure 11. Based on the response, Alice's UA learns that the URI is a conference URI and that the responding UA is focus that supports a number of SIP call control extensions.

The response details are as follows:

```
 F2   SIP/2.0 200 OK
      Via: SIP/2.0/UDP pc33.atlanta.example.com;branch=z9hG4bKhjhs8ass877
       ;received=192.0.2.4
      To: <sip:3402934234@example.com>;tag=93810874
      From: Alice <sip:alice@atlanta.example.com>;tag=1928301774
      Call-ID: a84b4c76e66710
      CSeq: 63104 OPTIONS
      Contact: <sip:3402934234@example.com>;isfocus
      Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
       SUBSCRIBE, NOTIFY
      Allow-Events: refer, conference
      Accept: application/sdp, application/conference-info+xml,
       message/sipfrag
      Accept-Language: en
      Supported: replaces, join, gruu
      Content-Type: application/sdp
      Content-Length: ...

      v=0
      o=focus431 2890844563 2890842835 IN IP4 ms5.conf.example.com
      s=Example Subject
      i=Example Conference Hosted by Example.com
      u=http://conf.example.com/3402934234
      e=3402934234@conf-help.example.com
      p=+18882934234
      c=IN IP4 ms5.conf.example.com
      t=0 0
      m=audio 49170 RTP/AVP 0 1 3 5 7
      m=video 51372 RTP/AVP 31 32
```

Useful information from each of these headers is detailed in the next sections.

Allow. The support of methods such as REFER, SUBSCRIBE, and NOTIFY indicate that the user agent supports call control and SIP Events.

Accept. The support of bodies such as message/sipfrag [14], application/conference-info+xml [5] also indicates support of call control and conferencing.

Allow-Events. The support of event packages such as refer [3], conference [5].

Supported. The support of extensions such as replaces, join, and gruu.

Contact. The presence of the "isfocus" feature parameter in the Contact header indicates that the URI is a conference URI and that the UA is a focus.

5. Security Considerations

This document discusses call control for SIP conferencing. Both call control and conferencing have specific security requirements which will be summarized here. Conferences generally have authorization rules about who may or may not join a conference, what type of media may or may not be used, etc. This information is used by the Focus to admit or deny participation in a conference. It is recommended that these types of authorization rules be used to provide security for a SIP conference. For this authorization information to be used, the focus needs to be able to authenticate potential participants. Normal SIP mechanisms including Digest authentication and certificates can be used. These conference specific security requirements are discussed further in the requirements and framework documents.

For call control security, a user agent must maintain local policy on who is permitted to perform call control operations, initiate REFERs, and replace dialogs. Normal SIP authentication mechanisms are also appropriate here. The specific authentication and authorization schemes are described in the multiparty call control framework document.

6. Contributors

We would like to thank Rohan Mahy, Jonathan Rosenberg, Roni Even, Petri Koskelainen, Brian Rosen, Paul Kyzivat, Eric Burger, and others in list discussions.

7. Changes since -02

- Added reference and text about use of GRUUs.

- Updated for latest version of conference package.

- Clarified that conference package subscription should use a separate dialog from INVITE dialog.

8. Changes since -01

- Added messages details of selected INVITE, 200 OK, SUBSCRIBE, REFER, and NOTIFY messages.

9. Changes since -00

- Showed separation between conference factory application and focus by having the application redirect to the newly created focus in the ad-hoc creation scenario.

- Removed inclusion of "isfocus" parameter in Refer-To header field - this may be a useful extension to the REFER mechanism in the future, however.

- Updated reference from Caller Prefs document to the new Capabilities of User Agents document.

- Added scenario of participant changing user agents during a conference.

- Added requirement on focus to support Replaces header field.

- Added discussion about termination of dialog using BYE and subscription using SUBSCRIBE/NOTIFY to flows involving termination of session with the focus.

Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[3] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[4] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[5] Rosenberg, J. and H. Schulzrinne, "A Session Initiation Protocol (SIP) Event Package for Conference State", draft-ietf-sipping-conference-package-02 (work in progress), October 2003.

[6] Mahy, R. and D. Petrie, "The Session Inititation Protocol (SIP)

'Join' Header", draft-ietf-sip-join-02 (work in progress), July 2003.

[7] Rosenberg, J., "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", draft-ietf-sip-callee-caps-03 (work in progress), January 2004.

[8] Biggs, B., Dean, R. and R. Mahy, "The Session Inititation Protocol (SIP) 'Replaces' Header", draft-ietf-sip-replaces-04 (work in progress), August 2003.

[9] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-00 (work in progress), January 2004.

Informative References

[10] Levin, O. and R. Even, "High Level Requirements for Tightly Coupled SIP Conferencing", draft-ietf-sipping-conferencing-requirements-00 (work in progress), April 2003.

[11] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-01 (work in progress), October 2003.

[12] Mahy, R., "A Call Control and Multi-party usage framework for the Session Initiation Protocol (SIP)", draft-ietf-sipping-cc-framework-03 (work in progress), October 2003.

[13] Campbell, B. and R. Sparks, "Control of Service Context using SIP Request-URI", RFC 3087, April 2001.

[14] Sparks, R., "Internet Media Type message/sipfrag", RFC 3420, November 2002.

[15] Johnston, A., Donovan, S., Sparks, R., Cunningham, C. and K. Summers, "Session Initiation Protocol (SIP) Basic Call Flow Examples", BCP 75, RFC 3665, December 2003.

[16] Johnston, A. and R. Sparks, "Session Initiation Protocol Service Examples", draft-ietf-sipping-service-examples-05 (work in progress), September 2003.

[17] Sparks, R. and A. Johnston, "Session Initiation Protocol Call Control - Transfer", draft-ietf-sipping-cc-transfer-01 (work in progress), February 2003.

[18] Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-03 (work in progress), October 2003.

Authors' Addresses

Alan Johnston
MCI
100 South 4th Street
St. Louis, MO 63102

EMail: alan.johnston@mci.com

Orit Levin
Microsoft
One Microsoft Way
Redmond, WA 75024

EMail: oritl@microsoft.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING
Internet-Draft
Expires: August 15, 2004

J. Rosenberg
dynamicsoft
H. Schulzrinne
Columbia University
O. Levin, Ed.
Microsoft Corporation
February 15, 2004

A Session Initiation Protocol (SIP) Event Package for Conference State

draft-ietf-sipping-conference-package-03

Status of this Memo

Copyright Notice

Abstract

This document defines a conference event package for the Session Initiation Protocol (SIP) Events framework, along with a data format used in notifications for this package. The conference package allows users to subscribe to a conference URI. Notifications are sent about changes in the membership of this conference, the status of users' participation in the conference, and the sidebars in the conference.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [3] Events framework [2] defines general mechanisms for subscribing to, and receiving notifications of, events within SIP networks. It introduces the notion of a package, which is a specific "instantiation" of the events framework for a well-defined set of events. Here, we define an event package for SIP conferences. This package provides the conference notification service as outlined in the SIP conferencing framework [9]. As described there, subscriptions to a conference URI are routed to the focus that is handling the conference. It acts as the notifer, and provides clients with updates on conference state.

The information provided by this package is comprised of conference identifier(s), conference participants (optionally with their statuses and media types) and conference sidebars.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [1] and indicate requirement levels for compliant implementations.

3. Conference Event Package

The conference event package allows a user to subscribe to a conference. In SIP, conferences are represented by URIs. These URIs route to a SIP user agent, called a focus, that is responsible for ensuring that all users in the conference can communicate with each other [9]. The focus has sufficient information about the state of the conference to inform subscribers about it.

It is possible a participant in the conference may in fact be another focus. In order to provide a more complete participant list, the focus MAY subscribe to the conference package of the other focus to discover the participant list in the cascaded conference. This information can then be included in notifications by using of the "cascaded-focus" attribute as specified by this package.

This section provides the details for defining a SIP Events package, as specified by [2].

3.1 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 [2].

3.2 SUBSCRIBE Bodies

A SUBSCRIBE for a conference package MAY contain a body. This body defines a filter to apply to the subscription. Filter documents are not specified in this document, and at the time of writing, are expected to be the subject of future standardization activity.

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 state of the conference.
- Notifications do not normally contain full state; rather, they only indicate the state that has changed. The exception is a NOTIFY sent in response to a SUBSCRIBE. These NOTIFYs contain the full state of the information requested by the subscriber.

3.3 Subscription Duration

The default expiration time for a subscription to a conference is one hour. Once the conference ends, all subscriptions to that particular

conference are terminated, with a reason of "noresource" [2].

3.4 NOTIFY Bodies

As described in RFC 3265 [2], the NOTIFY message will contain bodies that describe the state of the subscribed resource. This body is in a format listed in the Accept header field of the SUBSCRIBE, or a package-specific default if the Accept header field was omitted from the SUBSCRIBE.

In this event package, the body of the notification contains a conference information document. This document describes the state of a conference. All subscribers and notifiers MUST support the "application/conference-info+xml" data format described in Section 4. The subscribe request MAY contain an Accept header field. If no such header field is present, it has a default value of "application/conference-info+xml". If the header field is present, it MUST include "application/conference-info+xml", and MAY include any other types capable of representing dialog state.

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

3.5 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.

3.6 Notifier Generation of NOTIFY Requests

Notifications SHOULD be generated for the conference whenever there is a change in the state in any of the information delivered to the subscriber.

The changes generally occur when a new participant joins, a participant leaves, or a participant is put on-hold. Subject to a local focus policy, changes in media types and other optional media attributes MAY be reported by the focus. In addition, creation and deletion of sidebars together with their rosters MAY be reported by the focus, subject to its local policy.

3.7 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 in the conference has changed. To construct a coherent view of the total state of all users, a subscriber to the conference package will need to combine NOTIFYs received over time.

Notifications within this package can convey partial information; that is, they can indicate information about a subset of the state associated with the subscription. This means that an explicit algorithm needs to be defined in order to construct coherent and consistent state. The details of this mechanism are specific to the particular document type. See Section 4.2 for information on constructing coherent information from an application/ conference-info+xml document.

3.8 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.

3.9 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.

3.10 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.

4. Conference Data Format

Conference information is an XML document that MUST be well-formed and SHOULD be valid. Dialog information documents MUST be based on XML 1.0 and MUST be encoded using UTF-8. This specification makes use of XML namespaces for identifying dialog information documents and document fragments. The namespace URI for elements defined by this specification is a URN [4], using the namespace identifier 'ietf' defined by [5] and extended by [6]. This URN is:

urn:ietf:params:xml:ns:conference-info

A conference information document begins with the root element tag "conference-info".

4.1 Conference Information

Conference information begins with the top level element "conference-info". This element has three mandatory and one optional attributes:

version: This mandatory attribute allows the recipient of conference information documents to properly order them. Versions start at 0 and increment by one for each new document sent to a subscriber. Versions are scoped within a subscription. Versions MUST be represented using a 32 bit integer.

state: This mandatory attribute indicates whether the document contains the full conference information, or whether it contains only the information that has changed since the previous document (partial).

entity: This mandatory attribute contains the conference URI that identifies the conference being described in the document.

recording: This optional attribute indicates whether the conference is being recorded at this moment ("on") or not ("off").

The "conference-info" element has zero or more "user" sub-elements which contain information on the users in the conference. This is followed by zero or more "sidebar" sub-elements which contain information on the sidebars in the conference. This is followed by zero or more "conf-uri" sub-elements which contain information on additional URIs that the conference can be accessed by. This is followed by zero or more "policy-uri" sub-elements which contain information on additional URIs that the conference policies can be accessed by.

4.1.1 User Element

The user element has one mandatory attribute, "uri" that indicates the URI for the user in the conference. This is a logical identifier, not a machine specific one (i.e., it's taken from the authenticated identity of the participant). The optional attribute "display-name" contains a display name for the user. The standard "xml:lang" language attribute can also be present to indicate the language of the display-name.

The optional attribute "cascaded-focus" contains a conference URI (different from the main conference URI) for users that are connected to the main conference as a result of focus cascading. In accordance with the SIP conferencing framework [9], this defined package allows for representation of peer-to-peer (i.e. "flat") focus cascading only. The actual cascading graph is not explicitly expressed in this package because most applications do not care about the actual topology of the cascaded focuses as long as the information about their participants is available. In addition, an advanced application can construct the graph by subscribing to both this package and the Dialog Package [10] of the involved focuses and correlating the required information.

4.1.1.1 User Statuses

Three optional status elements are defined: status, joining-mode, and disconnection-reason.

- "status": provides information about user's current level of participation in the conference.
- "joining-mode": if present, provides information about the way the user joined the conference.
- "disconnection-reason": if present, provides information about the way the user left the conference.

The following statuses are defined for the "status" element:

connected: The user is a participant in the conference. Depending on the media policies, he/she can send and receive media to and from other participants.

disconnected: The user is not a participant in the conference and no active dialog exists between the user and the focus.

on-hold: Active SIP dialog exists between a user and a focus, but user is "on-hold" for this conference, i.e. neither he/she is "hearing" the conference mix, nor is his/her media being mixed in the conference. As an example, the user has asked to join the conference using SIP, but his/her participation is pending based on moderator approval. In the meantime he/she is hearing music-on-hold or some other kind of related content.

muted-by-focus: Active SIP dialog exists between a user and a focus and the user can "listen" to the conference, but user's media is not being mixed into the conference.

The following statuses are defined for the "joining-mode" element:

dialed-in: The user dialed into the conference, i.e. sent INVITE to the focus, which resulted in successful dialog establishment.

dialed-out: The focus has brought the user into the conference by sending a successful INVITE to the user.

focus-owner: The user is the focus itself for this conference.

The following statuses are defined for the disconnection-reason element:

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

booted: The user was sent a BYE by the focus, booting him/her out of the conference. Alternatively, the user tried to dial into to conference without success because was rejected by the focus according to local policy decisions.

failed: The server tried to bring the user into the conference, but its attempt to contact the specific user resulted in a non-200 class final response. Alternatively, the user tried to dial into the conference without success due to technical reasons.

4.1.1.2 Media Stream Information

Each user has zero or more "media-stream" sub-elements.

Each "media-stream" element indicates the media stream that the user is currently connected to. Here, "connected to" implies that a user has a media line in their SDP [12]. With this definition, a user is connected to a media stream even if they are not sending any media.

The "media-stream" element has a mandatory "media-type" attribute

which identifies the media type (e.g. audio, video, message and application) and MUST have one of the values registered for "media" of SDP [12].

The "media-stream" element has also an optional "proto" sub-element, which MUST has the value registered for "proto" of SDP [12]).

An optional "ssrc" sub-element, if present, carries the value of SSRC (RTP/RTCP [8]) as generated by the user for the stream it sends.

When an RTP mixer generates a CSRC list according to RTP/RTCP [8], it inserts a list of the SSRC identifiers of the sources that contributed to the generation of a particular packet into the RTP header of that packet. "An example application is audio conferencing where a mixer indicates all the talkers whose speech was combined to produce the outgoing packet, allowing the receiver to indicate the current talker, even though all the audio packets contain the same SSRC identifier (that of the mixer)."

4.1.2 Sidebar

The sidebar element has one attribute - "entity" that indicates the URI which identifiers the sidebar. A sidebar has zero or more users that are of type "user-type" as the users of the main conference are.

4.1.3 Additional Conference Identifiers

In addition to the Conference URI present in the "entity" attribute, a conference MAY have additional URIs of various types. Connecting to these URIs will result in joining to the same conference.

4.1.4 Policy URIs

A policy URI specifies where and how a certain policy pertaining to the conference can be accessed. The actual policy name and usage is deduced from the URI schema name.

An example for the "policy-uri" usage is inclusion of the URI of the CPCP [11]. A subscriber to the Conference package can use the Policy URI to access and modify the conference policy.

4.2 Constructing Coherent State

The conference information subscriber maintains a table for the list of users in the conference. The table contains a row for each user. Each row is indexed by a URI, present in the "uri" attribute of the "user" element. The contents of each row contain the state of that user as conveyed in the document.

The table is associated with a version number. The version number MUST be initialized with the value of the "version" attribute from the "conference-info" element in the first document received. Each time a new document is received, the value of the local version number, and the "version" attribute in the new document, are compared. If the value in the new document is one higher than the local version number, the local version number is increased by one, and the document is processed. If the value in the document is more than one higher than the local version number, the local version number is set to the value in the new document, the document is processed, and the subscriber SHOULD generate a refresh request to trigger a full state notification. If the value in the document is less than the local version, the document is discarded without processing.

The processing of the conference information document depends on whether it contains full or partial state. If it contains full state, indicated by the value of the "state" attribute in the "conference-info" element, the contents of the table is flushed. It is repopulated from the document. A new row in the user table is created for each "user" element. If the document contains partial state, as indicated by the value of the "state" attribute in the "conference-info" element, the document is used to update the table. For each "user" element in the document, the subscriber checks to see whether a row exists for that user in the user table. This check is done by comparing the URI in the "uri" attribute of the "user" element with the URI associated with the row. If the user doesn't exist in the table, a row is added, and its state is set to the information from that "user" element. If the user does exist, its state is updated to be the information from that "user" element. If a row is updated or created, such that its state is now disconnected, booted, failed or departed, that entry MAY be removed from the table at any time.

4.3 Schema

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:conference-info" xmlns:tns="urn:
<!--
 This import brings in the XML language attribute xml:lang
  -->
  <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="http:

<xs:element name="conference-info">

<xs:complexType>
```

```
   <xs:sequence>
     <xs:element name="user" type="user-type" minOccurs="0" maxOccurs="unbounded" />
     <xs:element name="sidebar" type="sidebar-type" minOccurs="0" maxOccurs="unbounded
     <xs:element name="conf-uri" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"
     <xs:element name="policy-uri" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded
     <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded" />
   </xs:sequence>

     <xs:attribute name="version" type="xs:nonNegativeInteger" use="required" />
     <xs:attribute name="state" use="required">
       <xs:simpleType>
        <xs:restriction base="xs:string">
           <xs:enumeration value="full" />
           <xs:enumeration value="partial" />
        </xs:restriction>
       </xs:simpleType>
     </xs:attribute>

     <xs:attribute name="entity" type="xs:anyURI" use="required" />
     <xs:attribute name="recording" type="tns:recording-type" use="optional" />

     <xs:anyAttribute />

   </xs:complexType>
   </xs:element>

   <xs:complexType name="user-type">
    <xs:sequence>
     <xs:element name="status" type="tns:status-type" minOccurs="0" />
     <xs:element name="joining-mode" type="tns:joining-mode-type" minOccurs="0" />
     <xs:element name="disconnection-reason" type="tns:disconnection-reason-type" minO
     <xs:element name="media-stream" type="tns:media-stream-type" minOccurs="0" maxOcc
     <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded" />
     </xs:sequence>


     <xs:attribute name="uri" type="xs:anyURI" use="required" />
     <xs:attribute name="display-name" type="xs:string" use="optional" />
     <xs:attribute ref="xml:lang" use="optional" />
     <xs:attribute name="cascaded-focus" type="xs:anyURI" use="optional" />
     <xs:anyAttribute />

   </xs:complexType>

   <xs:complexType name="sidebar-type">
    <xs:sequence>
     <xs:element name="user" type="user-type" minOccurs="0" maxOccurs="unbounded" />
     <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded" />
```



```
    </xs:sequence>

     <xs:attribute name="entity" type="xs:anyURI" use="required" />
   </xs:complexType>

   <xs:complexType name="media-stream-type">
    <xs:sequence>
     <xs:element name="proto" type="xs:string" minOccurs="0" />
     <xs:element name="ssrc" type="xs:nonNegativeInteger" minOccurs="0" />
     <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>

     <xs:attribute name="media" type="xs:string" use="required" />
   </xs:complexType>

   <xs:simpleType name="status-type">
    <xs:restriction base="xs:string">
     <xs:enumeration value="connected" />
     <xs:enumeration value="disconnected" />
     <xs:enumeration value="on-hold" />
     <xs:enumeration value="muted-by-focus" />
    </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name="joining-mode-type">
    <xs:restriction base="xs:string">
     <xs:enumeration value="dialed-in" />
     <xs:enumeration value="dialed-out" />
     <xs:enumeration value="focus-owner" />
    </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name="disconnection-reason-type">
    <xs:restriction base="xs:string">
     <xs:enumeration value="departed" />
     <xs:enumeration value="booted" />
     <xs:enumeration value="failed" />
    </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name="recording-type">
    <xs:restriction base="xs:string">
     <xs:enumeration value="on" />
     <xs:enumeration value="off" />
    </xs:restriction>
   </xs:simpleType>

   </xs:schema>
```

4.4 Example

The following is an example conference information document:

```
<conference-info version="0" state="full" entity="sip:conf233@example.com" recordin

  <user uri="sip:bob@example.com" display-name="Bob Jones">
    <status>connected</status>
    <joining-mode>dialed-in</joining-mode>
    <media-stream media-type="audio">
       <proto> RTP/AVP </proto>
       <ssrc> 583398 </ssrc>
    </media-stream>
  </user>

  <user uri="sip:barbara@example.com" display-name="Barbara Jones">
    <status>on-hold</status>
  </user>

  <user uri="sip:bill@example.com" display-name="Bill Minelli">
    <status>on-hold</status>
  </user>

  <sidebar entity="sip:conf233.1@example.com">
    <user uri="sip:barbara@example.com">
    <user uri="sip:bill@example.com">
  </sidebar>

  <conf-uri>tel:+18005671234</conf-uri>
  <conf-uri>h323:conf545@example.com</conf-uri>

</conference-info>
```

This conference currently has three users, two of which are in a sidebar conversation. The conference is being recorded. There are additional means to join the conference either by phone using tel URI [14] or by H.323 protocol using h323 URL [13].

5. Security Considerations

Subscriptions to 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.

6. IANA Considerations

This document registers a SIP event package, a new MIME type, application/conference-info+xml, a new XML namespace, and a new XML schema.

6.1 conference Event Package Registration

This specification registers an event package, based on the registration procedures defined in RFC 3265 [2]. The following is the information required for such a registration:

Package Name: conference

Package or Template-Package: This is a package.

Published Document: RFC XXXX (Note to RFC Editor: Please fill in XXXX with the RFC number of this specification).

Person to Contact: Jonathan Rosenberg, jdrosen@jdrosen.net.

6.2 application/conference-info+xml MIME Registration

MIME media type name: application

MIME subtype name: conference-info+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [7].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [7].

Security considerations: See Section 10 of RFC 3023 [7] and Section 5 of this specification.

Interoperability considerations: none.

Published specification: This document.

Applications which use this media type: This document type has been used to support SIP conferencing applications.

Additional Information:

Magic Number: None

File Extension: .cif or .xml

Macintosh file type code: "TEXT"

Personal and email address for further information: Jonathan Rosenberg, <jdrosen@jdrosen.net>

Intended usage: COMMON

Author/Change controller: The IETF.

6.3 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:conference-info

This section registers a new XML namespace, as per the guidelines in [6].

URI: The URI for this namespace is urn:ietf:params:xml:ns:conference-info.

Registrant Contact: IETF, SIPPING working group, <sipping@ietf.org>, Jonathan Rosenberg <jdrosen@jdrosen.net>.

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
     content="text/html;charset=iso-8859-1"/>
  <title>Conference Information Namespace</title>
</head
<body>
  <h1>Namespace for Conference Information</h1>
  <h2>urn:ietf:params:xml:ns:conference-info</h2>
  <p>See <a href="[[[URL of published RFC]]]">RFCXXXX</a>.</p>
</body>
</html>
END
```

6.4 XML Schema Registration

This specification registers a schema, as per the guidelines in in [6].

URI: please assign.

Registrant Contact: IETF, SIPPING Working Group (sipping@ietf.org), Jonathan Rosenberg (jdrosen@jdrosen.net).

XML: The XML can be found as the sole content of Section 4.3.

7. Acknowledgements

The authors would like to thank Dan Petrie, Sean Olson, and Alan Johnston for their comments.

8. Changes since -02

- State "muted-by-focus" is added to user's status.
- Optional conference attribute "recording" is added.
- Policy URI placeholder (i.e. element "policy-uri") is created.
- ExampleÆs syntax is corrected.
- Optional attribute "cascaded-focus" URI per user is added.
- Optional additional conference identifiers (i.e. element "conf-uri") are added.
- In order to cover all possible cases, participant's status is expressed using three optional statuses: "status", "joining-mode" and "disconnection-reason". That is instead of "activity-status", "history-status" and "is-on-dial-out-list".

9. Changes since -01

- Package parameters are removed. Decision about performing "recursive" membership algorithm is perceived as a focus local policy.
- General information (i.e. pointers to additional available services) is removed. The defined XML schema can be extended in future to include those when XCON work matures.
- Dialog information is removed. It can be obtained by direct subscription to a dialog package of a participant.
- Media stream information is aligned with SDP definitions (media and proto) and SSRC attribute is added.
- Participant's status is expressed using two optional statuses: "activity" and "history". Optional "is-on-a-dial-out-list" indication is added.
- Normative references to XCON work are removed.
- Optional sidebar rosters are added.

Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[3] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[4] Moats, R., "URN Syntax", RFC 2141, May 1997.

[5] Moats, R., "A URN Namespace for IETF Documents", RFC 2648, August 1999.

[6] Mealling, M., "The IETF XML Registry", draft-mealling-iana-xmlns-registry-05 (work in progress), June 2003.

[7] Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[8] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

Informative References

[9] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-01 (work in progress), October 2003.

[10] Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-03 (work in progress), October 2003.

[11] Koskelainen, P., "Requirements for Conference Policy Control Protocol", draft-ietf-xcon-cpcp-reqs-02 (work in progress), February 2004.

[12] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[13] Levin, O., "H.323 Uniform Resource Locator (URL) Scheme Registration", RFC 3508, April 2003.

[14] Schulzrinne, H., "The tel URI for Telephone Numbers", draft-ietf-iptel-tel-rfc2806bis-03 (work in progress), February 2004.

Authors' Addresses

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
EMail: jdrosen@dynamicsoft.com
URI: http://www.jdrosen.net

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027
US

EMail: schulzrinne@cs.columbia.edu
URI: http://www.cs.columbia.edu/~hgs

Orit Levin (editor)
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA

EMail: oritl@microsoft.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP J. Rosenberg
Internet-Draft dynamicsoft
Expires: April 26, 2004 October 27, 2003

A Framework for Conferencing with the Session Initiation Protocol
draft-ietf-sipping-conferencing-framework-01

Status of this Memo

Copyright Notice

Abstract

The Session Initiation Protocol (SIP) supports the initiation, modification, and termination of media sessions between user agents. These sessions are managed by SIP dialogs, which represent a SIP relationship between a pair of user agents. Because dialogs are between pairs of user agents, SIP's usage for two-party communications (such as a phone call), is obvious. Communications sessions with multiple participants, generally known as conferencing, are more complicated. This document defines a framework for how such conferencing can occur. This framework describes the overall architecture, terminology, and protocol components needed for multi-party conferencing.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [1] supports the initiation, modification, and termination of media sessions between user agents. These sessions are managed by SIP dialogs, which represent a SIP relationship between a pair of user agents. Because dialogs are between pairs of user agents, SIP's usage for two-party communications (such as a phone call), is obvious. Communications sessions with multiple participants, however, are more complicated. SIP can support many models of multi-party communications. One, referred to as loosely coupled conferences, makes use of multicast media groups. In the loosely coupled model, there is no signaling relationship between participants in the conference. There is no central point of control or conference server. Participation is gradually learned through control information that is passed as part of the conference (using the Real Time Control Protocol (RTCP) [2], for example). Loosely coupled conferences are easily supported in SIP by using multicast addresses within its session descriptions.

In another model, referred to as fully distributed multiparty conferencing, each participant maintains a signaling relationship with each other participant, using SIP. There is no central point of control; it is completely distributed amongst the participants. This model is outside the scope of this document.

In another model, sometimes referred to as the tightly coupled conference, there is a central point of control. Each participant connects to this central point. It provides a variety of conference functions, and may possibly perform media mixing functions as well. Tightly coupled conferences are not directly addressed by RFC 3261, although basic participation is possible without any additional protocol support.

This document is one of a series of specifications that discusses tightly coupled conferences. Here, we present the overall framework for tightly coupled conferencing, referred to simply as "conferencing" from this point forward. This framework presents a general architectural model for these conferences, presents terminology used to discuss such conferences, and describes the sets of protocols involved in a conference. The aim of the framework is to meet the general requirements for conferencing that are outlined in [3].

2. Terminology

Conference: Conference is an overused term which has different meanings in different contexts. In SIP, a conference is an instance of a multi-party conversation. Within the context of this specification, a conference is always a tightly coupled conference.

Loosely Coupled Conference: A loosely coupled conference is a conference without coordinated signaling relationships amongst participants. Loosely coupled conferences frequently use multicast for distribution of conference memberships.

Tightly Coupled Conference: A tightly coupled conference is a conference in which a single user agent, referred to as a focus, maintains a dialog with each participant. The focus plays the role of the centralized manager of the conference, and is addressed by a conference URI.

Focus: The focus is a SIP user agent that is addressed by a conference URI and identifies a conference (recall that a conference is a unique instance of a multi-party conversation). The focus maintains a SIP signaling relationship with each participant in the conference. The focus is responsible for ensuring, in some way, that each participant receives the media that make up the conference. The focus also implements conference policies. The focus is a logical role.

Conference URI: A URI, usually a SIP URI, which identifies the focus of a conference.

Participant: The software element that connects a user or automata to a conference. It implements, at a minimum, a SIP user agent, but may also include a conference policy control protocol client, for example.

Conference Notification Service: A conference notification service is a logical function provided by the focus. The focus can act as a notifier [4], accepting subscriptions to the conference state, and notifying subscribers about changes to that state. The state includes the state maintained by the focus itself, the conference policy, and the media policy.

Conference Policy Server: A conference policy server is a logical function which can store and manipulate the conference policy. The conference policy is the overall set of rules governing operation of the conference. It is broken into membership policy and media policy. Unlike the focus, there is not an instance of the

conference policy server for each conference. Rather, there is an instance of the membership and media policies for each conference.

Conference Policy: The complete set of rules for a particular conference manipulated by the conference policy server. It includes the membership policy and the media policy. There is an instance of conference policy for each conference.

Membership Policy: A set of rules manipulated by the conference policy server regarding participation in a specific conference. These rules include directives on the lifespan of the conference, who can and cannot join the conference, definitions of roles available in the conference and the responsibilities associated with those roles, and policies on who is allowed to request which roles.

Media Policy: A set of rules manipulated by the conference policy server regarding the media composition of the conference. The media policy is used by the focus to determine the mixing characteristics for the conference. The media policy includes rules about which participants receive media from which other participants, and the ways in which that media is combined for each participant. In the case of audio, these rules can include the relative volumes at which each participant is mixed. In the case of video, these rules can indicate whether the video is tiled, whether the video indicates the loudest speaker, and so on.

Conference Policy Control Protocol (CPCP): The protocol used by clients to manipulate the conference policy.

Mixer: A mixer receives a set of media streams of the same type, and combines their media in a type-specific manner, redistributing the result to each participant. This includes media transported using RTP \cite{rfc1889}. As a result, the term defined here is a superset of the mixer concept defined in RFC 1889, since it allows for non-RTP-based media such as instant messaging sessions [5].

Conference-Unaware Participant: A conference-unaware participant is a participant in a conference that is not aware that it is actually in a conference. As far as the UA is concerned, it is a point-to-point call.

Cascaded Conferencing: A mechanism for group communications in which a set of conferences are linked by having their focuses interact in some fashion.

Simplex Cascaded Conferences: a group of conferences which are linked such that the user agent which represents the focus of one conference is a conference-unaware participant in another conference.

Conference-Aware Participant: A conference-aware participant is a participant in a conference that has learned, through automated means, that it is in a conference, and that can use a conference policy control protocol, media policy control protocol, or conference subscription, to implement advanced functionality.

Conference Server: A conference server is a physical server which contains, at a minimum, the focus. It may also include a conference policy server and mixers.

Mass Invitation: A conference policy control protocol request to invite a large number of users into the conference.

Mass Ejection: A conference policy control protocol request to remove a large number of users from the conference.

Sidebar: A sidebar appears to the users within the sidebar as a "conference within the conference". It is a conversation amongst a subset of the participants to which the remaining participants are not privy.

Anonymous Participant: An anonymous participant is one that is known to other participants through the conference notification service, but whose identity is being withheld.

Hidden Participant: A hidden participant is one that is not known to other participants in the conference. They may be known to the moderator, depending on conference policy.

3. Overview of Conferencing Architecture

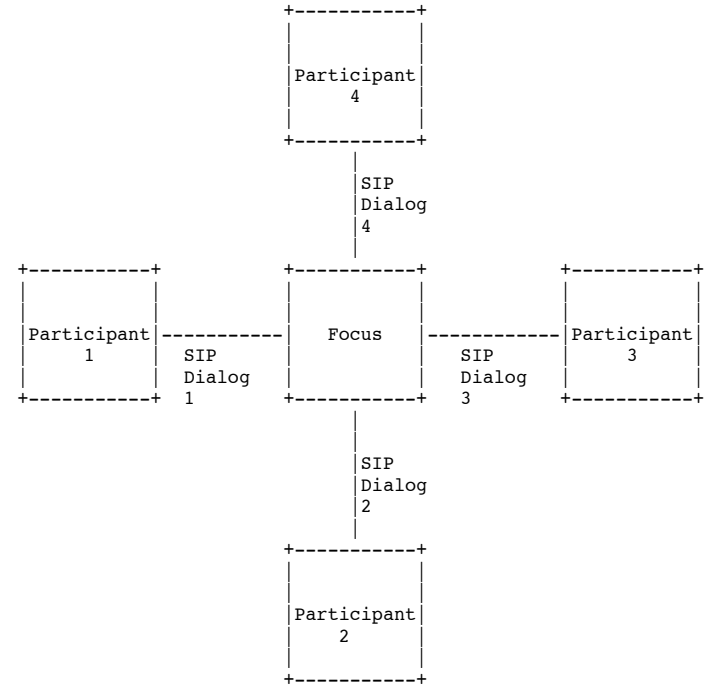


Figure 1

The central component (literally) in a SIP conference is the focus. The focus maintains a SIP signaling relationship with each participant in the conference. The result is a star topology, shown in Figure Figure 1.

The focus is responsible for making sure that the media streams which constitute the conference are available to the participants in the conference. It does that through the use of one or more mixers, each of which combines a number of input media streams to produce one or

more output media streams. The focus uses the media policy to determine the proper configuration of the mixers.

The focus has access to the conference policy (composed of the membership and media policies), an instance of which exist for each conference. Effectively, the conference policy can be thought of as a database which describes the way that the conference should operate. It is the responsibility of the focus to enforce those policies. Not only does the focus need read access to the database, but it needs to know when it has changed. Such changes might result in SIP signaling (for example, the ejection of a user from the conference using BYE), and most changes will require a notification to be sent to subscribers using the conference notification service.

The conference is represented by a URI, which identifies the focus. Each conference has a unique focus and a unique URI identifying that focus. Requests to the conference URI are routed to the focus for that specific conference.

Users usually join the conference by sending an INVITE to the conference URI. As long as the conference policy allows, the INVITE is accepted by the focus and the user is brought into the conference. Users can leave the conference by sending a BYE, as they would in a normal call.

Similarly, the focus can terminate a dialog with a participant, should the conference policy change to indicate that the participant is no longer allowed in the conference. A focus can also initiate an INVITE, should the conference policy indicate that the focus needs to bring a participant into the conference.

The notion of a conference-unaware participant is important in this framework. A conference-unaware participant does not even know that the UA it is communicating with happens to be a focus. As far as it's concerned, its a UA just like any other. The focus, of course, knows that its a focus, and it performs the tasks needed for the conference to operate.

Conference-unaware participants have access to a good deal of functionality. They can join and leave conferences using SIP, and obtain more advanced features through stimulus signaling, as discussed in [6]. However, if the participant wishes to explicitly control aspects of the conference using functional signaling protocols, the participant must be conference-aware.

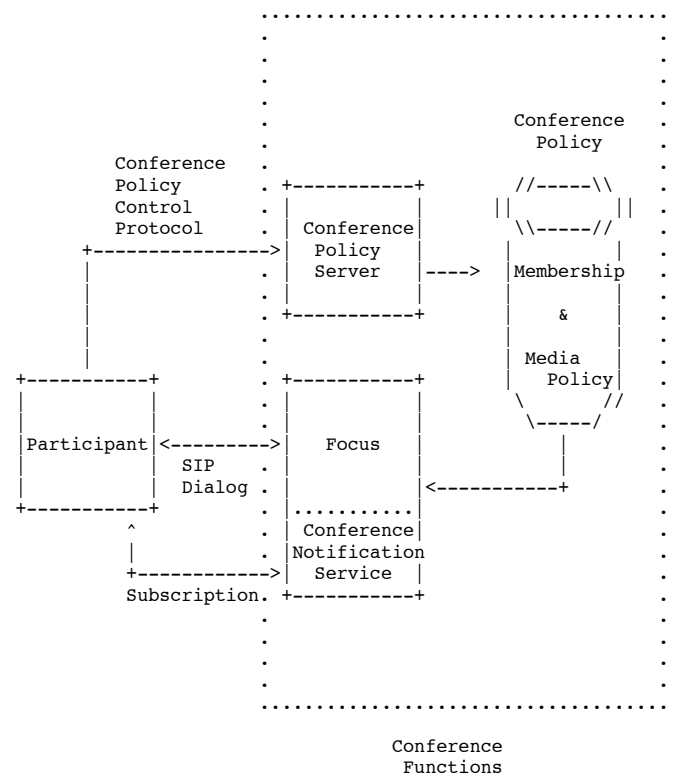


Figure 2

A conference-aware participant is one that has access to advanced functionality through additional protocol interfaces. The client uses these protocols to interact with the conference policy server and the focus. A model for this interaction is shown in Figure Figure 2. The participant can interact with the focus using extensions, such as REFER, in order to access enhanced call control functions [7]. The participant can SUBSCRIBE to the conference URI, and be connected to the conference notification service provided by the focus. Through this mechanism, it can learn about changes in participants

(effectively, the state of the dialogs), the media policy, and the membership policy.

The participant can communicate with the conference policy server using a conference policy control protocol. Through this protocol, it can affect the conference policy. The conference policy server need not be available in any particular conference, although there is always a conference policy.

The interfaces between the focus and the conference policy, and the conference policy server and the conference policy, are not subject to standardization at the time of this writing. They are intended primarily to show the logical roles involved in a conference, as opposed to suggesting a physical decomposition. The separation of these functions is documented here to encourage clarity in the requirements and to allow individual implementations the flexibility to compose a conferencing system in a scalable and robust manner.

3.1 Usage of URIs

It is fundamental to this framework that a conference is uniquely identified by a URI, and that this URI identifies the focus which is responsible for the conference. The conference URI is unique, such that no two conferences have the same conference URI. A conference URI is always a SIP or SIPS URI.

The conference URI is opaque to any participants which might use it. There is no way to look at the URI, and know for certain whether it identifies a focus, as opposed to a user or an interface on a PSTN gateway. This is in line with the general philosophy of URI usage [8]. However, contextual information surrounding the URI (for example, SIP header parameters) may indicate that the URI represents a conference.

When a SIP request is sent to the conference URI, that request is routed to the focus, and only to the focus. The element or system that creates the conference URI is responsible for guaranteeing this property.

The conference URI can represent a long-lived conference or interest group, such as "sip:discussion-on-dogs@example.com". The focus identified by this URI would always exist, and always be managing the conference for whatever participants are currently joined. Other conference URIs can represent short-lived conferences, such as an ad-hoc conference.

Ideally, a conference URI is never constructed or guessed by a user. Rather, conference URIs are learned through many mechanisms. A

conference URI can be emailed or sent in an instant message. A conference URI can be linked on a web page. A conference URI can be obtained from a conference policy control protocol, which can be used to create conferences and the policies associated with them.

To determine that a SIP URI does represent a focus, standard techniques for URI capability discovery can be used. Specifically, the callee capabilities specification [9] provides the "isfocus" feature tag to indicate that the URI is a focus. Caller preferences parameters are also used to indicate that a focus supports the conference notification service. This is done by declaring support for the SUBSCRIBE method and the relevant package(s) in the caller preferences feature parameters associated with the conference URI.

The other functions in a conference are also represented by URIs. If the conference policy server is implemented through web pages, this server is identified by HTTP URIs. If it is accessed using an explicit protocol, it is a URI defined for that protocol.

Starting with the conference URI, the URIs for the other logical entities in the conference can be learned using the conference notification service.

4. Functions of the Elements

This section gives a more detailed description of the functions typically implemented in each of the elements.

4.1 Focus

As its name implies, the focus is the center of the conference. All participants in the conference are connected to it by a SIP dialog. The focus is responsible for maintaining the dialogs connected to it. It ensures that the dialogs are connected to a set of participants who are allowed to participate in the conference, as defined by the membership policy. The focus also uses SIP to manipulate the media sessions, in order to make sure each participant obtains all the media for the conference. To do that, the focus makes use of mixers.

When a focus receives an INVITE, it checks the membership policy. The membership policy might indicate that this participant is not allowed to join, in which case the call can be rejected. It might indicate that another participant, acting as a moderator, needs to approve this new participant. In that case, the INVITE might be parked on a music-on-hold server, or a 183 response might be sent to indicate progress. A notification, using the conference notification service, would be sent to the moderator. The moderator then has the ability to manipulate the policies using the conference policy control protocol. If the policies are changed to allow this new participant, the focus can accept the INVITE (or unpark it from the music-on-hold server). The interpretation of the membership policy by the focus is, itself, a matter of local policy, and not subject to standardization.

If a participant manipulated the membership policy to indicate that a certain other participant was no longer allowed in the conference, the focus would send a BYE to that other participant to remove them. This is often referred to as "ejecting" a user from the conference. The process of ejecting fundamentally constitutes these two steps - the establishment of the policy through the conference policy protocol, and the implementation of that policy (using a BYE) by the focus.

Similarly, if a user manipulated the membership policy to indicate that a number of users need to be added to the conference, the focus would send an INVITE to those participants. This is often referred to as the "mass invitation" function. As with ejection, it is fundamentally composed of the policy functions that specify the participants which should be present, and the implementation of those functions. A policy request to add a set of users might not require an INVITE to execute it; those users might already be participants in the conference.

A similar model exists for media policy. If the media policy indicates that a participant should not receive any video, the focus might implement that policy by sending a re-INVITE, removing the media stream to that participant. Alternatively, if the video is being centrally mixed, it could inform the mixer to send a black screen to that participant. The means by which the policy is implemented are not subject to specification.

4.2 Conference Policy Server

The conference policy server allows clients to manipulate and interact with the conference policy. The conference policy is used by the focus to make authorization decisions and guide its overall behavior. Logically speaking, there is a one-to-one mapping between a conference policy and a focus.

The conference policy is represented by a URI. There is a unique conference policy for each conference. The conference policy URI points to a conference policy server which can manipulate that conference policy. A conference policy server also has a "top level" URI which can be used to access functions that are independent of any conference. Perhaps the most important of these functions is the creation of a new conference. Creation of a new conference will result in the construction of a new focus and a corresponding conference URI, which can then be used to join the conference itself, along with a media policy and conference policy.

The conference policy server is accessed using a client-server transactional protocol. The client can be a participant in the conference, or it can be a third party. Access control lists for who can modify a conference policy are themselves part of the conference policy.

The conference policy server is responsible for reconciliation of potentially conflicting requests regarding the policy for the conference.

The client of the conference policy control protocol can be any entity interested in manipulating the conference policy. Clearly, participants might be interested in manipulating them. A participant might want to raise or lower the volume for one of the other participants it is hearing. Or, a participant might want to add a user to the conference.

A client of the conference policy protocol could also be another server whose job is to determine the conference policy. As an example, a floor control server is responsible for determining which participant(s) in a conference are allowed to speak at any given

time, based on participant requests and access rules. The floor control server would act as a client of the conference policy server, and change the media policy based on who is allowed to speak.

The client of the conference policy control protocol could also be another conference policy server.

4.3 Mixers

A mixer is responsible for combining the media streams that make up the conference, and generating one or more output streams that are distributed to recipients (which could be participants or other mixers). The process of combining media is specific to the media type, and is directed by the focus, under the guidance of the rules described in the media policy.

A mixer is not aware of a "conference" as an entity, per se. A mixer receives media streams as inputs, and based on directions provided by the focus, generates media streams as outputs. There is no grouping of media streams beyond the policies that describe the ways in which the streams are mixed.

A mixer is always under the control of a focus. The focus is responsible for interpreting the media policy, and then installing the appropriate rules in the mixer. If the focus is directly controlling a mixer, the mixer can either be co-resident with the focus, or can be controlled through some kind of protocol.

However, a focus need not directly control a mixer. Rather, a focus can delegate the mixing to the participants, each of which has their own mixer. This is described in Section Section 6.4.

4.4 Conference Notification Service

The focus can provide a conference notification service. In this role, it acts as a notifier, as defined in RFC 3265 [4]. It accepts subscriptions from clients for the conference URI, and generates notifications to them as the state of the conference changes.

This state is composed of two separate pieces. The first is the state of the focus and the second is the conference policy. A subscriber to the conference notification service can use capabilities defined in the SIP events framework [4] to request that it receive focus state changes only, conference policy changes only, or both.

The state of the focus includes the participants connected to the focus, and information about the dialogs associated with them. As new participants join, this state changes, and is reported through the

notification service. Similarly, when someone leaves, this state also changes, allowing subscribers to learn about this fact.

As described previously, the conference policy includes the membership policy and the media policy. As those policies change, due to usage of the CPCP, direct change by the focus, or through an application, the conference notification service informs subscribers of these changes.

4.5 Participants

A participant in a conference is any SIP user agent that has a dialog with the focus. This SIP user agent can be a PC application, a SIP hardphone, or a PSTN gateway. It can also be another focus. A conference which has a participant that is the focus of another conference is called a simplex cascaded conference. They can also be used to provide scalable conferences where there are regional sub-conferences, each of which is connected to the main conference.

4.6 Conference Policy

The conference policy contains the rules that guide the operation of the focus. The rules can be simple, such as an access list that defines the set of allowed participants in a conference. The rules can also be incredibly complex, specifying time-of-day based rules on participation conditional on the presence of other participants. It is important to understand that there is no restriction on the type of rules that can be encapsulated in a conference policy.

The conference policy can be manipulated using web applications or voice applications. It can also be manipulated with proprietary protocols. However, the conference policy control protocol can be used as a standardized means of manipulating the conference policy. By the nature of conference policies, not all aspects of the policy can be manipulated with the conference policy control protocol.

The conference policy includes the membership policy and the media policy. The membership policy includes per-participant policies that specify how the focus is to handle a particular participant. These include whether or not the participant is anonymous, for example.

The media policy describes the way in which the set of inputs to a mixer are combined to generate the set of outputs. Media policies can span media types. In other words, the policy on how one media stream is mixed can be based on characteristics of other media streams. Media policies can be based on any quantifiable characteristic of the media stream (its source, volume, codecs, speaking/silence, etc.), and they can be based on internal or external variables accessible by

the media policy.

Some examples of media policies include:

- The video output is the picture of the loudest speaker (video follows audio).
- The audio from each participant will be mixed with equal weight, and distributed to all other participants.
- The audio and video that is distributed is the one selected by the floor control server.

5. Common Operations

There are a large number of ways in which users can interact with a conference. They can join, leave, set policies, approve members, and so on. This section is meant as an overview of the major conferencing operations, summarizing how they operate. More detailed examples of the SIP mechanisms can be found in [7].

5.1 Creating Conferences

There are many ways in which a conference can be created. The creation of a conference actually constructs several elements all at the same time. It results in the creation of a focus and a conference policy. It also results in the construction of a conference URI, which uniquely identifies the focus. Since the conference URI needs to be unique, the element which creates conferences is responsible for guaranteeing that uniqueness. This can be accomplished deterministically, by keeping records of conference URIs, or by generating URIs algorithmically, or probabilistically, by creating random URI with sufficiently low probabilities of collision.

When a media and conference policy are created, they are established with default rules that are implementation dependent. If the creator of the conference wishes to change those rules, they would do so using the conference policy control protocol (CPCP), for example.

Of course, using the CPCP requires that an element know the URI for manipulating the policy. That requires a means to learn the conference policy URI from the conference URI, since the conference URI is frequently the sole result returned to the client as a result of conference creation. Any other URIs associated with the conference are learned through the conference notification service. They are carried as elements in the notifications.

5.1.1 SIP Mechanisms

SIP can be used to create conferences hosted in a central server by sending an INVITE to a conferencing application that would automatically create a new conference and then place a user into it.

Creation of conferences where the focus resides in an endpoint operates differently. There, the endpoint itself creates the conference URI, and hands it out to other endpoints which are to be the participants. What differs from case to case is how the endpoint decides to create a conference.

One important case is the ad-hoc conference described in Section 6.2. There, an endpoint unilaterally decides to create the conference

based on local policy. The dialogs that were connected to the UA are migrated to the endpoint-hosted focus, using a re-INVITE to pass the conference URI to the newly joined participants.

Alternatively, one UA can ask another UA to create an endpoint-hosted conference. This is accomplished with the SIP Join header [10]. The UA which receives the Join header in an invitation may need to create a new conference URI (a new one is not needed if the dialog that is being joined is already part of a conference). The conference URI is then handed to the recently joined participants through a re-INVITE.

5.1.2 CPCP Mechanisms

Another way to create a conference is through interaction with the conference policy server. Using the conference policy control protocol, a client can instruct the conference policy server to create a new conference and return the conference URI and conference policy URI.

5.1.3 Non-Automated Mechanisms

One way to create a conference is through interaction with an IVR application. The user would send a SIP INVITE to the conferencing application. This application would interact with the user, collect information about the desired conference, and create it. The user can then be placed into their newly created conference.

Of course, a user can also create conferences by interacting with a web server. The web server would prompt the user for the neccessary information (start and stop times of the conference, participants, etc.) and return the conference URI to the user. The user would copy this URI into their SIP phone, and send it an INVITE in order to join the newly-created conference.

5.2 Adding Participants

There are many mechanisms for adding participants to a conference. These include SIP, the conference policy control protocol, and non-automated means. In all cases, participant additions can be first party (a user adds themself) or third party (a user adds another user).

5.2.1 SIP Mechanisms

First person additions using SIP are trivially accomplished with a standard INVITE. A participant can send an INVITE request to the conference URI, and if the conference policy allows them to join, they are added to the conference.

If a UA does not know the conference URI, but has learned about a dialog which is connected to a conference (by using the dialog event package, for example [11]), the UA can join the conference by using the Join header to join the dialog.

Third party additions with SIP are done using REFER [12]. The client can send a REFER request to the participant, asking them to send an INVITE request to the conference URI. Additionally, the client can send a REFER request to the focus, asking it to send an INVITE to the participant. The latter technique has the benefit of allowing a client to add a conference-unaware participant that does not support the REFER method.

5.2.2 CPCP Mechanisms

A basic function of the conference policy control protocol is to add participants. A client of the protocol can specify any SIP URI (which may identify themself) that is to be added. If the URI does not identify a user that is already a participant in the conference, the focus will send an INVITE to that URI in order to add them in.

5.2.3 Non-Automated Mechanisms

There are countless non-automated means for asking a participant to join the conference. Generally, they involve conveying the conference URI to the desired participant, so that they can send an INVITE to it. These mechanisms all require some kind of human interaction.

As an example, a user can send an instant message [13] to the third party, containing an HTML document which requests the user to click on the hyperlink to join the conference:

```
<html>
Hey, would you like to <a href="sip:9sf88fk-99sd@conferences.example.com">join
</a> the conference now?
</html>
```

5.3 Conditional Joins

In many cases, a new participant will not wish to join the conference unless they can join with a particular set of policies. As an example, a participant may want to join anonymously, so that other participants know that someone has joined, but not who. To accomplish this, the conference policy control protocol is used to establish these policies prior to the generation or acceptance of an invitation to the conference. For example, if a user wishes to join a conference

with a known conference URI, the user would obtain the URI for the conference policy, manipulate the policy to set themself as an anonymous participant, and then actually join the conference by sending an INVITE request to the conference URI.

5.4 Removing Participants

As with additions, there are several mechanisms for departures. These include SIP mechanisms and CPCP mechanisms. Removals can also be first person or third person.

5.4.1 SIP Mechanisms

First person departures are trivially accomplished by sending a BYE request to the focus. This terminates the dialog with the focus and removes the participant from the conference.

Third person departures can also be done using SIP, through the REFER method.

5.4.2 CPCP Mechanisms

The CPCP can be used by a client to remove any participant (including themself). When CPCP is used for this purpose, the focus will send a BYE request to the participant that is being removed. The focus will execute any other signaling that is needed to remove them (for example, manipulate other dialogs in order to manage the change in media streams).

The conference policy control protocol can also be used to remove a large number of users. This is generally referred to as mass ejection.

5.4.3 Non-Automated Mechanisms

As with the other common conferencing functions, there are many non-automated ways to remove a participant. The identity of the participant can be entered into a web form. When the user clicks submit, the focus sends a BYE to that participant, removing them from the conference. Alternatively, the conference can expose an IM interface, where the user can send an IM to the conference saying "remove Bob", causing the conference server to remove Bob.

5.5 Approving Policy Changes

OPEN ISSUE: The basic mechanism described here depends on the actual protocols used for conference and media policy manipulation. If the protocol itself provides change

notifications, sip-events may not be needed for that purpose. Thus, this description here is tentative.

A conference policy for a particular conference may designate one or more users as moderators for some set of media policy or conference policy change requests. This means that those moderators need to approve the specific policy change. Typically, moderators are used to approve member additions and removals. However, the framework allows for moderators to be associated with any policy change that can be made.

Moderating a policy request is done using a combination of the conference notification service and the CPCP protocol.

First, a client makes a policy change. This can be directly, using the CPCP, or indirectly. An indirect policy change request is any non-CPCP action that requires approval. The simplest example is an INVITE to the focus from a new participant. That represents a request to change the membership of the conference. From a moderation perspective, it is handled identically to the case where a client used the CPCP to request that the same user to be added to the conference.

Part of the conference policy itself may designate any policy change as moderated. This means that they change cannot be performed by the client directly. As a result, the CPCP request will be answered with a response saying that the action will be done pending authorization. That completes the CPCP transaction. In the case of a policy change requested indirectly through some other means, the behavior depends on the mechanism. For example, if a user sends a SIP INVITE request to the conference in order to join, and that join request is moderated, the focus would normally accept it and play music-on-hold until the request is approved.

Even though the CPCP transaction failed, it does result in a change in internal state. Specifically, the requested change shows up as a "pending" state within the media and conference policies. This means that the change has been requested, but has not taken effect. It is almost a form of change request history. However, because it is a state change, it is something that can result in notifications through the conference notification service.

Therefore, in order to moderate requests, the moderator subscribes to the conference policy notification service. Normally, the notifications from the focus do not reflect pending state changes. That is, the service will not normally send a notification informing a subscriber that a policy change request was made and failed due to lack of authorization. However, notifications to the moderator do

reflect these changes. That is because the policy of the focus is to inform moderators, and only moderators, of these changes. Indeed, different users can be moderators for different parts of the conference and media policies. For example, one user can be a moderator for membership changes, and another, a moderator for whether users can be anonymously joined or not.

There are two ways that the focus knows whether a subscriber to the conference notification service is a moderator. The first is configured policy (once again through CPCP). That policy can specify that a particular user is the moderator for a particular piece of policy. Therefore, if that user subscribes to the conference notification service, any notification sent to that user will include pending changes to that piece of policy. As an alternative, a SUBSCRIBE request from a user can include a filter [14] that requests receipt of these pending state changes. If the conference policy allows, that request is honored, and the subscriber will receive notifications about pending state changes.

Once the moderator receives a notification about the pending state change, they use the CPCP to implement their decision. If the moderator decides to approve the change, they use the CPCP or MPCP to actually perform the change themselves. Since the moderator for a piece of policy is allowed to change that piece of policy, by definition, their change is accepted and performed. If the moderator decides to reject the change, they use the CPCP to remove the pending state from the database.

The pending state persists in the database for a period of time which is, itself, part of the conference policy. If the moderator does not either approve or reject the change, the pending state eventually disappears, as if the change was explicitly rejected.

If the pending state is approved, a real change to the conference or media policy takes place, and this change will be reflected in the conference notification service. In this way, if a client makes a policy change, and their request is rejected because they are not authorized, the client can subscribe to the conference notification service to learn if their change is eventually approved or rejected.

This general mechanism for moderating policy requests is consistent with the moderation of presence subscriptions [15][16].

5.6 Creating Sidebars

A sidebar is a "conference within a conference", allowing a subset of the participants to converse amongst themselves. Frequently, participants in a sidebar will still receive media from the main

conference, but "in the background". For audio, this may mean that the volume of the media is reduced, for example.

A sidebar is represented by a separate conference URI. This URI is a type of "alias" for the main conference URI. Both route to the same focus. Like any other conference, the sidebar conference URI has a conference policy and a media policy associated with it. Like any other conference, one can join it by sending an INVITE to this URI, or ask others to join by referring them to it. However, it differs from a normal conference URI in several ways. First, users in the main conference do not need to establish a separate dialog to the sidebar conference. The focus recognizes the sidebar as a special URI, and knows to use the existing dialog to the main conference as a "virtual" connection to the sidebar URI.

The second difference is the way in which conference and media policies are implemented. If the conference policy control protocol is used to add a user to a normal conference, the focus will typically send an INVITE to the participant to ask them to join. For a sidebar conference, it is done differently. If the conference policy control protocol is used to add a user to it, and that user is already part of the main conference, the focus will use the conference notification service to alert the existing participant that they have been asked to join the sidebar. The invited user can then make use of the CPCP to formally add themselves to the sidebar.

5.7 Destroying Conferences

Conferences can be destroyed in several ways. Generally, whether those means are applicable for any particular conference is a component of the conference policy.

When a conference is destroyed, the conference and media policies associated with it are destroyed. Any attempts to read or write those policies results in a protocol error. Furthermore, the conference URI becomes invalid. Any attempts to send an INVITE to it, or SUBSCRIBE to it, would result in a SIP error response.

Typically, if a conference is destroyed while there are still participants, the focus would send a BYE to those participants before actually destroying the conference. Similarly, if there were any users subscribed to the conference notification service, those subscriptions would be terminated by the server before the actual destruction.

5.7.1 SIP Mechanisms

There is no explicit means in SIP to destroy a conference. However, a

conference may be destroyed as a by-product of a user leaving the conference, which can be done with BYE. In particular, if the conference policy states that the conference is destroyed once the last user leaves, when that user does leave (using a SIP BYE request), the conference is destroyed.

5.7.2 CPCP Mechanisms

The CPCP contains mechanisms for explicitly destroying a conference.

5.7.3 Non-Automated Mechanisms

As with conference creation, a conference can be destroyed by interacting with a web application or voice application that prompts the user for the conference to be destroyed.

5.8 Obtaining Membership Information

A participant in a conference will frequently wish to know the set of other users in the conference. This information can be obtained many ways.

5.8.1 SIP Mechanisms

The conference notification service allows a conference aware participant to subscribe to it, and receive notifications that contain the list of participants. When a new participant joins or leaves, subscribers are notified. The conference notification service also allows a user to do a "fetch" [4] to obtain the current listing.

5.8.2 CPCP Mechanisms

The CPCP contains mechanisms for querying for the current set of conference participants.

5.8.3 Non-Automated Mechanisms

Users can also interact with applications to obtain conference membership. There may be a conference web page associated with the conference, which has a link that will fetch the current list of participants and display them in the browser. Similarly, an interactive voice response application connected to the focus can be used to obtain the current membership. A user in the conference could press the pound key on their phone, and hear a listing of the current participants.

5.9 Adding and Removing Media

Each conference is composed of a particular set of media that the focus is managing. For example, a conference might contain a video stream and an audio stream. The set of media streams that constitute the conference can be changed by participants. When the set of media in the conference change, the focus will need to generate a re-INVITE to each participant in order to add or remove the media stream to each participant. When a media stream is being added, a participant can reject the offered media stream, in which case it will not receive or contribute to that stream. Rejection of a stream by a participant does not imply that that the stream is no longer part of the conference - just that the participant is not involved in it.

There are several ways in which a media stream can be added or removed from a conference.

5.9.1 SIP Mechanisms

A SIP re-INVITE can be used by a participant to add or remove a media stream. This is accomplished using the standard offer/answer techniques for adding media streams to a session [17]. This will trigger the focus to generate its own re-INVITEs.

5.9.2 CPCP Mechanisms

The CPCP can be used to add or remove a media stream. This too will trigger the focus to generate a re-INVITE to each participant in order to affect the change.

5.9.3 Non-Automated Mechanisms

As with most of the other common functions, addition and removal of media streams can be accomplished with a web application or interactive voice application.

5.10 Conference Announcements and Recordings

Conference announcements and recordings play a key role in many real conferencing systems. Examples of such features include:

- Asking a user to state their name before joining the conference, in order to support a roll call
- Allowing a user to request a roll call, so they can hear who else is in the conference
- Allowing a user to press some keys on their keypad in order to record the conference

- Allowing a user to press some keys on their keypad in order to be connected with a human operator

- Allowing a user to press some keys on their keypad to mute or unmute their line

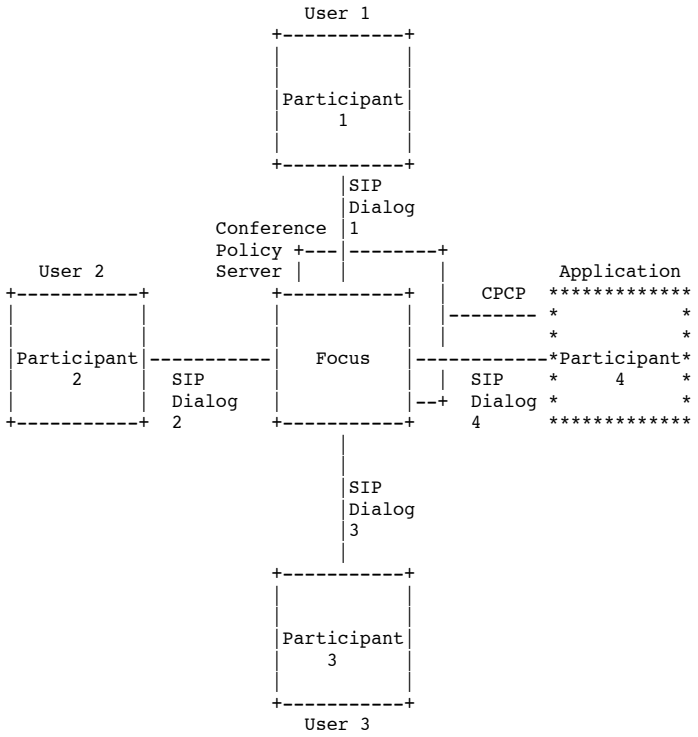


Figure 4

In this framework, these capabilities are modeled as an application which acts as a participant in the conference. This is shown

pictorially in Figure 4. The conference has four participants. Three of these participants are end users, and the fourth is the announcement application.

If the announcement application wishes to play an announcement to all the conference members (for example, to announce a join), it merely sends media to the mixer as would any other participant. The announcement is mixed in with the conversation and played to the participants.

Similarly, the announcement application can play an announcement to a specific user by using the CPCP to configure its media policy so that the media it generates is only heard by the target user. The application then generates the desired announcement, and it will be heard only by the selected recipient.

The announcement application can also receive input from a specific user through the conference. The announcement application would use the CPCP to cause in-band DTMF to be dropped from the mix, and sent only to itself. When a user wishes to invoke an operation, such as to obtain a roll call, the user would press the appropriate key sequence. That sequence would be heard only by the announcement application. Once the application determines that the user wishes to hear a roll call, it can use the CPCP to set the media policy so that media from that user is delivered only to the announcement application. This "disconnects" the user from the rest of the conference so they can interact with the application. Once the interaction is done, and announcement application uses the CPCP to "reconnect" the user to the conference.

5.11 Floor Control

Floor control is similar to a conference announcement application. Within this framework, floor control is managed by an application (possibly one that is not a participant) that uses the CPCP to enforce the resulting floor control decisions.

[[Need more work here]]

5.12 Camera and Video Controls

OPEN ISSUE: Originally, I was just going to say that this is outside the scope of conferencing. But, it does impact conferencing. Effectively, camera control is treated like a media stream. The mixer would combine the various requests across participants and direct them to the appropriate device. How does that work though? In a video conference with 4 participants, the camera control needs to identify the specific user whose camera is

to be controlled. That is something unique to conferencing.

6. Physical Realization

In this section, we present several physical instantiations of these components, to show how these basic functions can be combined to solve a variety of problems.

6.1 Centralized Server

In the most simplistic realization of this framework, there is a single physical server in the network which implements the focus, the conference policy server, and the mixers. This is the classic "one box" solution, shown in Figure 5.

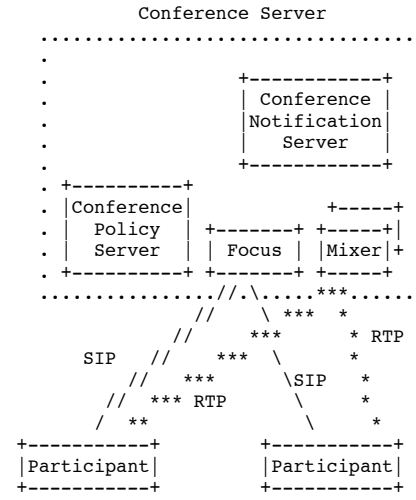


Figure 5

6.2 Endpoint Server

Another important model is that of a locally-mixed ad-hoc conference. In this scenario, two users (A and B) are in a regular point-to-point call. One of the participants (A) decides to conference in a third participant, C. To do this, A begins acting as a focus. Its existing

dialog with B becomes the first dialog attached to the focus. A would re-INVITE B on that dialog, changing its Contact URI to a new value which identifies the focus. In essence, A "mutates" from a single-user UA to a focus plus a single user UA, and in the process of such a mutation, its URI changes. Then, the focus makes an outbound INVITE to C. When C accepts, it mixes the media from B and C together, redistributing the results. The mixed media is also played locally. Figure 6 shows a diagram of this transition.

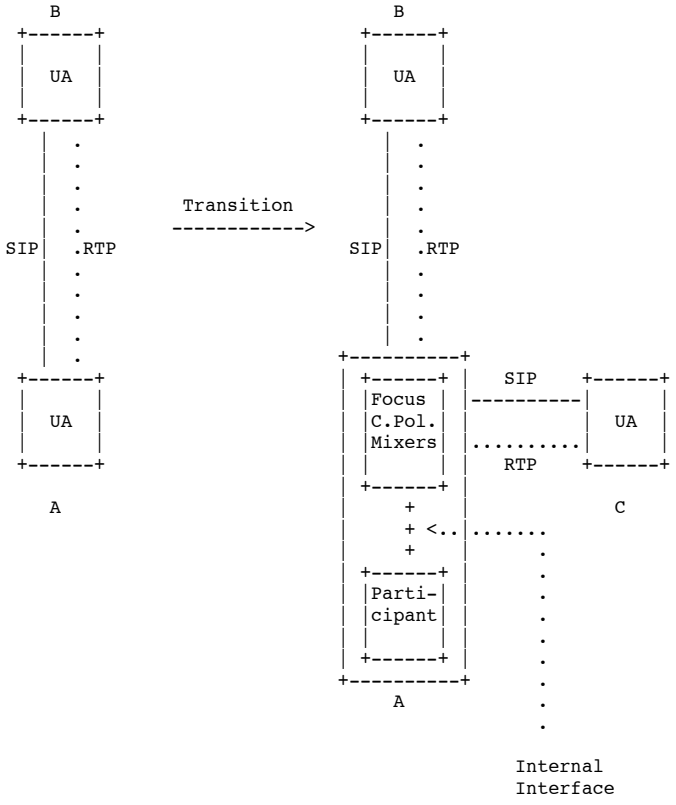


Figure 6

It is important to note that the external interfaces in this model, between A and B, and between B and C, are exactly the same to those that would be used in a centralized server model. B could also include a conference policy server and conference notification service, allowing the participants to have access to them if they so desired. Just because the focus is co-resident with a participant does not mean any aspect of the behaviors and external interfaces will change.

6.3 Media Server Component

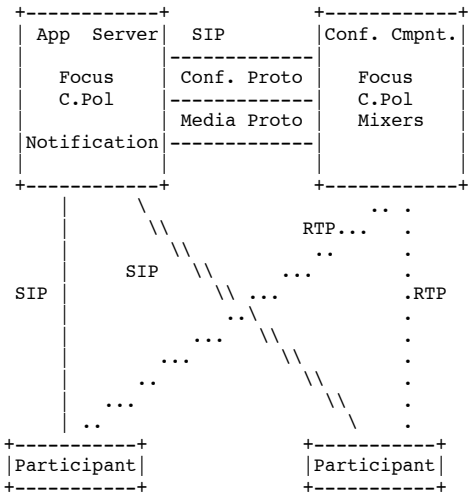


Figure 7

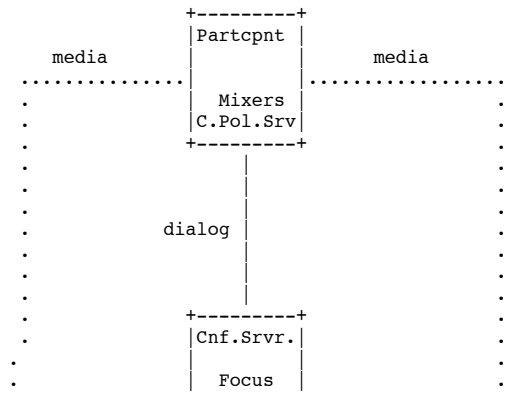
In this model, shown in Figure 7, each conference involves two centralized servers. One of these servers, referred to as the "application server" owns and manages the membership and media policies, and maintains a dialog with each participant. As a result, it represents the focus seen by all participants in a conference. However, this server doesn't provide any media support. To perform the actual media mixing function, it makes use of a second server, called the "mixing server". This server includes a focus, and a

conference policy server, but has no conference notification service. It has a default membership policy, which accepts all invitations from the top-level focus. Its conference policy server accepts any controls made by the application server. The focus in the application server uses third party call control to connect the media streams of each user to the mixing server, as needed. If the focus in the application server receives a conference policy control command from a client, it delegates that to the media server by making the same media policy control command to it.

This model allows for the mixing server to be used as a resource for a variety of different conferencing applications. This is because it is unaware of any conference or media policies; it is merely a "slave" to the top-level server, doing whatever it asks.

6.4 Distributed Mixing

In a distributed mixed conference, there is still a centralized server which implements the focus, conference policy server, and media policy server. However, there are no centralized mixers. Rather, there are mixers in each endpoint, along with a conference policy server. The focus distributes the media by using third party call control [18] to move a media stream between each participant and each other participant. As a result, if there are N participants in the conference, there will be a single dialog between each participant and the focus, but the session description associated with that dialog will be constructed to allow media to be distributed amongst the participants. This is shown in Figure 8.



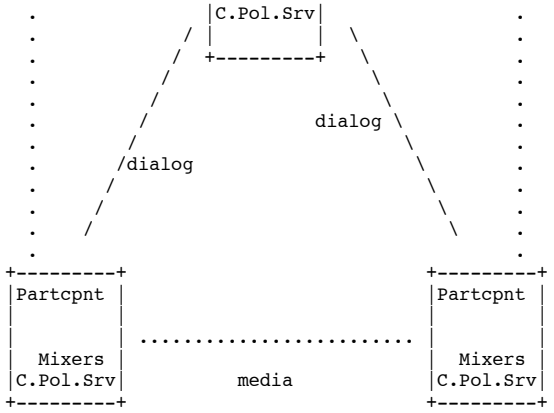


Figure 8

There are several ways in which the media can be distributed to each participant for mixing. In a multi-unicast model, each participant sends a copy of its media to each other participant. In this case, the session description manages N-1 media streams. In a multicast model, each participant joins a common multicast group, and each participant sends a single copy of its media stream to that group. The underlying multicast infrastructure then distributes the media, so that each participant gets a copy. In a single-source multicast model (SSM), each participant sends its media stream to a central point, using unicast. The central point then redistributes the media to all participants using multicast. The focus is responsible for selecting the modality of media distribution, and for handling any hybrids that would be necessitated from clients with mixed capabilities.

When a new participant joins or is added, the focus will perform the necessary third party call control to distribute the media from the new participant to all the other participants, and vice-a-versa.

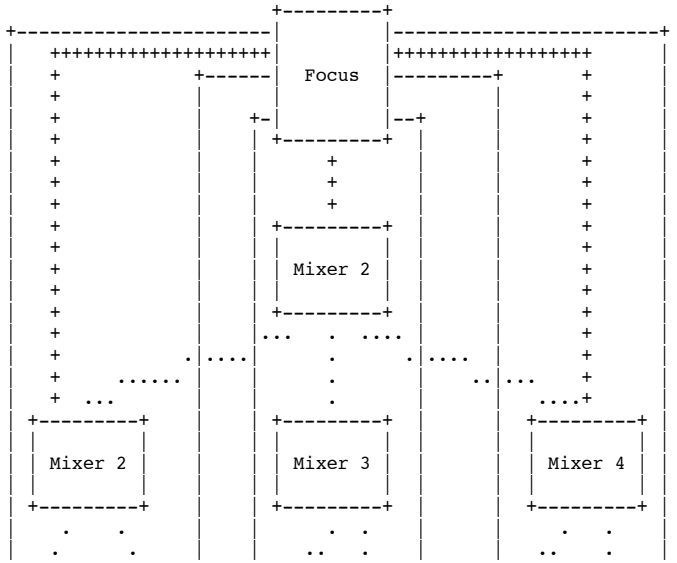
The central conference server also includes a conference policy server. Of course, the central conference server cannot implement any of the media policies directly. Rather, it would delegate the implementation to the conference policy servers co-resident with a participant. As an example, if a participant decides to switch the overall conference mode from "voice activated" to "continuous presence", they would communicate with the central conference policy

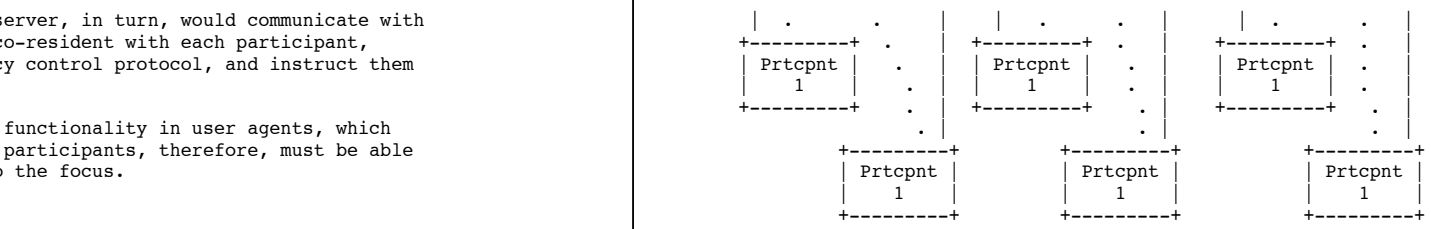
server. The conference policy server, in turn, would communicate with the conference policy servers co-resident with each participant, using the same conference policy control protocol, and instruct them to use "continuous presence".

This model requires additional functionality in user agents, which may or may not be present. The participants, therefore, must be able to advertise this capability to the focus.

6.5 Cascaded Mixers

In very large conferences, it may not be possible to have a single mixer that can handle all of the media. A solution to this is to use cascaded mixers. In this architecture, there is a centralized focus, but the mixing function is implemented by a multiplicity of mixers, scattered throughout the network. Each participant is connected to one, and only one of the mixers. The focus uses some kind of control protocol to connect the mixers together, so that all of the participants can hear each other.





------- SIP Dialog
..... Media Flow
+++++++ Control Protocol

Figure 9

This architecture is shown in Figure 9.

7. Security Considerations

Conferences frequently require security features in order to properly operate. The conference policy may dictate that only certain participants can join, or that certain participants can create new policies. Generally speaking, conference applications are very concerned about authorization decisions. Mechanisms for establishing and enforcing such authorization rules is a central concept throughout this document.

Of course, authorization rules require authentication. Normal SIP authentication mechanisms should suffice for the conference authorization mechanisms described here.

Privacy is an important aspect of conferencing. Users may wish to join a conference without anyone knowing that they have joined, in order to silently listen in. In other applications, a participant may wish just to hide their identity from other participants, but otherwise let them know of their presence. These functions need to be provided by the conferencing system.

8. Contributors

This document is the result of discussions amongst the conferencing design team. The members of this team include:

Alan Johnston
Brian Rosen
Rohan Mahy
Henning Schulzrinne
Orit Levin
Roni Even
Tom Taylor
Petri Koskelainen
Nermeen Ismail
Andy Zmolek
Joerg Ott
Dan Petrie

9. Changes from draft-ietf-sipping-conferencing-framework-00

Updated references and formatting cleanup.

10. Changes since draft-rosenberg-sipping-conferencing-framework-01

- Clarified that the conference notification service uses a single package with some kind of filtering to select whether you get the focus or policy state.

11. Changes since draft-rosenberg-sipping-conferencing-framework-00

- Rework of terminology.
- More details on moderating policy changes.
- Rework of the overview, and in particular, a shift of focus from basic/complex conferences (a term which has been removed) to conference aware/unaware participants.
- Removal of explicit reference to megaco for controlling a mixer.
- Discussion of a lot more conferencing operations.
- New sidebar mechanism.

Informative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

[3] Levin, O. and R. Even, "High Level Requirements for Tightly Coupled SIP Conferencing", draft-levin-sipping-conferencing-requirements-03 (work in progress), March 2003.

[4] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[5] Campbell, B., "Instant Message Sessions in SIMPLE", draft-ietf-simple-message-sessions-02 (work in progress), October 2003.

[6] Rosenberg, J., "A Framework for Application Interaction in the Session Initiation Protocol (SIP)", draft-ietf-sipping-app-interaction-framework-00 (work in progress), October 2003.

[7] Johnston, A. and O. Levin, "Session Initiation Protocol Call Control - Conferencing for User Agents", draft-ietf-sipping-cc-conferencing-01 (work in progress), July 2003.

[8] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[9] Rosenberg, J., "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", draft-ietf-sip-callee-caps-01 (work in progress), October 2003.

[10] Mahy, R. and D. Petrie, "The Session Inititation Protocol (SIP) 'Join' Header", draft-ietf-sip-join-02 (work in progress), July 2003.

[11] Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-02 (work in progress), July 2003.

[12] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[13] Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C. and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[14] Khartabil, H., Leppanen, E. and T. Moran, "Requirements for Presence Specific Event Notification Filtering", draft-ietf-simple-pres-filter-reqs-02 (work in progress), August 2003.

[15] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", draft-ietf-simple-presence-10 (work in progress), January 2003.

[16] Rosenberg, J., "A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)", draft-ietf-simple-winfo-package-05 (work in progress), January 2003.

[17] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[18] Rosenberg, J., Peterson, J., Schulzrinne, H. and G. Camarillo, "Best Current Practices for Third Party Call Control in the Session Initiation Protocol", draft-ietf-sipping-3pcc-04 (work in progress), July 2003.

Author's Address

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
EMail: jdrosen@dynamicsoft.com
URI: http://www.jdrosen.net

Intellectual Property Statement

Full Copyright Statement

HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING Working Group
Internet-Draft
Expires: October 22, 2003

O. Levin
RADVISION
R. Even
Polycom
April 23, 2003

High Level Requirements for Tightly Coupled SIP Conferencing
draft-ietf-sipping-conferencing-requirements-00

Status of this Memo

Copyright Notice

Abstract

This document examines a wide range of conferencing requirements for tightly coupled SIP conferences. Separate documents will map the requirements to existing protocol primitives, define new protocol extensions, and introduce new protocols as needed. Together, these documents will provide a guide for building interoperable SIP conferencing applications.

Table of Contents

1. Scope

This document examines a wide range of conferencing requirements for tightly coupled SIP (RFC 3261 [2]) conferencing.

The requirements are grouped by subjects in various areas allowing solutions to progress in parallel.

Separate documents will map the requirements to existing protocol primitives, define new protocol extensions, and introduce new protocols as needed.

Together, these documents will provide a guide for building interoperable SIP conferencing applications.

2. An Overview

A SIP conference is an association of SIP user agents (i.e. conference participants) with a central point (i.e. a conference focus) where the focus has direct peer-wise relationships with the participants by maintaining a separate SIP dialog with each.

The focus is a SIP user agent which has abilities to host SIP conferences including their creation, maintenance, and manipulation using SIP call control means and potentially other non-SIP means.

In this tightly coupled model, the SIP conference graph is always a star. The conference focus maintains the correlation among conference's dialogs internally.

The conference focus can be implemented either by a participant or by a separate application server.

In the first case, a focus is typically capable of hosting a simple ad-hoc conference only. We envision that such basic conference can be established using SIP call control primitives only.

A dedicated conference server, in addition to the basic features, offers richer functionality including simultaneous conferences, large scalable conferences, reserved conferences, and managed conferences. A conferencing server can support any subset of the advanced conferencing functions presented in this document.

The media graph of a SIP conference can be centralized, de-centralized, or any combination of both and potentially differ per media type. In centralized case, the media sessions are established between the focus and each one of the participants. In de-centralized (i.e. distributed) case, the media graph is a (multicast or

multi-unicast) mesh among the participants. Consequently, the media processing (e.g. mixing) can be performed either by the focus alone or by the participants.

Conference participants and third parties can have different roles and privileges in a certain conference. For example, conferencing policy can state that the rights to disconnect from and to invite to a conference are limited to the conference chair only.

Throughout the document, by conference policies we mean a set of parameters and rules (e.g. maximum number of participants, needs chair-person supervision or not, password protected or not, duration, a way of media mixing, etc.) that are defined at the onset of a conference. Typically, conference policies would be specified by a conference creator and need special privileges to be manipulated.

Throughout the document, by a conference state we mean a set of information describing the conference in progress. This includes participantsÆ information (such as dialog identifiers), media sessions in progress, the current loudest speaker, the current chair, etc.

3. High Level Requirements

In addition to the requirements presented in this document, supplementary requirements for conferencing policy, media mixing and other manipulations, floor control, privileges control, etc. will be discussed in separate documents.

3.1 Discovery Phase

Some of the requirements presented in this section can be met either by configuration means or by using proprietary conventions. Nevertheless, we feel that standard means for implementing these functions by automata MUST be defined.

REQ -1: Discovery of a location of an arbitrary SIP conferencing server(s).

EditorÆs Note: No solution currently exists.

REQ -2: Given a SIP AOR of a certain entity, resolution whether the SIP entity has focus capabilities.

EditorÆs Note: No solution currently exists.

REQ -3: Given a global identifier of a particular conference, locating the conference focus.

REQ -4: Given a global identifier of a particular conference, obtaining the conference properties.

REQ -5: Given a global identifier of a particular conference, obtaining the conference state information.

3.2 Conference Creation

Given a focus location, a means MUST be defined for an interested entity (including a user agent) to implement the procedures below:

REQ -1: Creation of an ad-hoc conference identifier and the conference with specified properties.

REQ -2: Creation of a reserved conference identifier for a conference with specified properties.

REQ -3: Specifying properties upon conference creation in any of the following ways: default, profiles and explicitly.

3.3 Conference Termination

REQ -1: Given a conference identifier, a means MUST be defined for a user agent to disconnect all participants from the conference and terminate the conference including the release of the associated resources.

REQ -2: A means MAY be defined for requesting a focus to revert a two-party conference to a basic SIP point-to-point session including the release of the associated conferencing resources.

3.4 ParticipantsÆ Manipulations

Some of the requirements presented in this section can be met by human intervention, configuration means, or by using proprietary conventions. Nevertheless, we feel that standard means for implementing these functions by automata MUST be defined.

3.4.1 Participation of a Conference-unaware User Agent

REQ -1: Focus MUST be able to invite and disconnect an RFC 3261 compliant only SIP user agent to and from a SIP conference.

REQ -2: RFC 3261 compliant only SIP user agent MUST be able to dial-in a particular SIP conference. In this case, only the human knows that he/she is connected to the conference.

3.4.2 Dial-Out Scenarios

REQ -1: A means MUST be defined for a focus to invite another user agent to one of the focusÆ conferences. This procedure MUST result in establishing of a single SIP dialog between the two.

REQ -2: Given an existent SIP dialog between two user agents, where at least one with focus capabilities, a means MUST be defined for the conference focus to invite the other user agent to one of the focusÆ conferences without additional SIP dialog establishment.

REQ -3: An invitation to a user agent to join a conference MUST include a standard indication that it is "a conference" and the conference identifier.

3.4.3 Dial-In Scenarios

REQ -1: A means MUST be defined for a user agent to create an ad-hoc conference with default properties (as per "Conference Creation" REQ -1 above) and to become its participant using a single SIP dialog.

REQ -2: Given a reserved conference identifier, a means MUST be defined for a user agent to activate the conference and to become its participant using a single SIP dialog.

REQ -3: Given a conference identifier of an active conference, a means MUST be defined for a user agent to dial-in the conference and to become its participant using a single SIP dialog between the two.

REQ -4: Given an identifier of one of the dialogs of a particular active conference, a means MUST be defined for a user agent to dial-in the conference and to become its participant.

3.4.4 Third Party Invitation to a Conference

REQ -1: Given a conference identifier, a means MUST be defined for a user agent to invite another user agent to this conference.

REQ -2: Given an identifier of one of the dialogs of a particular active conference, a means MUST be defined for a user agent to invite another user agent to this conference.

REQ -3: Given a conference identifier, a means SHOULD be defined for a user agent to invite a list of user agents to this conference (a so-called "mass invitation").

3.4.5 ParticipantsÆ Removal

REQ -1: A means MUST be defined for a conference focus to remove a conference participant from the conference.

REQ -2: Given a conference identifier, a means MUST be defined for a user agent to remove a participant from the conference.

REQ -3: Given an identifier of one of the dialogs of a particular active conference, a means MUST be defined for a user agent to remove a participant from the conference.

REQ -4: Given a conference identifier, a means MUST be defined for a user agent to remove all the participants from the conference.

REQ -5: Given a conference identifier and a sub-list of participants, a means MAY be defined for a user agent to remove the specified participants from the conference (a so-called "mass ejection").

3.4.6 ParticipantsÆ Privacy

A conference focus SHOULD support the procedures described in this section. A conference participant MAY support the procedures described in this section. The requirements imply that "anonymizing" operations MUST be performed on all: the call control, the media control and the media content when appropriate.

REQ -1: A conference participant joins the conference "anonymously", i.e. his/her presence can be announced but without disclosing his/her identity.

REQ -2: A conference participant requests a focus for anonymous participation in the conference.

REQ -3: A conference participant joins a conference in a "hidden mode", i.e. his/her both presence and identity are not to be disclosed to other participants.

REQ -4: A conference participant requests a focus for participation in the conference in a hidden mode.

3.5 Conference State Information

3.5.1 Description

By a conference state we mean a virtual database describing the conference in progress. This includes different conference aspects - participantsÆ information (such as dialog identifiers and state), media sessions in progress (such as current stream contributing sources and encoding schemes), the current loudest speaker, the

current chair, etc. Conference state is the latest conference snapshot triggered by changes in participantsÆ state, conference policy changes, etc.

REQ -1: Conference state virtual database MUST have a modular definition, i.e. it MUST be possible to access different conference aspects independently.

REQ -2: It MUST be possible to aggregate information relating to different conference aspects in a single report.

REQ -3: A mechanism for extensible definition and registration of conference state evolving aspects MUST be present.

REQ -4: A default conference state report MUST be defined. It SHOULD contain minimal useful to participants information (e.g. a list of current conference participants).

3.5.2 Dissemination of Changes

REQ -1: A means MUST be defined for reporting the conference state changes to interested parties (including non-conference participants) in a timely manner.

REQ -2: A means MUST be defined for a SIP user agent to express its interest in selected state changes only.

REQ -3: A means MUST be defined for a SIP user agent to express the minimum interval between receiving state change reports.

REQ -4: It MUST be possible to aggregate recent changes in a single reporting event.

REQ -5: Default conference state change reports MUST be defined. They SHOULD contain minimal useful to the participants information (e.g. participantsÆ joining and leaving the conference).

3.5.3 On-demand Information Dissemination

REQ -1: A means MUST be defined to disseminate any conference state information to interested parties (including SIP user agents) on-demand.

REQ -2: A means MUST be defined for an interested party (including SIP user agents) to request conference state information of a particular conference defined by the conference identifier.

REQ -3: A means MUST be defined for an interested party (including

SIP user agents) to specify the subset of the conference state information it wants and capable to receive.

3.6 Focus Role Migration

Editor's Note: We should decide whether the requirements below can be met by using SIP or non-SIP means.

REQ -1: A procedure for delegating a focus role by the current focus to another participant MUST be defined.

REQ -2: A procedure for requesting a conference focus to transfer its role to another participant MUST be defined.

REQ -3: A procedure for on-demand unconditional transfer of the focus role to a different participant MUST be defined.

REQ -4: A detection procedure for a focus failure condition MUST be defined.

3.7 Side-bar Conferences

A standard means MUST be defined in order to implement the operations defined in this section below.

REQ -1: A user agent (not a conference participant) joins a side-bar within the conference by SIP means.

REQ -2: A user agent (not a conference participant) is invited to a side-bar within the conference by SIP means.

REQ -3: A conference participant creates a side-bar conference with one or more participants in a conference by SIP means.

REQ -4: A conference participant joins a side-bar within the conference by SIP means.

REQ -5: A conference participant is invited to a side-bar within the conference by SIP means.

REQ -6: A conference-unaware user agent (a participant or not) creates and participates in side-bar conferences. It MAY be achieved by non-SIP means.

REQ -7: A conference participant creates side-bar conferences within the conference without establishing any additional SIP dialogs with the focus. It MAY be achieved by non-SIP means.

REQ -8: A conference participant joins any number of side-bars within the conference without establishing any additional SIP dialogs with the focus. It MAY be achieved by non-SIP means.

REQ -9: A conference participant is invited to any number of side-bars within the conference without establishing any additional SIP dialogs with the focus. It MAY be achieved by non-SIP means.

3.8 Cascading of Conferences

"Cascading of Conferences" is a term that has different meanings in different contexts. Some examples are listed below:

- Peer-to-peer chaining of signaling. (Many ways exist to build the media graph in this case.)
- Conferences have hierarchal signaling relations. (Many ways exists to build the media graph in this case.)
- "Cascading" is used to distribute the media "mixing" only. The distribution of signaling is not required.

As it can be seen from the examples, each will define a different set of requirements.

Editor's Note: We need to discuss which of the architectures require our attention as a part of the SIP conferencing force.

3.9 SIMPLE and SIP Conferencing Coordination

REQ -1: SIMPLE-based Presence and Instant Messaging architecture SHOULD fit into the general SIP Conferencing architecture.

REQ -2: A scenario where a multimedia SIP conference and a multiparty IM conversation take place among the same group of participants MUST be addressed.

REQ -3: A scenario where a side-bar or/and a sub-IM-conference is being held as a part of SIP conference MUST be addressed.

4. Security Considerations

Editor's Note: Will be provided in the next version of the document.

5. Contributors

This work is based on the discussions among the members of the SIP Conferencing design team.

Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

Informative References

Authors' Addresses

Orit Levin
RADVISION
266 Harristown Road
Glen Rock, NJ 75024

EMail: orit@radvision.com

Roni Even
Polycom
94 Derech Em Hamoshavot
Petach Tikva, Israel

EMail: roni.even@polycom.co.il

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING
Internet-Draft
Expires: August 14, 2004

D. Petrie
Pingtel Corp.

February 14, 2004

A Framework for SIP User Agent Profile Delivery
draft-ietf-sipping-config-framework-02.txt

Status of this Memo

Copyright Notice

Abstract

This document defines the application of a set of protocols for providing profile data to SIP user agents. The objective is to define a means for automatically providing profile data a user agent needs to be functional without user or administrative intervention. The framework for discovery, delivery, notification and updates of user agent profile data is defined here. As part of this framework a new SIP event package is defined here for the notification of profile changes. This framework is also intended to ease on going administration and upgrading of large scale deployments of SIP user agents. The contents and format of the profile data to be defined is outside the scope of this document.

Table of Contents

1. Motivation

Today all SIP user agent vendors use proprietary means of delivering user or device profiles to the user agent. The profile delivery framework defined in this document is intended to enable a first phase migration to a standard means of providing profiles to SIP user agents. It is expected that UA vendors will be able to use this framework as a means of delivering their existing proprietary user and device data profiles (i.e. using their existing proprietary binary or text formats). This in itself is a tremendous advantage in that a SIP environment can use a single profile delivery server for profile data to user agents from multiple vendors. Follow-on standardization activities can:

1. define a standard profile content format framework (e.g. XML with name spaces [??] or name-value pairs [RFC0822]).
2. specify the content (i.e. name the profile data parameters, xml schema, name spaces) of the data profiles.

One of the objectives of the framework described in this document is to provide a start up experience similar to that of users of an analog telephone. When you plug in an analog telephone it just works (assuming the line is live and the switch has been provisioned). There is no end user configuration required to make analog phone work (at least in a basic sense). So the objective here is to be able to take a new SIP user agent out of the box, plug it in (or install the software) and have it get its profiles without human intervention (other than security measures). This is necessary for cost effective deployment of large numbers of user agents.

Another objective is to provide a scalable means for on going administration of profiles. Administrators and users are likely to want to make changes to user and device profiles.

Additional requirements for the framework defined in this document are described in: [I-D.ietf-sipping-ua-prof-framewk-reqs], [I-D.sinnreich-sipdev-req]

2. Introduction

2.1 Requirements Terminology

Keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "MAY" that appear in this document are to be interpreted as described in RFC 2119[RFC2119].

2.2 Profile Delivery Framework Terminology

profile - data set specific to a user or device.
device - SIP user agent, either software or hardware appliance.
profile content server - The server that provides the content of the profiles using the protocol specified by the URL scheme.
notifier - The SIP user agent server which processes SUBSCRIBE requests for events and sends NOTIFY requests with profile data or URI(s) point to the data.
profile delivery server - The logical collection of the SIP notifier and the server which provides the contents of the profile URI(s).

2.3 Overview

The profile life cycle can be described by five functional steps. These steps are not necessarily discrete. However it is useful to describe these steps as logically distinct. These steps are named as follows:

Discovery - discover a profile delivery server
Enrollment - enroll with the profile delivery server
Profile Retrieval - retrieve profile data
Profile Change Notification - receive notification of profile changes
Profile Change Upload - upload profile data changes back to the profile delivery server

Discovery is the process by which a UA SHOULD find the address and port at which it SHOULD enroll with the profile delivery server. As there is no single discovery mechanism which will work in all network environments, a number of discovery mechanisms are defined with a prescribed order in which the UA SHOULD try them until one succeeds.

Enrollment is the process by which a UA SHOULD make itself known to the profile delivery server. In enrolling the UA MUST provide identity information, name requested profile type(s) and supported protocols for profile retrieval. It SHOULD also subscribe to a mechanism for notification of profile changes. As a result of enrollment, the UA receives the data or the URI for each of the profiles that the profile delivery server is able to provide. Each profile type (set) requires a separate enrollment or SUBSCRIBE session.

Profile Retrieval is the process of retrieving the content for each of the profiles the UA requested.

Profile Change Notification is the process by which the profile delivery server notifies the UA that the content of one or more of the profiles has changed. If the content is provided indirectly the

UA SHOULD retrieve the profile from the specified URI upon receipt of the change notification.

Profile Upload is the process by which a UA or other entity (e.g. OSS, corporate directory or configuration management server) pushes a change to the profile data back up to the profile delivery server.

This framework defines a new SIP event package [RFC3265] to solve enrollment and profile change notification steps.

The question arises as to why SIP should be used for the profile delivery framework. In this document SIP is used for only a small portion of the framework. Other existing protocols are more appropriate for transport of the profile contents (upstream and downstream of the user agent) and are suggested in this document. The discovery step is simply a specified order and application of existing protocols. SIP is only needed for the enrollment and change notification functionality of the profile delivery framework. In many SIP environments (e.g. carrier/subscriber and multi-site enterprise) firewall, NAT and IP addressing issues make it difficult to get messages between the profile delivery server and the user agent requiring the profiles.

With SIP the users and devices already are assigned globally routable addresses. In addition the firewall and NAT problems are already presumably solved in the environments in which SIP user agents are to be used. Therefore SIP is the best solution for allowing the user agent to enroll with the profile delivery server which may require traversal of multiple firewalls and NATs. For the same reason the notification of profile changes is best solved by SIP.

It is assumed that the content delivery server MUST be either in the public network or accessible through a DMZ. The user agents requiring profiles may be behind firewalls and NATs and many protocols, such as HTTP, may be used for profile content retrieval without special consideration in the firewalls and NATs.

A conscious separation of user and device profiles is made in this document. This is useful to provide features such as hoteling as well as securing or restricting user agent functionality. By maintaining this separation, a user may walk up to someone else's user agent and direct that user agent to get their profile data. In doing so the user agent can replace the previous user's profile data while still keeping the devices profile data that may be necessary for core functionality and communication described in this document.

3. Profile Change Event Notification Package

This section defines a new SIP event package [RFC3265]. The purpose of this event package is to send to subscribers notification of content changes to the profile(s) of interest and to provide the location of the profile(s) via content indirection [I-D.ietf-sip-content-indirect-mech] or directly in the body of the NOTIFY. If the profile is large enough to cause packet fragmentation over the transport protocol, the profile SHOULD use content indirection. The user agent SHOULD specify the profile delivery means and format via the MIME type in the Accepts header.

3.1 Event Package Name

The name of this package is "sip-profile". This value appears in the Event header field present in SUBSCRIBE and NOTIFY requests for this package as defined in [RFC3265].

3.2 Event Package Parameters

This package defines the following new parameters for the event header: profile-name, vendor, model, version, effective-by. The effective-by parameter is for use in NOTIFY requests only. The others are for use in the SUBSCRIBE request, but may be used in NOTIFY requests as well.

The profile-name parameter is used to indicate the token name of the profile type the user agent wishes to obtain URIs for or to explicitly specify the URI to which it is to be notified of change. Using a token in this parameter allows the URL semantics to be opaque to the subscribing user agent. All it needs to know is the token value for this parameter. However in some cases the user agent may know the URI of the profile and only wishes to know about changes to the profile. The user agent MAY supply the URI for the profile as the value of the profile-name parameter. This document defines two type categories of profiles and their token names. The contents or format of the profiles is outside the scope of this document. The two types of profiles define here are "user" and "device". Specifying device type profile(s) indicates the desire for the URI(s) and change notification of all profiles that are specific to the device or user agent. Specifying user type profile(s) indicates the desire for the URI(s) and change notification of all profile(s) that are specific to the user. The user or device is identified in the URI of the SUBSCRIBE request. The Accept header of the SUBSCRIBE request MUST include the MIME types for all profile content types that the subscribing user agent wishes to retrieve profiles or receive change notifications.

The user or device token in the profile-name parameter may represent a class or set of profiles as opposed to a single profile. As standards are defined for specific profile contents related to the user or device, it may be desirable to define additional tokens for the profile-name header. This is to allow a user agent to subscribe to that specific profile as opposed to the entire class or set of user or device profiles.

The rational for the separation of user and device type profiles is provided in section Section 2.3. It should be noted that either type may indicate that zero or more URIs are provided in the NOTIFY request. As discussed, a default user may be assigned to a device. In this scenario the profile delivery server may provide the URI(s) in the NOTIFY request for the default user when subscribing to the device profile type. Effectively the device profile type becomes a superset of the user profile type subscription. The user type is still useful in this scenario to allow the user agent to obtain profile data or URIs for a user other than the default user. This provides the ability to support a hoteling function where a user may "login" to any user agent and have it use a users profile(s).

The vendor, model and version parameters are tokens specified by the vendor of the user agent. These parameters are useful to the profile delivery server to effect the profiles provided. In some scenarios it is desirable to provide different profiles based upon these parameters. For example feature parameter X in a profile may work differently on two versions of user agent. This gives the profile deliver server the ability to compensate for or take advantage of the differences.

The "effective-by" parameter in the Event header of the NOTIFY specifies the maximum number of seconds before the user agent MUST make the new profile effective. A value of 0 (zero) indicates that the user agent MUST make the profiles effective immediately (despite possible service interruptions). This gives the profile delivery server the power to control when the profile is effective. This may be important to resolve an emergency problem or disable a user agent immediately.

SUBSCRIBE request example:
Event: sip-profile;profile-name=device;
vendor=acme;model=Z100;version=1.2.3

Event: sip-profile;profile-name=
"http://example.com/services/user-profiles/users/freds.xml";
vendor=premier;model=trs8000;version=5.5

NOTIFY request examples:
Event:sip-profile;effective-by=0

Event:sip-profile;effective-by=3600

3.3 SUBSCRIBE Bodies

This package defines no new use of the SUBSCRIBE request body.

3.4 Subscription Duration

As profiles are generally static with infrequent changes, it is recommended that default subscription duration be 86400 seconds (one day).

3.5 NOTIFY Bodies

The size of profile content is likely to be hundreds to several thousand bytes in size. Frequently even with very modest sized SDP bodies, SIP messages get fragmented causing problems for many user agents. For this reason the NOTIFY body MUST use content indirection [I-D.ietf-sip-content-indirect-mech] for providing the profiles if the Accept header of the SUBSCRIBE included the MIME type: message/external-body indicating support for content indirection.

When delivering profiles via content indirection the profile delivery server MUST include the Content-ID defined in [I-D.ietf-sip-content-indirect-mech] for each profile URL. This is to avoid unnecessary download of the profiles. Some user agents are not able to make a profile effective without rebooting or restarting. Rebooting is probably something to be avoided on a user agent performing services such as telephony. In this way the Content-ID allows the user agent to avoid unnecessary interruption of service as well. The Content-Type MUST be specified for each URI.

Initially it is expected that most user agent vendors will use a proprietary content type for the profiles retrieved from the URIs(s). It is hoped that over time a standard content type will

be specified that will be adopted by vendors of user agents. One direction that appears to be promising for this content is to use XML with name spaces [??] to segment the data into sets that the user agent implementer may choose to support based upon desired feature set. The specification of the content is out of the scope of this document.

Likewise the URL scheme used in the content indirection is outside the scope of this document. This document is agnostic to the URL schemes as the profile content may dictate what is required. It is expected that TFTP [RFC3617], FTP [??], HTTP [RFC2616], HTTPS [RFC2818], LDAP [RFC3377], XCAP [I-D.rosenberg-simple-xcap] and other URL schemes are supported by this package and framework.

3.6 Notifier processing of SUBSCRIBE requests

The general rules for processing SUBSCRIBE requests [RFC3265] apply to this package. The notifier does not need to authenticate the subscription as the profile content is not transported in the SUBSCRIBE or NOTIFY transaction messages. Only URLs are transported in the NOTIFY request which may be secured using the techniques in section Section 6.

The behavior of the profile delivery server is left to the implementer. The profile delivery server may be as simple as a SIP SUBSCRIBE UAS and NOTIFY UAC front end to a simple HTTP server delivering static files that are hand edited. At the other extreme the profile delivery server can be part of a configuration management system that integrates with a corporate directory and IT system or carrier OSS, where the profiles are automatically generated. The design of this framework intentionally provides the flexibility of implementation from simple/cheap to complex/expensive.

If the user or device is not known to the profile delivery server, the implementer MAY accept the subscription or reject it. It is recommended that the implementer accept the subscription. It is useful for the profile delivery server to maintain the subscription as an administrator may add the user or device to the system, defining the profile contents. This allows the profile delivery server to immediately send a NOTIFY request with the profile URIs. If the profile delivery server does not accept the subscription from an unknown user or device, the administer or user must manually provoke the user agent to reSUBSCRIBE. This may be difficult if the user agent and administrator are at different sites.

3.7 Notifier generation of NOTIFY requests

As in [RFC3265], the profile delivery server MUST always send a

NOTIFY request upon accepting a subscription. If the device or user is unknown to the profile delivery server and it chooses to accept the subscription, the implementer has two choices. A NOTIFY MAY be sent with no body or content indirection containing the profile URI(s). Alternatively a NOTIFY MAY be sent with URI(s) pointing to a default data set. Typically this data set allows for only limited functionality of the user agent (e.g. a phone user agent with data to call help desk and emergency services.). This is an implementation and business policy decision.

A user or device known and fully provisioned on the profile delivery server SHOULD send a NOTIFY with profile data or content indirection containing URIs for all of the profiles associated with the user or device (i.e. which ever specified in the profile-name parameter). The device may be associated with a default user. The URI(s) for this default user profiles MAY be included with the URI(s) of the device if the profile type specified is device.

A user agent can provide Hoteling by collecting a user?s AOR and credentials needed to SUBSCRIBE and retrieve the user profiles from the URI(s). Hoteling functionality is achieved by subscribing to the AOR and specifying the "user" profile type. This same mechanism can be used to secure a user agent, requiring a user to login to enable functionality beyond the default user?s restricted functionality.

The profile delivery server MAY specify when the new profiles MUST be made effective by the user agent. By default the user agent makes the profiles effective as soon as it thinks that it is non-obtrusive. However the profile delivery server MAY specify a maximum time in seconds (zero or more), in the effective-by event header parameter, by which the user agent MUST make the new profiles effective.

3.8 Subscriber processing of NOTIFY requests

The user agent subscribing to this event package MUST adhere to the NOTIFY request processing behavior specified in [RFC3265]. The user agent MUST make the profiles effective as specified in the NOTIFY request (see section Section 3.7). The user agent SHOULD use one of the techniques specified in section [RFC3265] to securely retrieve the profiles.

3.9 Handling of forked requests

This event package allows the creation of only one dialog as a result of an initial SUBSCRIBE request. The techniques to achieve this are described in section 4.4.9 of [RFC3265].

3.10 Rate of notifications

It is anticipated that the rate of change for user and device profiles will be very infrequent (i.e. days or weeks apart). For this reason no throttling or minimum period between NOTIFY requests is specified for this package.

3.11 State Agents

State agents are not applicable to this event package.

3.12 Examples

```
SUBSCRIBE sip:00df1e004cd0@example.com SIP/2.0
Event: sip-profile;profile-name=device;vendor=acme;
                        model=Z100;version=1.2.3
From: sip:00df1e004cd0@acme.com;tag=1234
To: sip:00df1e004cd0@acme.com;tag=abcd
Call-ID: 357385334292342210.1.1.44
CSeq: 2131 SUBSCRIBE
Contact: sip:00df1e004cd0@10.1.1.44
Content-Length: 0
```

```
NOTIFY sip:00df1e004cd0@10.1.1.44 SIP/2.0
Event: sip-profile;effective-by=3600
From: sip:00df1e004cd0@acme.com;tag=abcd
To: sip:00df1e004cd0@acme.com;tag=1234
Call-ID: 357385334292342210.1.1.44
CSeq: 321 NOTIFY
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=boundary42
Content-Length: ...

--boundary42
Content-Type: message/external-body;
    access-type="URL";
    expiration="Mon, 24 June 2002 09:00:00 GMT";
        URL="http://www.example.com/devices/fsmith";
        size=2222

Content-Type: application/z100-user-profile
Content-ID: <69ADF2E92@example.com>

--boundary42
Content-Type: message/external-body;
```



```
        access-type="URL";
        expiration="Mon, 24 June 2002 09:00:00 GMT";
            URL="http://www.example.com/devices/ff00000036c5";
            size=1234

Content-Type: application/z100-device-profile
Content-ID: <39EHF78SA@example.com>

--boundary42--
```

3.13 Use of URIs to Retrieve State

The profile type specified determines what goes in the user part of the SUBSRIBE URI. If the profile type requested is "device", the user part of the URI is an identity that MUST be unique across all user agents from all vendors. This identity must be static over time so that the profile delivery server can keep a specific device and its identity associated with its profiles. For Ethernet hardware type user agents supporting only a single user at a time this is most easily accomplished using its MAC address. Software based user agents running on general purpose hardware may also be able to use the MAC address for identity. However in situations where multiple instances of user agents are running on the same hardware it may be necessary to use a another scheme, such as using a unique serial number for each software user agent instance.

For example a device having a MAC address of 00df1e004cd0 might subscribe to the device profile URI: sip:00df1e004cd0@sipuaconfig.example.com. When subscribing to a user profile for user Fred S. the user agent would subscribe to the URI: sip:freds@sipuaconfig.example.com

If the profile type request is "user" the URI in the SUBSCRIBE request is the address of record for the user. This allows the user to specify (e.g. login) to the user agent by simply entering their known identity.

4. Profile Delivery Framework Details

The following describes how different functional steps of the profile delivery framework work. Also described here is how the event package defined in this document provides the enrollment and notification functions within the framework.

4.1 Discovery of Subscription URI

The discovery function is needed to bootstrap user agents to the point of knowing where to enroll with the profile delivery server.

Section Section 3.13 describes how to form the URI used to sent the SUBSCRIBE request for enrollment. However the bootstrapping problem for the user agent (out of the box) is what to use for the host and port in the URI. Due to the wide variation of environments in which the enrolling user agent may reside (e.g. behind residential router, enterprise LAN, ISP, dialup modem) and the limited control that the administrator of the profile delivery server (e.g. enterprise, service provider) may have over that environment, no single discovery mechanism works everywhere. Therefore a number of mechanisms SHOULD be tried in the specified order: SIP DHCP option [RFC3361], SIP DNS SRV [RFC3263], DNS A record and manual.

1. The first discovery mechanism that SHOULD be tried is to construct the SUBSCRIBE URI as described in Section 3.13 using the host and port of out bound proxy discovered by the SIP DHCP option as described in [RFC3361]. If the SIP DHCP option is not provided in the DHCP response, no SIP response or a SIP failure response other than for authorization is received for the SUBSCRIBE request to the sip-profile event, the next discovery mechanism SHOULD be tried.
2. The local IP network domain for the user agent, either configured or discovered via DHCP, should be used with the technique in [RFC3263] to obtain a host and port to use in the SUBSCRIBE URI. If no SIP response or a SIP failure response other than for authorization is received for the SUBSCRIBE request to the sip-profile event, the next discovery mechanism SHOULD be tried.
3. The fully qualified host name constructed using the host name "sipuaconfig" and concatenated with the local IP network domain should be tried next using the technique in [RFC3263] to obtain a host and port to use in the SUBSCRIBE URI. If no SIP response or a SIP failure response other than for authorization is received for the SUBSCRIBE request to the sip-profile event, the next discovery mechanism SHOULD be tried.
4. If all other discovery techniques fail, the user agent MUST provide a manual means for the user to enter the host and port used to construct the SUBSCRIBE URI.

Once a user agent has successfully discovered, enrolled, received a NOTIFY response with profile data or URI(s), the user agent SHOULD cache the SUBCRIBE URI to avoid having to rediscover the profile delivery server again in the future. The user agent SHOULD NOT cache the SUBSCRIBE URI until it receives a NOTIFY with profile data or URI(s). The reason for this is that a profile delivery server may send 202 responses to SUBSCRIBE requests and NOTIFY responses to unknown user agent (see section Section 3.6) with no URIs. Until the profile delivery server has sent a NOTIFY request with profile data or URI(s), it has not agreed to provide profiles.

To illustrate why the user agent should not cache the SUBSCRIBE URI until profile URI(s) are provided in the NOTIFY, consider the following example: a user agent running on a laptop plugged into a visited LAN in which a foreign profile delivery server is discovered. The profile delivery server never provides profile URIs in the NOTIFY request as it is not provisioned to accept the user agent. The user then takes the laptop to their enterprise LAN. If the user agent cached the SUBSCRIBE URI from the visited LAN (which did not provide profiles), the user agent would not attempt to discover the profile delivery server in the enterprise LAN which is provisioned to provide profiles to the user agent..

4.2 Enrollment with Profile Server

Enrollment is accomplished by subscribing to the event package described in section Section 3. The enrollment process is useful to the profile delivery server as it makes the server aware of user agent to which it may delivery profiles (those user agents the profile delivery server is provisioned to provide profiles to; those present that the server may be provide profiles in the future; and those that the server can automatically provide default profiles). It is an implementation choice and business policy as to whether the profile delivery server provides profiles to user agents that it is not provisioned to do so. However the profile server SHOULD accept (with 2xx response) SUBSCRIBE requests from any user agent.

4.3 Notification of Profile Changes

The NOTIFY request in the sip-profile event package serves two purposes. First it provides the user agent with a means to obtain the profile data or URI(s) for desired profiles without requiring the end user to manually enter them. It also provides the means for the profile delivery server to notify the user agent that the content of the profiles have changed and should be made effective.

4.4 Retrieval of Profile Data

The user agent retrieves it’s needed profile(s) via the URI(s) provide in the NOTIFY request as specified in section Section 3.5. The profile delivery server SHOULD secure the content of the profiles using one of the techniques described in Section 6. The user agent SHOULD make the new profiles effective in the timeframe described in section Section 3.2.

The contents of the profiles SHOULD be cached by the user agent. This it to avoid the situation where the content delivery server is not available, leaving the user agent non-functional.

4.5 Upload of Profile Changes

The user agent or other service MAY push changes up to the profile delivery server using the technique appropriate to the profile's URL scheme (e.g. HTTP PUT method, FTP put command). The technique for pushing incremental or atomic changes MUST be described by the specific profile data framework.

5. IANA Considerations

There are several IANA considerations associated with this specification.

5.1 SIP Event Package

This specification registers a new event package as defined in [RFC3265]. The following information required for this registration:

Package Name: sip-profile
Package or Template-Package: This is a package
Published Document: RFC XXXX (Note to RFC Editor: Please fill in XXXX with the RFC number of this specification).
Person to Contact: Daniel Petrie dpetrie@pingtel.com
New event header parameters: profile-name, vendor, model, version, effective-by

6. Security Considerations

Profiles may contain sensitive data such as user credentials. The protection of this data depends upon how the data is delivered. If the data is delivered in the NOTIFY body, SIP authentication MUST be used for SUBSRIPTION and SIPS and/or S/MIME MAY be use to encrypt the data. If the data is provided via content indirection, SIP authentication is not necessary for the SUBSCRIBE request. With content indirection the data is protected via the authentication, authorization and encryption mechanisms provided by the profile URL scheme. Use of the URL scheme security mechanisms via content indirection simplifies the security solution as the SIP event package does not need to authenticate, authorize or protect the contents of the SIP messages. Effectively the profile delivery server will provide profile URI(s) to anyone. The URLs themselves are protected via authentication, authorization and snooping (e.g. via HTTPS).

6.1 Symmetric Encryption of Profile Data

If the URL scheme used for content indirection does not provide a an authentication, authorization or encryption, a technique to provide this is to encrypted the profiles on the content delivery server using a symmetric encryption algorithm using a shared key. The

encrypted profiles are delivered by the content delivery server via the URIs provided in the NOTIFY requests. Using this technique the profile delivery server does not need to provide authentication or authorization for the retrieval as the profiles are obscured. The user agent must obtain the username and password from the user or other out of band means to generate the key and decrypt the profiles.

7. Differences from Simple XCAP Package

The author of this document had an action item from the July 2003 IETF SIPPING WG meeting to consider resolving the differences of the sip-profile and simple XCAP package [I-D.ietf-simple-xcap-package]. It is the author's opinion that XCAP [I-D.rosenberg-simple-xcap] can be supported by the framework and event package defined in this document and that this package provides a superset of the functionality in the XCAP package. The following lists the differences between the event packaged defined in this document vs. the one defined in [I-D.ietf-simple-xcap-package].

The simple XCAP package requires that the relative path be known and specified by the user agent when subscribing for change notification. The event package in this document requires a token or complete URI be known and specified when subscribing. The advantage of the token is that bootstrapping is easier and well defined. It also leaves the freedom of specifying and changing the entire path of the profile URL up to the profile delivery server.

The event package defined in this document allows multiple URIs to be provided in the NOTIFY request body as a result of a single token specified in the SUBSCRIBE event parameter: profile-name. This allows the profile delivery server to provide sets of profiles that the user agent may not have enough information to specify in the SUBSCRIBE URI (e.g. at boot strapping time the user agent may not know the user's identity, but the profile delivery server may know the default user for the device's identity) or the doc-component of the simple XCAP package.

All other functional differences between draft-ietf-sipping-config-framework-00 and draft-ietf-simple-xcap-package-00 are believed to be resolved in this version of this document.

8. Open Issues

9. Change History

9.1 Changes from draft-ietf-sipping-config-framework-01.txt

Changed the name of the profile-type event parameter to profile-name. Also allow the profile-name parameter to be either a token or or an explicit URI.

Allow content indirection to be optional. Clarified the use of the Accept header to indicate how the profile is to be delivered.

Added some content to the Iana section.

9.2 Changes from draft-ietf-sipping-config-framework-00.txt

This version of the document was entirely restructured and re-written from the previous version as it had been micro edited too much.

All of the aspects of defining the event package are now organized in one section and is believed to be complete and up to date with [RFC3265].

The URI used to subscribe to the event package is now either the user or device address or record.

The user agent information (vendor, model, MAC and serial number) are now provided as event header parameters.

Added a mechanism to force profile changes to be make effective by the user agent in a specified maximum period of time.

Changed the name of the event package from sip-config to sip-profile

Three high level securityapproaches are now specified.

9.3 Changes from draft-petrie-sipping-config-framework-00.txt

Changed name to reflect SIPPING work group item

Synchronized with changes to SIP DHCP [RFC3361], SIP [RFC3261] and [RFC3263], SIP Events [RFC3265] and content indirection [I-D.ietf-sip-content-indirect-mech]

Moved the device identity parameters from the From field parameters to User-Agent header parameters.

Many thanks to Rich Schaaf of Pingtel, Cullen Jennings of Cisco and Adam Roach of Dyamicsoft for the great comments and input.

9.4 Changes from draft-petrie-sip-config-framework-01.txt

Changed the name as this belongs in the SIPPING work group.

Minor edits

9.5 Changes from draft-petrie-sip-config-framework-00.txt

Many thanks to those who contributed and commented on the previous draft. Detailed comments were provided by Jonathan Rosenberg from Dynamicsoft, Henning Schulzrinne from Columbia U., Cullen Jennings from Cisco, Rohan Mahy from Cisco, Rich Schaaf from Pingtel.

Split the enrollment into a single SUBSCRIBE dialog for each profile. The 00 draft sent a single SUBSCRIBE listing all of the desired. These have been split so that each enrollment can be routed differently. As there is a concept of device specific and

user specific profiles, these may also be managed on separate servers. For instance in a roaming situation the device might get it's profile data from a local server which knows the LAN specific profile data. At the same time the user specific profiles might come from the user's home environment profile delivery server.

Removed the Config-Expires header as it is largely superfluous with the SUBSCRIBE Expires header.

Eliminated some of the complexity in the discovery mechanism.

Suggest caching information discovered about a profile delivery server to avoid an avalanche problem when a whole building full of devices powers up.

Added the User-Profile From header field parameter so that the device can a request a user specific profile for a user that is different from the device's default user.

References

[I-D.ietf-simple-xcap-package]
Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Modification Events for the Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Managed Documents", draft-ietf-simple-xcap-package-00 (work in progress), June 2003.

[I-D.ietf-sip-content-indirect-mech]
Olson, S., "A Mechanism for Content Indirection in Session

Initiation Protocol (SIP) Messages", draft-ietf-sip-content-indirect-mech-03 (work in progress), June 2003.

[I-D.ietf-sipping-ua-prof-framewk-reqs] Petrie, D. and C. Jennings, "Requirements for SIP User Agent Profile Delivery Framework", draft-ietf-sipping-ua-prof-framewk-reqs-00 (work in progress), March 2003.

[I-D.rosenberg-simple-xcap] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", draft-rosenberg-simple-xcap-00 (work in progress), May 2003.

[I-D.sinnreich-sipdev-req] Butcher, I., Lass, S., Petrie, D., Sinnreich, H. and C. Stredicke, "SIP Telephony Device Requirements, Configuration and Data", draft-sinnreich-sipdev-req-03 (work in progress), February 2004.

[RFC0822] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, March 1997.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[RFC3265] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[RFC3361] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers", RFC 3361, August 2002.

[RFC3377] Hodges, J. and R. Morgan, "Lightweight Directory Access Protocol (v3): Technical Specification", RFC 3377, September 2002.

[RFC3617] Lear, E., "Uniform Resource Identifier (URI) Scheme and Applicability Statement for the Trivial File Transfer Protocol (TFTP)", RFC 3617, October 2003.

Author's Address

Daniel Petrie
Pingtel Corp.
400 W. Cummings Park
Suite 2200
Woburn, MA 01801
US

Phone: "Dan Petrie (+1 781 938 5306)"<sip:dpetrie@pingtel.com>
EMail: dpetrie@pingtel.com
URI: http://www.pingtel.com/

Appendix A. Acknowledgments

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

The IETF has been notified of intellectual property rights claimed in regard to some or all of the specification contained in this document. For more information consult the online list of claimed rights.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING K. Ono
Internet-Draft S. Tachimoto
Expires: August 16, 2004 NTT Corporation
February 16, 2004

Requirements for End-to-middle Security for the Session Initiation Protocol (SIP)

draft-ietf-sipping-e2m-sec-reqs-01

Status of this Memo

Copyright Notice

Abstract

A SIP User Agent (UA) does not always trust all proxy servers in a request path to decide whether or not to inspect the message bodies and/or headers contained in a message. The UA might want to protect the message bodies and/or headers from proxy servers excluding the particular proxy that provides some services based on their content. This situation requires a mechanism for securing information passed between the UA and an intermediary proxy, also called "end-to-middle security", which does not interfere with end-to-end security. This document defines a set of requirements for a mechanism to achieve end-to-middle security.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [1].

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [2] supports hop-by-hop security using Transport Layer Security (TLS) [3] and end-to-end security using Secure MIME (S/MIME) [4]. This assumes that a SIP UA trusts all proxy servers in a request path to decide whether or not to inspect the message bodies contained in a message.

However, there is a model where trusted and partially-trusted proxy servers are mixed along a message path. The partially-trusted proxy servers are only trusted by users in terms of the SIP routing. The proxy servers are not trusted by users to inspect data except routing headers. Hop-by-hop confidentiality services using TLS are not suitable for this model. End-to-end confidentiality services using S/MIME are also not suitable when the intermediaries provide services based on reading the message bodies and/or headers. This problem is described in Section 23 of [2].

One example of such services is a firewall traversal. A firewall entity that supports the SIP protocol or a midcom [5] agent co-located with a proxy server controls a firewall based on certain Session Description Protocol (SDP) attributes in a SIP transaction.

Another example is transcoding [6]. A transcoder related to a proxy server transfers coding based on certain SDP attributes in a SIP transaction or transfers text-to-speech based on a message body in the MESSAGE [7] method.

A third example is the archiving of instant messaging traffic, where the archiving function co-located with a proxy server logs the message bodies in the MESSAGE method. This service might be deployed for financial or health care applications, where achiving communications is required by policies, as well as other applications.

In these cases, a UA might want to protect the message bodies and/or headers from proxy servers excluding the particular proxy server that provides these services. Conversely, a proxy server might want to view the message bodies and/or headers to provide these services. Such a proxy server is not always the first hop for the UA. These situations require security between the UA and the intermediary proxy server for the message bodies and/or message headers. We call this "end-to-middle security".

End-to-middle security consists of authentication, data integrity and data confidentiality. Above examples mainly require data confidentiality for end-to-middle security. For authentication, proxy servers usually require to authenticate a user that sends a request

message. The user also requires to authenticate the proxy that has the user's credential. HTTP digest authentication described in [2] can be used for mutual authentication for the request message. The authenticating proxy is not limited to the first hop for the UA. Thus, HTTP digest authentication can be used for end-to-middle security. To avoid replay attacks, the HTTP digest authentication needs to be used with a security mechanism for confidentiality such as TLS. HTTP digest authentication does not support authentication for an originator of a response message. Digital signatures obtained from a Public Key Infrastructure, S/MIME Cryptographic Message Syntax (CMS) [8] SignedData body, can be used for the authentication. Since these mechanisms achieve authentication for end-to-middle security, the requirements are not discussed in this document.

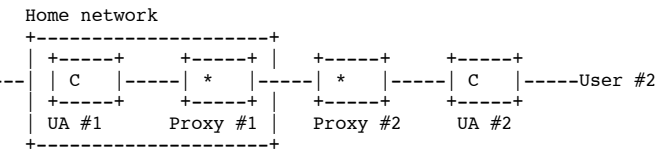
As for data integrity, proxy servers require to validate the content to be used for providing some services. The CMS SignedData body might be used in a mechanism for end-to-middle security. The CMS SignedData body can be created with the original data and the originator's private key, and anyone can verify the data integrity by using the originator's public key and the certificate. That is, proxy servers can verify the data integrity whenever they require. Thus, the CMS SignedData body could be used to implement end-to-middle security at the same time as using end-to-end security. Currently, proxy servers cannot require UAs to send a message with the CMS SignedData body. Some new mechanisms are needed to achieve data integrity for end-to-middle security.

This document mainly discusses requirements for data confidentiality and the integrity of end-to-middle security. Proposed mechanisms are discussed in [9].

2. Problems with the Existing Situations

We describe here examples of models in which trusted and partially trusted proxy servers both exist in a message path. These situations demonstrate the reasons why end-to-middle security are required in certain scenarios.

In the following example, User #1 does not know the services provided by or security policies of Proxy #1. User#1 sends an INVITE request including S/MIME-encrypted SDP for end-to-end security as shown in Figure 1. Proxy #1 may reject the request because it cannot offer a firewall traversal service. Or Proxy #1 may erase the encrypted data in the request based on a strict security policy that prohibits the forwarding of unknown data. Thus, the UA will need to discover if information requirements to receive intermediary's services or security policies will conflict with end-to-end confidentiality.

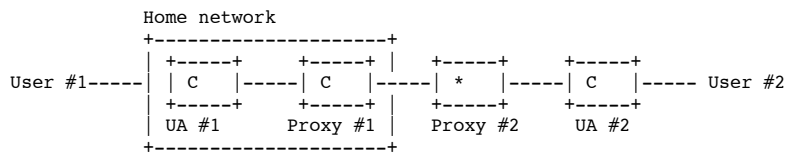


C: Content that UA #1 allows the entity to inspect
*: Content that UA #1 prevents the entity from inspecting

Figure 1: Deployment example #1

In the second example, Proxy server #1 (Proxy #1) is the home proxy server of User #1 using UA #1. User #1 communicates with User #2 through Proxy #1 and Proxy #2 as shown in Figure 2. UA #1 already knows the public key certificate of Proxy #1, and it allows Proxy #1 to inspect the message bodies in a request for some purpose. However, User #1 does not know whether Proxy #2 is trustworthy, and thus wants to protect the message bodies in the request. The UA will need to be able to grant a trusted intermediary permission to inspect message bodies while preserving their confidentiality with respect to other intermediaries.

Even if UA #1's request message authorizes a selected proxy server (Proxy #1) to see the message body, UA #1 is unable to authorize the same proxy server to see the message body in the response from UA #2. The originating UA will need to designate and share a key that can be reused as a content encryption key (CEK) for bidirectional exchanges of S/MIME-secured messages in SIP.

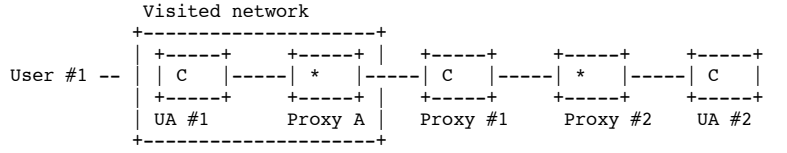


C: Content that UA #1 needs to disclose
*: Content that UA #1 needs to protect

Figure 2: Deployment example #2

In the third example, User #1 connects UA #1 to a proxy server in a Visited (potentially hostile) network, e.g. a hotspot service or a roaming service. Since User #1 wants to utilize certain home network services, UA #1 connects to a home proxy server, Proxy #1. However, UA #1 must connect to Proxy #1 via the proxy server of the visited network (Proxy A), because User#1 must follow the policy of that network. Proxy A may perform access control based on the destination addresses of calls. User #1 trusts Proxy A to route requests, but not to inspect the message bodies they contain as shown in Figure 3. User #1 trusts Proxy #1 both to route requests and to inspect the message bodies for some purpose.

The same problems as in the second example also exist here.



C: Content that UA #1 needs to disclose
*: Content that UA #1 needs to protect

Figure 3: Deployment example #3

3. Requirements for a Solution

We describe here requirements for a solution. The requirements are mainly applied during the phase of a dialog creation or sending a MESSAGE method.

3.1 General Requirements

Following are gerenal requirements for end-to-middle confidentiality and the integrity.

1. It SHOULD have little impact on the way a UA handles messages with S/MIME bodies.

2. It SHOULD have no impact on proxy servers that do not provide services based on S/MIME bodies in terms of handling the existing SIP headers.

3. It SHOULD have little impact on the standardized mechanism of proxy servers that provide services based on S/MIME bodies.

   When a proxy server receives an S/MIME message, it should be able to quickly and easily determine the necessity to investigate the S/MIME body. This can be restated as:

   + It SHOULD allow proxy servers to quickly and easily determine whether to handle S/MIME bodies and, if so, how and which ones.

4. It SHOULD allow a proxy server to notify a UA about the proxy server's security policy for a request/response.

5. It SHOULD allow a proxy server to notify a UA what data in a request/response is needed in order to provide a service.

3.2 Requirements for End-to-middle Confidentiality

1. The solution MUST be compatible with end-to-end encryption. The encrypted data can be shared with the end user and selected proxy server, if needed.

2. It MUST NOT violate end-to-end encryption when the encrypted data does not need to be shared with any proxy servers.

   For example, keying materials for secure RTP (SRTP) in SDP [11] can be included only in the end-to-end encryption, if the UA's policy is such.

3. It SHOULD allow a UA to discover which proxy server needs to view some data in a request/response message for a certain service, and discover what data is needed.

   This requirement is necessary when the UA does not know which proxy or domain provides the service in advance.

4. It MUST allow a UA to request selected proxy servers to view specific message bodies. The request itself SHOULD be secure.

5. It SHOULD allow a UA to request the recipient UA to disclose the same information that the requesting UA is providing to the proxy server to the same proxy server. The request itself SHOULD be secure.

   It is not reasonable to expect the recipient UA have knowledge of the public key certificate of the proxy server on the originating network. This can be restated as:

   + The solution SHOULD allow a UA to request the opposite-side UA to reuse a CEK in subsequent messages during a dialog.

   + It SHOULD allow a UA to request a selected proxy server to keep a CEK in a message during a dialog. The requests themselves SHOULD be secure.

6. It MAY allow a UA to notify the opposite-side UA which proxy server needs to view some data in a request/response for the services.

7. It MAY allow a UA to notify the opposite-side UA what data the proxy server is permitted to view in a request/response for the services.

   These last two requirements might be needed when there are a firewall in the network on UAS's side. A UAS need to notify a UAC to disclose the SDP in an INVITE message to a proxy server that control the firewall in the UAS side. Such notification might be applied to a registration phase.

3.3 Requirements for End-to-middle Integrity

1. It SHOULD work even with SIP end-to-end integrity service enabled.

2. It SHOULD allow a UA to discover what data in a request/response the proxy needs to verify in order to provide the service.

This requirement is necessary when the UA does not know what data is used to provide the service in advance.

3. It MUST allow a UA to request selected proxy servers to verify specific message bodies. The request itself SHOULD be secure.

4. It SHOULD allow a UA to request the recipient UA to send the verification data of the same information that the requesting UA is providing to the proxy server. The request itself SHOULD be secure.

5. It MAY allow a UA to notify the opposite-side UA what data the proxy server needs to verify in a request/response for the services.

4. Security Considerations

This documents present requirements including security viewpoints in Section 3.

5. IANA Considerations

This document requires no additional considerations.

6. Changes from 00.txt

- Reworked the sub-sections in Section 3 to clarify the objectives, separating end-to-middle confidentiality and integrity.

7. Acknowledgments

Thanks to Rohan Mahy and Cullen Jennings for their initial support of this concept, and to Jon Peterson, Gonzalo Camarillo, and Sean Olson for their helpful comments.

References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[3] Allen, C. and T. Dierks, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[4] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1992.

[5] Srisuresh, P., Kuthan, J., Rosenberg, J., Brim, S., Molitor, A. and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, August 2002.

[6] Camarillo, G., "Framework for Transcoding with the Session Initiation Protocol", draft-ietf-sipping-transc-framework-00.txt (work in progress), February 2004.

[7] Campbell, Ed., B., Rosenberg, J., Schulzrinne, H., Huitema, C. and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[8] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[9] Ono, K. and S. Tachimoto, "End-to-middle security in the Session Initiation Protocol(SIP)", draft-ono-sipping-end2middle-security-01 (work in progress), Feb. 2004.

[10] Baugher, M., Carrara, E., McGrew, D., Naslund, M., McGrew, D. and K. Norrman, "The Secure Real-time Transport Protocol", draft-ietf-avt-srtp-09.txt (work in progress), July 2003.

[11] Andreasen, F., Baugher, M. and D. Wing, "Session Description Protocol Security Descriptions for Media Streams", draft-ietf-mmusic-sdescriptions-03.txt (work in progress), February 2004.

Authors' Addresses

Kumiko Ono
Network Service Systems Laboratories
NTT Corporation
9-11, Midori-Cho 3-Chome
Musashino-shi, Tokyo 180-8585
Japan

EMail: ono.kumiko@lab.ntt.co.jp

Shinya Tachimoto
Network Service Systems Laboratories
NTT Corporation
9-11, Midori-Cho 3-Chome
Musashino-shi, Tokyo 180-8585
Japan

EMail: tachimoto.shinya@lab.ntt.co.jp

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the
Internet Society.

SIPPING
Internet-Draft
Expires: August 2, 2004

E. Burger
SnowShore Networks, Inc.
M. Dolly
AT&T Labs
February 2, 2004

Keypad Stimulus Protocol (KPML)
draft-ietf-sipping-kpml-02

Status of this Memo

Copyright Notice

Abstract

The Key Press Stimulus Protocol uses the SIP SUBSCRIBE/NOTIFY mechanism and Keypad Markup Language (KPML) to provide instructions to SIP User Agents for the reporting of user key presses.

Conventions used in this document

RFC2119 [1] provides the interpretations for the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" found in this document.

In the narrative discussion, the "user device" is a User Agent that

will report stimulus. it could be, for example, a SIP phone, edge media processor, or media gateway. An "application" is a User Agent requesting the user device to report stimulus. The "user" is an entity that stimulates the user device. In English, the user device is a phone, the application is an application server or proxy server, and the user presses keys to generate stimulus.

Table of Contents

1. Introduction

This document describes the Key Press Stimulus Protocol. The Key Press Stimulus Protocol exchanges messages using the SUBSCIBE and NOTIFY methods of SIP [2] with message bodies formed from the Keypad Markup Language, KPML. KPML is a markup [7] that enables "dumb phones" to report user key-press events. Colloquially, this mechanism provides for "digit reporting" or "DTMF reporting."

A goal of KPML is to fit in an extremely small memory and processing footprint. Note KPML has a corresponding lack of functionality. For those applications that require more functionality, please refer to VoiceXML [8] and MSCML [9].

We strongly discourage the use of non-validating XML parsers, as one can expect problems with future versions of KPML.

The name of the markup, KPML, reflects its legacy support role. The public switched telephony network (PSTN) accomplished end-to-end signaling by transporting Dual-Tone, Multi-Frequency (DTMF) tones in the bearer channel. This is in-band signaling.

From the point of view of an application being signaled, what is important is the fact the stimulus occurred, not the tones used to transport the stimulus. For example, an application may ask the caller to press the "1" key. What the application cares about is the key press, not that there were two cosine waves of 697 Hz and 1209 Hz transmitted.

A SIP-signaled [3] network transports end-to-end signaling with RFC2833 [10] packets. In RFC2833, the signaling application inserts RFC2833 named signal packets as well as or instead of generating tones in the media path. The receiving application gets the signal information, which is what it wanted in the first place.

RFC2833 correlates the time the end user pressed a digit with the user's media. However, out-of-band signaling methods, as are appropriate for user device to application signaling, do not need millisecond accuracy. On the other hand, they do need reliability, which RFC2833 does not provide.

An interested application could request notifications of every key press. However, many of the use cases for such signaling has the application interested in only one or a few keystrokes. Thus we need a mechanism for specifying to the user device what stimulus the application would like notification of.

2. Key Press Stimulus Protocol

2.1 Model

There are two usage models for the protocol. Functionally, they are both equivalent. However, it is useful to understand the use cases.

The first model is that of a SIP User Agent (UA) that directly interacts, on a given dialog, with the end device. Figure 1 shows a two-party SIP dialog. In this scenario, the SIP UA requests the End Point to report on key press events that would normally emanate from End Point port B. This could represent, for example, a toll by-pass scenario where the End Point is an ingress gateway and the SIP UA is an egress gateway.

In this case, the requesting User Agent requests digit notification on the same dialog established for the call, between SIP ports A and X.

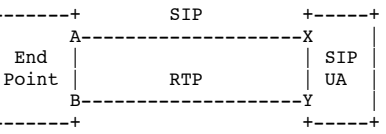


Figure 1: Endpoint Model

The second model is that of a third-party application that is interested in entered key presses. Figure 2 shows an established two-party SIP dialog between the End Point and the SIP UA. The requesting application addresses the particular media stream by referencing the established dialog identifier referring to the dialog between SIP ports A and X.

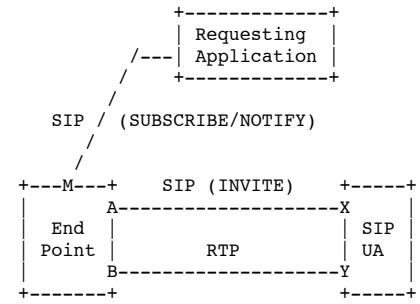


Figure 2: Third-Party Model

The third model is that of a media proxy. A media proxy is a media relay in the terminology of RFC1889 [11]. However, in addition to the RTP forwarding capability of a RFC1889 media relay, the media proxy can also do light media processing, such as tone detection, tone transcoding (tones to RFC2833 [10], and so on.

The Requesting Application uses dialog identifiers to identify the stream to monitor. The default is to monitor the media entering the End Point. For example, if the Requesting Application in Figure 3 uses the dialog represented by SIP ports V-C, then the media coming from SIP UAa RTP port W gets monitored. Likewise, the dialog represented by A-X directs the End Point to monitor the media coming from SIP UAb RTP Port Y.

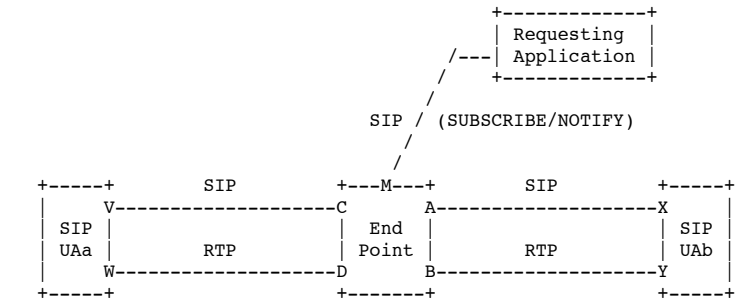


Figure 3: Media Proxy Model

2.2 Monitoring Leg

The default leg to monitor is the leg represented by the local tag of the SIP dialog at the monitoring End Point. A requesting application MAY request monitoring of the leg represented by the remote tag of the SIP dialog at the monitoring End Point.

Not all End Point devices are able to monitor the remote media stream. However, the End Point MUST be able to report on local (End Point-generated) key press events.

If the requesting application wishes to monitor both legs at a given End Point, the application will establish two subscriptions, one for each leg.

Section 4.1.5 describes how to specify to the End Point which leg of the dialog to monitor.

2.3 Operation

The key press stimulus protocol uses explicit subscription requests and notification requests, using the semantics of SUBSCRIBE/NOTIFY [2].

Following the semantics of SUBSCRIBE, if the user device receives a second subscription on the same dialog, the user device MUST terminate the existing KPML request (if any) and replace it with the new request.

An application may register multiple digit patterns in a single KPML

request.

If the user device supports multiple, simultaneous KPML requests, the application registers the separate requests either in a new SUBSCRIBE-initiated dialog or on an existing SUBSCRIBE-initiated dialog with a new event id tag.

If the user device does not support multiple, simultaneous KPML requests, it responds with an error response code. See Section 4.1.6 for more information.

A KPML request can be persistent or one-shot. Persistent requests are active until either the dialog terminates, including normal subscription expiration, the client replaces them, the client deletes them by sending a null document on the dialog, or the client deletes the subscription by sending a SUBCRIBE with an expires of zero (0).

Standard SUBSCRIBE processing dictates the end point sends a NOTIFY response if it receives a SUBSCRIBE with an expires of zero.

One-shot requests terminate themselves once a match occurs. The "persist" KPML element specifies whether the subscription remains registered for the duration specified in the SUBSCRIBE message or if it automatically terminates after a pattern matches.

KPML requests route to the user device using standard SIP request routing. A KPML request identifies the leg in question in one of two ways. The first method is to send the request on an existing, INVITE-initiated dialog. The second method is to explicitly identify the call leg by its dialog identifiers.

Response messages are KPML documents (messages). If the user device matched a digit map, the response indicates the digits detected and whether the user device suppressed digits. If the user device had an error, such as a timeout, it will indicate that instead.

3. Protocol Machinery

The Key Press Stimulus Protocol uses the SIP [3]SUBSCRIBE/NOTIFY [2] mechanism.

The registration of a digit map is simply setting a digit event notification filter. When the device detects the digits, it sends an event notification to the application.

The following sub-sections are the formal specification of the KPML SIP-specific event notification package.

3.1 Event Package Name

The name for the Key Press Stimulus Protocol package is "kpml".

3.2 Event Package Parameters

The "leg" parameter identifies the call leg being monitored.

If the "leg" parameter is not present, the SUBSCRIBE MUST be on an established INVITE-initiated SIP dialog. In this case, the leg the end device monitors is the call leg associated with the established dialog. If there is no corresponding dialog or call leg, the end device will send a 481 result code in a KPML notification.

NOTE: The SUBSCRIBE may succeed, resulting in a SIP 200 OK. However, the "current state" will be the KPML 481 result, and the subscription state will be "terminated."

SIP identifies call legs by their dialog identifier. The dialog identifier is the remote-tag, local-tag, and Call-ID entities.

To identify a specific dialog, all three of these parameters MUST be present. Usually, the local-tag is the To: entity with the To tag, the remote-tag is the From: entity including tag, and the call-id matches the Call-ID. Although semantically different, the important entities are the To: and From: tags.

Note there may be ambiguity in specifying only the SIP dialog to monitor. The dialog may specify multiple SDP streams that could carry key press events. For example, a dialog may have multiple audio streams. Wherever possible, the End Point MAY apply local policy to disambiguate which stream or streams to monitor. In order to have an extensible mechanism for identifying streams, the mechanism for specifying streams is as an element content to the <stream> tag. The only content defined today is the <reverse/> tag.

For most situations, such as a monaural point-to-point call with a single codec, the stream to monitor is obvious. In such situations the Application need not specify which stream to monitor.

The BNF for these parameters is as follows. The definitions of callid, token, EQUAL, SWS, and DQUOTE are from RFC3261 [3].

```
call-id   =  "call-id" EQUAL DQUOTE callid DQUOTE
from-tag  =  "from-tag" EQUAL token
to-tag    =  "to-tag" EQUAL token
```

The call-id parameter is a quoted string. This is because the BNF for word (which is used by callid) allows for characters not allowed

within token. One usually just copies these elements from the Call-Id, to, and from fields of the SIP INVITE.

One can use any method of determining the dialog identifier. One method available, particularly for third-party applications, is the SIP Dialog Package [12].

3.3 SUBSCRIBE Bodies

Key press filtering requests use KPML, as described in Section 4.1. The MIME type for KPML is application/kpml+xml.

Because of the potentially sensitive nature of the information reported by KPML, subscribers SHOULD use sips: and SHOULD consider the use of S/MIME on the content.

Subscribers MUST be prepared for the notifier to insist on authentication at a minimum and encryption as a likelihood.

3.4 Subscription Duration

The subscription lifetime should be longer than the expected call time. The default subscription lifetime (Expires value) MUST be 7200 seconds. This two-hour subscription time is entirely arbitrary. Please contact the editor if you have a better suggestion, and why.

Subscribers MUST be able to handle the end device returning an Expires value smaller than the requested value. Per RFC3265 [2], the subscription duration is the value returned by the end device in the 200 OK response Expires entity.

3.5 NOTIFY Bodies

The key press notification uses KPML, as described in Section 4.2. The MIME type for KPML is application/kpml+xml. The default MIME type for the kpml event package is application/kpml+xml.

If the requestor is not using a secure transport protocol such as TLS (e.g., by using a sips: URI), the end device SHOULD use S/MIME to protect the user information in responses.

3.6 Notifier Processing of SUBSCRIBE Requests

The user information transported by KPML is potentially sensitive. For example, it could include calling card or credit card numbers. Thus the first action of the end device (notifier) SHOULD be to authenticate the requesting party.

End devices MUST support digest authentication at a minimum.

End devices MUST support the sips: scheme and TLS.

Upon authenticating the requesting party, the end device determines if the requesting party has authorization to monitor the user's key presses. Determining authorization policies and procedures is beyond the scope of this specification.

NOTE: While it would be good to require both authorization and user notification for KPML, some uses, such as lawful intercept pen registers, have very strict authorization requirements yet have a requirement of no user notification. Conversely, pre-paid applications running on a private network may have no authorization requirements and already have implicit user acceptance of key press monitoring. Thus we cannot give any guidelines here.

After authorizing the request (RECOMMENDED), the end device checks to see if the request is to terminate a subscription. If the request will terminate the subscription, the end device does the appropriate processing, including the procedures described in Section 3.7.4.

If the request has no KPML body, than any KPML document running on that dialog, and addressed by the event id, if present, immediately terminates. This is a mechanism for unloading a KPML document while keeping the SUBSCRIBE-initiated dialog active. This can be important for secure sessions that have high costs for session establishment, such as TLS. The end device follows the procedures described in Section 3.7.1.

If the SUBSCRIBE request arrived on an INVITE-initiated dialog, and there is no "leg" parameter to the kpml subscription, then the KPML document acts upon the call legs created by the INVITE-initiated dialog.

If the SUBSCRIBE request has a "leg" parameter to the kpml subscription, then the KPML document acts upon the call leg referred to by the "leg" parameter. If appropriate, the end device SHOULD validate the requestor has authorization to monitor a given leg.

If the SUBSCRIBE request has a "leg" parameter to the kpml subscription, but the referenced leg does not exist, the end device follows the procedures in Section 3.7.5 Note the end device MUST issue a 200 OK before issuing the NOTIFY, as the SUBSCRIBE itself is well-formed.

If the request has a KPML body, the end device parses the KPML document. The end device SHOULD validate the XML document against

the schema presented in Section 6. If the document is not valid, the end device performs the procedures described in Section 3.7.6. If there is a loaded KPML document on the dialog (and given event id, if present), the end device unloads the document.

If the KPML document is valid, and the end device is capable of performing the monitoring, the end device performs the filtering specified by the KPML document. See Section 4 for the specification of KPML.

3.7 Notifier Generation of NOTIFY Requests

3.7.1 SIP Protocol-Generated

The end device (notifier in SUBSCRIBE/NOTIFY parlance) generates NOTIFY requests based on the requirements of RFC3265 [2]. Specifically, unless a SUBSCRIBE request is not valid, all SUBSCRIBE requests will result in an immediate NOTIFY.

The KPML payload distinguishes between a NOTIFY that RFC3265 mandates and a NOTIFY informing of key presses. If there are no digits quarantined at the time of the SUBSCRIBE (see Section 4.1 below) or the quarantined digits do not match the new KPML document, then the immediate NOTIFY MUST NOT contain a KPML body. If end device has digits quarantined that result in a digit match using the new KPML document, then the NOTIFY MUST return the appropriate KPML document.

3.7.2 Match

During the subscription lifetime, the end device may detect a key press stimulus that triggers a KPML event. In this case, the end device (notifier) MUST return the appropriate KPML document.

3.7.3 Inter-Digit Timeout No Match

Once a user starts to enter digits, it is highly likely they will enter all of the digits of interest within a specific time period. There is a temporal locality of reference for key presses. It is possible for users to accidentally press a key, however. Moreover, users may start pressing a key and then be lost as to what to do next. For applications to handle this situation, KPML allows applications to request notification if the user starts to enter digits but then stops before a digit map matches.

Once the end point detects a key press that matches the first character of a digit map, the end point starts the interdigit timer specified in the <pattern> tag. Every subsequent key press detected restarts the interdigit timer. If the interdigit timer expires, the

end point generates a KPML report with the KPML status code 423, Timer Expired. The report also includes the digits collected up to the time the timer expired. This could be the null string. After sending the NOTIFY, the end point will resume quarantining additional detected digits.

Applications may have different requirements for the interdigit timer. For example, applications targeted to user populations that tend to key in information slowly may require longer interdigit timers. The specification of the interdigit timer is in milliseconds. The default value is 4000, for 4 seconds. A value of zero indicates disabling the interdigit timer. The End Device MUST round up the requested interdigit timer to the nearest time increment it is capable of detecting.

3.7.4 Dialog Terminated

It is possible for a dialog to terminate during key press collection. The cases enumerated here are explicit SUBSCRIPTION termination, automatic SUBSCRIPTION termination, and underlying (INVITE-initiated) dialog termination.

If a SUBSCRIBE request has an expires of zero (explicit SUBSCRIBE termination), includes a KPML request, and there are quarantined digits, then the end device attempts to process the quarantined digits against the document. If there is a match, the end device generates the appropriate KPML report with the KPML status code of 200. The SIP NOTIFY body terminates the subscription by setting the subscription state to "terminated" and a reason of "timeout". If the subscription was on a SUBSCRIBE-initiated dialog, and there are no more active event id's associated with the dialog, then the end point MUST consider the dialog terminated. If the subscription was on an INVITE-initiated dialog, then the end point MAY release KPML-specific resources related to the dialog, but it MUST NOT alter the state of the INVITE-initiated dialog.

If the requesting party issues a SUBSCRIBE with an expires of zero and no KPML body or the expires timer on the SUBSCRIBE-initiated dialog fires at the end device (notifier), then the end device issues a KPML report with the KPML status code 487, Subscription Expired. The report also includes the digits collected up to the time the expires timer expired or when the subscription with expires equal to zero was processed. This could be the null string. Also, note that the digits in this case cannot match a digit map. If they did, the end device would have generated a KPML match report if they did.

Again, per the mechanisms of RFC3265 [2], the end device will terminate the SIP SUBSCRIBE dialog. The end device does this via the

SIP NOTIFY body transporting the final report described in the preceding paragraph. In particular, the subscription state will be "terminated" and a reason of "timeout". If the subscription was on a SUBSCRIBE-initiated dialog, then the end point MUST consider the dialog terminated. If the subscription was on an INVITE-initiated dialog, then the end point MAY release KPML-specific resources related to the dialog, but it MUST NOT alter the state of the INVITE-initiated dialog.

3.7.5 No Call Leg

If a SUBSCRIBE request references a dialog that is not present at the endpoint, usually by specifying a dialog identifier through the leg parameter to the kpml event package, the end point generates a KPML report with the KPML status code 481, Dialog Not Found. The end device terminates the subscription by setting the subscription state to "terminated". If the subscription was on a SUBSCRIBE-initiated dialog, and there are no more active event id's associated with the dialog, then the end point MUST consider the dialog terminated. If the subscription was on an INVITE-initiated dialog, then the end point MAY release KPML-specific resources related to the dialog, but it MUST NOT alter the state of the INVITE-initiated dialog.

IMPORTANT: The end device can invoke this procedure if the dialog underlying a subscription terminates. For example, a SUBSCRIBE-initiated dialog subscribes to the state of a different dialog (call) via the leg kpml parameter. That different call may terminate before the SUBSCRIBE-initiated dialog terminates. In this case, the end device MUST terminate the SUBSCRIBE-initiated dialog. This ensures reauthorization (if necessary) for attaching to subsequent call legs.

3.7.6 Bad Document

If the KPML document is not valid, the end device generates a KPML report with the KPML status code 501, Bad Document. The end device terminates the subscription by setting the subscription state to "terminated". If the subscription was on a SUBSCRIBE-initiated dialog, and there are no more active event id's associated with the dialog, then the end point MUST consider the dialog terminated. If the subscription was on an INVITE-initiated dialog, then the end point MAY release KPML-specific resources related to the dialog, but it MUST NOT alter the state of the INVITE-initiated dialog.

3.7.7 One-Shot vs. Persistent Requests

A one-shot kpml subscription is one that the KPML document does not mark as persistent. If the end device detects a key press stimulus

that triggers a one-shot KPML event, then the end device (notifier) MUST set the "Subscription-State" in the NOTIFY message to "terminated". At this point the end device MUST consider the subscription destroyed. The end device MUST quarantine digits per the controls specified in Section 4.1.

For persistent kpml subscriptions, the KPML document remains active for the lifetime of the subscription.

3.8 Subscriber Processing of NOTIFY Requests

3.8.1 No KPML Body

If there is no KPML body, it means the SUBSCRIBE was successful. This establishes the dialog if there are no quarantined digits to report.

3.8.2 KPML Body

If there is a KPML document, and the KPML status code is 200, then a match occurred.

If there is a KPML document, and the KPML status code is 4xx, then an error occurred with digit collection. The most likely cause is a timeout condition.

If there is a KPML document, and the KPML status code is 5xx, then an error occurred with the subscription. See Section 7 for more on the meaning of error codes.

The subscriber MUST be mindful of the subscription state. The end device may terminate the subscription at any time.

3.9 Handling of Forked Requests

The SUBSCRIBE behavior described in Section 3.6 ensures that it is only possible to have a subscription where there is an active (e.g., voice) dialog. Thus the case of multiple subscription installation cannot occur.

3.10 Rate of Notifications

The end device MUST NOT generate messages faster than one message every 40 milliseconds. This is the minimum time period for MF digit spills. Even 30 millisecond DTMF, as one sometimes finds in Japan, has a 20 millisecond off-time, resulting in a 50 millisecond interdigit time. This document strongly RECOMMENDS AGAINST using KPML for digit-by-digit messaging, such as would be the case if the

only <regex> is "x".

Because there is no meaningful metric for throttling requests. In addition, the end device MUST reliably deliver notifications. Thus the end device SHOULD send NOTIFY messages over a congestion-controlled transport, such as TCP or SCTP.

End devices MUST at a minimum implement SIP over TCP.

3.11 State Agents

Not applicable.

4. Message Format - KPML

The Key Press Stimulus Protocol exchanges KPML messages. There are two, mutually exclusive elements to KPML: the request and response.

4.1 KPML Request

A KPML request document (message) contains a <request> entity containing a <pattern> tag with a series of <regex> tags. The <regex> element specifies a digit pattern for the device to report on. Section 5 describes the DRegex, or digit regular expression, language.

Some devices can buffer entered digits. Subsequent KPML requests first apply their patterns against the buffered digits. Some applications use modal interfaces where the first few key presses determine what the following digits mean. For a novice user, the application may play a prompt describing what mode the application is in. However, "power users" often barge through the prompt.

The protocol provides a <flush> tag in the <pattern> element. The default is not to flush digits. Flushing digits means the user device flushes any buffered digits. This has the effect of ignoring digits entered before the KPML request. To flush digits, the KPML includes <flush>yes</flush>.

The End Device MUST be able to receive <flush>no</flush>. This directive is effectively a no-op.

Other string values for <flush> may be defined in the future. If the End Device receives a string it does not understand, it MUST treat the string as a no-op.

If the user presses a key not matched by the <regex> tags, the user device MUST discard the key press from consideration against the

current or future KPML messages. However, as described above, once there is a match, the user device quarantines any key presses the user entered subsequent to the match.

NOTE: This behavior allows for applications to only receive digits that interest them. For example, a pre-paid application only wishes to monitor for a long pound. If the user enters other digits, presumably for other systems, the pre-paid applicationd does not want notification of those digits. This feature is fundamentally different than the behavior of every system receiving every digit that TDM-based equipment provides.

The end device MAY support an inter-digit timeout value. This is the amount of time the end device will wait for user input before returning a timeout error result on a partially matched pattern. The application can specify the inter-digit timeout as an integer number of milliseconds by using the interdigittimer attribute to the <pattern> tag. The default is 4000 milliseconds. If the end device does not support the specification of an inter-digit timeout, the end device MUST silently ignore the specification. If the end device supports the specification of an inter-digit timeout, but not to the granularity specified by the value presented, the end device MUST round up the requested value to the closest value it can support.

KPML messages are independent. Thus it is not possible for the current document to know if a following document will enable barging or want the digits flushed. Therefore, the user device MUST quarantine all digits detected between the time of the report and the interpretation of the next script, if any. If the next script indicates a buffer flush, then the interpreter MUST flush all collected digits from consideration from KPML documents received on that dialog with the given event id. If the next script does not indicate flushing the quarantine digits, then the interpreter MUST apply the collected digits (if possible) against the digit maps presented by the script's <regex> tags. If there is a match, the interpreter MUST follow the procedures in Section 3.7.2 If there is no match, the interpreter MUST flush all of the collected digits.

Unless there is a suppress indicator in the digit map, it is not possible to know if the signaled digits are for local KPML processing or for other recipients of the media stream. Thus, in the absence of a digit suppression indicator, the user device transmits the digits to the far end in real time, using either RFC2833, generating the appropriate tones, or both.

The section Digit Suppression (Section 4.1.2) describes the operation of the suppress indicator.

4.1.1 Pattern Matching

4.1.1.1 Inter-Digit Timing

The pattern matching logic works as follows. KPML endpoints MUST follow the logic presented in this section so that multiple implementations will perform deterministically on the same KPML document given the same key press input.

The pattern match algorithm matches the longest regular expression. This is the same mode as H.248.1 [13] and not the mode presented by MGCP [14]. The pattern match algorithm choice has an impact on determining when a pattern matches. Consider the following KPML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
      version="1.0">
 <request>
  <pattern>
   <regex>0</regex>
   <regex>011</regex>
  </pattern>
 </request>
</kpml>
```

Figure 5: Greedy Matching

In Figure 5, if we were to match on the first found pattern, the string "011" would never match. This happens because the "0" rule would match first.

While this behavior is what most applications desire, it does come at a cost. Consider the following KPML document snippet.

```
   <regex>x{7}</regex>
   <regex>x{10}</regex>
```

Figure 6: Timeout Matching

Figure 6 is a typical NANP dial plan. From an application perspective, users expect a seven digit number to respond quickly, not waiting the typical inter-digit critical timer (usually four seconds). From a user's perspective, they do not want the system to cut off their ten digit number at seven digits because they did not enter the number fast enough.

One approach to this problem is to have an explicit dial string terminator. Typically, it is the pound key (#). Now, consider the following snippet.

```
<regex>x{7}#</regex>
<regex>x{10}#</regex>
```

Figure 7: Timeout Matching with Enter

The problem with the approach in Figure 7 is that the digit collector will still look for a digit after the "#" in the seven-digit case. Worse yet, the "#" will appear in the returned dial string.

The approach used in KPML is to have an explicit "Enter Key", as shown in the following snippet.

```
 <request>
  <pattern enterkey="#">
   <regex>xxxxxxx</regex>
   <regex>xxxxxxxxxx</regex>
  </pattern>
 </request>
```

Figure 8: Timeout Matching with Enter Key

In Figure 8 the enterkey parameter to the <pattern> tag specifies a string that terminates a pattern. In this situation, if the user enters seven digits followed by the "#" key, the pattern matches (or fails) immediately. KPML indicates a terminated nomatch with a KPML status code 402.

To address the various key press collection scenarios, we define three timers. The timers are the critical timer (criticaltimer), the inter-digit timer (interdigittimer), and the extra digit timer (extradigittimer). The critical timer is the time to wait for another digit if the collected digits can match a pattern. The extra timer is the time to wait after the longest match has occurred (presumably for the return key). The inter-digit timer inter-digit timer is the time to wait between digits in all other cases. Note there is no start timer, as that concept does not apply in the KPML context.

All of these timers are parameters to the <pattern> tag.

4.1.1.2 Intra-Digit Timing

Some patterns look for long duration key presses. For example, some applications look for long "#" or long "*".

KPML uses the "L" modifier to <regex> characters to indicate long key presses. The following KPML document looks for long pound.

```
<?xml version="1.0" encoding="UTF-8"?>
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
      version="1.0">
 <request>
  <pattern>
   <regex>L#</regex>
  </pattern>
 </request>
</kpml>
```

The request can specify what constitutes "long" by setting the long attribute to the <pattern>. This attribute is an integer representing the number of milliseconds. If the user presses a key for longer than longtimer milliseconds, the Long modifier is true.

NOTE: It is a local matter at the end device to consider multiple presses of the same key during the longtimer period to be equivalent to the Long version of that key. This is to support end devices that do not generate continuous key press tones.

4.1.2 Digit Suppression

Under basic operation, a KPML endpoint will transmit in-band tones (RFC2833 [10] or actual tone) in parallel with digit reporting.

NOTE: If KPML did not have this behavior, then a user device executing KPML could easily break called applications. For example, take a personal assistant that uses "*9" for attention. If the user presses the "*" key, KPML will hold the digit, looking for the "9". What if the user just enters a "*" key, possibly because they accessed an IVR system that looks for "*"? In this case, the "*" would get held by the user device, because it is looking for the "*9" pattern. The user would probably press the "*" key again, hoping that the called IVR system just did not hear the key press. At that point, the user device would send both "*" entries, as "**" does not match "*9". However, that would not have the effect the user intended when they pressed "*".

On the other hand, there are situations where passing through tones in-band is not desirable. Such situations include call centers that use in-band tone spills to effect a transfer.

For those situations, KPML adds a digit suppression tag, "pre", to the <regex> tag. There MUST NOT be more than one <pre> in any given

<regex>.

If there is only a single <pattern> and a single <regex>, the suppression processing is straightforward. The end-point passes digits until the stream matches the regular expression pre. At that point, the endpoint will continue collecting digits, but will suppress the generation or pass-through of any in-band digits.

If the endpoint suppressed digits, it MUST indicate this by including the attribute "suppressed" with a value of "yes" in the digit report.

Clearly, if the end device is processing the KPML document against quarantined digits, it is too late to suppress digits, as the end device has long sent the digits. This is a situation where there is a <pre> specification, but the "suppressed" attribute is not "yes" in the digit report.

A KPML endpoint MAY perform digit suppression. If it is not capable of digit suppression, it ignores the digit suppression attribute and will never send a suppressed indication in the digit report. In this case, it will match concatenated patterns of pre+value.

At some point in time, the endpoint will collect enough digits to the point it hits a <pre> pattern. The interdigittimer attribute indicates how long to wait once the user enters digits before reporting a time-out error. If the interdigittimer expires, the endpoint MUST issue a time-out report, transmit the suppressed digits on the media stream, and stop suppressing digit transmission.

Once the end device detects a match and it sends a NOTIFY request to report the digit string, the end device MUST stop digit suppression. Clearly, if subsequent digits match another <pre> expression, then the end device MUST start digit suppression.

After digit suppression begins, it may become clear that a match will not occur. For example, take the expression "<regex> <pre>*8</pre>xxx[2-9]xxxxxx</regex>". At the point the endpoint receives "*8", it will stop forwarding digits. Let us say that the next three digits are "408". If the next digit is a zero or one, the pattern will not match.

> NOTE: It is critically important for the endpoint to have a sensible inter-digit timer. This is because an errant dot (".") may suppress digit sending forever. See Section 4.1 for setting the inter-digit timer.

Applications should be very careful to indicate suppression only when they are fairly sure the user will enter a digit string that will

match the regular expression. In addition, applications should deal with situations such as no-match or time-out. This is because the endpoint will hold digits, which will have obvious user interface issues in the case of a failure.

4.1.3 One-Shot and Persistent Triggers

The KPML document specifies if the patterns are to be persistent by setting the persistent attribute to the <pattern> tag to "true". Otherwise, the request will be a one-shot subscription. If the end device does not support persistent subscriptions, it returns a KPML document with the KPML result code set to 531. If there are digits in the quarantine buffer and the digits match an expression in the KPML document, the end device prepares the appropriate KPML document.

4.1.4 Multiple Patterns

Some end devices may support multiple regular expressions in a given pattern request. In this situation, the application may wish to know which pattern triggered the event.

KPML provides a "tag" attribute to the <regex> tag. The "tag" is an opaque string that the end device sends back in the notification report upon a match in the digit map. In the case of multiple matches, the end device MUST chose the longest match in the KPML document. If multiple matches match the same length, the end device MUST chose the first expression listed in the subscription KPML document based on KPML document order.

If the end device does not support multiple regular expressions in a pattern request, the end device MUST return a KPML document with the KPML result code set to 532.

4.1.5 Monitoring Direction

By default, the end device monitors key presses emanating from the device. Given a dialog identifier of Call-ID, local-tag, and remote-tag, the end device monitors the key presses associated with the local-tag.

In the media proxy case, and potentially other cases, there is a need to monitor the key presses arriving from the remote user agent. The optional <stream> element to the >request> tag specifies which stream to monitor. The only legal value is "reverse", which means to monitor the stream associated with the remote-tag. The end point MUST ignore other values.

> NOTE: The reason this is a tag is so individual stream selection, if needed can be addressed in a backwards-compatible way.

4.1.6 Multiple, Simultaneous Subscriptions

Some end devices may support multiple key press event notification subscriptions at the same time. In this situation, the end device honors each subscription individually and independently.

A SIP user agent may request multiple subscriptions on the same SUBSCRIBE dialog, using the id parameter to the kpml event request.

One or more SIP user agents may request independent subscriptions on different SIP dialogs. In the body of the SUBSCRIBE is a leg parameter that indicates which leg to monitor. Section 3.2 describes the dialog addressing mechanism in detail.

If the end device does not support multiple, simultaneous subscriptions, the end device MUST return a KPML document with the KPML result code set to 533 on the dialog that requested the second subscription. The end device MUST NOT modify the state of the first subscription on the account of the second subscription attempt.

4.2 KPML Reports

When the user enters key press(es) that match a <regex> tag, the end device will issue a report.

After reporting, the interpreter terminates the KPML session unless the subscription has a persistence indicator. If the subscription does not have a persistence indicator, the end device MUST set the state of the subscription to "terminated" in the NOTIFY report.

If the subscription does not have a persistence indicator, to collect more digits the requestor must issue a new request.

> NOTE: This highlights the "one shot" nature of KPML, reflecting the balance of features and ease of implementing an interpreter. If your goal is to build an IVR session, we strongly suggest you investigate more appropriate technologies such as VoiceXML [8] or MSCML [9].

KPML reports have two mandatory attributes, code and text. These attributes describe the state of the KPML interpreter on the end device. Note the KPML code is not necessarily related to the SIP result code. An important example of this is where a legal SIP subscription request gets a normal SIP 200 OK followed by a NOTIFY, but there is something wrong with the KPML request. In this case, the NOTIFY would include the KPML failure code in the KPML report. Note that from a SIP perspective, the SUBSCRIBE and NOTIFY were successful. Also, if the KPML failure is not recoverable, the end

device will most likely set the Subscription-Sate to terminated. This lets the SIP machinery know the subscription is no longer active.

4.2.1 Pattern Match Reports

If a pattern matches, the end device will emit a KPML report. Since this is a success report, the code is "200" and the text is "OK".

The KPML report includes the actual digits matched in the digit attribute. The digit string uses the conventional characters '*' and '#' for star and octothorpe respectively. The KPML report also includes the tag attribute if the regex that matched the digits had a tag attribute.

If the subscription requested digit suppression (Section 4.1.2) and the end device suppressed digits, the suppressed attribute indicates "true". The default value of suppressed is "false".

> NOTE: KPML does not include a timestamp. There are a number of reasons for this. First, what timestamp would in include? Would it be the time of the first detected key press? The time the interpreter collected the entire string? A range? Second, if the RTP timestamp is a datum of interest, why not simply get RTP in the first place? That all said, if it is really compelling to have the timestamp in the response, it could be an attribute to the <response> tag.

4.2.2 KPML No Match Reports

There are a few circumstances in which the end device will emit a no match report. They are an immediate NOTIFY in response to SUBSCRIBE request (no digits detected yet), a request for service not supported by end device, or a failure of a digit map to match a string (timeout).

4.2.2.1 Immediate NOTIFY

The NOTIFY in response to a SUBSCRIBE request has no KPML if there are no matching quarantined digits. An example of this is in Figure 10.

If there are quarantined digits in the SUBSCRIBE request that match a pattern, then the NOTIFY message in response to the SUBSCRIBE request MUST include the appropriate KPML document.

```
NOTIFY sip:application@example.com SIP/2.0
Via: SIP/2.0/UDP proxy.example.com
Max-Forwards: 70
To: <sip:application@example.com>
From: <sip:endpoint@example.net>
Call-Id: 439hu409h4h09903fj0ioij
Subscription-State: active; expires=7200
CSeq: 49851 NOTIFY
Event: kpml
```

Figure 10: Immediate NOTIFY Example

5. DRegex Syntax

The Digit REGular EXpression (DRegex) syntax follows the Unix egrep and Java Regular Expression syntax.

White space is removed before parsing DRegex. This enables sensible pretty printing in XML without affecting the meaning of the DRegex string.

The following rules describe the use of DRegex in KPML.

Entity	Matches
digit	digit 0-9 and A-D
[digit selector]	Any digit in selector
[^digit selector]	Any digit NOT in selector
[digit-range]	Any digit in range
x	Any digit 0-9
.	Zero or more repetitions of previous pattern
|	Alternation
{m}	m repetitions of previous pattern
{m,}	m or more repetitions of previous pattern
{,n}	At most n (including zero) repetitions of previous pattern
{m,n}	at least m and at most n repetitions of previous pattern
Ldigit	Match the digit if it is "long"

Example	Description
1	Matches the digit 1
[179]	Matches 1, 7, or 9
[^01]	Matches 2, 3, 4, 5, 6, 7, 8, 9
[2-9]	Matches 2, 3, 4, 5, 6, 7, 8, 9
x	Any single digit
2|3	Matches 2 or 3; same as [23]
00|011	Matches the string 00 or 011
0.	Zero or more occurrences of 0
[2-9].	Zero or more occurrences of 2-9
011x{7,15}	011 followed by seven to fifteen digits
L*	Long star

6. Formal Syntax

The following syntax in Figure 11 uses the XML Schema [4].

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com)
     by Eric Burger (Snowshore Networks Inc.) -->
<xs:schema targetNamespace="urn:ietf:params:xml:ns:kpml"
 xmlns="urn:ietf:params:xml:ns:kpml"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified"
 attributeFormDefault="unqualified">
 <xs:element name="kpml">
  <xs:annotation>
   <xs:documentation>IETF Keypad Markup Language</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:choice>
    <xs:element name="request">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="pattern">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="flush" type="xs:string" minOccurs="0"/>
          <xs:element name="regex" maxOccurs="unbounded">
           <xs:complexType mixed="true">
            <xs:sequence>
             <xs:element name="pre" type="xs:string" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="tag" type="xs:string"
```

```
                        use="optional"/>
            </xs:complexType>
           </xs:element>
          </xs:sequence>
          <xs:attribute name="persistent" type="xs:boolean"
                        use="optional"/>
          <xs:attribute name="enterkey" type="xs:string"
                        use="optional"/>
          <xs:attribute name="interdigittimer" type="xs:integer"
                        use="optional"/>
          <xs:attribute name="criticaldigittimer" type="xs:integer"
                        use="optional"/>
          <xs:attribute name="extradigittimer" type="xs:integer"
                        use="optional"/>
          <xs:attribute name="longtimer" type="xs:integer"
                        use="optional"/>
         </xs:complexType>
        </xs:element>
        <xs:element name="stream" type="xs:string" minOccurs="0"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="response">
      <xs:complexType>
       <xs:attribute name="code" type="xs:string" use="required"/>
       <xs:attribute name="text" type="xs:string" use="required"/>
       <xs:attribute name="suppressed" type="xs:boolean"
                     use="optional"/>
       <xs:attribute name="digits" type="xs:string" use="optional"/>
       <xs:attribute name="tag" type="xs:string" use="optional"/>
      </xs:complexType>
     </xs:element>
    </xs:choice>
    <xs:attribute name="version" type="xs:string" use="required"/>
   </xs:complexType>
  </xs:element>
 </xs:schema>
```

Figure 11: XML Schema for KPML

7. Enumeration of KPML Status Codes

KPML failure codes broadly follow their SIP counterparts. Codes that start with a 2 indicate success. Codes that start with a 4 indicate failure. Codes that start with a 5 indicate a server failure, usually a failure to interpret the document or to support a requested feature.

KPML clients MUST be able to handle arbitrary status codes by examining the first digit only.

Any text can be in a KPML report document. KPML clients MUST NOT interpret the text field.

Code	Text
200	Success
402	User Terminated Without Match
423	Timer Expired
481	Dialog (call leg) Not Found
487	Subscription Expired
501	Bad Document
531	Persistent Subscriptions Not Supported
532	Multiple or Alternate Regular Expressions Not Supported
533	Multiple Subscriptions on a Call Leg Not Supported

Table 3: KPML Failure Codes

8. IANA Considerations

8.1 MIME Media Type application/kpml+xml

MIME media type name: application
MIME subtype name: kpml+xml
Required parameters: none
Optional parameters: charset

charset This parameter has identical semantics to the charset parameter of the "application/xml" media type as specified in XML Media Types [5].

Encoding considerations: See RFC3023 [5].

Interoperability considerations: See RFC2023 [5] and this document.

Published specification: This document.

Applications which use this media type: Session-oriented applications that have primitive user interfaces.

Intended usage: COMMON

8.2 URN Sub-Namespace Registration for urn:ietf:xml:ns:kpml

URI: urn:ietf:params:xml:ns:kpml

Registrant Contact: Eric Burger <eburger@ietf.org>

XML:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
          content="text/html;charset=iso-8859-1"/>
    <title>Key Press Markup Language</title>
  </head>
  <body>
    <h1>Namespace for Key Press Markup Language</h1>
    <h2>urn:ietf:params:xml:ns:kpml</h2>
    <p>
<a href="ftp://ftp.rfc-editor.org/in-notes/rfcXXXX.txt">RFCXXXX</a>.
    </p>
  </body>
</html>
```

8.3 KPML Schema Registration

Please register the XML Schema for KPML as referenced in Section 6.

9. Security Considerations

As an XML markup, all of the security considerations of RFC3023 [5] and RFC3406 [6] apply. Pay particular attention to the robustness requirements of parsing XML.

Key press information is potentially sensitive. Hijacking sessions allow unauthorized entities access to this sensitive information. Therefore, signaling SHOULD be secure, e.g., use of TLS and sips: SHOULD be used. Moreover, the information itself is sensitive. Thus if TLS is not used, S/MIME or other appropriate mechanism SHOULD be used.

End devices implementing this specification MUST implement TLS and SHOULD implement S/MIME at a minimum.

10. Examples

This section is informative in nature. If there is a discrepancy between this section and the normative sections above, the normative sections take precedence.

10.1 Monitoring for Octothorpe

A common need for pre-paid and personal assistant applications is to monitor a conversation for a signal indicating a change in user focus from the party they called through the application to the application itself. For example, if you call a party using a pre-paid calling card and the party you call redirects you to voice mail, digits you press are for the voice mail system. However, many applications have a special key sequence, such as the octothorpe (#, or pound sign) or *9 that terminate the called party leg and shift the user's focus to the application.

Figure 13 shows the KPML for long octothorpe. Note that the href is really on one line, but divided for clarity.

```
<?xml version="1.0">
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
      version="1.0">
  <request>
    <pattern>
      <regex>L#</regex>
    </pattern>
  </request>
</kpml>
```

Figure 13: Long Octothorpe Example

The regex value L indicates the following digit needs to be a long-duration key press.

10.2 Dial String Collection

In this example, the user device collects a dial string. The application uses KPML to quickly determine when the user enters a target number. In addition, KPML indicates what type of number the user entered.

```
<?xml version="1.0">
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
      version="1.0">
  <request>
    <pattern>
      <regex tag="local-operator">0</regex>
      <regex tag="ld-operator"/>00</regex>
      <regex tag="vpn">7[x][x][x]</regex>
      <regex tag="local-number7">9xxxxxxx</regex>
      <regex tag="RI-number">9401xxxxxxx</regex>
      <regex tag="local-number10">9xxxxxxxxxx</regex>
      <regex tag="ddd">91xxxxxxxxxx</regex>
      <regex tag="iddd">011x.</regex>
    </pattern>
  </request>
</kpml>
```

Figure 14: Dial String KPML Example Code

Note the use of the "tag" attribute to indicate which regex matched the dialed string. The interesting case here is if the user entered "94015551212". This string matches both the "9401xxxxxxx" and "9xxxxxxxxx" regular expressions. By following the rules described in Section 4.1.4, the KPML interpreter will pick the "9401xxxxxxx" string, as it occurs first in document order (both expressions match the same length). Figure 15 shows the response.

```
<?xml version="1.0"?>
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
      version="1.0">
  <response code="200" text="OK"
            digits="94015551212" tag="RI-number"/>
</kpml>
```

Figure 15: Dial String KPML Response

10.3 Interactive Digit Collection

This is an example where one would probably be better off using a full scripting language such as VoiceXML [8] or MSCML [9] or a device control language such as H.248.1 [13].

In this example, an application requests the user device to send the

user's signaling directly to the platform in HTTP, rather than monitoring the entire RTP stream. Figure 16 shows a voice mail menu, where presumably the application played a "Press K to keep the message, R to replay the message, and D to delete the message" prompt. In addition, the application does not want the user to be able to barge the prompt.

```
<?xml version="1.0">
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
      version="1.0">
  <request>
    <pattern>
      <flush>yes</flush>
      <regex tag="keep">5</regex>
      <regex tag="replay">7</regex>
      <regex tag="delete">3</regex>
    </pattern>
  </request>
</kpml>
```

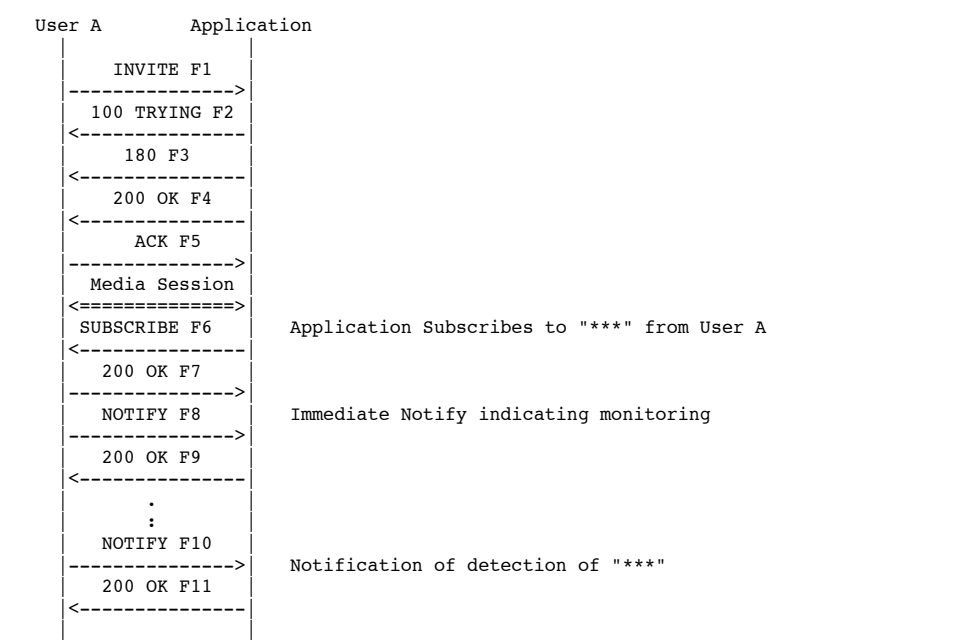
Figure 16: IVR KPML Example Code

NOTE: This usage of KPML is clearly inferior to using a device control protocol like H.248.1. From the application's point of view, it has to do the low-level prompt-collect logic. Granted, it is relatively easy to change the key mappings for a given menu. However, often more of the call flow than a given menu mapping gets changed. Thus there would be little value in such a mapping to KPML. We STRONGLY suggest using a real scripting language such as VoiceXML or MSCML for this purpose.

11. Call Flow Example

11.1 INVITE-Initiated Dialog

This section describes a successful subscription and notification from an Application with an End Device ("User A") in an INVITE-Initiated dialog. Note the Application can be a Record-Route Proxy, a B2BUA, or another end device.



Connection setup between User A and an Application subscribing to a DTMF event of "***" at User A.

F1 INVITE User A --> Application

```
INVITE sip:UserB@subB.example.com SIP/2.0
Via: SIP/2.0/UDP client.subA.example.com:5060;branch=z9hG4bK74
Max-Forwards: 70
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>
Call-ID: 12345601@subA.example.com
CSeq: 1 INVITE
Contact: <sip:UserA@client.subA.example.com>
Route: <sip:application.subA.example.com;lr>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, SUBCRIBE, NOTIFY
Allow-Events: kpml
Supported: replaces
```



```
Content-Type: application/sdp
Content-Length: ...

v=0
o=UserA 2890844526 2890844526 IN IP4 client.subA.example.com
s=Session SDP
c=IN IP4 client.subA.example.com
t=3034423619 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

F2 100 Trying Application --> User A

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP client.subA.example.com:5060;branch=z9hG4bK74
  ;received=192.168.12.22
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>
Call-ID: 12345601@subA.example.com
CSeq: 1 INVITE
Content-Length: 0
```

F3 180 Ringing Application --> User A

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP client.subA.example.com:5060;branch=z9hG4bK74
  ;received=192.168.12.22
Record-Route: <sip:application.subA.example.com;lr>
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 1 INVITE
Contact: <sip:UserB@client.subB.example.com>
Content Length: 0
```

F4 200 OK Application --> User A

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP client.subA.example.com:5060;branch=z9hG4bK74
  ;received=192.168.12.22
Record-Route: <sip:application.subA.example.com;lr>
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 1 INVITE
```

```
Contact: <sip:UserB@client.subB.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, SUBSCRIBE, NOTIFY
Supported: replaces
Content-Type: application/sdp
Content-Length: ...

v=0
o=UserB 2890844527 2890844527 IN IP4 client.subB.example.com
s=Session SDP
c=IN IP4 client.subB.example.com
t=3034423619 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

F5 ACK User A --> Application

```
ACK sip:UserB@subB.example.com SIP/2.0
Via: SIP/2.0/UDP client.subA.example.com:5060;branch=z9hG4bK74
Max-Forwards: 70
Route: <sip:application.subA.example.com;lr>
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 1 ACK
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY
Supported: replaces
Content-Length: 0
```

F6 SUBSCRIBE Application --> User A

```
SUBSCRIBE sip:UserA@subA.example.com SIP/2.0
Max-Forwards: 70
From: <sip:UserB@subB.example.com>;tag=567890
To: <sip:UserA@subA.example.com>;tag=1234567
Call-ID: 12345601@subA.example.com
CSeq: 1 SUBSCRIBE
Contact: <sip:UserB@client.subB.example.com>
Event: kpml
Expires: 7200
Accept: application/kpml+xml
Content-Type: application/kmpl+xml
Content-Length: ...

<?xml version="1.0">
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
      version="1.0">
```



```
  <request>
    <pattern>
      <regex value="***"/>
    </pattern>
  </request>
</kpml>
```

F7 200 OK User A --> Application

```
SIP/2.0 200 OK
To: <sip:UserA@subA.example.com>;tag=1234567
From: <sip:UserB@subB.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 1 SUBSCRIBE
Contact: <sip:UserB@client.subB.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, SUBSCRIBE, NOTIFY
Supported: replaces
Content-Length: 0
```

F8 NOTIFY User A --> Application

```
NOTIFY sip:UserB@subB.example.com SIP/2.0
Max-Forwards: 70
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 2 NOTIFY
Subscription-State: active;expires=3600
Content-Type: application/kpml+xml
Content-Length: ...
Event: kpml

<?xml version="1.0"?>
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
      version="1.0">
  <response code="100" text="TRYING"/>
</kpml>
```

F9 200 OK Application --> User A

```
SIP/2.0 200 OK
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>;tag=567890
```

```
Call-ID: 12345601@subA.example.com
CSeq: 2 NOTIFY
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, SUBSCRIBE, NOTIFY
Supported: replaces
Content-Type: application/sdp
Content-Length: 0
```

F10 NOTIFY User A --> Application

```
NOTIFY sip:UserB@subB.example.com SIP/2.0
Max-Forwards: 70
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@Application.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 3 NOTIFY
Subscription-State: active;expires=3125
Content-Type: application/kpml+xml
Content-Length: ...
Event: kpml

<?xml version="1.0"?>
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
      version="1.0">
  <response code="200" text="OK"
            digits="***"/>
</kpml>
```

F11 200 OK Application --> User A

```
SIP/2.0 200 OK
From: <sips:UserA@subA.net>;tag=1234567
To: <sips:UserB@Application.example.com>
Call-ID: 12345601@subA.com
JVD: CSeq: 3 NOTIFY
Contact: <sips:UserB@Application.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, SUBSCRIBE, NOTIFY
Supported: replaces
Content-Type: application/sdp
Content-Length: 0
```

11.2 Third-Party Subscription

Coming soon!

11.3 Remote-End Monitoring

Coming soon!

Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[3] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[4] Thompson, H., Beech, D., Maloney, M. and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC REC-xmlschema-1-20010502, May 2001.

[5] Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[6] Daigle, L., van Gulik, D., Iannella, R. and P. Faltstrom, "Uniform Resource Names (URN) Namespace Definition Mechanisms", BCP 66, RFC 3406, October 2002.

Informative References

[7] Bray, T., Paoli, J., Sperberg-McQueen, C. and E. Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C REC REC-xml-20001006, October 2000.

[8] World Wide Web Consortium, "Voice Extensible Markup Language (VoiceXML) Version 2.0", W3C Working Draft , April 2002, <http://www.w3.org/TR/voicexml20/>.

[9] Burger, E., Van Dyke, J. and A. Spitzer, "Media Server Control Markup Language (MSCML) and Protocol", draft-vandyke-mscml-02 (work in progress), June 2003.

[10] Schulzrinne, H. and S. Petrack, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals", RFC 2833, May 2000.

[11] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[12] Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP", draft-ietf-sipping-dialog-package-02 (work in progress), June 2003.

[13] Groves, C., Pantaleo, M., Anderson, T. and T. Taylor, "Gateway Control Protocol Version 1", RFC 3525, June 2003.

[14] Andreasen, F. and B. Foster, "Media Gateway Control Protocol (MGCP) Version 1.0", RFC 3435, January 2003.

[15] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[16] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[17] Olson, S., Camarillo, G. and A. Roach, "Support for IPv6 in Session Description Protocol (SDP)", RFC 3266, June 2002.

[18] Hunt, A. and S. McGlashan, "Speech Recognition Grammar Specification Version 1.0", W3C CR CR-speech-grammar-20020626, June 2002.

[19] Burger (Ed.), E., Van Dyke, J. and A. Spitzer, "Basic Network Media Services with SIP", draft-burger-sipping-netann-07 (work in progress), September 2003.

Authors' Addresses

Eric Burger
SnowShore Networks, Inc.
285 Billerica Rd.
Chelmsford, MA 01824-4120
USA

EMail: e.burger@ieee.org

Martin Dolly
AT&T Labs

EMail: mdolly@att.com

Appendix A. Contributors

Jeff Van Dyke worked enough hours and wrote enough text to be considered an author under the old rules.

Robert Fairlie-Cuninghame, Cullen Jennings, Jonathan Rosenberg, and I were the members of the Application Stimulus Signaling Design Team. All members of the team contributed to this work. In addition, Jonathan Rosenberg postulated DML in his "A Framework for Stimulus Signaling in SIP Using Markup" draft.

This version of KPML has significant influence from MSCML, the SnowShore Media Server Control Markup Language. Jeff Van Dyke and Andy Spitzer were the primary contributors to that effort.

That said, any errors, misinterpretation, or fouls in this document are my own.

Appendix B. Acknowledgements

Hal Purdy and Eric Cheung of AT&T Laboratories helped immensely through many conversations and challenges.

Steve Fisher of AT&T Laboratories suggested the digit suppression syntax.

Terence Lobo of SnowShore Networks made it all work.

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

Internet Engineering Task Force James M. Polk
Internet Draft Cisco Systems
Expiration: Aug 9th, 2004 Brian Rosen
File: draft-ietf-sipping-location-requirements-00.txt Marconi

Requirements for Session Initiation Protocol Location Conveyance

February 9th, 2003

Status of this Memo

Abstract

This document presents the framework and requirements for an extension to the Session Initiation Protocol (SIP) [1] for conveyance of user location information from a Session Initiation Protocol (SIP) user agent to another SIP entity. We consider cases where location information is conveyed from end to end, as well as cases where message routing by intermediaries is influenced by the location of the session initiator.

Table of Contents

1. Introduction

This document presents the framework and requirements for an extension to the Session Initiation Protocol (SIP) [1] for conveyance of user location information object described by [7] from a SIP User Agent to another SIP entity.

There are several situations in which it is appropriate for SIP to be used to convey Location Information (LI) from one SIP entity to another. This document specifies requirements when a SIP UAC knows its location by some means not specified herein, and needs to inform another SIP entity. One example is to reach your nearest pizza parlor. A chain of pizza parlors may have a single well known uri (sip:pizzaparlor.com), that is forwarded to the closest franchise by the pizzaparlor.com proxy server. The receiving franchise UAS uses the location information of the UAC to schedule your delivery.

Another important example is emergency calling. A call to sip:sos@example.com is an emergency call as in [3]. The example.com proxy server must route the call to the correct emergency response center (ERC) determined by the location of the caller. At the ERC, the UAS must determine the correct police/fire/ambulance/... service, which is also based on your location. In many jurisdictions, accurate location information of the caller in distress is a required component of a call to an emergency center.

A third example is a direction service, which might give you verbal directions to a venue from your present position. This is a case where only the destination UAS needs to receive the location information.

This document does not discuss how the UAC discovers or is

configured with its location (either coordinate or civil based). It also does not discuss the contents of the Location Object (LO). It does specify the requirements for the "using protocol" in [7].

1.1 Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [2].

1.2 Changes from Individual Submission Versions

This is a list of the changes that have been made from the -00 individual submission version of this ID:

- Brian Rosen was brought on as a co-author

- Requirements that a location header were negatively received in the previous version of this document. AD and chair advice was to move all location information into a message body (and stay away from headers)

- Added a section of "emergency call" specific requirements

- Added an Open Issues section to mention what hasn't been resolved yet in this effort

This is a list of the changes that have been made from the individual submission version -01

- Added the IPR Statement section

- Adjusted a few requirements based on suggestions from the Minneapolis meeting

- Added requirements that the UAC is to include from where it learned its location in any transmission of its LI

- Distinguished the facts (known to date) that certain jurisdictions relieve persons of their right to privacy when they call an ERC, while other jurisdictions maintain a person's right to privacy, while still others maintain a person's right to privacy - but only if they ask that their service be set up that way.

- Made the decision that TLS is the security mechanism for location conveyance in emergency communications (vs. S/MIME, which is still the mechanism for UA-to-UA non-emergency location conveyance cases).

- Added the Open Issue of whether a Proxy can insert location information into an emergency SIP INVITE message, and some of the open questions surrounding the implications of that action

- added a few names to the acknowledgements section

2. In the Body or in a Header

When one user agent wants to inform another user agent where they are, it seems reasonable to have this accomplished by placing the location information (coordinate or civil) in an S/MIME registered and encoded message body, and sending it as part of a SIP request or response. No routing of the request based on the location information is required in this case; therefore no SIP Proxies between these two UAs need to view the location information contained in the SIP messages.

Although SIP [1} does not permit a proxy server to modify or delete a body, there is no restriction on viewing bodies. However, S/MIME protection implemented on bodies is only specified between UAS and UAC, and if engaged, would render the location object opaque to a proxy server for any desired modification if it is not correct or precise enough from that proxy's point of view (were it to be able to view it). This problem is similar to that raised in Session Policy [8], where an intermediary may need information in a body, such as IP address of media streams or codec choices to route a call properly. Requirements in [8] are applicable to routing based on location, and are incorporated in these requirements by reference.

It is conceivable to create a new header for location information. However, [7] prefers S/MIME for security of Location Information, and indeed S/MIME is preferable in SIP for protecting one part of a message. Accordingly, these requirements specify location be carried in a body.

It is the use of S/MIME however, that limits routing based on location. Therefore, it seems appropriate to require that, where routing is dependent on location, protection of the location information object be accomplished by other mechanisms: here TLS ("sips:" from [1]). It is envisioned that S/MIME SHOULD be used when location information is not required by proxy servers, and TLS MUST be used when it is. The UAC will need to know the difference in the call's intent as to which security mechanism to engage for LI conveyance.

This document does not address the behavior or configuration of SIP Proxy Servers in these cases in order to accomplish location-sensitive routing. That is out of scope, and left for further (complementary) efforts.

3. Scope of Location in a Message Body

As concluded from the previous section, location information is to be contained within a message body. If either another body (SDP for example) is also to be sent in the message, or the LI is to be protected with S/MIME, the rules stated in section 7 of [1] regarding multipart MIME bodies MUST be followed. The format and privacy/security rules of the location information SHOULD be defined within the Geopriv WG.

4. Requirements for UA-to-UA Location Conveyance

The following are the requirements for UA-to-UA Location Conveyance Situations where routing is not based on the LI of either UA:

U-U1 - MUST work with dialog-initiating SIP Requests and responses, as well as the SIP MESSAGE method[4], and SHOULD work with most SIP messages.

U-U2 - UAC Location information SHOULD remain confidential in route to the destination UA.

U-U3 - The privacy and security rules established within the Geopriv Working Group that would categorize SIP as a 'using protocol' MUST be met [7].

U-U4 - The UAC SHOULD indicate in the SIP message that includes location information where the LI came from (IANA registered codes for GPS, Cell Tower Triangulation, WiFi, DHCP, manual entry - as examples).

5. Requirements for UA-to-Proxy Server Location Conveyance

The following are the requirements for UA-to-Proxy Server Location Conveyance situations:

U-PS1 - MUST work with dialog-initiating SIP Requests and responses, as well as the SIP MESSAGE method[4], and SHOULD work with most SIP messages.

U-PS2 - UAC location information SHOULD remain confidential with respect to entities to which the location information is not addressed, but MUST be useable by intermediary proxy servers.

U-PS3 - The privacy and security rules established within the Geopriv Working Group which would categorize SIP as a 'using protocol' MUST be met [7].

U-PS4 - Modification or removal of the LO by proxy servers MUST NOT be required (as [1] currently forbids this).

U-PS5 - any mechanism used to prevent unwanted observation of this Location Information CANNOT fail the SIP Request if not understood by intermediary SIP entities or the destination UAS.

U-PS6 - Proxy Servers that do not or cannot understand the Location Information in the message body for routing purposes MUST NOT fail the SIP Request.

U-PS7 ¡ It MUST be possible for a proxy server to assert the validity of the location information provided by the UA. Alternatively, it is acceptable for there to be a mechanism for a proxy server to assert a location object itself.

U-PS8 - The UAC SHOULD indicate in the SIP message that includes location information where the LI came from (IANA registered codes for GPS, Cell Tower Triangulation, WiFi, DHCP, manual entry - as examples).

6. Additional Requirements for Emergency Calls

Emergency calls have requirements that are not generally important to other uses for location in SIP:

Emergency calls presently have between 2 and 8-second call setup times. There is ample evidence that the longer call setup end of the range causes an unacceptable number of callers to abandon the call before it is completed. Two-second call completion time is a goal of many existing emergency call centers. Allocating 25% of the call set up for processing privacy concerns seems reasonable; 1 second would be 50% of the goal, which seems unacceptable; less than 0.5 second seems unachievable, therefore:

E-1 - Privacy mechanisms MUST add no more than 0.5 second of call setup time when implemented in present technology UAs and Proxy Servers.

It may be acceptable for full privacy mechanisms related to the location of the UAC (and it's user) to be tried on an initial attempt to place a call, as long as the call attempt may be retried without the mechanism if the first attempt fails. Abandoning privacy in cases of failure of the privacy mechanism might be subject to user preference, although such a feature would be within the domain of a UA implementation and thus not subject to standardization. It should be noted that some jurisdictions have laws that explicitly deny any expectation of location privacy when

making an emergency call, while others grant the user the ability to remain anonymous even when calling an ERC. So far, this has been offered in some jurisdictions, but the user within that jurisdiction must state this preference, as it is not the default configuration.

E-2 ¡ Privacy mechanisms MUST NOT be mandatory for successful conveyance of location during an (sos-type) emergency call.

E-3 - It MUST be possible to provide a privacy mechanism (that does not violate the other requirements within this document) to a user within a jurisdiction that gives that user the right to choose not to reveal their location even when contacting an ERC.

E-4 ¡ The retention and retransmission policy of the ERC MUST be able to be made available to the user, and override the user's normal policy when local regulation governs such retention and retransmission (but does not violate requirement E-3). As in E-2 above, requiring the use of the ERC's retention and/or retransmission policy may be subject to user preference although in most jurisdictions, local laws specify such policies and may not be overridden by user preference.

Location information is considered so important during emergency calls, that it is to be transmitted even when it is not considered reliable, or might even be wrong. For example, some application might know that the DHCP reply with location information was overwritten recently (or exactly) when a VPN connection was activated. This could, and likely will, provide any new location information to the UA from somewhere far away from the UA (perhaps the user's corporate facility).

E-5 Location information MUST be transmitted, if known to the UAC, in all calls to an ERC, even in the case it is not considered reliable.

E-6 The UAC SHOULD be able to inform the ERC that the location information provided in the SIP message might be wrong.

Requirements U-U4 and U-PS8 stipulate the inclusion of how the UAC learned its location. This can be especially useful to an ERC operator attempting to learn all that is possible from this remote person in distress. With that in mind, it is important to distinguish the location information learned locally from LI learned over a VPN; which in itself is useful additional information to that ERC operator.

E-7 The UA MUST not provide the (overwritten?) location information provided by a VPN (in lieu of the LI from the local network).

E-8 The UA SHOULD include within the location conveyance to the ERC that it is (or recently was) connected to a VPN.

7. Current Known Open issues

This is a list of open issues that have not yet been addressed to conclusion:

1) Whether SIP Proxies SHOULD be able to insert location information into an emergency call set-up (the INVITE)?

   1a) This has the additional implication of whether or not, or regardless of the fact the UAC already inserted location into the sos@localdomain INVITE.

   1b) Should the Proxy somehow differentiate its location information from that provided by the UAC (with each LI having a SIP entity (type?) originator label?

   1c) Should there be any behavior difference with respect to Open Issue #1b if the Proxy does not know or cannot tell if the UAC inserted location information (further emphasizing the need for some form of originator label)?

2) Whether SIP Proxies SHOULD be able to return location information in a Redirect message to the UAC making the emergency call?

3) If S/MIME is chosen as a SHOULD (in general, vs. TLS), this doc might consider stipulating a special purpose Proxy (an "emergency services" proxy) that can process location information (a Geopriv LO) and route the message directly to the appropriate ERC.

   At Issue: plain "vanilla" proxies probably won't have the capabilities to route based on location information in the near future, but should that timing be considered here?

8. Security Considerations

Conveyance of geo-location of a UAC is problematic for many reasons. This document calls for that conveyance to normally be accomplished through secure message body means (like S/MIME or TLS). In cases where a session set-up is routed based on the location of the UAC initiating the session or SIP MESSAGE, securing the location with an end-to-end mechanism such as S/MIME is problematic.

9. IANA Considerations

There are no IANA considerations within this document at this time.

10. Acknowledgements

To Dave Oran for helping to shape this idea. To Jon Peterson and Dean Willis on guidance of the effort. To Henning Schulzrinne, Jonathan Rosenberg, Dick Knight, and Keith Drage for constructive feedback.

11. References - Normative

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "SIP: Session Initiation Protocol ", RFC 3261, June 2002

[2] S. Bradner, "Key words for use in RFCs to indicate requirement levels," RFC 2119, Mar. 1997.

[3] H. Schulzrinne, "draft-schulzrinne-sipping-sos-04.txt", Internet Draft, Jan 03, Work in progress

[4] B. Campbell, Ed., J. Rosenberg, H. Schulzrinne, C. Huitema, D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging" , RFC 3428, December 2002

[5] J. Polk, J. Schnizlein, M. Linsner, " draft-ietf-geopriv-dhcp-lci-option-03.txt", Internet Draft, Dec 2003, Work in progress

[6] H. Schulzrinne, "draft-schulzrinne-geopriv-dhcp-civil-01.txt", Internet Draft, Feb 03, Work in progress

[7] J. Cuellar, J. Morris, D. Mulligan, J. Peterson. J. Polk, "draft-ietf-geopriv-reqs-04.txt", Internet Draft, Oct 03, Work in progress

[8] J. Rosenberg, "Requirements for Session Policy for the Session Initiation Protocolö, draft-ietf-sipping-session-policy-req-00", Internet Draft, "work in progress" June, 2003

12. Author Information

James M. Polk
Cisco Systems
2200 East President George Bush Turnpike
Richardson, Texas 75082 USA
jmpolk@cisco.com

Brian Rosen
Marconi Communications, Inc.
2000 Marconi Drive
Warrendale, PA 15086
Brian.rosen@marconi.com

The Expiration date for this Internet Draft is:

August 9th, 2004

SIPPING
Internet-Draft
Expires: August 16, 2004

J. Rosenberg
dynamicsoft

February 16, 2004

Requirements for Session Policy for the Session Initiation Protocol (SIP)

draft-ietf-sipping-session-policy-req-01

Status of this Memo

Copyright Notice

Abstract

The proxy server plays a central role as an intermediary in the establishment of sessions in the Session Initiation Protocol (SIP). In that role, they can define and impact policies on call routing, rendezvous, and other call features. However, there is no standard means by which proxies can have any influence on session policies, such as the codecs that are to be used. As such, ad-hoc and non-conformant techniques have been deployed to allow for such policy mechanisms. There is a need for a standards-based and complete mechanism for session policies. This document defines a set of requirements for such a mechanism.

Table of Contents

1. Introduction

The Session Initiation Protocol [2] enables the setup and management of interactive multimedia sessions on IP networks. A central element in SIP is the proxy server. Proxies are responsible for request routing, rendezvous, authentication and authorization, mobility, and other signaling services. However, proxies are divorced from the actual sessions - audio, video, and messaging - that SIP establishes. Details of the sessions are carried in the payload of SIP messages, and are usually described with the Session Description Protocol (SDP) [1]. Indeed, SIP provides end-to-end encryption features using S/MIME, so that all information about the sessions can be hidden from eavesdroppers and proxies alike.

However, experience has shown that there is a need for SIP intermediaries to impact aspects of the session. One aspect is the path that the media streams will take. Frequently, a SIP provider will need or want the media to traverse some kind of intermediary, such as a NAT. Indeed, the central concept of the midcom framework [4] is to define a model of how this can be done. In this model, a midcom agent, typically a proxy server, interacts with the middlebox to open and close media pinholes, obtain NAT bindings, and so on. In this role as a midcom agent, the proxy will need to examine and possibly modify the session description in the body of the SIP message. This modification is to achieve a specific policy objective: to force the media to route through an intermediary.

In another application, SIP is used in a wireless network. The network provider has limited resources for media traffic. During periods of high activity, the provider would like to restrict codec usage on the network to lower rate codecs.

In yet a third application, SIP is used in a network that has gateways which support a single codec type (say, G.729). When communicating with a partner network that uses gateways with a different codec (say, G.723), the network modifies the SDP to route the session through a converter that changes the G.729 to G.723.

The desire to impact aspects of the session inevitably occurs in domains where the administrator of the SIP domain is also the owner and administrator of an IP network over which it is known that the sessions will traverse. This includes enterprises, Internet access providers, and in some cases, backbone providers.

Since SIP is the protocol by which the details of these sessions are negotiated, it is natural for providers to wish to impose their session policies through some kind of SIP means. To date, this has been accomplished through SDP editing, a process where proxies dig

into the bodies of SIP messages, and modify them in order to impose their policies. However, this SIP editing technique has many drawbacks.

2. Problems with Existing Situation

RFC 3261 explicitly disallows proxy servers from manipulating the content of bodies. This is at odds with the common industry practice of extensive manipulation of bodies by proxies. Although a common practice, it is at odds with the SIP specification for many reasons:

End-to-End Encryption: SIP uses S/MIME to support end-to-end security security features. Authentication, message integrity, and encryption are provided. The encryption capabilities are important for end-to-end privacy services, for example. The end-to-end message integrity and authentication are important for preventing numerous attacks, including theft of calls, eavesdropping attacks, and so on. If end-to-end authentication is used, any manipulation of the body will cause the message integrity check to fail. If end-to-end encryption is used, the proxy won't even be able to look at the SDP to modify it. In this case, media may not function, and the call will fail.

Require Processing: A UA may require that an extension be applied to the SDP body. This is accomplished by including a Require header in the SIP message. Proxies do not look at such headers. If the proxy processes the SDP without understanding the extension, it may improperly modify the SDP, resulting in a call failure.

Consent: Ultimately, end users need to be in control of the media they send. If a user makes a call through a SIP network, they have the expectation that their media is delivered to the recipient. By having proxies modify the SDP in some way, they act in ways outside of expected behavior of the system.

Future Proofing: One of the benefits of the SIP architecture is that only the endpoints need to understand sessions, session descriptions, bodies, and so on. This facilitates the use of proxy networks to provide communications services for future session types, such as games and messaging. However, if proxies require an understanding of session types and session descriptions, the SIP network becomes locked in to providing features for a particular set of session types. If a new session description protocol, such as SDPng [10], were introduced, calls would not function even though the endpoints support SDPng. Furthermore, it would be hard to determine why it did not function, since the failure would occur transparently in some proxy in the middle of the network.

Robustness: Having a proxy manipulate the body introduces a host of new failure modes into the network. Firstly, the proxy itself will need to have state in some form in order to properly manipulate the SDP. This means that, should the proxy fail, the

call may not be able to continue. Secondly, proxies typically won't enforce the media policy. Rather, they leave that to some media middlebox somewhere on the media path. This media middlebox may fail as well. Since the user does not know of its existence, they may not be able to detect this failure or retry the media path around it.

Scalability: One of the reasons SIP scales so well is that proxies don't have to be aware of the details of the sessions being established through them. If a proxy needs to examine and/or manipulate session descriptions, this could require many additional processing steps. The proxy may need to traverse a multi-part body to find the SDP, in the case of SIP-T [5]. The proxy will need to parse, modify, and possibly re-serialize the session description. All of this requires additional processing that worsens the performance of the proxies.

We note that many of these problems are similar to those pointed out by the IAB regarding Open Pluggable Exchange Services (OPES) [6]. Indeed, the problems are similar. Both have to do with the involvement of intermediaries in manipulation of end-to-end content. Here, the content is not in the body itself, but is a session described by the body.

We believe a better solution is needed.

3. Requirements for a Solution

In order to prevent the continuing usage of SDP editing to achieve session policies, we believe explicit protocol support is needed to provide a mechanism that can overcome the limitations above. As per the IETF SIP change process [7], the first step in any such activity is to specify requirements for the solution. This section is an enumeration of those requirements.

3.1 General Requirements

REQ-GEN-1: The solution should work even with SIP end-to-end encryption and end-to-end authentication enabled.

REQ-GEN-2: The solution should not force a proxy to violate the SIP specification or any defined extensions.

REQ-GEN-3: The solution should not require substantial processing burden on the proxies.

REQ-GEN-4: The solution should not require proxies to understand a specific type of session description (i.e., SDP or SDPng).

REQ-GEN-5: The solution should have a minimal impact on call setup delays, and ideally, have no impact on call setup delays.

REQ-GEN-6: The solution should require minimal overhead, since it is anticipated to receive wide use in wireless networks.

REQ-GEN-7: The solution should be extensible, supporting new session policy types in the future.

REQ-GEN-8: The solution must not require that the proxies be in the same administrative domain as the media intermediaries.

3.2 Policy Requirements

REQ-POL-1: The solution should allow specification of independent policies by each proxy along the call setup path, without any coordination between proxies.

REQ-POL-2: The solution should allow a proxy to specify media policies on a stream-by-stream basis.

REQ-POL-3: When used in conjunction with the offer/answer model [3], the solution should allow a proxy to specify independent policies for the media streams in each direction.

REQ-POL-4: The mechanism must provide the ability to inform the UA about the set of session-independent session policies when the device starts up. These are session policies that do not depend on a particular session.

REQ-POL-5: The mechanism must allow the provider to change the session-independent policies at least a few times a day.

REQ-POL-6: The mechanism must allow the session independent policies to vary on a user by user basis.

REQ-POL-7 The mechanism must provide a way to inform the client about changes in session independent session policies when they occur.

3.3 Policy Types

REQ-POL-4: The solution should allow a proxy to request media sessions to traverse through one or more intermediaries.

REQ-POL-5: The solution should allow a proxy to request a specific source routing mechanism to be used (when applicable) in order to traverse those intermediaries. The source routing technique may be media-specific, or a generic technique, such as IP-in-IP [8]

REQ-POL-6: Intermediaries must be identifiable using either an IP address or an FQDN, in order to support DNS-based load balancing and failover techniques.

REQ-POL-7: The solution should allow a proxy to inspect the addresses for the media sessions, so that it can set policies in intervening firewalls.

REQ-POL-8: The solution should allow proxies to request that a particular media stream not be used (video, for example).

REQ-POL-9: The solution should allow proxies to request that a particular codec not be used.

REQ-POL-10: The solution should allow proxies to express preferences for the use of particular codecs.

REQ-POL-11: The solution should allow proxies to request that Quality of Service (QoS) should be requested for a stream.

REQ-POL-12: The solution should allow proxies to ask endpoints to use specific parameters in their QoS reservations.

REQ-POL-13: The solution should allow proxies to ask endpoints to provide a specific credential in their QoS requests. This requirement covers the functionality currently described in [9].

3.4 Consent Requirements

Consent plays a critical role for this problem. End users must be allowed control over how they communicate with each other. Indeed, with end-to-end IP connectivity, there is frequently little the provider can do to force users to communicate one way or another. Ultimately, any means a provider comes up with can be circumvented by some creative engineering in the clients. As such, policy requests by proxies are just that - requests, and are ultimately honored at the discretion of the end users. The mechanism needs to recognize this, and be engineered to work within this model, rather than try to work around it.

REQ-CON-1: The mechanism should allow the UAC to know the set of policies requested by the proxies along the call path. [[OPEN ISSUE: Is it more important for the UAC to know about changes requested for media in one direction or the other?]]

REQ-CON-2: The mechanism should allow the UAS to know the set of policies requested by the proxies along the call path.

REQ-CON-3: The mechanism should allow the UAC to reject any policy requests made by proxies.

REQ-CON-4: The mechanism should allow the UAS to reject any policy requests made by proxies.

REQ-CON-5: The mechanism should allow the proxies to know whether or not the UAC has accepted its policy requests.

REQ-CON-6: The mechanism should allow the proxies to know whether or not the UAS has accepted its policy requests.

REQ-CON-7: The mechanism should allow the proxies to inform the UAC and UAS of the consequences of non-compliance to the policies. Potential consequences include call rejection, degraded media quality, lack of connectivity for a media stream, and so on.

3.5 Security Requirements

REQ-SEC-1: The mechanism should allow user agents to verify the identity of the providers requesting the session policies.

REQ-SEC-2: The mechanism should allow user agents to verify the integrity of the session policies.

REQ-SEC-3: The mechanism must provide assurances to the UAC and UAS that only proxies on the actual SIP signaling path have requested session policies.

REQ-SEC-4: The mechanism should allow proxies to ensure the confidentiality of the session policies, so that no one but the UAC or UAS can observe them. [[OPEN ISSUE: Is this really a requirement?]]

REQ-SEC-5: The mechanism must not enable any new denial-of-service attacks to be launched. [[OPEN ISSUE: This is motherhood and apple pie - does it need to be here?]]

REQ-SEC-6: The mechanism shall still allow for media security through Secure RTP [11]. In the case of intermediaries which process the RTP in some way that would invalidate any signatures, the UAs must be aware of the presence of the intermediary, and perform key exchanges with it. [[OPEN ISSUE: This may be an impossible requirement to meet without using a B2BUA.]]

4. Security Considerations

Requirements related to security are considered in Section 3.5.

5. Acknowledgements

I would like to thank Volker Hilt, Gonzalo Camarillo, Miguel Garcia and Kumiko Ono for their input.

Informative References

[1] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[3] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[4] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A. and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, August 2002.

[5] Vemuri, A. and J. Peterson, "Session Initiation Protocol for Telephones (SIP-T): Context and Architectures", BCP 63, RFC 3372, September 2002.

[6] Floyd, S. and L. Daigle, "IAB Architectural and Policy Considerations for Open Pluggable Edge Services", RFC 3238, January 2002.

[7] Mankin, A., Bradner, S., Mahy, R., Willis, D., Ott, J. and B. Rosen, "Change Process for the Session Initiation Protocol (SIP)", BCP 67, RFC 3427, December 2002.

[8] Perkins, C., "IP Encapsulation within IP", RFC 2003, October 1996.

[9] Marshall, W., "Private Session Initiation Protocol (SIP) Extensions for Media Authorization", RFC 3313, January 2003.

[10] Kutscher, D., Ott, J. and C. Bormann, "Session Description and Capability Negotiation", draft-ietf-mmusic-sdpng-07 (work in progress), October 2003.

[11] Baugher, M., "The Secure Real-time Transport Protocol", draft-ietf-avt-srtp-09 (work in progress), July 2003.

Author's Address

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
EMail: jdrosen@dynamicsoft.com
URI: http://www.jdrosen.net

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

Network Working Group H. Schulzrinne
Internet-Draft Columbia U.
Expires: August 8, 2004 February 8, 2004

Emergency Services URI for the Session Initiation Protocol
draft-ietf-sipping-sos-00

Status of this Memo

Copyright Notice

Abstract

As part of an overall architecture for supporting emergency calling for the Session Initiation Protocol (SIP), this document defines universal emergency SIP URIs, sip:sos@domain and sips:sos@domain, that allows SIP user agents to contact the local emergency call center. It also defines conventions that increase the high probability of reaching the appropriate emergency call center. The document does not define any SIP protocol extensions.

Table of Contents

1. Introduction

Using the public switched telephone network (PSTN), emergency help can often be summoned at a designated, widely known number, regardless of where the telephone was purchased. However, this number differs between localities, even though it is often the same for a country or continent-size region (such as many countries in the European Union or North America). For end systems based on the Session Initiation Protocol (SIP) [RFC3261], it is desirable to have a universal identifier, independent of location, to simplify the user experience and to allow the device to perform appropriate processing. Here, we define a common user identifier, "sos", as the contact mechanism for emergency assistance. This identifier is meant to be used in addition to any local emergency numbers.

This document specifies only a small part of a comprehensive set of recommendations for operating emergency services. The overall architecture is described in [schulzrinne-sipping-emergency-arch]. That document describes, for example, how a device that identifies a call as an emergency call can route it to the appropriate emergency call center (ECC).

This document does not introduce any new SIP header fields, request methods, status codes, message bodies, or events. User agents unaware of the recommendations in this draft can place emergency calls, but may not be able to provide the same user interface functionality. The document suggests behavior for proxy servers, in particular outbound proxy servers.

2. Terminology

In this document, the key words "MUST", "MUSTNOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant implementations.

3. Requirements

- It should be possible for devices to provide user interfaces that can directly cause an emergency call, without the user having to "dial" or type a specific address.
- Even as each country is likely to operate their emergency calling infrastructure differently, SIP devices should be able to reach emergency help and, if possible, be located in any country.
- While traveling, users must be able to use their familiar "home" emergency identifier. Users should also be able to dial the local emergency number in the country they are visiting.
- Any mechanism must be deployable incrementally and work even if not all SIP entities support emergency calling. User agents conforming to the SIP specification [RFC3261], but unaware of this document, must be able to place emergency calls, possibly with restricted functionality.
- Given incremental deployment, emergency call functionality should be testable by the user without causing an emergency response.
- Emergency calling mechanisms must support existing emergency call centers based on circuit-switched technology as well as future ECC that are SIP-capable.

4. Emergency URIs

A single, global (set of) identifiers for emergency services is highly desirable, as it allows end system and network devices to be built that recognize such services and can act appropriately. Such actions may include restricting the functionality of the end system, providing special features, overriding user service constraints or routing session setup messages.

SIP user agents (UAs) that determine that a dialog or transaction relates to an emergency MUST use an emergency call identifier in the Request-URI. The Request-URI MUST be either an emergency SIP URI defined in Section Section 4.1 or an emergency tel URI defined in Section Section 4.2.

4.1 SIP URIs for Emergency Calls

It is RECOMMENDED that SIP-based [RFC3261] end systems and proxy servers support a uniform emergency call identifier, namely the reserved user name "sos" within any domain, e.g.,

```
sip:sos@example.com
sips:sos@example.com
```

The reserved name is case-insensitive.

The host part of the emergency URI SHOULD be the host portion of the address-of-record of the caller. The "sips" form SHOULD be used to ensure integrity and confidentiality. All SIP requests with URIs of this form are assumed to be emergency calls.

(The domain-of-record was chosen since a SIP user agent may not be able to determine the local domain it is visiting. This also allows each user to test this facility, as the user can ensure that such services are operational in his home domain. An outbound proxy in the visited domain can handle the call if it believes to be in a position to provide appropriate emergency services.)

In addition, we reserve user addresses beginning with the string "sos." for specific emergency services:

```
sos.fire      fire brigade
sos.rescue    ambulance (rescue)
sos.marine    marine guard
sos.police    police (law enforcement)
sos.mountain  mountain rescue
```

The sub-addresses are also case-insensitive. Additional subaddresses

can be registered with IANA (Section Section 8).

(In some areas, these emergency services use different numbers.)

The SIP URI user name "sos" and user names starting with "sos." MUSTNOT be assigned to any regular user.

4.2 Tel URIs for Emergency Calls

User agents SHOULD determine the local emergency numbers, either by consulting their manual configuration for devices that do not move across national borders, by DHCP, DNS NAPTR or some other configuration mechanism [schulzrinne-sipping-emergency-arch]. If a user agent has no knowledge of local emergency numbers, it MUST also recognize the digit strings 000, 08, 112, 110, 118, 119, 911 and 999 as emergency numbers.

(SIP user agents, such as Ethernet deskphones, that are unlikely to move frequently across national borders can easily implement a local dialing plan that recognizes local emergency numbers. Mobile devices, including PDAs and laptops, may not have a reliable way of determining their current location. Using automatic configuration avoids collisions with extensions that equal one of the eight numbers above. If a local network does not have an outbound proxy server, local dial plans also do not apply, so the problem of number collision does not arise. Collisions with non-emergency service numbers are still possible, albeit less likely. For example, 118 is used for directory assistance in Finland.)

If the user dials any of these digit strings, the UAC SHOULD generate a request with the "sos" URI described in Section Section 4.1 unless it has discovered a local outbound proxy. In that case, a UAC MAY use a "tel" URI [RFC2806] without 'phone-context', such as

tel:911
tel:112

Outbound proxy servers MUST be configurable to recognize additional local emergency numbers in "tel" URIs.

There are about 60 service numbers for emergency services in the world; including them all is not practical, as that would interfere with existing local two, three and four-digit dialing plans.

5. Request Handling

Once identified, a user agent can either determine the appropriate ECC locally or delegate this task to an outbound proxy. Details are in [schulzrinne-sipping-emergency-arch].

Outbound proxy servers MUST recognize all local emergency numbers as well as the tel URIs enumerated in Section Section 4.2. The proxy MAY use any additional information contained in the call request, such as Mobile Country Code and the Mobile Network Code for 3GPP devices, to recognize additional numbers as emergency numbers.

It is RECOMMENDED that gateway SIP MESSAGE requests are directed to a TTY-for-the-deaf translator or a short-message service (SMS) if the emergency call center cannot handle SIP instant messaging.

OPTIONS requests to the user "sos" and the "sos.*" addresses (sos.fire, etc.) can be used to test if the "sos" addresses are valid. As in standard SIP, a 200 (OK) response indicates that the address was recognized and a 404 (Not found) that it was not. Such request cause no further action. It is RECOMMENDED that user agents periodically automatically check for the availability of the "sos" identifier and alert the user if the check fails. The period of such automated checks SHOULDNOT be less than once per day and MUST be randomly placed over the testing interval.

6. Identifying the Local Emergency Numbers

There are many ways that a user agent can configure emergency numbers for use in analyzing calls made with telephony-type user input. Such numbers become part of the device dialplan. Mechanisms include configuration tokens such as SIM cards in mobile devices, network-specific solutions (e.g., for 3GPP networks) or protocol-based solutions. Protocol-based solutions, using XCAP and DNS, are discussed in [schulzrinne-sipping-emergency-arch.] Given the different trade-offs in user agent implementation complexity and deployment difficulty, it appears likely that multiple such mechanisms will co-exist.

7. Alternative Identifiers Considered

The "sos" SIP URI reserved user name proposed here follows the convention of RFC 2142 [RFC2142] and the "postmaster" convention documented in RFC 2822 [RFC2822]. One drawback is that it may conflict with locally assigned addresses of the form "sos@somewhere".

There are a number of possible alternatives, each with their own set of advantages and problems:

tel:sos This solution avoids name conflicts, but is not a valid "tel" URI. It also only works if every outbound proxy knows how to route requests to a proxy that can reach emergency services. The SIP URI proposed here only requires a user's home domain to be appropriately configured.

URI parameter: One could create a special URI, such as "aor-domain;user=sos". This avoids the name conflict problem, but requires mechanism-aware user agents that are capable of emitting this special URI.

Special domain: A special domain, such as "sip:fire@sos.int" could be used to identify emergency calls. This has similar properties as the "tel:sos" URI, except that it is indeed a valid URI.

8. IANA Considerations

Subaddresses of the "sos" address are registered with IANA This specification establishes the "sos" subaddres sub-registry under http://www.iana.org/assignments/sip-parameters.

Subaddresses are registered by the IANA when they are published in standards track RFCs. The IANA Considerations section of the RFC must include the following information, which appears in the IANA registry along with the RFC number of the publication.

- Name of the subaddress. The name MAY be of any length, but SHOULD be no more than twenty characters long. The name MUST consist of alphanumeric characters only and is case-insensitive.

- Descriptive text that describes the emergency service.

9. Security Considerations

The SIP specification [RFC3261] details security considerations that apply to emergency calls as well. Security for emergency calls has conflicting goals, namely to make it as easy and reliable as possible to reach emergency services, while discouraging and possibly tracing prank calls. It appears unlikely that classical authentication mechanisms can be required by emergency call centers, but SIP proxy servers may be able to add identifying information.

Given the sensitive nature of many emergency calls, it is highly desirable to use the "sips" URI to ensure transport-level confidentiality and integrity. However, this may cause the call to fail in some environments.

Allowing the user agent to clearly and unambiguously identify emergency calls makes it possible for the user agent to make appropriate policy decisions. For example, a user agent policy may reveal a different amount of information to the callee when making an emergency call. Local laws may affect what information network servers or service providers may be allowed or be required to release to emergency call centers. They may also base their decision on the user-declared destination of the call.

Additional security considerations related to call routing, destination authentication and other issues are detailed in [schulzrinne-sipping-emergency-arch].

10. Acknowledgements

Andrew Allen, Keith Drage, Mike Pierce, James Polk, Brian Rosen and John Schnizlein contributed helpful comments.

Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2806] Vaha-Sipila, A., "URLs for Telephone Calls", RFC 2806, April 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3361] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers", RFC 3361, August 2002.

Informative References

[RFC2142] Crocker, D., "MAILBOX NAMES FOR COMMON SERVICES, ROLES AND FUNCTIONS", RFC 2142, May 1997.

[RFC2822] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

Author's Address

Henning Schulzrinne
Columbia University
Department of Computer Science
450 Computer Science Building
New York, NY 10027
US

Phone: +1 212 939 7042
EMail: hgs@cs.columbia.edu
URI: http://www.cs.columbia.edu

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

Sipping C. Jennings
Internet-Draft Cisco Systems
Expires: August 14, 2004 February 14, 2004

Certificate Discovery for SIP
draft-jennings-sipping-certs-02

Status of this Memo

Copyright Notice

Abstract

This draft describes a scheme in which a SIP user agent can create self signed certificate for use with the SIP S/MIME mechanism and can store the certificate on a web server associated with the address of record (AOR) for the user. Other user agents that want to call that AOR can retrieve these certificates from the web server.

The result of this system is that, with no extra expense or effort for the end user, it is possible to have a reasonable degree of confidence about the identities of the parties in a SIP session.

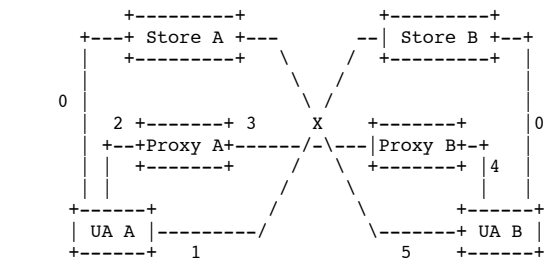
Table of Contents

1. Introduction

SIP RFC 3261 [1] defines an S/MIME based PKI mechanism for achieving end to end security. Among other things, it allows users to be confident that the party they are communicating with is likely the person they want. Like all PKI based schemes, distribution of the public keys is a hard problem. Failure to have a good and widely supported scheme for distributing public keys will result in users not using the S/MIME capabilities of SIP. Not knowing the identities of the other parties in a SIP session greatly reduces the usefulness of encrypted media such as SRTP.

This document describes an approach to using and combining existing schemes to build a trustworthy way of distributing certificates for SIP. An example use case makes this easier to understand. Say Alice meets Bob at a party and Bob says "Call me some time. Here is my AOR." Then Bob writes bob@example.com on the back of a napkin and hands it to Alice. Later Alice makes a call to bob@example.com but she wants to be sure that she really is talking to the person who owns the AOR bob@example.com. This document refers to Alice as the Caller, Bob as the Subscriber, and example.com as the Service.

The overall approach is fairly simple and is illustrated in the figure below. The "store" element in the network is an HTTP web server that is run by the same administrative domain as the proxy.



The goal is for UA A to sign and encrypt a message to UA B using securely acquired self signed certificates. Both sides save their public certificates in a well known store associated with their domain and get the other's certificate from the other domain's store. There are several steps.

- Step 0: At some point in time, both the UA generate a self signed certificate and store it in the the Store for their domain. This is done with a PUT over HTTPS that is digest challenged with the same credentials that are used to register with the proxy in the domain.
- Step 1: UA A fetches the certificate for UA B from Store B. This is done using a GET over HTTPS.
- Step 2,3,4: UA A uses its certificate to sign and UA B's certificate to encrypt and sends a message across the proxies in steps 2,3,and 4 to UA B. This is done using the normal SIP S/MIME bodies.
- Step 5: UA B needs to get UA A's certificate to check the signature. It gets this from Store A using a HTTPS GET. UA B can now decrypt the message and check the signature.

When one of the UA gets a certificate from a Store, the UA must check that the domain name in the AOR in the certificate matches the domain of the Store it is getting the result from. The UA knows this from the certificate presented in the TLS handshake. This one little part makes this scheme significantly different from a typical self signed certificate system. In a classical systems, such as SSH, the first time a certificate is received, there is no automatic way to validate it so the systems must make a "leap of faith" or provide manual out of bound validation which users are typically unwilling to do. This system does not require the leap of faith because the certificate in the TLS session with the store validates that the UA is getting the certificate for UA B from a trustworthy source.

The scheme described in this document meets the goal of allowing Alice to be confident she is communicating with the person with the AOR bob@example.com. It also has the following very desirable properties:

- Trivial to use, requiring no extra effort from the part of the Caller or Subscriber.
- Free in that it does not require any extra expense to the Caller or Subscriber.
- No requirement for a third party to know the Subscriber's private key.
- Allows the Subscriber to have more than one communication device associated with a single AOR.
- Does not require the Service to deploy additional equipment with strict security requirements beyond what they are already running.

None of the problems or ideas presented in this document are new. This presents work going on in the PKIX, SACRED, and SIP working groups in a SIP context and describes an approach to putting the parts together for SIP.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [3].

The term Subscriber refers to an end user that receives calls and has an AOR in a domain that is managed by the Service. The Service provides the SIP proxy and certificate Store. The term Caller refers to the UA that is trying to call the Subscriber. The Caller is often not in the same administrative domain as the Subscriber and therefore has no pre-existing relationship with the Service.

3. Overview

The approach is broken down into Enrollment, Location, and Retrieval phases. The general architecture is that the Service not only provides a SIP registrar service for the Subscriber but also provides certificate storage. In the Enrollment phase, the Subscriber puts their public certificate somewhere that others can find it. In the Locating phase, the Caller discovers where the person they are calling has stored their certificates. Finally in the Retrieval phase, the Caller gets a copy of the Subscriber's certificates. To meet the goal of being free, the certificates are assumed to be self signed.

4. Location and Retrieval

The goal of this stage is to allow the Caller to locate where the Subscriber stores their certificates. The only thing the Caller has is an AOR such as bob@example.com. The obvious solution is to use the host portion of the AOR to find a directory to look up the user portion.

4.1 Location with HTTP

The mechanism for location using HTTP is described in draft-ietf-pkix-certstore-http [2]. The approach first does a SRV lookup and if that fails, it tries a well known host formed from the AOR directly. For the AOR bob@example.com, first an SRV lookup of _certificates._tcp.example.com would be done. If this was successful and returned an address of a.example.com and a port of 7000 then the URL would be:

https://a.example.com:7000/search-cgi?email=bob%40example.com

If the SRV lookup was not successful, then the URL would be formed by adding the host name "certificates" to the domain. In this case the URL would be:

https://certificates.example.com/search-cgi?email=bob%40example.com

4.2 Location with SIP

An alternative scheme to locate the certificates could be based on SIP. The Caller would send an OPTIONS message to the Subscriber proxy. The reply to this would contain a content indirection body [6] or message/external type as defined in RFC 2017 [9] that references a MIME type of application/pkix-cert that could be retrieved using an https URL. The Caller would include a similar content indirection body pointing to their certificate in the messages sent to the Subscriber. This would avoid the need to have some well known URL for locating certificates, and each administrative domain could set up the certificates' locations as it wished.

4.3 Retrieval with HTTP

Once a URL for the certificate is known, the Caller needs to get it. There are several potential protocols that could work for this: HTTP, LDAP, FTP, SNMP, ACAP, and others. The existing tools for making HTTP scale and be reliable, the tools for managing attacks on servers, and the existing support for hardware acceleration of HTTPS make this a good choice from the server point of view. The ease of working through NATs and firewalls along with the fact that most SIP UAs need to implement HTTP for other reasons make it a good match on the client end. The MIME types in HTTP are useful for dealing with the various types of certificates. These points led to the selection of HTTPS as defined in draft-ietf-pkix-certstore-http [2] as a mechanism for getting the certificates. Getting the certificate with HTTP is defined in RFC 2585 [5] and will be in a MIME type of application/ pkix-cert and contain a DER encoded X509 certificate./

Since the certificates may be self signed, the Caller needs to be sure that they were not tampered with and that they came from the Service that was authorized to provide them. This means that the Caller MUST use HTTPS to get the certificate and the Service MUST present a certificate in the TLS handshake that has a domain name in the SubjectAltName field that matches the domain name in the AOR in the SubjectAltName in the retrieved certificate. In this example the original is example.com, not the result of any SRV lookup. The names are considered to match if the SubjectAltName matches the host

portion of the AOR using a case insensitive comparison. Sub-domains do not match. IP addresses do not match host names.

4.4 Multiple UAS for a Single AOR

It is possible to retrieve a list of several certificates for the same AOR when there are several different UA that may receive messages for this AOR. In this case the UA sending the messages needs to use every valid certificate it received for the public key operations. A certificate Store SHOULD not provide certificates that have become invalid.

4.5 Steps to Locate and Retrieve a Certificate

Both the Caller and Subscriber UA need to retrieve the other's certificate from the appropriate Store. This is done with the following steps:

- Determine the AOR of the certificate that is needed - for example, alice@example.com.
- Do a DNS SRV lookup for the service _certificates with a protocol of _tcp in the domain of the AOR. (In this example this would result in a DNS SRV query in the domain example.com). If this is found, form a URL using the hostname and port returned. If not, form the URL by using the default port for HTTPS and a hostname of certificates prepended to the the domain from the AOR. (In this example this would result in a hostname of certificates.example.com)
- Use the host and port found in the previous step to form a URL of the form "https:://host:port/serach-cgi?email=aor" where the "aor" is replaced with an appropriately escaped version of the AOR. For this example, this would become "https://certificates.example.com/search-cgi?email=alice%40example.com"
- Open a TLS connection to this URL. TLS extended hellos to indicate the requested domain SHOULD be used. The server MUST return a certificate with a SubjectAltName that matches the domain portion of the original AOR (example.com in this example). The UA MUST check this matches and if it does not, it must close the connection and not proceed.
- The UA then performs an HTTP GET on the URL. The Store returns the one or more bodies or an error if it has no certificates for this Subscriber. Each certificate is in an DER encoded X509 certificate and is in a body of type application/pkix-cert. A transfer encoding of binary is used.

- The UA MUST check that all the SubjectAltNames in all the certificates have a user and host portion that matches the original AOR. Schemes other than SIP are acceptable. In this example, a SubjectAltName that contained two URIs, "im:alice@example.com" and "sip:alice@example.com" would be acceptable. Any certificates that do not match MUST be discarded.
- The UA MUST check the expiry dates on the certificates. Any expired certificates MUST be discarded.

The UA now has a usable list of certificates for the AOR. If the UA is using them to decrypt, it uses the serial number and issuer to find the certificate it needs to decrypt the information. If it is using the certificates to encrypt some information, it must encrypt the CEK with each of the certificates so that the a UA in possession of the private key from any one of the certificates can decrypt the material.

5. Enrollment

The Subscriber must be able to authenticate to the Service and must be able to transfer the certificate in an integrity protected way to the Service. In SIP, the Service and the Subscriber already have a shared secret that is used for authentication during SIP registration; or the Service knows the certificate of the Subscriber by some out of band mechanism. This shared secret can be leveraged for enrollment of the Subscriber's public certificates.

The Subscriber would transfer acertificate to the Service using an HTTPS PUT with the same URL that would be used to get their certificate. This MUST happen over HTTPS so the transfer is integrity protected. The client MUST also check that the server's certificate name matches the name of the Subscriber's AOR. This matching follows the same rules as matching in retrieval of certificates. The client MUST authenticate to the server using DIGEST authentication with some shared secret. The same shared secret that is used for SIP registration SHOULD be used. This allows any Subscriber to generate a self signed certificate and store it at the Service. Note that authorization with TLS mutual authentication is not considered because in that case the Service already has the Subscriber's Certificate and there is no need to transfer it.

There is an additional problem of how to allow a user that has several communication clients to associate them all with the same AOR and still get the certificates to work. There are at least two approaches to this problem. One would be to upload a different certificate for each UA associated with the AOR and just let the Caller use all of them. This is the approach that is chosen here. The

other approach would be to use the work from the SACRED working group[8] which is solving the problem of security getting the same credential on all the clients.

In the chosen approach of using many certificates for a single AOR, the Caller would first get all the certificates from the Service. It would then send an INVITE to the Subscriber and sign it with its own certificate and encrypt the SDP (or whatever part of the messages was being encrypted) with each of the certificates retrieved. No matter which of the Subscribers UA's received the message, that UA would be able to decrypt the information.

The Service MUST provide some other authenticated, out of band mechanism for the Subscriber to revoke certificates. A web page accessed over HTTPS with digest authentication would work fine for this. A HTTPS DELETE with digest could work but there needs to be a way to tell which certificate needs to be deleted when the AOR has multiple certificates.

It is RECOMMENDED that the clients use fairly short-lived certificates (in the order of days to months) and enroll a new certificate before the old one expires. The Caller MAY cache the certificates that they retrieved for an AOR and use them in future calls. This cached result MUST expire after some short but configurable amount of time so that certificate revocation works. It MUST be possible to configure this time to be zero. If the Caller is using cached information and receives a certificate in the SIP signaling that is not cached, the Caller MUST update the cache and check that the certificate was not recently added to the Service.

When a UA registers, it SHOULD retrieve the certificates for its AOR and check that this UA's certificate is correctly enrolled. The HTTPS server MUST support a profile of TLS_RSA_WITH_AES_128_CBC_SHA as described in RFC 3268 [4] or a profile of TLS_RSA_WITH_3DES_CBC_SHA .

5.1 Steps to Enroll

The Subscriber UA needs to generate a self signed certificate and save it in the store. This is performed in the following steps:

- When the UA starts up, it needs to fetch its own certificate and check that it matches the certificate stored on the UA. If it does not, it should warn the user and generate a new certificate.
- The UA should check the expiration and arrange to generate a new certificate before the old one expires.
- TODO: Describe details of generating a self signed certificate.

- The UA forms a URL in the same way as locating a certificate but using its own AOR.
- The UA opens a a TLS connection and verifies the certificate returned in the TLS the same way as retrieving a certificate.
- The UA then does a HTTPS PUT of the certificate. The server MUST digest challenge this request. The UA computes the response to this digest and MAY use the same username and password as it would use to register with the proxy in this domain.
- The server must check that, for each URI in the SubjectAltName in the certificate, the user portion matches the username used in the digest authentication and the host portion matches the domain used for the TLS connection.
- When the certificate is close to expiring, the UA should create and store a new certificate.

At this point the UA has successfully stored its certificate in the Store. The Store may discard any certificates that have expired.

6. Delegated Crypto with Content Indirection

If the Subscriber or Caller wishes to use an authentication service to insert and verify S/MIME bodies on their behalf, they can do so by using content indirection [6] to specify URLs for the S/MIME bodies that can be filled in by the authentication service.

TODO - This needs significantly more detail if it is to be used

7. Security Considerations

This whole document is focused on security and must be considered from a security point of view.

It is important to remember that the scheme relies upon the Subscriber choosing a Service that does not lie. The Subscriber may wish to use contractual obligations to enforce this.

7.1 Security Analysis

This whole scheme is made possible because the Subscriber has a shared secret with the Service, the Service has a certificate that is signed by a well known certificate authority, and the Caller knows how to find the Service for the Subscriber they are calling.

To look at the security of this scheme one must consider the existing

SIP S/MIME trust model and what the trust relationships are. If Alice tells a secret to Bob, Bob can tell anyone. If Bob signs something and sends it to Alice, Alice can only believe this signature as much as she believes that Bob has securely managed his private key and has not posted it on an IRC channel. If Bob tells Alice that his AOR is bob@example.com, that may change in the future and someone else may get that AOR. Just because Alice manages to get a valid certificate bound to the AOR bob@example.com does not mean that Alice is going to talk to the right Bob. This last point is important in understanding why the scheme presented here is not significantly less secure than the use of S/MIME certificates in SIP that are signed by a well known certificate authority. All SIP has is the AOR - SIP can check that the name in the certificate matches the AOR but it can not check other things that are likely to make the identity unique. If the Service example.com gave the AOR bob@example.com to a new Bob, they would likely give away the email address bob@example.com to the new Bob as well. Furthermore, the certificate authority, after revoking the old certificates, would probably give the new Bob a new certificate if the new Bob could read email sent to the AOR. Alice would be talking to bob@example.com - but the new Bob instead of the old Bob.

The point of this is that you have to trust that the person providing your AOR will not give your AOR to someone else. Bob has some ability to choose a Service he trusts. He can enforce this contractually with the Service and by choosing one worthy of trust. Alice has to trust Bob on many things including that he picked a trustworthy party to manage his AOR and that he manages his private key appropriately.

If the Subscriber can trust the Service to manage the Subscriber's AOR, then the Subscriber can trust the Service not to lie about certificates they store for the Subscriber. If the Service wants to subvert Bob's communications, they can likely do this by getting a certificate authority to give them a certificate masquerading as Bob. The security of this scheme relies on the Service not lying about what Bob's public certificates are. If you buy this, the rest is fairly simple.

Only Bob's UAs have the shared secret to authenticate to the Service to upload a certificate. The UA will not accidentally authenticate to a rogue service because the UA checks the certificate the Service presents in TLS. The certificate is not tampered with because the HTTPS connection is integrity protected. When the Caller retrieves a certificate they know it is coming from the correct Service because the Service must have the certificate for the domain that represents the host portion of the AOR. The Caller knows the certificate was not tampered with in transit because the connection is integrity protected.

Certificates can be quickly revoked because the Caller gets the certificates on each new call to the Subscriber. This side steps some thorny CRL issues. The impact of getting these each time will probably make a relevant difference on the load of the Service's servers but does not make the scheme unworkable.

The Subscriber's UAs can use short lived self signed certificates. In fact UAs could upload a new certificate each time they boot. This would eliminate the need for UAs to store the private keys in NVRAM which might be a security advantage.

8. Open Issues

Is there a need for a SIP response code that indicates that a bad certificate was used and that the user should flush this certificate from their cache and try again?

It is likely that SIP requires a certificate separate from the one used for email. This would require an HTTP get of:

```
https://a.example.com:7000/search-cgi?sip=bob%40example.com
```

This is likely needed.

9. Comparison with Identity

The ietf-sip-identity [7] draft is about allowing the Service to assert the identity of a Subscriber to others. It does not deal with signing or encrypting messages from one user to another which is the focus of this draft. It does make the same primary assumption that the Service is trusted by the Subscriber and that the service is trustworthy enough to adequately authenticate the Subscribers.

10. IANA Considerations

There are no IANA considerations.

11. Conclusion

The procedure described in this document is easy and it can happen automatically with no extra expense or intervention from the Subscriber or Caller. It is easy for the Service to provide and does not require them to do much beyond running a normal HTTPS web service suitable for e-commerce application. It achieves about as good a job of identifying the participants of a call as the SIP S/MIME mechanism is capable of achieving. It does not require any modification of existing protocols or the invention of any new ones.

12. Acknowledgments

Many thanks to Eric Rescorla, Peter Gutmann, Rohan Mahy and Jason Fischl for comments.

Normative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Gutmann, P., "Internet X.509 Public Key Infrastructure Operational Protocols: Certificate Store Access via HTTP", draft-ietf-pkix-certstore-http-05 (work in progress), March 2003.

[3] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[4] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[5] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", RFC 2585, May 1999.

Informational References

[6] Olson, S., "A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages", draft-ietf-sip-content-indirect-mech-03 (work in progress), June 2003.

[7] Peterson, J., "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-01 (work in progress), August 2003.

[8] Gustafson, D., Just, M. and M. Nystrom, "Securely Available Credentials - Credential Server Framework", draft-ietf-sacred-framework-07 (work in progress), November 2003.

[9] Freed, N. and K. Moore, "Definition of the URL MIME External-Body Access-Type", RFC 2017, October 1996.

Author's Address

Cullen Jennings
Cisco Systems
170 West Tasman Drive
MS: SJC-21/2
San Jose, CA 95134
USA

Phone: +1 408 902-3341
EMail: fluffy@cisco.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING WG
Internet-Draft
Expires: August 7, 2004

C. Jennings
Cisco Systems, Inc.
February 7, 2004

Instance Identifiers for SIP User Agents
draft-jennings-sipping-instance-id-00.txt

Status of this Memo

Copyright Notice

Abstract

There are places in building SIP [2] based communications systems where it is useful to have a stable identifier for particular user agents that are used for user communications. This draft defines a convention for names that can be used to satisfy these needs.

Table of Contents

1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [3].

2. Introduction and Use Cases

There are a few cases in which it is convenient to be able to identify instances of a user agent. Some examples are described. They all require the name to be stable across reboots of the device.

In the config framework[4], a user agent sends a subscribe to fetch its configuration. It needs to get the same configuration each time.

A particular user, Alice, has several user agents that all register as Alice. A registrar wishes to report which user agent are currently registered to a network management system. For this reporting to make sense, each of Alice's user agents must have a stable name.

A system that is using the dialog package to monitor a particular user agent would like to be able to assign an alias like "My Office Phone" for display purposes to that particular user agent.

When several presence user agents are providing presence data, it must be possible to correlate a particular set of data with the particular device that provided it.

In all these cases, the user agent could be a software program running on a computer with more than one user.

3. Requirements

The identifier needs to be unique.

Identifiers are needed for user agents that are in dedicated pieces of hardware such as IP phones.

Identifiers are needed for software user agents running on multi-user computers.

In some of the cases with IP phones, it is desirable for this same identifier to be recorded as a bar code on the outside of the box that the IP phone comes in.

4. Solution

User agents that follow the convention of this document MUST put a unique identifier in a new tag, called "instance", of the Contact header when sending a SIP request. They MAY omit this for a particular sequence of SIP messages if the user has requested it be removed for privacy reasons.

The unique identifier has no real semantic information other than uniqueness. In cases in which the user agent runs on a single computer and this is the only user agent on that computer, the MAC address of the primary network card is the preferred identifier. In cases in which it is impossible to use the MAC address, then when the user agent is first run, it should generate a random 64 bit number and use this as the identifier. It MUST store this number in some non volatile storage that is stable over reboots and power outages. The user agent SHOULD use the same instance identifier tag even if it is registering different AOR or contacts.

If the identifier is a MAC address, it MUST be formatted as the letters "MAC-" followed by a 12 digit hexadecimal representation of the MAC address. The address can not include ":", whitespace, or other formatting. If the identifier is a random number, it MUST be formatted as the letters "RANID-" followed by a 16 digit hexadecimal representation of the number. Note that the identifiers are case sensitive and all alpha characters are upper case.

The MAC and RANDID identify the namespace for the unique identifier. In the future this unique identifer namespace may be extended with other namespaces that use unique identifiers from things like USB, Bluetooth, or Firewire.

These same identifiers may be used in the user portion of request URIs when that is appropriate. A SUBSCRIBE for configuration information is a good example.

5. Discussion

The contact header in a SIP request identifies an address that can be used to reach the device that is sending the request. This address may change each time the device running the user agent gets a new IP address, but it is very reasonable for the display name to give a unique identifier for what this instance of the user agent wishes to be known by. Right now SIP does not give any recommendation on what to place in the field. This document suggests a naming convention for this.

MAC addresses are usually put on the outside of the box for IP phones

in a form that humans can read and also by a barcode scanner.

6. BNF

The following ABNF follows the rules in RFC-2234 [1] and updates the BNF in RFC 3261.

```
contact-params = c-p-q / c-p-expires / c-p-instance
                 / contact-extentions
c-p-intance = "instance" EQUAL uniq-ident
UHEX  =  DIGIT / %x41-46 ;uppercase A-F
MAC  =  %x4d.41.43 ; MAC in caps
RANDID  = %x52.41.4e.44.49.44 ; RANDID in caps
uniq-ident = ( mac-ident / rand-ident )
mac-ident = MAC "-" 12UHEX
rand-ident = RANDID "-" 16UHEX
```

7. Example

The following are some valid Contact headers:

```
Contact: <sip:alice@host22.example.com>;instance=MAC-123456789ABC
Contact: <sip:alice@host22.example.com>;instance=
         RANDID-0123456789ABCDEF
```

8. Security Consideration

The unique identifer reveals further privacy related information to other people that see the SIP signalling. Currently user agents put an IP address or DNS name in the contact header, so the amount of extra information this reveals is very minimal. The MAC address may reveal the manufacturer of the user agent.

9. Open Issues

Would this be better in an "Instance-ID" header?

Would this be better in the User-Agent header? Some systems are doing already doing this.

Is 64 bits the right size for the random identifier?

Is requiring upper case appropriate?

10. Acknowledgments

Many thank for the useful comments and improvements from Louis Pratt, Steve Levy, Rohan Mahy, and Randy Baird as well as the list discussion from Jonathan Rosenberg.

Normative References

[1] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[3] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Informative References

[4] Petrie, D., "A Framework for SIP User Agent Configuration", draft-ietf-sipping-config-framework-00 (work in progress), March 2003.

Author's Address

Cullen Jennings
Cisco Systems, Inc.
170 West Tasman Dr.
MS: SJC-21/2
San Jose, CA 95134
USA

Phone: +1 408 902 3341
EMail: fluffy@cisco.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING Working Group
Internet-Draft
Expires: August 15, 2004

A. Johnston
H. Sinnreich
MCI
A. Clark
Telchemy Incorporated
A. Pendleton
Nortel Networks
February 15, 2004

RTCP Summary Report Delivery to Third Parties
draft-johnston-sipping-rtcp-summary-02

Status of this Memo

Copyright Notice

Abstract

This document discusses the motivation and requirements for the delivery of RTCP extended reports and other summary reports to non-participants in the session. Several solution mechanisms are also discussed and compared. A SIP events package is proposed as a solution. An event package "rtcp-xr" is defined in this document along with some example call flows.

Table of Contents

1. Introduction

There is a general need for real-time reporting of session quality in enterprise and service provider networks. While the approach discussed in this document is quite general, this document is limited in scope to the delivery of particular RTCP summary reports.

RTP Control Protocol (RTCP) [3] defines Sender Reports (SR) and Receiver Reports (RR) which are exchanged between the participants in a media session about the quality of the media session. RTCP Extended Reports (XR) [4] have also been defined to provide additional quality information. In particular, two summary reports are included: a statistics summary report and a VoIP (Voice over IP) metrics block.

This summary information is of particular interest to certain parties who may not be participants in the media session. For example, a service provider might be interested in logging a summary report of the QoS of a VoIP session. Alternatively, an enterprise might want to compile a summary of the QoS of multimedia sessions established over a wide area network.

In the case of a gateway or other high-density device, the device is likely to implement various AAA protocols and have the ability to log and export this type of RTCP summary reports. However, this is not practical in smaller endpoints such as SIP phones, clients, or mobile phones.

This document discusses the requirements of a mechanism to allow a third party which is not a participant in a session receive RTCP summary reports. Three possible mechanisms are discussed at a very high level.

The SIP events approach is found to be the best solution, and an event package is defined. Some sample call flows are also included.

2. Requirements

REQ-1: An authorized third party should be able to receive selected RTCP reports on a near real time basis.

REQ-2: The client should not have to store large amounts of information.

REQ-3: The client must be able to authenticate the third party.

REQ-4: The RTCP report information must be able to be transferred securely.

REQ-5: A client participating in a bi-directional session will store and send RTCP summary reports for both directions.

REQ-6: The reports will include or be associated with dialog identifiers for correlation purposes.

3. Possible Mechanisms

Four possible mechanisms could be used implement these requirements:
- o Forking RTCP to multiple locations,
- o SNMP,
- o Carrying RTCP information in a SIP header field or message body,
- o Using an events package to delivery RTCP information.

3.1 Forking RTCP

In general, one RTCP session is established per RTP media session. That is, if a session consists of a voice stream and a video stream, two separate RTCP sessions will be established in which the participants exchange QoS and other data. The RTCP reports are sent to the same IP address as the RTP media but the next higher port number. (There is also an extension [5] to SDP to explicitly list the RTCP IP address and port number.)

While RFC3550 (RTP/RTCP) proposes multicasting RTCP sessions, what is missing is a mechanism for communicating correlation identifiers for purposes of determining which reports are associated with each other, particularly where source/dest IP/port are not globally unique. Also, there is not a means for establishing an association between the session participants and a collector. In addition, authentication also not covered.

An extension to send RTCP reports to multiple locations could be defined. If this were implemented in an endpoint, the RTCP reports sent and received in a session could be sent to a third party which would listen on a particular IP address and port number.

An obvious difficulty of this approach is how the third party would signal this IP address and port number to the endpoint during session setup. A 3pcc could insert this extra information (in an SDP extension attribute) in the SDP at the time of call setup. However, there is no good solution for the peer-to-peer model without forcing a proxy to act as a B2BUA and modify SDP.

Another drawback is the lack of security in this approach.

This approach would not require any extensions to SIP but may require extensions to SDP and RTCP for mechanisms to signal the transport IP

address and port number of the third party.

3.2 SNMP

Since this type of QoS monitoring seems related to management, SNMP could possibly be used to collect this type of data. In general, SNMP may be used to manage the SIP user agent - the phone, soft phone or gateway. However, the information available in RTCP summary reports is of less interest to the management of the UA and more of interest to the VoIP service provider. In many cases, separate entities will be involved. For example, an enterprise may manage their own SIP phones using SNMP, but a service provider provides SIP and gateway services. It is unlikely a service provider will have SNMP privileges and may not be able to manage NAT/firewall traversal, etc. For these reasons, SNMP is not a good fit for this "service level" management function.

The next two approaches are closely coupled to SIP, which overcomes the disadvantages of the non-SIP approaches.

3.3 SIP Header Field or Message Body

In this approach, the desired RTCP reports could be carried in a SIP [6] request or response message which would then be available to proxies which had Record-Routed the dialog. For example, summary RTCP reports could be carried in a BYE message at the end of the session. Since the requirement is to make the information available to intermediary third parties, the information would best be carried in a header field rather than a message body. The compact nature of the binary encoded reports would not rule out inclusion in a header field.

The main disadvantage of this approach is that any third parties would need to Record-Route in order to receive the reports. Also, if the header field were only transported in an S/MIME encrypted message body, the information would not be available to the intermediaries. Finally, while the inclusion of this information at the end of a session in a BYE seems a good choice, there is no good candidates for mid-session delivery of this information (INFO would NOT be a good choice for this) although a re-INVITE could be used.

3.4 SIP Event Package

In this approach, a new SIP events package [6] would be defined. A third party could subscribe to the participant to receive notifications of RTCP reports transported using the NOTIFY method.

An advantage of this approach is that the third party does not need

to be a proxy that has Record-Routed a particular dialog. The SUBSCRIBE request from the third party can use any of the set of standard SIP authentication mechanisms to authorize the third party. In addition, the reports transported using NOTIFY can use TLS or S/MIME to secure the transport of the report data.

During the establishment of the subscription, the third party could request the type and frequency of RTCP reports. The event package could also define the rate limitations.

The subscription could either be for a particular dialog, in which the subscription would expire at the termination of the session. The third party could then subscribe to the dialog package to receive notifications whenever the endpoint began a new session, providing the third party the information about the session sufficient to make a decision as to whether to subscribe to the RTCP report package for this particular dialog. Alternatively, the subscription could be temporally bound in which the third party would receive notifications from all dialogs until the subscription expired.

A disadvantage of this approach is that that the endpoint must manage the subscription and support SIP events and the RTCP report event package. A third party wishing to receive reports from multiple endpoints would need to manage multiple subscriptions.

4. Event Package Formal Definition

4.1 Event Package Name

This document defines a SIP Event Package as defined in RFC 3265 [2]. The event-package token name for this package is:

"rtcp-xr"

OPEN ISSUE: Should a more general name be used so that different message bodies can be defined to carry different session information?

4.2 Event Package Parameters

The event package parameter "threshold" if present indicates that the subscriber wishes to receive mid-session threshold reports. That is, if the quality of the session degrades beyond a locally configured value during a session, the notifier should send a NOTIFY message.

If the event package is not present, the default is not to send the threshold reports, but to send a single NOTIFY at the end of each media session.

OPEN ISSUE: Is this the best way to do this or should a filter message body be defined?

OPEN ISSUE: Ideally, the threshold value could be negotiated during the establishment of the subscription, but this seems hard.

4.3 SUBSCRIBE Bodies

No SUBSCRIBE bodies are described by this specification.

4.4 Subscription Duration

Subscriptions to this event package MAY range from minutes to weeks. Subscriptions in hours or days are more typical and are RECOMMENDED. The default subscription duration for this event package is one hour.

4.5 NOTIFY Bodies

There are two notify bodies: a general report and a threshold report. The general report is used for periodic, mid-call reporting and end of call reporting. The general report can include both local and remote metrics.

The threshold report is used when call quality degrades. The general report is also included in the alert report to provide all of the necessary diagnostic information.

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC-2234 [7].

OPEN ISSUE: The message body should probably be a MIME type.

General Report Event:

```
VQEvent  =  LocalMetrics CLRF
            RemoteMetrics

LocalMetrics = ("LocalMetrics") HCOLON VoiceQualityMetrics
RemoteMetrics = ("RemoteMetrics") HCOLON VoiceQualityMetrics


VoiceQualityMetrics = ("VQMetrics") HCOLON CLRF
                      PacketLossMetrics CRLF
                      BurstMetrics CLRF
                      GapMetrics CLRF
                      DelayMetrics CLRF
                      SignalMetrics CLRF
                      QualityScores CLRF
```



```
PacketLossMetrics = ("plm") EQUALS loss-rate SPACE discard-rate
loss-rate         = ("loss") HCOLON HEX (HH)
discard rate = ("disc") HCOLON HEX (HH)

BurstMetrics = ("burst") EQUALS density SPACE length
GapMetrics = ("gap") EQUALS density SPACE length
density         = ("den") HCOLON HEX (HH)
length         = ("len") HCOLON HEX (HHHH)

DelayMetrics = ("delay") EQUALS round-trip SPACE end-system
round-trip = ("rt") COLON HEX (HHHH)
end-system = ("es") COLON HEX (HHHH)


SignalMetrics = ("signal") EQUALS signal SPACE echo-return-loss SPACE
noise
signal  = ("sig") HCOLON HEX (HH)
echo-return-loss = ("erl") HCOLON HEX (HH)
noise  = ("n") HCOLON HEX (HH)


QualityScores  = ("qs") EQUALS r-factor SPACE ext-r-factor SPACE mos-lq
SPACE mos-cq
r-factor      = ("r") HCOLON HEX (HH)
ext-r-factor   = ("xr") HCOLON HEX (HH)
mos-lq         = ("ml") HCOLON HEX (H"."H)
mos-cq         = ("mc") HCOLON HEX (H"."H)

DialogID       = ("DialogID") HCOLON callid *(SEMI dialogid-param)
dialogid-param  = to-tag / from-tag / generic-param
callid          = token
to-tag          = "to-tag" EQUAL token
from-tag        = "from-tag" EQUAL token
```

Alert Format:

```
VoiceQualityAlert = ("VQAlert") HCOLON SPACE ViolationMetric CRLF
                    VoiceQualityMetrics


ViolationMetric = ("AlertType") HCOLON ("rf"| "burst" | "erl" | "delay"
                          token )
```

OPEN ISSUE: Is this format human readable enough?

4.6 Metric Definitions

See RFC 3611 [4] for a full description of these metrics.

Packet Loss Ratio
The fraction of packets lost within the network.

Packet Discard Rate
The fraction of packets discarded due to jitter.

Burst Density
The fraction of packets lost and discarded within a burst (high loss rate) period.

Burst Length (mS)
The mean length of a burst.

Gap Density
The fraction of packets lost and discarded within a gap (low loss rate) period.

Gap Length (mS)
The mean length of a gap

Round Trip Delay (mS)
The round trip delay between RTP interfaces

End System Round Trip Delay (mS)
The "round trip" delay between the RTP interface and the analog or trunk interface.

Signal Level (dBm)
The signal level during talkspurts.

Noise Level (dBm)
The signal level during silence periods.

Residual Echo Return Loss (dB)
The residual (uncancelled) echo level from the analog or trunk interface.

R Factor
Estimated conversational call quality expressed in R factor terms.

External R Factor
An estimate of the call quality from an externally attached

network.

MOS-LQ
Estimated listening call quality expressed as a MOS score

MOS-CQ
Estimated conversational call quality expressed as a MOS score

4.7 Format Example

Call Alert Scenario

```
NOTIFY sip:collector@chicago.example.com SIP/2.0
Via: SIP/2.0/UDP pc22.example.com;branch=z9hG4bK3343d7
Max-Forwards: 70
To: <sip:collector@chicago.example.com>;tag=43524545
From: Alice <sip:alice@example.com>;tag=a3343df32
Call-ID: k3l43id034kevnx7334s
CSeq: 4321 NOTIFY
Contact: <sip:alice@pc22.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
 SUBSCRIBE, NOTIFY
Event: rtcp-xr
Accept: application/sdp, message/sipfrag
Subscription-State: active;expires=3600
Content-Type: text/plain
Content-Length: ...

Event:rtcp-xr
AlertType:rf
LocalMetrics:VQMetrics:
plm=loss:05 disc:02
burst=den:0 len:0
gap=den:2 len:0
delay=rt:200 es:140
signal=sig: rerl: n:
qs=r:82 xr:82 ml:3.4 mc:3.3
DialogID:38419823470834;to-tag=8472761;from-tag=9123dh311
```

This alert indicates that the quality of the call in progress has degraded to an unacceptable level. In this case, the packet loss rate was 5%, the packet discard rate (due to jitter) was 2%, there were no bursts, the gap loss/discard rate was 2%, the round trip delay was 160mS, the end system delay was 140mS, the R factor was 85, the MOS-LQ 3.6, the MOS-CQ 3.5. In this case, the remote metrics were unavailable and therefore not included.

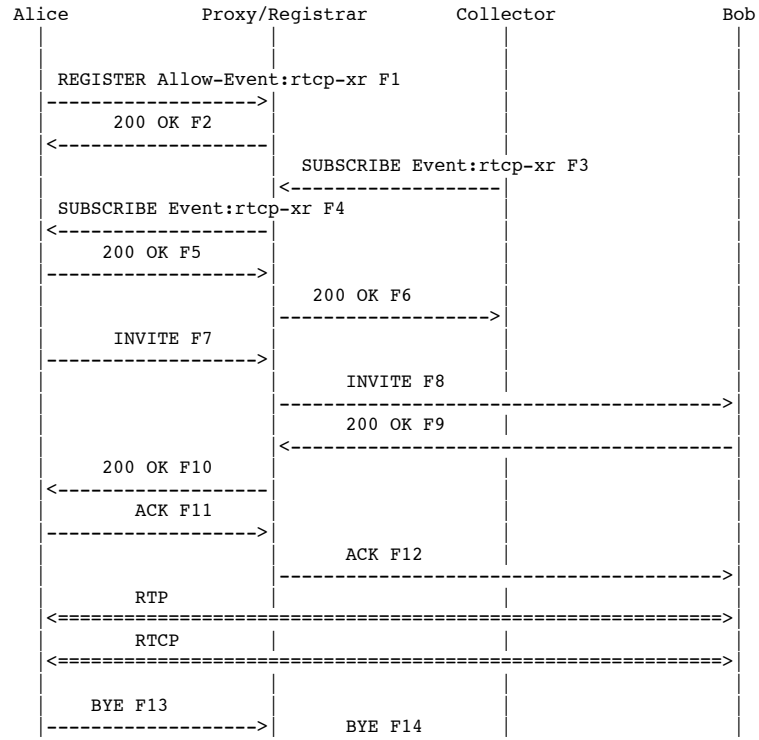
5. Call Flow Examples

This section shows a number of call flow examples showing how the event package works.

These flows assume that the summary report collector is notified by the registrar when a new User Agent registers which supports the event package.

OPEN ISSUE: The ways in which this can be done should probably be discussed in the document.

5.1 End of Session Notification Call Flow



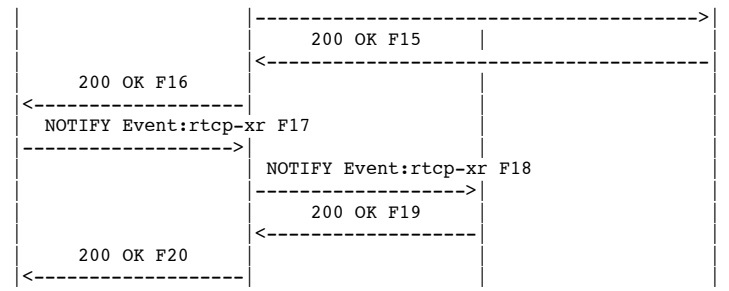
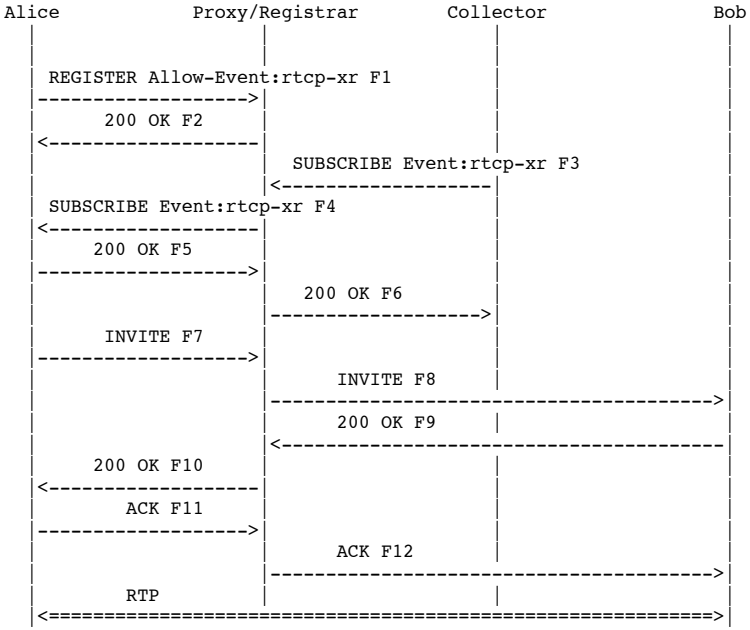


Figure 1. Summary report sent after session termination.

5.2 Mid Session Report



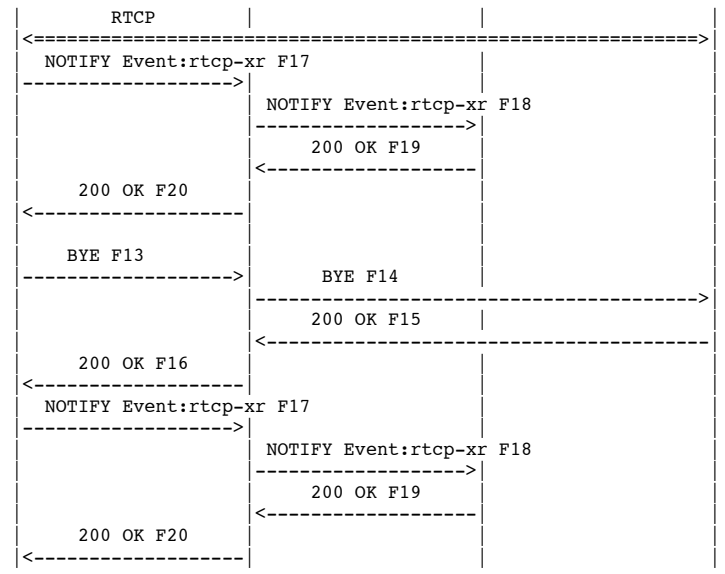


Figure 2. Summary report sent during session with threshold report.

6. Security Considerations

RTCP reports can contain sensitive information since they can provide information about the nature and duration of a session established between two endpoints. As a result, any third party wishing to obtain this information should be properly authenticated and the information transferred securely.

7. Contributors

The authors would like to thank Dave Oran and Tom Redman for their discussions.

Informative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[3] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

[4] Friedman, T., Caceres, R. and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, November 2003.

[5] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, October 2003.

[6] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[7] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

Authors' Addresses

Alan Johnston
MCI
100 South 4th Street
St. Louis, MO 63104

EMail: alan.johnston@mci.com

Henry Sinnreich
MCI
400 International Parkway
Richardson, TX 75081

EMail: henry.sinnreich@mci.com

Alan Clark
Telchemy Incorporated
3360 Martins Farm Road, Suite 200
Suwanee, GA 30024

EMail: alan@telchemy.com

Amy Pendleton
Nortel Networks
2380 Performance Drive
Richardson, TX 75081

EMail: aspen@nortelnetworks.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIMPLE WG
Internet-Draft
Expires: May 1, 2004

O. Levin
Microsoft Corporation
Nov 2003

Ad-hoc Resource Lists using SUBSCRIBE in SIMPLE
draft-levin-simple-adhoc-list-01.txt

Status of this Memo

Copyright Notice

Abstract

This document presents an extension to the Session Initiation Protocol (SIP)-Specific Event Notification mechanism for subscribing to a homogeneous list of resources. Instead of the subscriber sending a SUBSCRIBE for each resource individually, the subscriber can define an ad-hoc resource list, subscribe to it, and maintain it ? all within a single SUBSCRIBE dialog. Changes in the state of the resources are reported using NOTIFY within the same dialog in any standard SIMPLE format for conveying notifications for lists of resources or for individual resources and specified as "accepted" during the SUBSCRIBE dialog establishment.

Table of Contents

1. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [2].

This document uses the Resource List Server (RLS) definition from [8].

2. Introduction

This document presents an extension to the Session Initiation Protocol (SIP)-Specific Event Notification mechanism for subscribing to a homogeneous list of resources. Instead of the subscriber sending a SUBSCRIBE for each resource individually, the subscriber can define an ad-hoc resource list, subscribe to it, and maintain it ? all within a single SUBSCRIBE dialog. Changes in the state of the resources are reported using the same dialog in any standard SIMPLE format for conveying notifications for lists of resources (such as [8] and [9]) or individual resources (such as [6]) and agreed during the SUBSCRIBE dialog establishment.

This document defines a new XML schema for creation and maintenance of a homogeneous list of resources and a new SIP option tag for expressing support for this schema.

3. Overview of Operation

Before using the ad-hoc list subscription, a subscriber needs to know that the RLS supports this mode of operation. In order to do so, the subscriber SHOULD issue OPTIONS request and ensure that the SIP "adhoclist" option tag is included in the RLS response.

Once the subscriber knows that the RLS supports ad-hoc list operations, a Require header MUST be set to "adhoclist" and an initial resources list can be included in the SUBSCRIBE.

It is the responsibility of the subscriber to create the list name. The "adhoclist" attributes "uri" and "name" together MUST uniquely identify the list within the SUBSCRIBE dialog. The value in the Request URI in SUBSCRIBE MAY match the value of "uri" attribute of the "adhoclist". The RLS MUST store and maintain the ad-hoc list for the life of the SUBSCRIBE dialog.

Notification format is negotiated by including appropriate SIP option tags in the Accept headers of SUBSCRIBE and corresponding Responses and is out of scope of this document.

The subscriber can update the list by using the defined "create", "add", and "delete" primitives. "create" primitive means that the old list MUST be removed and replaced with the new data. Consequently, "create" primitive with an empty list means that the old list MUST be emptied. Adding an already existing resource MUST result in triggering of appropriate notifications.

Deleting of a non-existent resource SHOULD NOT result in an error condition. If a subset of the resources specified in (re-)SUBSCRIBE cannot be served within the context of this dialog, error or redirection indications (preferably specified per resource) SHOULD be reported using subsequent NOTIFY(s) in accordance with the notification format(s) established for this SUBSCRIBE dialog.

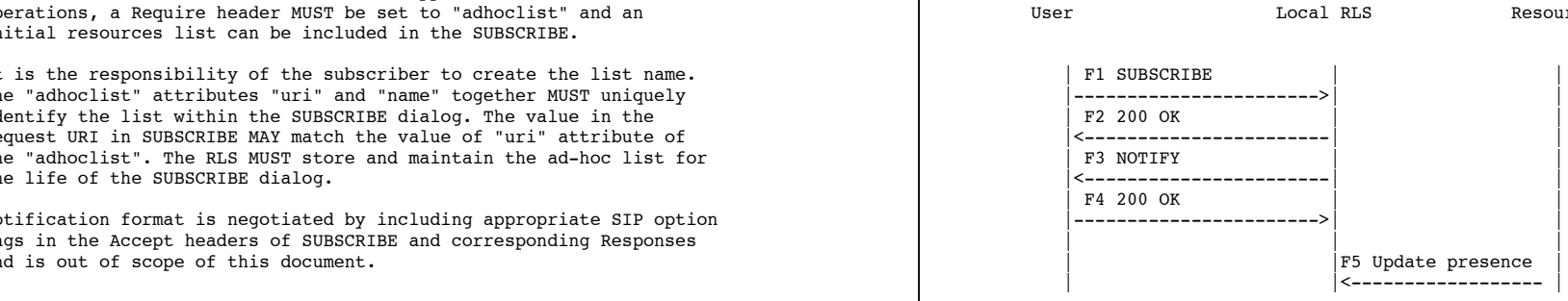
When any in-band operation is performed on a resource in the list, the RLS SHOULD generate a NOTIFY as if the operation has been performed by any possible out-of-band means and in accordance with the notification mechanism and the format established for this dialog.

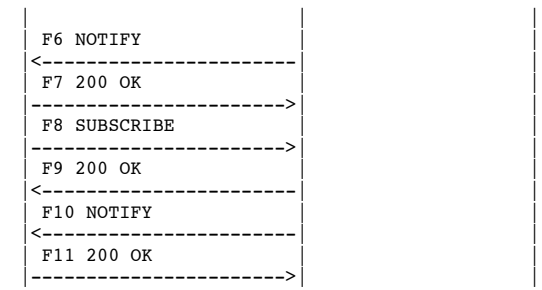
4. Example Message Flow

This message flow illustrates how a User (i.e. watcher) subscribes to its Presence Server for a list of resources, receives presence information about the resources, manipulates the list and receives updated information about the resources.

When the value of the Content-Length header field is "..." this means that the value is whatever the computed length of the body is.

For brevity some of the headers (e.g. Via, CSeq, and Max-Forwards) are omitted from the message flows.





F1 SUBSCRIBE user (watcher) -> server (local RLS)

```
SUBSCRIBE sip:user@pres.example.com SIP/2.0
To: <sip:user@pres.example.com>
From: <sip:user@example.com>;tag=22222
Call-ID: 2345@terminal.example.com
Event: presence
Require: adhoclist
Accept: application/cpim-pidf+xml
Accept: application/rlmi+xml
Contact: <sip:user@terminal.example.com>
Content-Type: application/adrl+xml
Content-Length: ...

[ADRL Document]
```

F2 200 OK server -> user

```
SIP/2.0 200 OK
To: <sip:user@pres.example.com>;tag=33333
From: <sip:user@example.com>;tag=22222
Call-ID: 2345@terminal.example.com
Event: presence
Accept: application/adrl+xml
Contact: sip:pres.example.com
Content-Length: 0
```

F3 NOTIFY server -> user

```
NOTIFY sip:user@terminal.example.com SIP/2.0
```



```
From: <sip:user@pres.example.com>;tag=33333
To: <sip:user@example.com>;tag=22222
Call-ID: 2345@terminal.example.com
Event: presence
Subscription-State: active;expires=750
Contact: sip:pres.example.com
Content-Type: application/rlmi+xml
Content-Length: ...

[RLMI Document]
```

F4 200 OK user -> server

```
SIP/2.0 200 OK
From: <sip:user@example.com>;tag=33333
To: <sip:user@pres.example.com>;tag=22222
Call-ID: 2345@terminal.example.com
Content-Length: 0
```

F5 Resources? information on the RLS is being updated by SIP or non-SIP means. D

F6 NOTIFY server -> user

```
NOTIFY sip:user@terminal.example.com SIP/2.0
From: <sip:user@pres.example.com>;tag=33333
To: <sip:user@example.com>;tag=22222
Call-ID: 2345@terminal.example.com
Event: presence
Subscription-State: active;expires=750
Contact: sip:pres.example.com
Content-Type: application/rlmi+xml
Content-Length: ...

[RLMI Document]
```

F7 200 OK user -> server

```
SIP/2.0 200 OK
From: <sip:user@example.com>;tag=33333
To: <sip:user@pres.example.com>;tag=22222
Call-ID: 2345@terminal.example.com
Content-Length: 0
```

F8 SUBSCRIBE user (watcher) -> server (local RLS)

```
SUBSCRIBE sip:user@pres.example.com SIP/2.0
To: <sip:user@pres.example.com>
```

```
From: <sip:user@example.com>;tag=22222
Call-ID: 2345@terminal.example.com
Event: presence
Require: adhoclist
Accept: application/cpim-pidf+xml
Accept: application/rlmi+xml
Contact: <sip:user@terminal.example.com>
Content-Type: application/adrl+xml
Content-Length: ...

[ADRL Document]
```

F9 200 OK server -> user

```
SIP/2.0 200 OK
To: <sip:user@pres.example.com>;tag=33333
From: <sip:user@example.com>;tag=22222
Call-ID: 2345@terminal.example.com
Event: presence
Accept: application/adrl+xml
Contact: sip:pres.example.com
Content-Length: 0
```

F10 NOTIFY server -> user

```
NOTIFY sip:user@terminal.example.com SIP/2.0
From: <sip:user@pres.example.com>;tag=33333
To: <sip:user@example.com>;tag=22222
Call-ID: 2345@terminal.example.com
Event: presence
Subscription-State: active;expires=650
Contact: sip:pres.example.com
Content-Type: application/rlmi+xml
Content-Length: ...

[RLMI Document]
```

F11 200 OK user -> server

```
SIP/2.0 200 OK
From: <sip:user@example.com>;tag=33333
To: <sip:user@pres.example.com>;tag=22222
Call-ID: 2345@terminal.example.com
Content-Length: 0
```


5. XML Schema

5.1 Formal Definition

The schema for the adrl+xml XML document is given below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:adrl"
           elementFormDefault="qualified"
           xmlns="urn:ietf:params:xml:ns:adrl"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="adhoclist">
    <xs:complexType>

      <xs:sequence>
        <xs:element name="delete" type="roster" minOccurs="0" maxOccurs="1"/>
        <xs:element name="create" type="roster" minOccurs="0" maxOccurs="1"/>
        <xs:element name="add"    type="roster" minOccurs="0" maxOccurs="1"/>
        <xs:any     minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>

      <xs:attribute name="uri" type="xs:anyURI" use="required" />
      <xs:attribute name="name" type="xs:string" use="required" />
      <xs:anyAttribute />
    </xs:complexType>
  </xs:element>

    <xs:complexType name="roster">
      <xs:sequence>
        <xs:element ref="resource" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:anyAttribute />
    </xs:complexType>

  <xs:element name="resource">
      <xs:attribute name="uri" type="xs:anyURI" use="required" />
      <xs:attribute name="name" type="xs:string" use="optional" />
      <xs:anyAttribute />
  </xs:element>

</xs:schema>
```

5.2 XML Document Examples

The example shows an update on Ann’s ad-hoc list of friends: three

friends are added to the list and one is deleted.

```
<?xml version="1.0"?>

<adhoclist uri="sip:ann@example.com" name="Ann?s Friends">

  <delete>
    <resource uri="ralph@examle.com">
  </delete>

  <add>
    <resource uri="vera@examle.com">
    <resource uri="donna@examle.com">
    <resource uri="calvin@examle.com">
  </add>

</adhoclist>
```

6. Security Considerations

The functionality described in this document doesn't introduce new security risks beyond described in [7] for subscription to a single resource. Note that security considerations related to conveying resources' information using NOTIFYs are addressed by corresponding drafts (e.g. [8], [9], and [6]).

7. IANA Considerations

7.1 New SIP Option Tag: adhoclist

This section defines a new option tag for the registry established by section 27.1 of RFC 3261[1].

Option Tag Name: adhoclist

Description: Extension to allow creation of and subscription to lists of resources

Published specification: RFC xxxx [[Note to RFC editor: replace xxxx with the RFC number of this document when published]]

7.2 New MIME type for Resource List Meta-Information

MIME Media Type Name: application

MIME subtype name: adrl+xml

Required parameters: None

Optional parameters: charset

See RFC 3023 [12] for a discussion of the charset parameter on XML-derived MIME types. Since this MIME type is used exclusively in SIP, the use of UTF-8 encoding is strongly encouraged.

Encoding considerations: 8-bit text

Security considerations: Security considerations specific to uses of this MIME type are discussed in RFC xxxx [[Note to RFC editor: replace xxxx with the RFC number of this document when published]]. RFC-1874 [4] and RFC-3023 [5] discuss security issues common to all uses of XML.

Interoperability considerations: The use of this MIME body is intended to be generally interoperable. No unique considerations have been identified.

Published specification: RFC xxxx [[Note to RFC editor: replace xxxx with the RFC number of this document when published]]

Applications which use this media type: This media type is used to convey meta-information for the state of lists of resources within a Session Initiation Protocol (SIP) subscription.

Additional information:

Magic Number(s): None.

File Extension(s): None.

Macintosh File Type Code(s): None.

Object Identifier(s) or OID(s): None.

Intended usage: Limited Use

Other Information/General Comment: None.

Person to contact for further information:

Name: Orit Levin

E-Mail: oritl@microsoft.com

7.3 URN Sub-Namespace

URI: urn:ietf:params:xml:ns:adrl

Description: This is the XML namespace URI for XML elements defined by [RFCXXXX] to describe identifiers of resources when the information about such resources is aggregated within a single SIP subscription. It is used in the application/adrl+xml body type.

Registrant Contact:

Name: Orit Levin

E-Mail: oritl@microsoft.com

XML:

```
BEGIN
  <?xml version="1.0"?>
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
       content="text/html;charset=utf-8"/>
    <title>Namespace for SIP Ad-hoc Resource List</title>
  </head>
  <body>
    <h1>Namespace for SIP Ad-hoc Resource List</h1>
    <h2>application/adrl+xml</h2>
    <p>See <a href="[[[URL of published RFC]]]">
       RFCXXXX</a>.</p>
  </body>
  </html>
END
```

8. Acknowledgments

Many thanks to Dhigha Sekaran and Sean Olson.

Normative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Rosenberg, J., Schulzrinne, H. and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", draft-ietf-sip-callerprefs-10 (work in progress), October 2003.

Informational References

[4] Levinson, E., "SGML Media Types", RFC 1874, December 1995.

[5] Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[6] Sugano, H. and S. Fujimoto, "Presence Information Data Format (PIDF)", draft-ietf-impp-cpim-pidf-08 (work in progress), May 2003.

[7] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", draft-ietf-simple-presence-10 (work in progress), January 2003.

[8] Roach, A., Rosenberg, J. and B. Campbell, "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists", draft-ietf-simple-event-list-04 (work in progress), June 2003.

[9] Lonnfors, M., "Partial Notification of Presence Information", draft-ietf-simple-partial-notify-00 (work in progress), September 2003.

Author's Address

Orit Levin
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA

EMail: oritl@microsoft.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIP WG
Internet-Draft
Expires: August 1, 2004

R. Mahy
Cisco Systems, Inc.
O. Levin
Microsoft Corporation
F. Audet
Nortel Networks
Feb 2004

Remote Call Control in SIP using the REFER method and the session-oriented dialog package
draft-mahy-sip-remote-cc-01.txt

Status of this Memo

Copyright Notice

Abstract

This document describes how to use the SIP REFER method and the dialog package to manipulate conversations, dialogs, and sessions on remote User Agents. This functionality is most useful for collections of loosely coupled User Agents that wish to present a coordinated user experience. It does not require a Third-Party Call Control controller to be involved in any of the manipulated dialogs.

Table of Contents

1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [2].

To simplify discussions related to the REFER method and its extensions, three new terms will be used:

REFER-Issuer: the UA issuing the REFER request. Sometimes this document will also use the term "controller".

REFER-Recipient: the UA receiving the REFER request

REFER-Target: the UA designated in the Refer-To URI

2. Introduction

The SIP [1] core protocol describes how User Agents originate and terminate sessions. The SIP call control framework [13] also describes how User Agents involved in these sessions can manipulate conversations based on the sessions to provide functionality such as transfer, pickup, and barge-in. Third-Party Call Control [15] goes on to describe how a controller can setup dialogs with a number of participants in order to manipulate sessions among the participants.

Remote call control is the manipulation of conversations and session-oriented dialogs by a UA that is not directly involved in any of the relevant conversations, dialogs, or sessions. This manipulation generally involves sending REFER [4] requests to a UA which is directly involved, using information obtained via the dialog package [5]. (Although many are familiar with REFER only as used to implement call transfer [14], the authors of the REFER method never intended this limitation. In fact the REFER method was created when the SIP working group realized that a generic request to ask another UA to do something on your behalf was much more powerful than just doing transfers.) The Extensions to the REFER mechanism [6] describes the use of REFER for that purpose.

Unlike the Third-Party Call Control (3pcc) model which requires its controller to act as a B2BUA and maintain dialog state for all relevant dialogs, all the SIP entities involved in remote call control using REFER are just regular SIP User Agents. For convenience we can still describe the SIP entity that sends requests to manipulate remote sessions "the controller", but this is just a logical role. A UA that acts as a controller for one request can terminate and originate its own sessions, and even receive remote

call control requests as other requests.

Some readers may question if remote call control is an appropriate use of SIP, instead possibly something more appropriate for MGCP [19] or Megaco [20]. The authors believe that remote call control is an appropriate and natural extension of SIP. Manipulating sessions and dialogs is certainly consistent with core functionality of SIP. This usage of SIP is much different from an MGCP or Megaco master/slave approach. For example, multiple UAs can send remote call control requests. All remote call control requests can be refused based on local authorization policy or if the request doesn't make sense. Finally, each UA is still fully responsible and authoritative for their own dialog and session state. In other words, each UA still has the last word on its sessions and dialogs, even if asked to perform manipulations on that state by another entity. This seems completely appropriate with the design of SIP. In fact these requirements and goals are well documented in the SIP Call Control Framework.

Remote call control is especially useful for collections of loosely coupled User Agents which would like to present a coordinated user experience. Among other things, this allows User Agents which handle orthogonal media types but which would like to be present in a single conversation to add and remove each other from the conversation as needed. This is especially appropriate when coordinating conversations among organizers, general purpose computers, and special purpose communications appliances like telephones, Internet televisions, in-room video systems, electronic whiteboards, and gaming devices.

For example using remote call control, an Instant Messaging client could initiate a multiplayer gaming session and an audio session to a chat conversation. Likewise a telephone could add an electronic whiteboard session to a voice conversation. Finally, a computer or organizer could cause a nearby phone to dial from numbers or URIs in a document, email, or address book; allow users to answer or deflect incoming calls without removing hands from the computer keyboard; place calls on hold; and join other sessions on the phone or otherwise.

Remote call control can also be used in two directions. A computer could remote control a nearby phone and make it dial a SIP URI, but the SIP phone could then also remote control the computer into terminating the session upon the user hanging up the phone.

3. Remote control operations

Remote call control can be used to request a variety of operations.

Commonly used operations include the following:

Make Session - Initiate a new session.

Clear Session - Terminate a session.

Answer - Succesfully respond to a session invitation.

Deflect Session - Redirect a session invitation.

Reject Session - Reject a session invitation.

Single Step Transfer Session - Transfer a session to another UA in a single step. The transferring device is no longer involved with the session after single step transfer is completed. This is described as a "Blind Transfer" in [14]

Complete Transfer Between Sessions - Transfer the remote UA of one existing session to communicate directly with the remote UA of another existing session. Once the transfer completes, the remote controlled UA is no longer involved with either session.

Hold Session - Holds a call at the holding UA. Note that this operation would cause whatever call control would occur locally when this operation is selected (for example a simple hold which makes the call inactive, or a service such as music on hold using a remote stream.

Retrieve Session - Retrieves a held call at the retrieving device.

Merge Sessions - Conferences together two existing sessions at a UA.

Single Step Conference Call - Initiate another session and merge it to an existing session into a new conference.

Alternate Sessions - Place an existing session on hold, and retrieves a previously held session. This operation is a combination of the Hold Call and Retrieve Call operations.

Consultation Session - Places an existing session on hold at the UA and initiates a new session from the UA. This operation is a combination of the Hold Call and Make Call operations.

Set Do Not Disturb - Will cause the remote controlled UA to reject further session invitations with a proper response indicating that it is not availble. This operation does not require the participation of the controller for subsequent session

invitations. The target may cause this operation via local processing or for example by updating presence [17] status which is consumed by systems performing call routing.

Set Forwarding - Will cause the remote controlled UA to redirect further session invitations to another URI. This operation does not require the participation of the controller for subsequent session invitations. The target may cause this operation via local processing or for example by manipulating SIP registrations.

4. Implementing these operations

In order to convey requests for remote call control operations, there are several syntactic approaches possible. The most obvious is to use the existing Refer-To URI syntax. However, escaping long URIs is error-prone and obfuscates the intent of a request. Another option mentioned as a REFER extension is carrying the Refer-To target as a message/sipfrag [12] body. However, encoding remote call control operations which deal with with more that one session in a single URI are still cumbersome. Also, both these approaches rely on implicit behavior or undefined URI conventions. This document uses this approach for operations which only require a straightforward encoding.

Alternatively, the Refer-To URI could be a Universal Resource Name (URN) [21] which could describe a particular operation such as Hold or Retrieve. Combined with the dialog-identifiers of an existing session conveyed as parameters of the Refer-To header, this would permit explicit operations which do not need additional parameters or handle more than a single session. For example, the following could represent a Hold operation of a session with the Call-ID "123":

```
Refer-To: <urn:ietf:params:sip:remotecc:hold>
  ;call-id=123;remote-tag=aaa;local-tag=bbb
```

Note however that the most interesting remote call control operations (such as Complete Transfer Between Sessions and Merge) operate on more than one session and may require additional parameters. These are still abstract operations, but they operate on more than one target. Using an explicit description of these parameters in a new MIME body is an ideal way to provide this additional functionality, and the only approach which works with all the sample remote call control operations in this document.

An additional benefit of a remote call control body is that certain details of these operations can be abstracted. For example, a Clear Session operation can cause either a CANCEL, BYE or appropriate

response to be sent depending on context. A Hold operation can result in whatever user-visible functionality occurs when a Hold is selected locally (for example a simple hold, tone-on-hold, music-on-hold, animated cartoon characters, etc.). A Merge Sessions operation can use whatever conference resource would be used by the UA itself (a local conferencing focus, a discovered focus, or an administratively configured focus).

This document therefore describes a MIME body for remote call control operations conveyed in the body of a REFER request. Remote call control operations using a remote call control MIME type body are operations that are typically more abstract or complex information than can be practically be achieved with a message/sipfrag body or a Refer-to URI.

This document makes frequent use of the REFER extensions defined in [6] to carry out these operation. In particular, we frequently reference bodies in the Refer-To header using a Content-ID URI (cid:).

While a remote call control MIME body is not defined in this document, we use the MIME type application/remotecc in our examples. The following is an example of a REFER with a Remote Call Control operation with such a body:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
Via: SIP/2.0/TCP issuer.example.com.com;branch=z9hG4bK-a-1
To: "Alice's phone" <sip:reg2@10.1.1.3>
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123@issuer.example.com
CSeq: 2 REFER
Max-Forwards: 70
Contact: sip:alice1@10.1.1.2
Accept: application/dialog-info+xml
Require: extended-refer
Refer-To: <cid:1239103912039@issuer.example.com>
 ;call-id=
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...

 ----------------------------
 | Remote Call Control Body |
 ----------------------------
```

The application/dialog-info+xml package can be used to provide information about the status of dialogs. The examples in this specification assume that the dialog event package is sufficient to

provide the necessary feedback for remote call control operations.

5. Examples of Remote Call Control Operations SIP Call Flows

This entire section provide non-normative examples of functionality where a computer or PDA manipulates a telephone. The behavior for remote call control with other types of devices is similar, but describing similar manipulations for other media or device types would naturally use a different set of vocabulary.

The following sub-sections provide an example for every operation described in the previous section.

The following notes are applicable to all the call flows in the subsections below:

It is assumed that Alice's PC or PDA has subscribed to Alice's Phone dialog package. All of the NOTIFY messages are notifications about changes in the dialog state at Alice's phone. No additional remote call control event packages are shown, but it is not precluded that one be defined later.

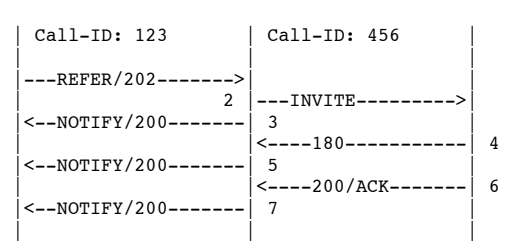
As specified in [6], there is no no implicit subscrition on all REFER messages between Alice's PDA or PC and Alice's Phone with the extended REFER mechanism.

Via and Max-Forward headers and session descriptions are omitted for brevity and clarity. In some cases, display names are added for simplify the task of the reader following the examples. Note that URIs in SIP cannot wrap lines. Due to RFC formatting conventions, this draft splits URIs across lines where the URI would exceed 72 characters. A backslash character marks where this line folding has taken place.

5.1 Make Call Operation

In message 1, Alice's PC or PDA asks her phone to "call Bob" (message 2), which eventually results in an early dialog (3) with one of Bob's Contacts. Bob sends a ringing indication (4) which triggers Alice's phone to send a notification (5) of "early" to Alice's PC or PDA. Then Bob answers the phone (6) wich triggers Alice's phone to send a notification (7) of "confirmed" to Alice's PC or PDA.

```
  Alice's              Alice's              Bob
  PC or PDA            Phone
```



In this first example, in Message 1a, traditional Refer-To encoding is used. Message 1b shows how to request this same operation with an embedded remote call control MIME body.

Message 1a:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Refer-To: "Bob" <sip:bob@example.net;method=INVITE>
```

Message 1b:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: extended-refer
Refer-To: cid:1239103912039@issuer.example.com
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...
 --------------------------------
| Remote Call Control Body       |
| MakeCall                       |
|   From: sip:reg2@10.1.1.3      |
|   To: bob@example.net          |
|   other parameters             |
 --------------------------------
```


Message 2:

```
INVITE sip:bob@example.net SIP/2.0
To: "Bob" <sip:bob@example.net>
From: "Alice" <sip:alice@example.com>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Alice's Phone" <sip:reg2@10.1.1.3>
Content-Type: application/sdp
Content-Length: xxx
```

Message 3:

NOTIFY indicates "trying".

Message 4:

```
SIP/2.0 180 Ringing
To: "Bob" <sip:bob@example.net>;tag=uvw
From: "Alice's phone" <sip:reg2@10.1.1.3>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Bob's Contact" <sip:line1@192.168.0.5>
```

Message 5:

NOTIFY will indicates "early".

Message 6:

```
SIP/2.0 200 OK
To: "Bob" <sip:bob@example.net>;tag=uvw
From: "Alice's phone" <sip:reg2@10.1.1.3>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Bob's Contact" <sip:line1@192.168.0.5>
Content-Type: application/sdp
Content-Length: xxx
```

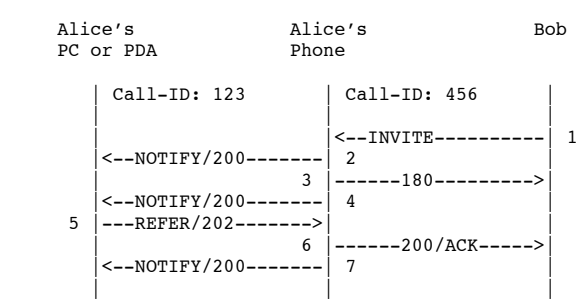
Message 7:

NOTIFY indicates "confirmed".

5.2 Answer Call Operation

In message 1, Bob makes a call to Alice's Phone. A notification (2)

of "trying" is sent to Alice. Alice's phone automatically sends a "ringing" (3) to Bob. Another notification (4) of "early" is then sent to Alice's PC. Alice then instructs (5) her PDA to tell the phone to answer the call (6). Alice's phone sends a notification (7) of "confirmed" to Alice's PDA.



Message 1:

```
INVITE sip:alice@example.com SIP/2.0
To: "Alice" <sip:alice@example.com>
From: "Bob" <sip:bob@example.net>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Bob's Contact" <sip:line1@192.168.0.5>
Content-Type: application/sdp
Content-Length: xxx
```

Message 2:

NOTIFY indicates "trying".

Message 3:

```
SIP/2.0 180 Ringing
To: "Alice" <sip:alice@example.com>;tag=uvw
From: "Bob" <sip:bob@example.net>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Alice's Phone" <sip:reg2@10.1.1.3>
```

Message 4:

NOTIFY indicates "early".

Message 5:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: extended-refer
Refer-To: cid:1239103912039@issuer.example.com
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...
```

```
Remote Call Control Body
answercall
  Call=
    sip:line1@192.168.0.5
    call-id:456
    remote-tag=uvw
    local-tag=xyz
  other parameters
```

Meassage 6:

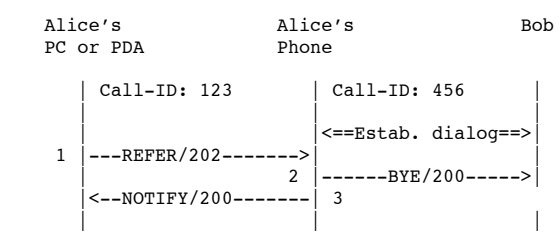
```
SIP/2.0 200 OK
To: "Alice" <sip:alice@example.com>;tag=uvw
From: "Bob" <sip:bob@example.net>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Alice's Phone" <sip:reg2@10.1.1.3>
Content-Type: application/sdp
Content-Length: xxx
```

Message 7:

NOTIFY indicates "confirmed".

5.3 Clear Connection

Alice's Phone and Bob's contact are currently in an established dialog. In message 1, Alice's PC or PDA asks her phone to "clear the connection" with Bob's phone. (message 2).



Message 1:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: extended-refer
Refer-To: cid:1239103912039@issuer.example.com
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...
```

Remote Call Control Body
clearconnection
Call=
sip:line1@192.168.0.5
call-id:456
remote-tag=uvw
local-tag=xyz
other parameters

Message 2:

BYE is sent to Bob's contact.

Message 3:

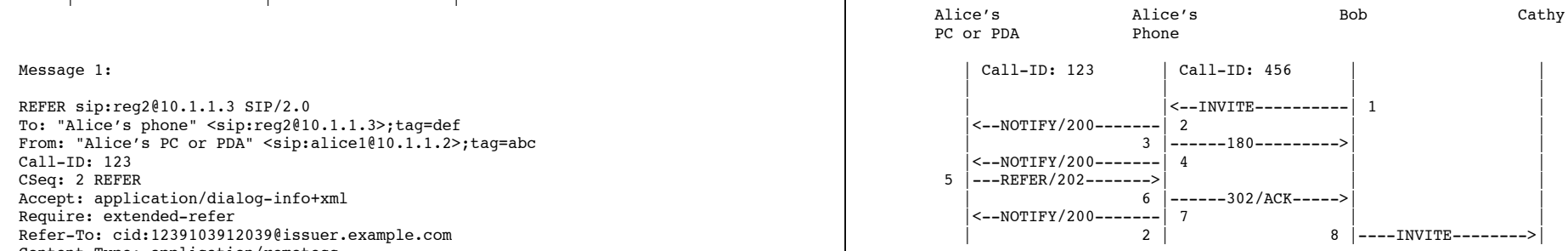
NOTIFY indicates "trying".

Message 3:

NOTIFY will indicates "terminated".

5.4 Deflect Call

In message 1, Bob makes a call to Alice's Phone. A notification (2) of "trying" is sent to Alice. Alice's phone automatically sends a "ringing" (3) to Bob. Another notification (4) of "early" is then sent to Alice's PC. Alice then instructs (5) her PDA to tell the phone to deflect the call (6) to Cathy. Alice's phone sends a notification (7) of "terminated" to Alice's PDA. Bob's will attempt the call to Cathy (8).



Message 1:

```
INVITE sip:alice@example.com SIP/2.0
To: "Alice" <sip:alice@example.com>
From: "Bob" <sip:bob@example.net>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Bob's Contact" <sip:line1@192.168.0.5>
Content-Type: application/sdp
Content-Length: xxx
```

Message 2:

NOTIFY indicates "trying".

Message 3:

```
SIP/2.0 180 Ringing
To: "Alice" <sip:alice@example.com>;tag=uvw
From: "Bob" <sip:bob@example.net>;tag=xyz
Call-ID: 456
```

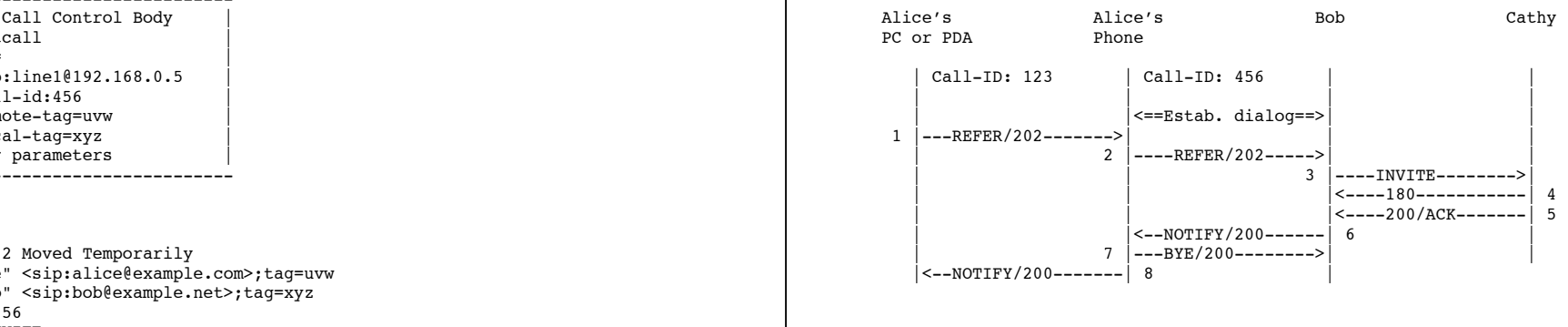
```
   CSeq: 1 INVITE
   Contact: "Alice's Phone" <sip:reg2@10.1.1.3>

   Message 4:

   NOTIFY indicates "early".

   Message 5:

   REFER sip:reg2@10.1.1.3 SIP/2.0
   To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
   From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
   Call-ID: 123
   CSeq: 2 REFER
   Accept: application/dialog-info+xml
   Require: extended-refer
   Refer-To: cid:1239103912039@issuer.example.com
   Content-Type: application/remotecc
   Content-Id: <1239103912039@issuer.example.com>
   Content-Length: ...
    --------------------------------
    | Remote Call Control Body     |
    | deflectcall                  |
    |   Call=                      |
    |     sip:line1@192.168.0.5    |
    |     call-id:456              |
    |     remote-tag=uvw           |
    |     local-tag=xyz            |
    |   other parameters           |
    --------------------------------

   Message 6:

   SIP/2.0 302 Moved Temporarily
   To: "Alice" <sip:alice@example.com>;tag=uvw
   From: "Bob" <sip:bob@example.net>;tag=xyz
   Call-ID: 456
   CSeq: 1 INVITE
   Contact: "Cathy" <sip:cathy@example.net>

   Message 7:

   NOTIFY indicates "rejected".

   Mesage 8:

   INVITE sip:cathy@example.net SIP/2.0
   To: "Cathy" <sip:cathy@example.net>
```



```
   From: "Bob" <sip:bob@example.net>;tag=pqr
   Call-ID: 789
   CSeq: 1 INVITE
   Contact: "Bob's Contact" <sip:line1@192.168.0.5>
   Content-Type: application/sdp
   Content-Length: xxx
```

5.5 Single Step Transfer Call

Alice's Phone and Bob's contact are currently in an established dialog. In message 1, Alice's PC or PDA requests that a request be made to transfer the call to Cathy. Alice's phone sends a request (2) to Bob's contact to transfer the call to Cathy (3). Call from Bob's contact to Cathy rings (4), is answered (5). Bob's contact sends a notification (6) to Alice's phone because of the REFER implicit subsription. Alice's phone then terminates the session with Bob's contact (7) and sends a notification of "terminated" to Alice's PC or PDA.




```
   Message 1:

   REFER sip:reg2@10.1.1.3 SIP/2.0
   To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
   From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
   Call-ID: 123
   CSeq: 2 REFER
   Accept: application/dialog-info+xml
   Require: extended-refer
```

```
Refer-To: cid:1239103912039@issuer.example.com
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...
 --------------------------------
 | Remote Call Control Body     |
 | transfer                     |
 |   FirstCall=                 |
 |     sip:line1@192.168.0.5    |
 |     call-id:456              |
 |     remote-tag=uvw           |
 |     local-tag=xyz            |
 |   SecondCall=                |
 |     sip:cathy@example.net    |
 |   other parameters           |
 --------------------------------
```

Message 2:

```
REFER sip:bob@example.net SIP/2.0
To: "Bob" <sip:bob@example.net>
From: "Alice" <sip:alice@example.com>;tag=xyz
Call-ID: 456
CSeq: 1 REFER
Refer-To: "Cathy" <sip:cathy@example.net;method=INVITE>
```

Mesage 3:

```
INVITE sip:cathy@example.net SIP/2.0
To: "Cathy" <sip:cathy@example.net>
From: "Bob" <sip:bob@example.net>;tag=pqr
Call-ID: 789
CSeq: 1 INVITE
Contact: "Bob's Contact" <sip:line1@192.168.0.5>
Content-Type: application/sdp
Content-Length: xxx
```

Messages 4 & 5:

180, 200, ACK when call is set up with Cathy.

Message 6:

NOTIFY will include the sigfrag as per the REFER implicit subsription.

Message 7:

Bob's contact clears the call.

Message 8:

NOTIFY indicates "confirmed".

5.6 Complete Transfer Between Sessions

TBD

5.7 Hold Call

In message 1, Alice's PC or PDA asks her phone to put on hold the already established dialog with Bob. Alice's phone sends a re-INVVITE to Bob's contact to put the media stream on hold. Note that a call hold is different concept than held media. In fact, a user can be placed on hold, and be provided with music on hold. A held call is a logical state which could be useful for a number of things such as monitoring the amount of time a user stays in a queue. This diagram does not illustrate any event package to illustrate that a can can be held.

Message 1:

```
Accept: application/dialog-info+xml
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Require: extended-refer
Refer-To: cid:1239103912039@issuer.example.com
```

```
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...
 --------------------------------
 | Remote Call Control Body     |
 | hold                         |
 |   Call=                      |
 |     sip:line1@192.168.0.5    |
 |     call-id:456              |
 |     remote-tag=uvw           |
 |     local-tag=xyz            |
 |   other parameters           |
 --------------------------------
```

Message 2:

```
INVITE sip:bob@example.net SIP/2.0
To: "Bob" <sip:bob@example.net>
From: "Alice" <sip:alice@example.com>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Alice's Phone" <sip:reg2@10.1.1.3>
Content-Type: application/sdp
Content-Length: xxx
   SDP to indicate held media for example.
```

Message 3:

```
NOTIFY indicates "trying".
```

Message 4:

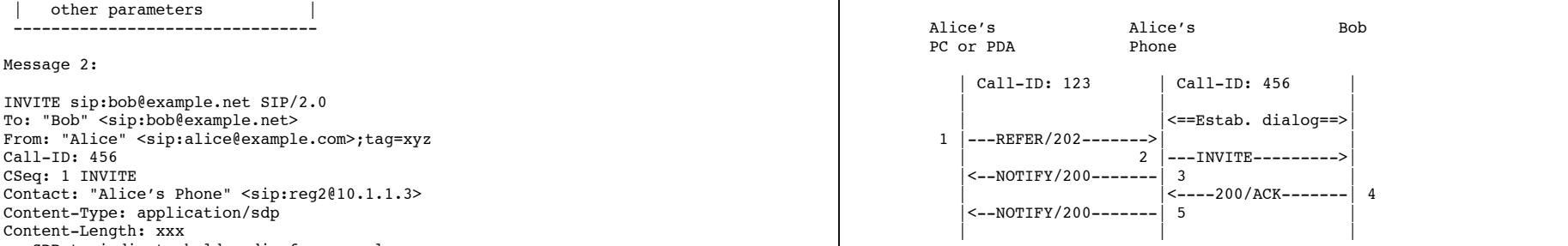
```
SIP/2.0 200 OK
To: "Bob" <sip:bob@example.net>;tag=uvw
From: "Alice's phone" <sip:reg2@10.1.1.3>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Bob's Contact" <sip:line1@192.168.0.5>
Content-Type: application/sdp
Content-Length: xxx
```

Message 5:

```
NOTIFY indicates "confirmed".
```


5.8 Retrieve Call

In message 1, Alice's PC or PDA asks her phone to retreive an held call with Bob. Alice's phone sends a re-INVVITE to Bob's contact to resume the media stream which was already on hold. Note that a call hold is different concept than held media. In fact, a user can be placed on hold, and be provided with music on hold. A held call is a logical state which could be useful for a number of things such as monitoring the amount of time a user stays in a queue. This diagram does not illustrate any event package to illustrate that a can can be held.



Message 1:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: extended-refer
Refer-To: cid:1239103912039@issuer.example.com
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...
 --------------------------------
 | Remote Call Control Body     |
 | retreive                     |
 |   Call=                      |
 |     sip:line1@192.168.0.5    |
 |     call-id:456              |
 |     remote-tag=uvw           |
 |     local-tag=xyz            |
```

```
 |   other parameters           |
 -------------------------------
```

Message 2:

```
INVITE sip:bob@example.net SIP/2.0
To: "Bob" <sip:bob@example.net>
From: "Alice" <sip:alice@example.com>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Alice's Phone" <sip:reg2@10.1.1.3>
Content-Type: application/sdp
Content-Length: xxx
   SDP to indicate re-established media.
```

Message 3:

```
NOTIFY indicates "trying".
```

Message 4:

```
SIP/2.0 200 OK
To: "Bob" <sip:bob@example.net>;tag=uvw
From: "Alice's phone" <sip:reg2@10.1.1.3>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Bob's Contact" <sip:line1@192.168.0.5>
Content-Type: application/sdp
Content-Length: xxx
```

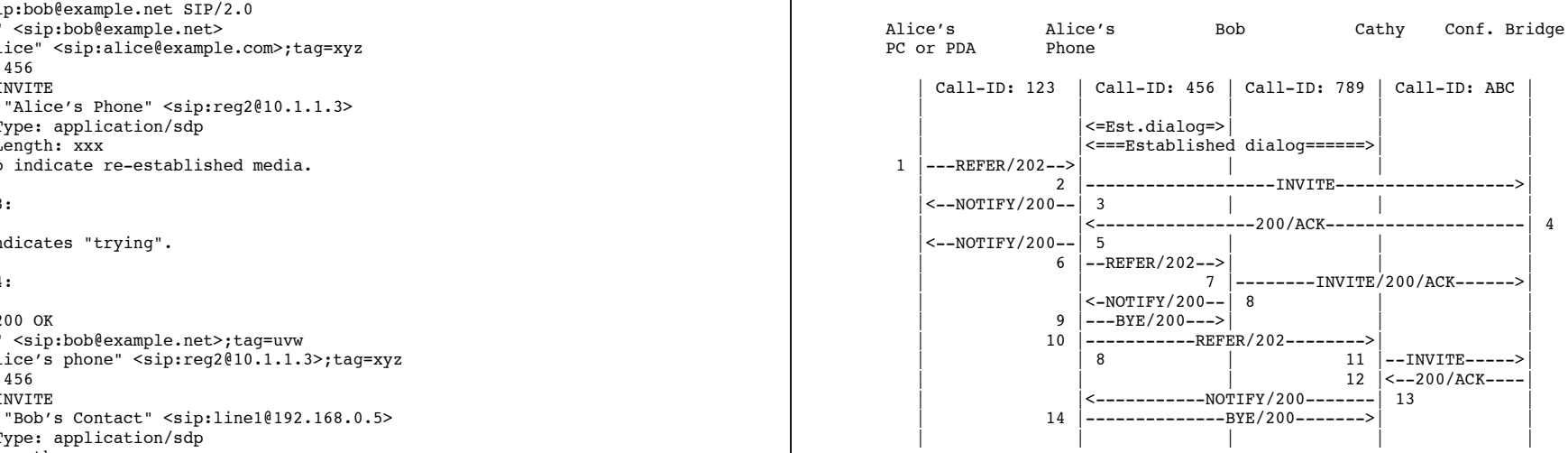
Message 5:

```
NOTIFY indicates "confirmed".
```

5.9 Conference Call

Alice's Phone and Bob's contact are currently in an established dialog. Alice's Phone and Cathy's contact are also currently in an established dialog. In message 1, Alice's PC or PDA requests that a conference be established between the two calls (i.e., a conference between Alice's Phone, Bob's contact and Cathy's contact. Alice's phone establish a call with a conference bridge (2-5). Alice's phone sends a request (6) to Bob's contact to transfer the call to the same conference bridge (7). Alice's phone is notified (implicit REFER subscription) of the successful transfer to the conference bridge (8) and clears the call with Bob (9). Alice's phone sends a request (10)

to Cathy's contact to transfer the call to the same conference bridge (11). Alice's phone is notified (implicit REFER subscription) of the successful transfer to the conference bridge (12) and clears the call with Cathy (13). The call flow does not show an event package for the successful remote conference invocation.



Message 1:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: extended-refer
Refer-To: cid:1239103912039@issuer.example.com
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...
 -------------------------------
 | Remote Call Control Body     |
```

```
| conference                    |
|   FirstCall=                  |
|     sip:line1@192.168.0.5     |
|     call-id:456               |
|     remote-tag=uvw            |
|     local-tag=xyz             |
|   SecondCall=                 |
|     sip:cathy-pc@192.168.8.8  |
|     call-id:789               |
|     remote-tag=abc            |
|     local-tag=def             |
|   other parameters            |
 -------------------------------
```

Message 3:

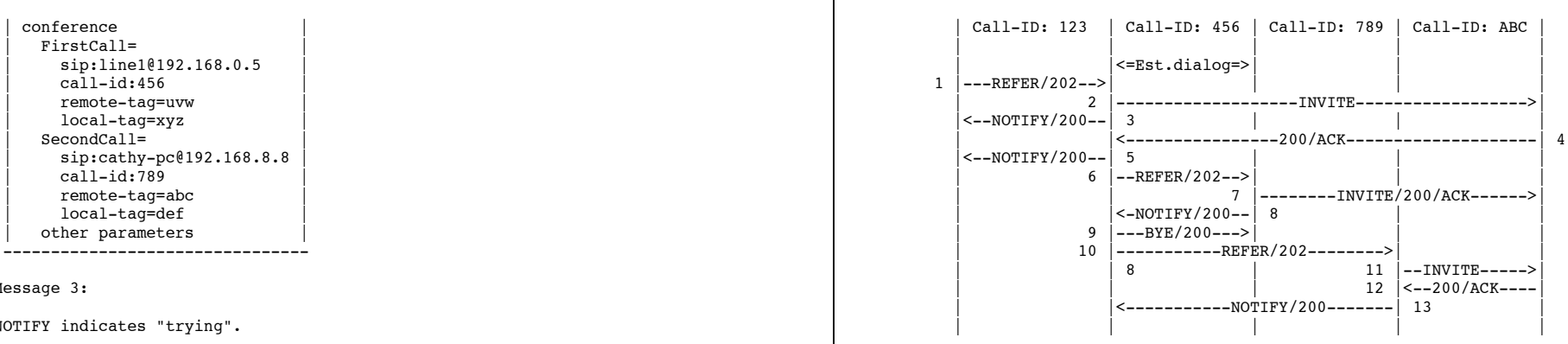
NOTIFY indicates "trying".

Mesage 5:

NOTIFY indicates "confirmed".

5.10 Single Step Conference Call

A single step conference call is the same operation as a conference call, except that one of the legs is a SIP URI instead of an established dialog. Alice's Phone and Bob's contact are currently in an established dialog. In message 1, Alice's PC or PDA requests that a conference be established between the the existing call with Bob's contact and with Cathy (i.e., a conference between Alice's Phone, Bob's contact and Cathy. Alice's phone establish a call with a conference bridge (2-5). Alice's phone sends a request (6) to Bob's contact to transfer the call to the same conference bridge (7). Alice's phone is notified (implicit REFER subscription) of the successful transfer to the conference bridge (8) and clears the call with Bob (9). Alice's phone sends a request (10) to Cathy's contact to transfer the call to the same conference bridge (11). Alice's phone is notified (implicit REFER subscription) of the successful transfer to the conference bridge (12). The call flow does not show an event package for the successful remote single step conference invocation.



Message 1:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: extended-refer
Refer-To: cid:1239103912039@issuer.example.com
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...
 -------------------------------
 | Remote Call Control Body |
 | conference                    |
 |   FirstCall=                  |
 |     sip:line1@192.168.0.5     |
 |     call-id:456               |
 |     remote-tag=uvw            |
 |     local-tag=xyz             |
 |   SecondCall=                 |
 |     sip:cathy@example.net     |
 |   other parameters            |
 -------------------------------
```

Message 3:

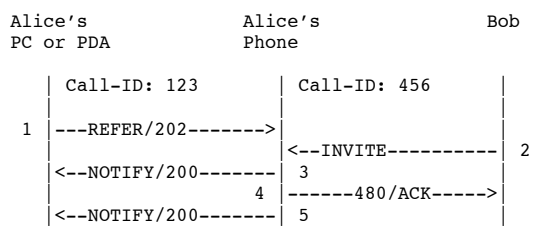
NOTIFY indicates "trying".

Mesage 5:

NOTIFY indicates "confirmed".

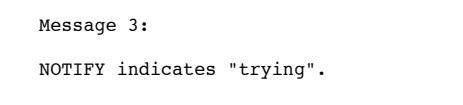
5.11 Set Do Not Disturb

In message 1, Alice sends a request so that her phone will be in "Do not disturb" or "Make set busy" mode. Any subsequent invitation (2) send to Alice's phone will result in the session being rejected with response 480 "Temporarily not available" (or 486 "Busy Here", or any other appropriate code) without any interaction from Alice's PC or PDA.



Message 1:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: extended-refer
Refer-To: cid:1239103912039@issuer.example.com
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...
```

Message 2:

```
INVITE sip:alice@example.com SIP/2.0
To: "Alice" <sip:alice@example.com>
From: "Bob" <sip:bob@example.net>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Bob's Contact" <sip:line1@192.168.0.5>
Content-Type: application/sdp
Content-Length: xxx
```

Message 3:

NOTIFY indicates "trying".

Message 4:

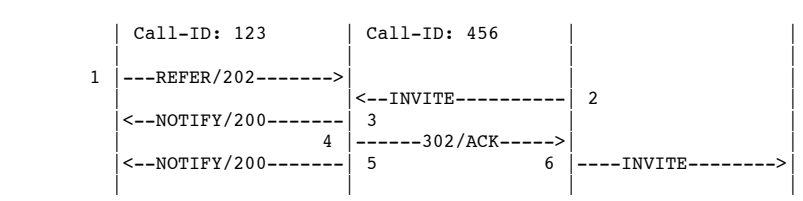
```
SIP/2.0 480 Temporarily unavailable
To: "Alice" <sip:alice@example.com>;tag=uvw
From: "Bob" <sip:bob@example.net>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Alice's Phone" <sip:reg2@10.1.1.3>
```

Message 5:

NOTIFY indicates "rejected".

5.12 Set Forwarding

In message 1, Alice sends a request so that her phone will be "call forwarded" to Cathy. Any subsequent invitation (2) send to Alice's phone will result in the session being forewared with response 302 "Move temporarily" without any interaction from Alice's PC or PDA.



Message 1:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: extended-refer
Refer-To: cid:1239103912039@issuer.example.com
Content-Type: application/remotecc
Content-Id: <1239103912039@issuer.example.com>
Content-Length: ...
```

```
Remote Call Control Body
setforwarding
  destination
    sip:cathy@example.net
    forwardingtype=always
  other parameters
```

Message 2:

```
INVITE sip:alice@example.com SIP/2.0
To: "Alice" <sip:alice@example.com>
From: "Bob" <sip:bob@example.net>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Bob's Contact" <sip:line1@192.168.0.5>
Content-Type: application/sdp
Content-Length: xxx
```

Message 3:

NOTIFY indicates "trying".

Message 4:

```
SIP/2.0 302 Moved temporarily
To: "Alice" <sip:alice@example.com>;tag=uvw
From: "Bob" <sip:bob@example.net>;tag=xyz
Call-ID: 456
CSeq: 1 INVITE
Contact: "Cathy" <sip:cathy@example.net>
```

Message 5:

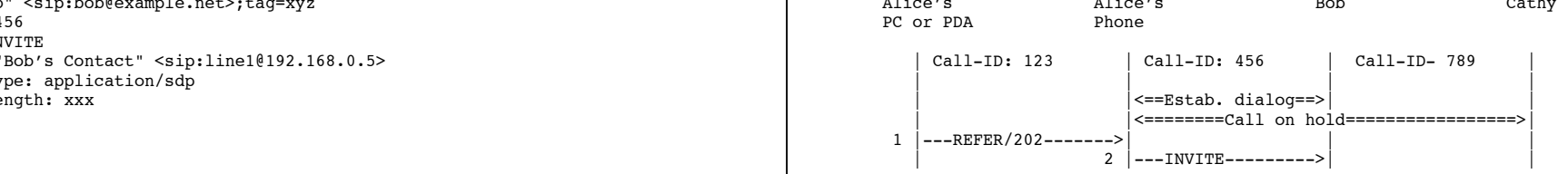
NOTIFY indicates "rejected".

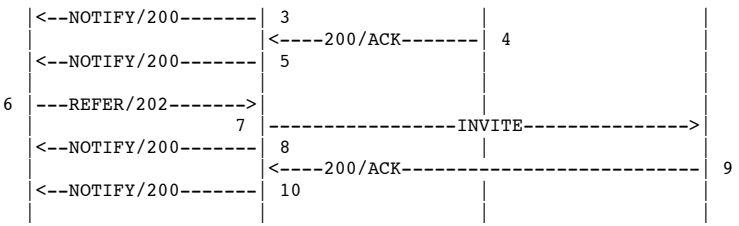
Message 6:

```
INVITE sip:cathy@example.net SIP/2.0
To: "Cathy" <sip:cathy@example.net>
From: "Bob" <sip:bob@example.net>;tag=pqr
Call-ID: 789
CSeq: 1 INVITE
Contact: "Bob's Contact" <sip:line1@192.168.0.5>
Content-Type: application/sdp
Content-Length: xxx
```

5.13 Alternate Call

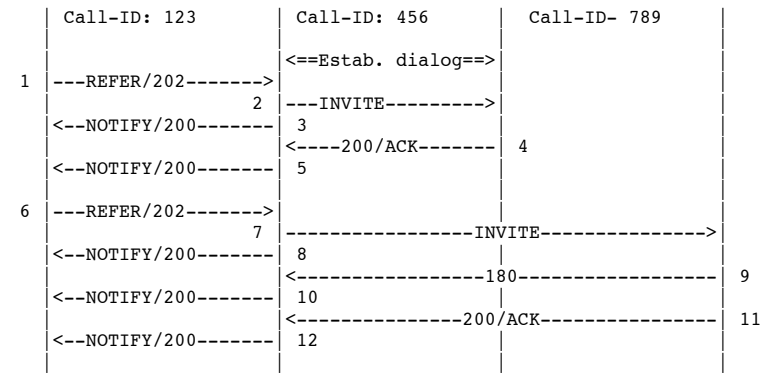
Alternate call is not really an operation by itself. It is a a hold operation followed by a retrieve operation. This section is included only to illustrate how those two operations can be combined to provide an Alternate Call service. In message 1, Alice's PC or PDA asks her phone to put on hold the already established dialog with Bob. Alice's phone sends a re-INVITE to Bob's contact (2) to put the media stream on hold. Alice's PC or PDA then asks her phone (6) to retrieve her previously held call with Cathy (7).





5.14 Consultation Call

Consultation call is not really an operation by itself. It is a hold opeeraiton followed by a make call operation. This section is included only to illustrate how those two operation can be combined to provide a Consultation Call service. In message 1, Alice's PC or PDA asks her phone to put on hold the already established dialog with Bob. Alice's phone. Alice's phone sends a re-INVITE to Bob's contact (2) to put the media stream on hold. Alice's PC or PDA then asks her phone (6) to make a call to Cathy (7).



6. Examples of implementing remote call control operations with Refer-To URI

This section provided examples of how to implement some of the simple operations of the previous sections without using a REFER MIME body and relying instead of the Refer-To URI. All the call flows are assumed to be the same as per the previous section. Only the changed REFER message is shown.

6.1 Make Call Operation

This example is already discussion in a previous section.

6.2 Answer Call Operation

Message 1:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: response-refer
Refer-To: "Bob's Contact" <sip:line1@192.168.0.5;method=INVITE;response=200?
  Call-ID=456&To=alice%40example.com;tag=uvw&From=bob%40example.net;tag=xyz>
```

6.3 Clear Connection

Message 1:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: response-refer
Refer-To: "Bob's contact" <sip:line1@192.168.0.5;method=BYE?
  Call-ID=456&To=alice%40example.com;tag=uvw&From=bob%40example.net;tag=xyz>
```

6.4 Deflect Call

Message 5:

```
REFER sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 2 REFER
Accept: application/dialog-info+xml
Require: response-refer
Refer-To: "Bob's Contact" <sip:line1@192.168.0.5;method=INVITE;response=302?
  Call-ID=456&To=alice%40example.com;tag=uvw&From=bob%40example.net;tag=xyz>
```

6.5 Complete Transfer Between Calls

7. User Agent Behavior

7.1 Organizing requests within dialogs

REFER messages used for call transfer always arrive within an existing dialog which was created with the INVITE method. In general, REFER messages can be sent within an existing dialog, or they can start a new dialog (the dialog used by the implicit subscription they create). In many use cases of remote call control, receiving notifications about the status of a REFER request are superfluous, as the Refer-Issuer typically maintains a long duration subscription to the dialog package. This situation is complicated by the possible presence of the norefersub option-tag, defined in section 7 of [6]. When the norefersub option tag is present, a REFER request which would have created a new subscription and dialog becomes a standalone transaction instead. Each such standalone REFER transaction MUST use a new (unique) Call-Id header field value. The following three use cases are suggested:

1. In the most common usage, the controller maintains a long duration subscription to the dialog package, and sends REFER requests within that dialog. Each REFER is sent within the context of the dialog created for the subscription to the dialog package, and could include the norefersub option-tag in a Supported header field value.

2. Occasionally the dialog package is only supported via a dialog state agent separate from the Refer-Receiver, in which case the controller maintains a long duration subscription to the dialog package to a dialog state agent, and the controller sends these

individual REFER requests as standalone requests each with a different (unique) Call-ID header field value, which could also include the norefersub option-tag in a Supported header field value.

3. In some cases, the controller does not typically maintain a dialog package subscription for the Refer-Receiver. This might be the case for a "webdialer" or other application which associates with other UAs on an adhoc and intermitent basis. An initial REFER request is sent to start a new dialog, which is followed by notifications for the refer event type (the norefersub option-tag SHOULD NOT be used in this case). These notifications could contain message/sipfrag or application/dialog-info+xml notification bodies as described in Section 4 of [6].

Message 1:

```
SUBSCRIBE sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's phone" <sip:reg2@10.1.1.3>
From: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
Call-ID: 123
CSeq: 1 SUBSCRIBE
Event: dialog
Contact: <sip:alice1@10.1.1.2>
```

Message 2:

```
NOTIFY sip:reg2@10.1.1.3 SIP/2.0
To: "Alice's PC or PDA" <sip:alice1@10.1.1.2>;tag=abc
From: "Alice's phone" <sip:reg2@10.1.1.3>;tag=def
Call-ID: 123
CSeq: 1 NOTIFY
Event: dialog
Contact: <sip:reg2@10.1.1.3>
Subscription-State: active;expires=3600
Content-Type: application/dialog-info+xml
Content-Length: xxx
```

7.2 Addressing the relevant parties

REFER requests contain a number of URIs which need to address the appropriate parties. A list of the relevant fields include the Request-URI, To header URI, From header URI, Contact header URI, Refer-To header URI, and the Referred-By header URI. This section attempts to clarify what needs to be placed in each field.

In most cases, remote call control seeks to manipulate dialogs or

sessions on a specific UA. For this reason, the Request URI of the REFER request MUST be a valid Globally Routable Unique URI (GRUU) [9] for a single UA (a Contact URI). Contact URIs for a UA can be discovered by subscribing to the Registration Package [22] for the relevant AORs.

For remote call control requests to operate as expected, the Refer-Issuer needs to be confident that the Refer-Receiver supports the extensions and conventions described here. Otherwise, the triggered request might have completely different semantics from the request which was indicated in the Refer-To header. (Most implementations ignore unknown URI and header parameters). For example a REFER intended to cause the Refer-Receiver to send a 486 Busy Here response for an existing dialog, might instead trigger a new INVITE to the sender of the original INVITE. Implementations which send remote call control requests MUST include the refer-response option-tag in a Require header field value in each REFER request. (Note that support for this option-tag also implies support for the response URI parameter in a Refer-To header.)

The To header field in the REFER request should contain the same URI as in the Request-URI, and the From identifies the AOR of the controller. The Refer-To is set to whatever URI would normally appear in the triggered request if the request were initiated autonomously by the Refer-Receiver. A REFER triggering a standalone request or dialog starting request, could send to either an AOR or a Contact address, but typically to an AOR. A REFER request triggering a request which is in a dialog MUST always place a Contact URI in the Refer-To header.

When set, the Referred-By [7] header field SHOULD be the same URI as the URI in the Contact address of the REFER. If included by the Refer-Issuer, it SHOULD be protected with a signed authenticated identity body [8] as recommended in the Referred-By specification.

7.3 Selecting an existing dialog context for the triggered request

Many uses of remote call control require that the Refer-Receiver generate a new request or response in the context of an existing dialog. For example, the controller might want the Refer-Receiver to send a BYE, CANCEL, or response to an INVITE in the context of a dialog created with INVITE. For subscriptions, the controller might want the Refer-Receiver to unsubscribe (send a SUBSCRIBE with an Expires header field of 0).

To select the appropriate dialog from which to source the request, this document proposes a few new (header) parameters to the Refer-To header (the call-id, remote-tag, and local-tag parameters). Explicit

header parameters were selected because they can apply to non SIP URIs. For example, the following URI, loads a "How To" website in the context of an existing dialog (presumably one created with an INVITE). When the associated dialog completes, the content may be hidden or dismissed with the context with which it was associated

```
Refer-To: <http://support.example.com/howto.html>
 ;call-id=xyz;remote-tag=123;local-tag=456
```

When describing the context of a subscription, the event and event-id parameters are also used. These correspond to the event type and the event-id parameter in the Event header (if present).

Explicit matching of target dialogs and subscriptions was intentonally selected instead of including the appropriate values in embedded Call-ID, To, From, and Event headers. Among other benefits, this reduces the length of the URI portion of the Refer-To header and simplifies URI encoding requirements dramatically.

> OPEN ISSUE: These parameter extensions should be incorporated in the REFER extensions draft.

8. Authorizing remote call control requests

User Agents MUST authorize all remote call control requests. Requests from user agents which can authenticate themselves (using Digest authentication, mutual TLS authentication, or S/MIME) as representing the AOR on the target UA SHOULD be authorized unless local policy directs otherwise. In addition, some user agents may need introduction using one-time credentials which have additional authorization restrcitions. For example, an electronic whiteboard in a conference room could authorize participants only if they had scheduled a meeting in the corresponding conference room for the current time. [More explanation needed.]

9. Security Considerations

The functionality described in this document allows an authorized party to manipulate SIP sessions and dialogs in arbitrary ways. Implementations need to take reasonable precautions to insure authenticity of remote call control request, which MUST be sent using either hop-by-hop TLS [11] via a SIPS URI, or individually signed using SMIME [10]. Signing remote call control requests with SMIME is RECOMMENDED. In addition, UAs which support remote call control SHOULD sign Referred-By headers in remote call control requests in an appropriate authenticated identity body. UAs which support remote call control MUST implement SIPS, SHOULD implement SMIME signing and

verification, and SHOULD implement separate signing of Referred-By headers in an appropriate authenticated identity body.

10. IANA Considerations

No action by IANA is required.

11. Acknowledgments

Many thanks to Sean Olson, Robert Sparks, and Alan Johnston.

Normative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[4] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[5] Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP", draft-ietf-sipping-dialog-package-01 (work in progress), March 2003.

[6] Olson, S., "Extensions to the REFER mechanism for Third Party Call Control", draft-olson-sipping-refer-extensions-00 (work in progress), June 2003.

[7] Sparks, R., "The SIP Referred-By Mechanism", draft-ietf-sip-referredby-01 (work in progress), February 2003.

[8] Peterson, J., "SIP Authenticated Identity Body (AIB) Format", draft-ietf-sip-authid-body-01 (work in progress), March 2003.

[9] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-00 (work in progress), January 2004.

[10] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[11] Dierks, T., Allen, C., Treese, W., Karlton, P., Freier, A. and P. Kocher, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[12] Sparks, R., "Internet Media Type message/sipfrag", RFC 3420, November 2002.

Informational References

[13] Mahy, R., "A Call Control and Multi-party usage framework for the Session Initiation Protocol (SIP)", draft-ietf-sipping-cc-framework-02 (work in progress), March 2003.

[14] Sparks, R. and A. Johnston, "Session Initiation Protocol Call Control - Transfer", draft-ietf-sipping-cc-transfer-01 (work in progress), February 2003.

[15] Rosenberg, J., Schulzrinne, H., Camarillo, G. and J. Peterson, "Best Current Practices for Third Party Call Control in the Session Initiation Protocol", draft-ietf-sipping-3pcc-03 (work in progress), March 2003.

[16] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-00 (work in progress), May 2003.

[17] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", draft-ietf-simple-presence-10 (work in progress), January 2003.

[18] Rosenberg, J., Schulzrinne, H. and P. Kyzivat, "Caller Preferences and Callee Capabilities for the Session Initiation Protocol (SIP)", draft-ietf-sip-callerprefs-08 (work in progress), March 2003.

[19] Andreasen, F. and B. Foster, "Media Gateway Control Protocol (MGCP) Version 1.0", RFC 3435, January 2003.

[20] Groves, C., Pantaleo, M., Anderson, T. and T. Taylor, "Gateway Control Protocol Version 1", RFC 3525, June 2003.

[21] Moats, R., "URN Syntax", RFC 2141, May 1997.

[22] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", draft-ietf-sipping-reg-event-00 (work in progress), October 2002.

Authors' Addresses

Rohan Mahy
Cisco Systems, Inc.
5617 Scotts Valley Drive, Suite 200
Scotts Valley, CA 95066
USA

EMail: rohan@cisco.com

Orit Levin
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA

EMail: oritl@microsoft.com

Francois Audet
Nortel Networks
4655 Great America Parkway
Santa Clara, CA 95054
USA

EMail: audet@nortelnetworks.com

Intellectual Property Statement

Full Copyright Statement

HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

Internet Engineering Task Force SIPPING WG
Internet Draft G. Hellström (editor)
Document: <draft-manyfolks-sipping-ToIP-01.txt> R. R. Roy (editor)
Feb 2004 A. van Wijk (editor)
Expires: August 2004 Omnitor, AT&T,
Informational Viataal

Framework of requirements for real-time text conversation using SIP.

Status of this Memo

Abstract

This document provides the framework of requirements for text conversation with real time character-by-character interactive flow over the IP network using the Session Initiation Protocol. The requirements for general real-time text-over-IP telephony, point-to point and conference calls, transcoding, relay services, user mobility, interworking between text-over-IP telephony and existing text-telephony, and some special features including instant messaging have been described.

Table of Contents

1. Introduction

Text-over-IP (ToIP) is becoming popular as a part of total conversation among a range of users although this medium of communications may be the most convenient to certain categories of people (e.g., deaf, hard of hearing and speech-impaired individuals). The Session Initiation Protocol (SIP) has become the protocol of choice for control of Multimedia IP telephony and Voice-over-IP (VoIP) communications. Naturally, it has become essential to define the requirements for how ToIP can be used with SIP to allow text conversations as an equivalent to voice. This document defines the framework of requirements for using ToIP, either by itself or as a part of total conversation using SIP for session control.

2. Scope

The primary scope of this document is to define the requirements for using ToIP with SIP, either stand-alone or as a part of a total conversation approach. In general, the scope of the requirements is:

a. Features in Real-Time ToIP
b. Real-time Multimedia Conversational Sessions using SIP
c. General Requirements for Real-Time ToIP using SIP
d. Interworking Requirements for ToIP
e. Text gateways in the different networks

The subsequent sections describe those requirements in detail.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL"

in this document are to be interpreted as described in RFC 2119 [2].

4. Definitions

Full duplex ? user information is sent independently in both directions.

Half duplex ? user information can only be sent in one direction at a time or, if an attempt to send information in both directions is made, errors can be introduced into the user information.

TTY ? name for text telephone, often used in USA, see textphone.

Textphone ?text telephone. A terminal device that allow end-to-end real time text communication. A variety of textphone protocols exists world-wide, both in the PSTN and other networks. A textphone can often be combined with a voice telephone, or include voice communication functions for simultaneous or alternating use of text and voice in a call.

Text telephony ? Analog textphone services

Text Relay Service - A third-party or intermediary that enables communications between deaf, hard of hearing and speech-impaired people, and voice telephone users by translating between voice and text in a call.

Transcoding Services - Services of a third-party user agent (human or automated) that transcodes one stream into another.

Total Conversation - A multimedia service offering real time conversation in video, text and voice according to interoperable standards. All media flow in real time. Further defined in ITU-T F.703 Multimedia conversational services description.

Text gateway ? A multi functionalgateway that sits at the border of a network able to transcode RFC 2793 Interactive text (ToIP) into a different text medium and vice versa. E.g. ToIP into Boudot and vice versa in the PSTN.

Acronyms:

2G Second generation cellular (mobile)
2.5G Enhanced second generation cellular (mobile)
3G Third generation cellular (mobile)
CDMA Code Division Multiple Access
CTM Cellular Text Telephone Modem
GSM Global System of Mobile Communication
ISDN Integrated Services Digital Network
ITU-T International Telecommunications Union ? Telecommunications standardisation Sector
PSTN Public Switched Telephone Network

SIP	Session Initiation Protocol
TDD	Telecommunication Device for the Deaf
TDMA	Time Division Multiple Access
ToIP	Text over Internet Protocol
UTF-8	Universal Transfer Format ? 8

5. Background and General Requirements

The main purpose of this document is to provide a set of requirements for real-time text conversation over the IP network using the Session Initiation Protocol (SIP) [3]. The overall requirements described are such that the real-time text can be expressed as a part of the session description as a part of the total conversation like any other media. Participants can negotiate all media including real-time text conversation[4, 5]. This is a highly desirable function for all IP telephony users,irrespective of whether the users are or are not deaf, hard of hearing, or speech impaired.

It is important to understand that real-time text conversations are significantly different from other text based communications like email or instant messaging. Real-time text conversations deliver an equivalent mode to voice conversations by providing transmission of text character by character as it is entered, so that the conversation can be followed closely and immediate interaction take place, therefore providing the same mode of interaction as voice telephony does. Store-and-forward systems like email or messaging on mobile networks or non-streaming systems like instant messaging are unable to provide that functionality.

One particular application where real-time text is absolutely essential, is the use of relay services between conversational modes, like between text and voice.

Direct text emergency service calls, where time and continuous connection are of the essence, is another essential application.

6. Features in Real-time Text-over-IP

While real-time Text-over-IP will be used for a wide variety of services, an important field of application will be to provide a text equivalent to voice conversation, in particular for deaf, hard of hearing and speech-impaired users.
As such, it is crucial that the conversational nature of this service is maintained. Text based communications exist in a variety of forms, some non-conversational (SMS, text paging, E-mail, newsgroups, message boards, etc.), others conversational (TTY/TDD, Textphone, etc).

Real-time Text-over-IP will sometimes be used in conjunction with a relay service [I] to allow text users to communicate with voice users. With relay services, it is crucial that text characters are

sent as soon as possible after they are entered. While buffering MAY be done to improve efficiency, the delays SHOULD be kept as small as possible. In particular, buffering of whole lines of text MUST NOT be used.

In order to make Real-Time Text-over-IP the equivalent of what voice is to hearing people, it needs to offer equivalent features in terms of conversation as voice communications provides to hearing people. To achieve that, real-time Text-over-IP MUST:
a. Offer Real-Time presentation of the conversation. This means that text MUST be sent as soon as available, or with very small delays. The delay MUST not be longer than 500 milliseconds,
b. Provide simultaneous transmission in both directions,
c. Except for the case of interworking with other networks and protocols (e.g. TTY on PSTN) allow users to interrupt/barge in at any time in the conversation.
d. Except for the case of interworking with other networks and protocols, Real-Time Text-over-IP MUST support a transmission rate of at least 30 characters/second.
e. Support sending redundant data as described in RFC 2793 [5].
f. Be possible to merge with video transmission.

The end-to-end delay in transmission MUST be less than 2000 milliseconds.

Many users will want to use multiple modes of communication during the conversation, either at the same time or by switching between modes e.g. between real-time Text-over-IP and voice. Native real-time Text-over-IP systems MUST support at least the alternate use of modalities and MAY support simultaneous use of modalities.

When communicating via a gateway to other networks and protocols, the system MUST completely support the functionality for alternating or simultaneous modalities as offered by the gateway. When voice is supported on the terminal, the terminal MUST provide volume control.

7. Real-Time Multimedia Conversational Sessions using SIP

The Session Initiation Protocol (SIP) [3] provides mechanisms for creating, modifying, and terminating sessions for real-time conversation with one or more participants using any combination of media: Text, Video and Audio. However, participants are allowed to negotiate on a set of compatible media types (e.g., Text, Video, Audio) with session descriptions used in SIP invitations.

The standardized T.140 real-time text conversation [4], in addition to audio and video communications, will be valuable services to many. Real-time text can be expressed as a part of the session description in SIP and will be a useful subset of the Total Conversation (e.g., Real-time text, Video and Audio).

This specification describes the framework for using the T.140 text conversation in SIP as a part of the multimedia session establishment in real-time over a SIP network.

The session establishment using SIP defines procedures for how T.140 text conversation can be supported using a RTP payload defined in RFC 2793 [5]. The performance characteristics of T.140 will be determined using RTCP.

The session will not only define procedures between the SIP devices having text conversation capability, but will also define how sessions in SIP can be established between the text conversation and audio/video/text capable devices transparently.

If there is any incompatibility between the terminals, e.g. T.140 only and audio-only terminals, the necessary transcoding services will need to be invoked. This important service feature invites a variety of rich capabilities in the transcoding server. For example, speech-to-text (STT), text-to-speech (TTS), text bridging after conversion from speech, audio bridging after conversion from text, and other services can also be provided by the transcoding and/or translation server. The session description protocol (SDP) [6] used in SIP to describe the session also needs to be capable of expressing these attributes of the session (e.g., uniqueness in media mapping for conversion from one media to another for each communicating party).

Real-time texts can also be presented in conjunction with video.

Alerting for T.140 terminals needs to be provided. Users may set up text conversation sessions using SIP from any location. In addition, user privacy and security MUST be provided for text conversation sessions at least equal to that for voice.

The transcoding/translation services can be invoked in SIP using different session establishment models [7]: Third party call control [8] and Conference Bridge model [9].

Both point-to-point and multipoint communication need to be defined for the session establishment using T.140 text conversation. In addition, the interworking between T.140 text conversation and text telephony conversation [10] is needed.

The general requirements for real-time text conversation using SIP can be described as follows:

a. Session setup, modification and teardown procedures for point-to-point and multimedia calls
b. Registration procedures and address resolutions
c. Negotiation procedures for device capabilities
d. Discovery and invocation of transcoding/translation services between the media in the call

e. Different session establishment models for transcoding/translation services invocation: Third party call control and Conference bridge model
f. Uniqueness in media mapping to be used in the session for conversion from one media to another by the transcoding/translation server for each communicating party
g. Media bridging services for T.140 real-time text, audio, and video for multipoint communications
h. Transparent session setup, modification, and teardown between text conversation capable and voice/video capable devices
i. Conversations to be carried out using T.140-over-RTP and RTCP will provide performance report for T.140
j. Altering capability using text conversation during the session establishment
k. T.140 real-time text presentation mixing with voice and video
l. T.140 real-time text conversation sessions using SIP, allowing users to move from one place to another
m. Users? privacy and security for sessions setup, modification, and teardown as well as for media transfer
n. Interoperability between T.140 conversations and text telephony

8. General Requirements for Real-Time Text-over-IP using SIP

The communications environments for ToIP using SIP to set up the conversation in real-time may vary from a simple point-to-point call to multipoint calls in addition to the fact that ToIP can be used in combination with other media like audio and video. In order to establish the session in real-time, the communicating parties SHOULD be provided with experiences like those of normal telephony call setup. There may also be some need for pre-call setup e.g. storing registration information in the SIP registrar to provide information about how a user can be contacted. This will allow calls to be set up rapidly and with proper addressing.

Similarly, there are requirements that need to be satisfied during call set up when another media is preferred by a user. For instance, some users may prefer to use audio while others want to use text as their preferred choice of conversational mode. In this case, transcoding services will need to be invoked for text-to-speech (TTS) and speech-to-text (STT). The requirements for transcoding services need to be negotiated in real-time to set up the session.

The subsequent subsections describe those requirements in great detail.

8.1 Pre-Call Requirements

The desire of the users for using ToIP as a medium of communications can be expressed during registration time. Two situations need to be considered in the pre-call setup environment:

a. User Preferences: It MUST be possible for a user to indicate a preference for ToIP by registering that preference in a SIP server. If the user is called by other party, preferences can be invoked by the SIP server to accept or reject the call based on the rules defined by the user. If the rules require that a transcoding server is needed, the call can be re-directed or handled accordingly.

b. Server to support User Preferences: SIP servers MUST have the capability to act on users preferences for ToIP, based on the users? preferences defined during the pre-call setup registration time.

8.2 Basic Point-to-Point Call Requirements

The point-to-point call will take place between two parties. The requirements are described in subsequent sub-sections. They assume that one or both of the communicating parties will indicate ToIP as the preferred medium for conversation using SIP in the session setup.

8.2.1 General Requirements

The general requirements are that ToIP will be chosen from the available media as the preferred means of communication for the session. However, there may be a need to invoke some underlying capabilities in some cases, for example, a transcoding server may be invoked if one of the users want to use a communication medium other than ToIP.
The following entities MAY need to be involved to facilitate the session establishment using ToIP as another medium:

a. Caller Preferences: SIP headers (e.g., Contact) can be used to show that ToIP is the medium of choice for communications.
b. Called Party Preferences: The called party being passive can formulate a clear rule indicating how a call should be handled either using ToIP as a preferred medium or not, and whether a designated SIP proxy needs to handle this call or it is handled in the SIP user agent (UA).
c. SIP Server support for User Preferences: SIP servers can also handle the incoming calls in accordance to preferences expressed for ToIP. The SIP Server can also enforce ToIP policy rules for communications (e.g., use of the transcoding server for ToIP).

8.2.2 Session Setup

Users will set up a session by identifying the remote party or the service they will want to connect to. However, conversations could be started using a mode other than real-time Text-over-IP. For instance, the conversation might be established using voice and the user could elect to switch to text, or add text, during the conversation. Systems supporting real-time Text-over-IP MUST allow

users to select any of the supported conversation modes at any time, including mid-conversation.

Systems SHOULD allow the user to specify a preferred mode of communication, with the ability to fall back to alternatives that the user has indicated are acceptable.

If the user requests simultaneous use of text and voice, and this is not possible either because the system only supports alternate modalities or because of resource management on the network, the system MUST try to establish a text-only communication. and the user MUST be informed of this change throughout the process, either in text or in a combination of modalities that MUST include text.

Session setup, especially through gateways to other networks, MAY require the use of prefixes or the use of specially formatted URLs.
This MUST be supported by the terminal.

8.2.3 Addressing

The SIP [3] addressing schemes MUST be used for all entities. For example SIP URL and Tel URL will be used for caller, called party, user devices, and servers (e.g., SIP server, Transcoding server).

The right to include a transforming or translating service MUST NOT require user registration in any specific SIP registrar.

8.2.4 Alerting

Systems supporting real-time Text-over-IP MUST have an alerting method (e.g., for incoming calls and messages) that can be used by deaf and hard of hearing people or provide a range of alternative, but equivalent, alerting methods that are suitable for all users, regardless of their abilities and preferences.

It should be noted that general alerting systems exist, and one common interface for triggering the alerting action is a contact closure between two conductors.

Among the alerting options are alerting on the user equipment and specific alerting user agents registered to the same registrar as the main user agent.

If present, identification of the originating party (for example in the form of a URL or CLI) MUST be clearly presented to the user in a form suitable for the user BEFORE answering the request. When the invitation to initiate a conversation involving real-time Text-over-IP originates from a gateway, this MAY be signalled to the user.

8.2.5 Call Negotiations

The Session Description Protocol (SDP) used in SIP [3] provides the capabilities to indicate ToIP as a media for the call setup. RFC 2793 [5] provides the RTP payload type for support of ToIP which can be indicated in the SDP as a part of SDP INVITE, OK and SIP/200/ACK for media negotiations. In addition, SIP?s offer/answer model can also be used in conjunction with other capabilities including the use of a transcoding server for enhanced call negotiations [7,8,9].

8.2.6 Answering

Systems SHOULD provide a best-effort approach to answering invitations for session set-up and users should be kept informed at all times about the progress of session establishment. On all systems that both inform users of session status and support real-time Text-over-IP, this information MUST be available in text, and may be provided in other visual media.

8.2.6.1 Auto-Answer

Systems for real-time Text-over-IP MAY support an auto-answer function, equivalent to answering machines on telephony networks. If an auto-answer function is supported, it MUST support at least 160 characters for the recorded message. It MUST support incoming text message storage of a minimum of 16000 characters, although systems MAY support much larger storage.

When the auto-answer function is activated, user alerting MUST still take place. The user MUST be allowed to monitor the auto-answer progress and MUST be allowed to intervene during any stage of the auto-answer and take control of the session.

8.2.7 Session progress and status presentation

During a conversation that includes real-time Text-over-IP, status and session progress information MUST be provided in text. That information MUST be equivalent to session progress information delivered in any other format, for example audio. Users MUST be able to manage the session and perform all session control functions based on the textual session progress information.

The user MUST be informed of any change in modalities.

Session progress information MUST use simple language as much as possible so that it can be understood by as many users as possible.
The use of jargon or ambiguous terminology SHOULD be avoided at all times. It is RECOMMENDED to let text information be used together with icons symbolising the items to be reported.

There MUST be a clear indication, both visually as well as audibly whenever a session gets connected and disconnected. The user

should never be in doubt as to what the status of the connection is, even if he/she is not able to use audio feedback or vision.

8.2.8 Actions During Calls

Certain actions need to be performed for the ToIP conversation during the call and these actions are describe briefly as follows:

a. Text transmission SHALL be done character by character as entered, or in small groups transmitted so that no character is delayed between entry and transmission by more than 300 milliseconds.
b. The text transmission SHALL allow a rate of at least 30 characters per second so that human typing speed as well as speech to text methods of generating conversation text can be supported.
c. After text connection is established, the mean end-to-end delay of characters SHALL be less than two seconds, measured between two ToIP users. This requirement is valid as long as the text input rate is lower or equal to the text reception and display rate.
d. The character corruption rate SHALL be less than 1% in conditions where users experience the quality of voice transmission to be low but useable. This is in accordance with ITU-T F.700 Annex A.3 quality level T1.
e. When interoperability functions are invoked, there may be a need for intermediate storage of characters before transmission to a device receiving slower than the typing speed of the sender. Such temporary storage SHALL be dimensioned to adjust for receiving at 30 characters per second and transmitting at 6 characters per second during at least 4 minutes [less than 3k].
f. If text is detected to be missing after transmission, there SHALL be an indication in the text marking the loss.
g. When used from a terminal designed for PSTN text telephony, or in interworking with such a terminal, ToIP shall enable alternating between text and voice in a similar manner as the PSTN text telephone handles this mode of operation. (This mode is often called VCO/HCO in USA).
h. The transmission of the text conversation SHALL be made according to an internationally suitable character set and control protocol for text conversation as specified in ITU-T T.140.
i. When display of the conversation on end user equipment is included in the design, display of the dialogue SHALL be made so that it is easy to read text belonging to each party in the conversation.

8.2.8.1 Text and other Media Handling Between ToIP Devices

The native ToIP devices do not need transcoding from speech to text and can communicate directly.

I. When used between terminals designed for native ToIP, it SHALL be possible to send and receive text simultaneously with the other media (text, audio and/or video) supported by the same terminals.

II. When used between terminals designed for native ToIP, it SHALL be possible to send and receive text simultaneously.

8.2.8.2 General Actions

a. It SHALL be possible to establish a session with text capabilities enabled at the beginning of a Call. Note: a call is in this document defined as one or more sessions).
b. It SHALL be possible to place a call without text capabilities, and to add text capabilities later in the call.
c. It SHALL be possible to transfer text at at least 30 characters per second
d. It SHALL be possible to talk and listen simultaneously with typing and reading.

8.2.8.3 Call Action with Native ToIP Devices

a. It SHALL be possible to answer a callwith text capabilities enabled.
b. It SHOULD be possible to use video simultaneously with the other media in the call.
c. It SHALL be possible to answer a callin voice or video without text enabled, and add text later in the call.
d. It SHALL be possible to disconnect the call.
e. It SHOULD be possible to control IVR (Interactive Voice Response) services from a numeric keypad.
f. It SHOULD be possible to control ITR (Interactive Text Response) services from the alphanumeric keyboard.
g. It SHOULD be possible to invoke multi-party calls.
h. It SHALL be possible to transfer the call.
i. It SHOULD be possible to use text characters (numbers) instead of DTMF tones (numbers) in interactions where the person is using a keyboard to interact with a service and the service asks for a number.

8.2.8.4 Audio/Visual/Tactile Indicators

It SHALL be possible to observe visual or tactile indicators about:
- Call progress
- Availability of text, voice and video channels.
- Incoming call.
- Incoming text.
- Typed and transmitted text.
- Any loss in incoming text.

8.2.9 Additional session control

Systems that support additional session control features, for example call waiting, forwarding, hold etc on voice calls, MUST offer equivalent functionality for real-time Text-over-IP functions. In addition, all these features MUST be controllable by text users at any time, in an equivalent way as for other users.

It SHOULD be possible to use text characters (numbers) instead of DTMF tones (numbers) in interactions where the person is using a keyboard to interact with a service and the service asks for a number.

8.2.10 File storage

Systems that support real-time Text-over-IP MAY save the text conversation to a file. This SHOULD be done using a standard file format. It is recommended to use an xhtml [11] format.

8.3 Conference Call Requirements

The conference call requirements deal with multipoint conferencing calls where there will be at least one or more ToIP capable devices along with other end user devices where the total number end user devices will be at least three.

8.4 Transport

ToIP SHALL use RTP as the default transport protocol for transmission of real-time text as specified in RFC 2793 [5]. Signaling and other media will use the transport protocol specified in SIP [3] and/or their revised versions as specified in standards.

The redundancy method of RFC 2198 SHOULD be used for making text transmission reliable with transmission of three generations.

Text capability SHOULD be announced in SDP by a declaration in line with this example:

```
m=text 11000 RTP/AVP 98 100
a=rtpmap:98 t140/1000
a=rtpmap:100 red/1000
a=fmtp:100 98/98
```

Characters SHOULD BE buffered for transmission and transmitted every 300 ms.

By having this single coding and transmission scheme for real time text defined, in the SIP call control environment, the opportunity for interoperability is optimised.

However, if good reasons exist, other transport mechanisms MAY be offered and used for the T.140 coded text, provided that proper negotiation is introduced, and RFC 2793 transport is used as the defaut fallback solution.

8.5 Character Set

a. Real-Time Text-over-IP protocols MUST use UTF-8 encoding as specified in ITU-T T.140 [12]. A number of characters used in traditional text telephony have special meanings.
b. Real-time Text-over-IP SHALL handle characers with editing effect such as new line, erasure and alerting during session as specified in ITU-T T.140.

8.6 Transcoding

Transcoding of text may need to take place in gateways between ToIP and other forms of text conversation. ToIP make use of ISO 10646 character set.
Most PSTN textphones use a 7-bit character set, or a character set that is converted to a 7-bit character set by the V.18 modem.

When transcoding between these character sets and T.140 in gateways, special consideration MUST be paid to the national variants of the 7 bit codes, with national characters mapping into different codes in the ISO 10 646 code space. The national variant to be used SHOULD be possible to select by the user per call, or be configured as a national default for the gateway.

The missing text indicator in T.140, specified in T.140 amendment 1, cannot be represented in the 7 bit character codes. Therefore these characters SHALL be translated to be represented by the ' (apostrophe) character in legacy text telephone systems where this character exists. For legacy systems where the character ' does not exist, the character . (full stop) SHALL be used instead.

8.7 Relay Services

The relay service acts as an intermediary between 2 or more callers.
The basic relay service allows a translation of speech to text and text to speech, which enables hearing and speech impaired callers to communicate with hearing callers. Even though this document focuses on ToIP, we do not exclude video relay services for e.g., speech to sign language and vice versa and other possible relay services. It will be possible to use ToIP simultaneously with other relay services if desired.

It is very important for the users that a relay session is invoked as transparently as possible. It SHOULD happen automatically when the call is being set-up or by a simple user action. A transcoding framework document using SIP [7] describes invoking relay services, where the relay acts as a conference bridge or uses the third party control mechanism.

Adding or removing a relay service MUST be possible without disrupting the current call.

When setting up a call, the relay service MUST be able to determine the type of service requested (e.g. speech to text or text to speech), to indicate if the caller wants voice carry over, the language of the text including the sign language being used.

The user MUST be provided with a method to indicate which service is desired.

It MUST be possible to identify ToIP sessions as emergency sessions.

The relay service operator MUST be able to process such a session correctly and quickly.

a. The relay service operator?s network must give priority to this incoming call.
b. The relay service operator MUST forward this session if they are unable to process it to an alternative emergency relay operator.
c. The relay service MUST label the transcoded stream as an emergency call (in case of text to speech and/or vice versa).
d. The relay service MUST provide all session information to the emergency centre (e.g., location information of the caller if available).

Relay services must be available all the time, even if the users are roaming.

8.8 Emergency services

a. It SHALL be possible to support emergency service calls with text only or simultaneously with voice.
b. All session information that accompanies a voice session to the emergency centre, shall also be provided to the emergency centre if it is a ToIP session.(e.g, phone number and location information of the user placing the emergency call).
c. A text over IP stream must be labelled as an emergency stream to ensure that the emergency service center is able to receive this call.

8.9 User Mobility

ToIP terminals SHALL use the same mechanisms as other terminals to resolve mobility issues. It is RECOMMENDED to use a SIP-adress for the users, resolved by a SIP REGISTRAR, to enable basic user mobility. Further mechanisms are defined for the 3G IP multimedia systems.

8.10 Confidentiality and Security

Users? confidentiality and privacy need to be met as described in SIP [3]. For example, nothing should reveal the fact that the user of ToIP is a person with a disability unless the user prefers to

make this information public. If a transcoding server is being used, this SHOULD be transparent. Encryption SHOULD be used on end-to-end or hop-by-hop basis as described in SIP [3].

Authentication needs to be provided for users in addition to the message integrity and access control.

Protection against Denial-of-service (DoS) attacks needs to be provided considering the case that the ToIP users might need transcoding servers.

8.11 Call Flows

ToIP is a way of establishing the real-time conversation. Call flow for ToIP SHOULD be as similar to audio and video session establishment. For example, ToIP services MAY be invoked in the following situations (among others):

- Noisy environment (e.g., in a machine room of a factory where listening is difficult)Busy with another call and want to participate in two calls at the same time
- Text and/or speech recording services (e.g., text documentation/audio recording for legal/clarity/flexibility purposes)
- Overcoming of language barriers through speech translation and/or transcoding services
- Not hearing well or at all (e.g., hearing loss due to aging, heard of hearing, deaf)

NOTE: In many of the above scenarios, text may accompany speech in a caption like fashion. This would occur for individuals who are hard of hearing and also for mixed calls with a hearing and deaf person listening to the call.

All call flows either for the point-to-point or for the multipoint need to consider that ToIP services may be invoked for many different reasons by users as explained. When the transcoding/translation services are needed, call flows will be shown for both session establishment models: Third-party call control model and Conferencing bridge model.

8.11.1 Call Scenarios

There are 2 different terminal types possible:

1. The terminal itself has the intelligence to initiate a relay service for incoming and outgoing calls (based on address book, user preferences programmed on the terminal etc. This terminal can be used in a conference bridge call as well as a third party control call.

2. Dumb terminals, so that the relay service server actually initiates the correct call handling (the dumb terminal can only

REFER the call to the relay center, which then sets up the call using the conference bridge flow.).

The following call scenarios are shown:

- Communications between two ToIP/Multimedia capable, end user devices using the same language.
- Communications between ToIP capable, end user devices using translation services to provide language translation.
- Communications between ToIP/Multimedia capable and Audio (non-ToIP) capable end user devices.
- Communications between ToIP/Multimedia and/or Audio (non-ToIP)/Multimedia end user devices maintaining privacy.

8.11.2 Point-to-Point Call Flows

The point-to-point calls will contain at least one or both ToIP/Multimedia devices in setting up the session. The detail call flows need to be provided in the following scenarios:

- ToIP/Multimedia devices that use the same language.
- ToIP/Multimedia devices invoke translation services for using different languages.
  * Third-party call control model.
  * Conference bridge service model.
- ToIP/Multimedia devices invoke translation services for using different languages maintaining privacy.
  * Third-party call control model.
  * Conference bridge service model.
- ToIP/Multimedia device and Audio (non-ToIP)/Multimedia device invoking transcoding server.
  * Call initiated by Audio (non-ToIP)/Multimedia user
    - Third-party call control model.
    - Conference bridge service model.
  * Call initiated by ToIP user.
    - Third-party call control model.
    - Conference bridge service model.
- ToIP/Multimedia device and Audio (non-ToIP)/Multimedia device invoking transcoding server maintaining privacy.
  * Call initiated by Audio (non-ToIP)/Multimedia user
    - Third-party call control model.
    - Conference bridge service model.
  * Call initiated by ToIP user.
    - Third-party call control model.
    - Conference bridge service model.

8.11.3 Conference Call Flows

Conference call flows only contain the multipoint communications scenarios, and only the centralized bridge model is considered. The following multipoint conference call flow scenarios will contain at least one more ToIP/Multimedia devices:

- ToIP/Multimedia devices that use the same language.
- ToIP/Multimedia devices invoke translation services for using different languages.
- ToIP/Multimedia devices invoke translation services for using different languages maintaining privacy.
- ToIP/Multimedia device and Audio (non-ToIP)/Multimedia device invoking transcoding server.
  * Call initiated by Audio (non-ToIP)/Multimedia user.
  * Call initiated by ToIP/Multimedia user.
- ToIP/Multimedia device and Audio (non-ToIP)/Multimedia device invoking transcoding server maintaining privacy.
  * Call initiated by Audio (non-ToIP)/Multimedia user.
  * Call initiated by ToIP/Multimedia user.

9. Interworking Requirements for Text-over-IP

A number of systems for real time text conversation already exist as well as a number of message oriented text communication systems. Interoperability is of interest between ToIP and some of these systems. This section describes requirements on this interoperability.

9.1 Real-Time Text-over-IP Interworking Gateway Services

Interactive texting facilities exist already in various forms and on various networks. On the PSTN, it is commonly referred to as text telephony. The simultaneous or alternating use of voice and text is used by a large number of users who can send voice, but must receive text or who can hear but must send text due to a speech disability.

9.2 Text-over-IP and PSTN/ISDN Text-Telephony

On PSTN networks, transmission of interactive text takes place using a variety of codings and modulations, including ITU-T V.21 [II], Baudot, DTMF, V.23 [III] and others. Many difficulties have arisen as a result of this variety in text telephony protocols and the ITU-T V.18 [10] standard was developed to address some of these issues.

ITU-T-V.18 [10] offers a native text telephony method plus it defines interworking with current protocols. In the interworking mode, it will recognise one of the older protocols and fall back to that transmission method when required.

In order to allow systems and services based on Real-time Text-over-IP to communicate with PSTN text telephones, text gateways are the recommended approach. These gateways MUST use the ITU-T V.18 [10] standard at the PSTN side.

Buffering MUST be used to support different transmission rates. At least 1K buffer MUST be provided. 2K is recommended. In addition, the gateway MUST provide a minimum throughput of at least 30

characters/second or the highest speed supported by the PSTN text telephony protocol side, whichever is the lowest.

PSTN-Real-time Text-over-IP gateways MUST allow alternating use of text and voice.

PSTN and ISDN to real-time Text-over-IP gateways that receive CLI information from the originating party MUST pass this information to the receiving party as soon as possible.

Priority MUST be given to calls labeled as emergency calls.

9.3 Text-over-IP and Cellular Wireless circuit switched Text-Telephony

Cellular wireless (or Mobile) circuit switched connections provide a digital real-time transport service for voice or data.
The access technologies include GSM, CDMA, TDMA, iDen and various 3G technologies.

Alternative means of transferring the Text telephony data have been developed when TTY services over cellular was mandated by the FCC in the USA. They are a) ?No-gain? codec solution, b) the Cellular Text Telephony Modem (CTM) solution and c) ?Baudot mode? solution.

The GSM and 3G standards from 3GPP make use of the CTM modem in the voice channel for text telephony.
However, implementations also exist that use the data channel to provide such functionality. Interworking with these solutions SHOULD be done using text gateways that set up the data channel connection at the GSM side and provide real-time Text-over-IP at the other side.

9.3.1 ?No-gain?

The ?No-gain? text telephone transporting technology uses specially modified EFR [15] and EVR [16] speech vocoders in both mobile terminals used provide a text telephony call. It provides full duplex operation and supports alternating voice and text.("VCO/HCO").

9.3.2 Cellular Text Telephone Modem (CTM)

CTM [17] is a technology independent modem technology that provides the transport of text telephone characters at up to 10 characters/sec using modem signals that are at or below 1 kHz and uses a highly redundant encoding technique to overcome the fading and cell changing losses. On any interface that uses analog transmission, half-duplex operation must be supported as the ?send? and ?receive? modem frequencies are identical. The use of CTM may have to be modified slightly to support half-duplex operation.

9.3.3 ?Baudot mode?

This term is often used by cellular terminal suppliers for a GSM cellular phone mode that allows TTYs to operate into a cellular phone and to communicate with a fixed line TTY.

9.3.4 Data channel mode

Many mobile terminals allow the use of the data channel to transfer data in real-time. Data rates of 9600 bit/s are usually supported.

9.3.5 Common Text Gateway Functions

Text Gateways MUST support the differences that result from different text protocols. The protocols to be supported will depend on the service requirements of the Gateway.

Different data rates of different protocols MAY require text buffering.

Interoperation of half-duplex and full-duplex protocols MAY require text buffering and some intelligence to determine when to change direction when operating in half-duplex.

Identification may be required of half-duplex operation either at the ?user? level (ie. users must inform each other) or at the ?protocol? level (where an indication must be sent back to the Gateway).

A Text Gateway MUST be able to route text calls to emergency service providers when any of the recognised emergency numbers that support text communications for the country are called eg. ?911? in USA.

A text gateway (MUST)/SHOULD act transparantly on the IP side. It acts then as a virtual end-point terminal.

9.4 Text-over-IP and Cellular Wireless Text-over-IP

Text-over-IP MAY be supported over the cellular wireless packet switched service. It interfaces to the Internet.

A Text gateway with cellular wireless packet switched services MUST be able to route text calls into emergency service providers when any of the recognized emergency numbers that support text communication for the country are called.

9.5 Instant Messaging Support

Instant Messaging is used by many people to communicate using text via the Internet. Instant Messaging transfers blocks of text

rather than streaming as is used for real-time Text-over-IP. As such, it is not a replacement for real-time Text-over-IP and in particular does not meet the needs for real time conversations of deaf, hard of hearing and speech-impaired users. It is unsuitable for communications through a relay service [I]. The streaming character of real-time Text-over-IP provides a better user experience and, when given the choice, users often prefer real-time Text-over-IP.

However, since some users might only have Instant Messaging available, text gateways might be developed that allow interworking between Instant Messaging systems and real-time Text-over-IP solutions.

Because Instant Messaging is based on blocks of text, rather than on a continuous stream of characters, such gateways need to transform between these two formats. Text gateways for interworking between Instant Messaging and real-time Text-over-IP MUST concatenate individualcharacters originating at the real-time Text-over-IP side into blocks of text and:

a. When the length of the concatenated message becomes longer than 50 characters, the buffered text MUST be transmitted to the Instant Messaging side as soon as any non-alphanumerical character is received from the real-time Text-over-IP side.

b. When a single carriage return, a single line feed, a carriage return/line feed pair or a line feed/carriage return pair is received from the real-time Text-over-IP side, the buffered characters up to that point, including the carriage return and/or line feed characters, MUST be transmitted to the Instant Messaging side.

c. When the real-time Text-over-IP side has been idle for at least 5 seconds, all buffered text up to that point MUST be transmitted to the Instant Messaging side.

Many Instant Messaging protocols signal that a user is typing to the other party in the conversation. Text gateways between Instant Messaging and real-time Text-over-IP MAY provide this signaling to the Instant Messaging side when characters start being received, either at the beginning of the conversation.

It is also possible to introduce the chat feature of certain Instant Messaging protocols. When the chat feature is selected, the IM client should use real-time text over IP. In this way, an IM client can also be used for real-time streaming text over IP.

9.6 IP Telephony with Traditional RJ-11 Interfaces

Analogue adapters using SIP based IP communication and RJ-11 connectors for connecting traditional PSTN devices SHOULD enable connection of legacy PSTN text telephones [18]. These adapters

SHOULD contain V.18 modem functionality, voice handling functionality, and conversion functions to/from SIP based ToIP with T.140 transported in according to RFC 2793, in a similar way as it provides interoperability for voice calls. If a call is set up and RFC2793 capability is not declared by the endpoint (by the end-point terminal or the text gateway in the network at the end-point), a method for invoking a transcoding server shall be used. If no such server is available, the signals from the textphone MAY be transmitted in the voice channel as audio with high quality of service.
NOTE: It is preferred that such analogue adaptors do use RFC2793 on board and thus act as a text gateway. Sending textphone signals over the voice channel is undesirable due posible filtering and compression between the 2 end-points. Which can result in dropping characters in the textphone conversation or even not allowing the textphones to connect with each other.

9.7 Interworking Call Flows

<< this chapter will change depending on how chapter 9.10 works out>>

The call flows in chapter 8.11 deal with end to end ToIP. These call flows do not change on the IP network when one end-point is actually a text gateway. The text gateway actually acts like a ToIP/Multimedia device. Separate call flows will show the interworking between the ToIP/Multimedia devices [4] over the IP network and the text telephony devices [10] over the PSTN/ISDN network using the IP-PSTN/ISDN interworking functional (IWF) entity. It is assumed that the IWF will provide ToIP and text telephony interworking in addition to other capabilities. Thus acting as a Text gateway.

?The point-to-point call flows will contain at least one ToIP/Multimedia and one text telephony/multimedia (or POTS) device for the following cases:

- ToIP/Multimedia device and text telephony/multimedia device that use the same/different language.
- ToIP/Multimedia device and PSTN/ISDN-based POTS/Multimedia device.

For multipoint conferencing calls, it is assumed that only the centralized conferencing will be considered, and the media bridge is supposed to be located somewhere in the SIP network. However, it is considered that the ToIP and text telephony interworking function will be located in the IWF.

The multipoint conference call flows will contain at least one ToIP/Multimedia, at least one text telephony/multimedia device, and other devices where total number of devices will be three or more for the following cases:

- ToIP/Multimedia and text telephony/multimedia devices that use the same/different language.
- ToIP/Multimedia devices, telephony/multimedia devices, and/or PSTN/ISDN-based POTS/Multimedia devices.?

9.8 Multi-functional gateways

The scenarios described in this document deal with single pairs of interworking protocols or services. However, in practice many of these interworking systems will be implemented as gateways that combine different functions. As such, a text gateway could be build to have modems to interwork with the PSTN and support both Instant Messaging as well as real-time ToIP. Such interworking functions are called Combination gateways.

Combination gateways MUST provide interworking between all of their supported text based functions. For example, a text gateway that has modems to interwork with the PSTN and that support both Instant Messaging and real-time ToIP MUST support the following interworking functions:

- PSTN text telephony to real-time ToIP.
- PSTN text telephony to Instant Messaging.
- Instant Messaging to real-time ToIP.

9.9 Gateway Discovery

To get a smooth invocation of the text gateways, where those gateways are transparant on the IP side, it requires a method how and when to invoke the text gateway. As described previously in this draft. The text gateways must act as the end-terminal. The capabilities of the text gateway will in that call be determined by the call capabilities of the terminal that is using the gateway. For example, a PSTN textphone is only able to receive voice and streaming text. Thus the text gateway will only allow ToIP and, in case of VCO or HCO, audio.

The PSTN devices or other non IP multimedia devices that require the text gateways to connect to the IP must be able to locate the text gateway. And ensure that the correct call capabilities of the non IP multimedia device is used by the text gateway.

The following possible solutions for using the text gateway are:

- PSTN Textphone users using a prefix before dialing out.
- In band text dialogue. (???!!!)
- separate text subscriptions, linked to the phone number or terminal identifier/ IP address.
- text capability indicators.
- text preference indicator.
- listen for text activity in all calls.
- call transfer request by the called user.

- placing a call via the web.
- text gateways with its own telephone number and/or SIP address. (this requires user interaction with the text gateway to place a call).
- ENUM.
- etc

9.10 Text Gateway in the call Scenarios

9.10.1 IP terminal calling an analogue textphone (PSTN)

The ToIP stream will be converted into an analogue text telephone protocol (using the voice channel) and vice versa by the text gateway.

The PSTN knows it is a textphone call thanks to the SDP description (for example: m=text 11000 RTP/AVP 98 a=rtpmap:98 t140/1000 for T.140 text on port 11000).

The PSTN will also know that all those incoming calls are only for analogue textphones. Thus the speed of the text stream is adjusted to the selected analogue textphone protocol.
If there is no analogue textphone on the called number, the call setup will be terminated by the text gateway.

The text gateway can be implemented in 2 ways: The PSTN has its own text gateway (the IWF), or it redirects the media stream to the nearest IP-PSTN gateway with text transcoding abilities.

Text gateway detection: In the SIP messages.

9.10.2 IP terminal calling a mobile text telephone (CTM)

The ToIP stream will be converted into CTM and vice versa by the text gateway located in the network of the cellular/mobile operator. It is similar to the PSTN.

Text gateway detection: In the SIP messages.

9.10.3 IP terminal calling a mobile telephone (GPRS based)

A text gateway located in the mobile network converts the incoming T.140/RTP stream into for example T.140 over TCP (T.140/TCP) or tunnels the T.140 stream over HTTP (T.140/HTTP). Or any other temporarily non standard solution necessary to connect the text gateway with the text telephone client on the mobile phone.

This is necessary, since RTP over GPRS is not possible (especially on GPRS phones with Symbian OS).
Note, those server-client solutions are ONLY acceptable for the GPRS and non RTP stack phones. It is encouraged to use T.140/RTP as soon as possible for all mobile phones.

Allowing UDP transport over the GPRS link will enable RFC2793 text over GPRS.

Text gateway detection: In the SIP messages.

9.10.4 IP terminal calling a mobile telephone(UMTS)

No text gateway is required here since this will be end to end IP.

9.10.5 Analogue textphone (PSTN) user calling an IP terminal

The PSTN is unable to distinguish between an analogue voice call and an analogue textphone, both use the voice channel. The text gateway needed to transcode the analogue textphone protocol into T.140/RTP needs to be invoked.

The easiest way for a PSTN to separate an incoming voice call into text telephony or normal voice is by using a prefix number for all incoming text telephone calls to the PSTN. For example , the text telephone user (e.g Boudot) places a call and enters a prefix e.g. 600 and then continues with the original number. The PSTN will recognize all incoming 600 calls as an analogue textphone call and redirects the call to a text gateway (unless it is a number connecting the same PSTN).

It is undesirable to allow a PSTN to transport all the analogue textphone tones/signals through a VoIP stream! (In band text dialogue).

Text gateway detection: Prefix number for incoming textphone calls.

9.10.6 Mobile text telephone (CTM) user calling an IP terminal

The voice channel of the cellular network is used. The MSC is able to separate between the text call and voice only, it is just a matter of redirecting the voice channel to the text gateway.

Text gateway detection: CTM signal detection.

9.10.7 Mobile telephone user (GPRS) calling an IP terminal

The text telephone client on the mobile telephone connects the text gateway located in the network. The text gateway transcodes the text stream into ToIP.

Text gateway detection: pre-programmed in the mobile textphone client.

9.10.8 Mobile telephone (UMTS) user calling an IP terminal

No text gateway is required here since this will be end to end IP.

9.10.9 Voice over DSL user using an analogue text telephone.

In Europe, Voice over DSL is introduced. It is likely that analogue text telephones just use the voice channel. The VoDSL gateway located in the network of the (A)DSL operator itself should connect with a text gateway as soon it turns into VoIP.

Text gateway detection: prefix number similar to the PSTN.

9.10.10 VoIP user via a building telephone switch (at an apartment building) owning an analogue text telephone.

This is the case where only VoIP is possible and no other IP traffic between the telephone switch and the apartments. The question is if this will be implemented.
The only solution would be a forced analogue text telephone protocol over the Voice channel, in band text dialogue . If that must happen. Then the telephone switch MUST convert the analogue text telephone protocol into ToIP and vice versa before the telephone switch connects the IP network.
Note: The in band text dialogue is undesirable. This scenario SHOULD be avoided at any cost.

Text gateway detection: prefix number or in band text signalling.

9.10.11 VoIP user via a gateway/box connected to his/her own Broadband connection owning an analogue text telephone.

The gateway box should natively transcode analogue text telephony into ToIP and vice versa when an analogue text phone is plugged in the RJ-11 socket [18].

Text gateway detection: RJ-11 socket preconfigured by the box via jumpers or software.

10. Terminal Features

Implementers of products that support interactive Text-over-IP SHOULD NOT assume that all users of text are able to use mainstream input and output devices. People with arthritis or other dexterity problems might not be able to use very small keyboards. Visually impaired people might not be able to use standard sized characters on a display. Colour-blind people might suffer from badly chosen colour-schemes. People with motor disabilities might require specialised input devices.

Implementers SHOULD try to make their products as open as possible with regard to this wide range of abilities and preferences and they MUST use standard interfaces wherever they provide such interfaces.

10.1 Text input

Systems that support real-time interactive Text-over-IP SHOULD support suitable input mechanisms, either built-in or connectable through the use of a standard interface: PS/2, USB, Bluetooth, or virtual keyboard. In particular Braille users should be able to connect Braille keyboards to the terminal. Terminals MAY support a web interface for input and output of text.

It is recommended that systems that fixed terminals that support real-time interactive Text-over-IP allow the user to enter the standard alphanumerical characters directly, rather than through a cycle of key presses or other indirect means. This could be done using full-sized keyboards, smaller sized keyboards or fastap keyboards for example. It is highly recommended to provide a standard interface to allow attachment of an external input device, especially for terminals that have only limited input systems built-in.

All IP phones with a display of 12 or more characters MUST support at least text input through the regular phone keypad (and display of any incoming text) in order to provide basic emergency text communication from any IP phone.

Input devices that have automatic key repeat MUST allow the user to specify the key-repeat rate.

10.2 Text presentation

Systems that support real-time interactive Text-over-IP SHOULD support suitable displays, either built-in or connectable through the use of a standard interface: S-VGA, USB, Bluetooth or IP. Braille readers should be connectable to the terminal using a standard interface.

Terminals MAY support a web interface for input and output of text.

While a variety of handsets and terminals might be developed for a number of equally varied scenarios, implementers MUST:

In the case of fixed terminals or software applications on Personal Computers:

a. Use either separate screen areas for displaying sent and received text OR clearly indicate the difference between sent and received text. Systems MAY allow the user to chose either on of these presentation methodologies.

b. Provide at least 5 lines of 35 monospaced characters each for each direction (sent and received text) OR at least 10 lines of 35 characters when sent and received text are presented together.

In the case of Mobile terminals:

c. Use either separate screen areas for displaying sent and received text OR clearly indicate the difference between sent and received text. Systems MAY allow the user to chose either on of these presentation methodologies.

d. Provide at least 3 lines of 20 monospaced characters each for each direction (sent and received text) OR at least 6 lines of 20 characters when sent and received text are presented together.

On both types of terminals, scrolling back through both sent and received text MUST be supported, including after the conversation has ended. Lines SHOULD be wrapped at word boundaries and this is strongly recommended.

There MUST be an easy-to-use function to clear the screen at any time during the session, and if the implementation has chosen to present sent and received text separately, clearing the screen SHOULD be possible as a separate function for sent and received text.

The function of the [CR], [LF] and [BACKSPACE] characters as explained in section 9.5. MUST be supported by the presentation. Presentation layers MUST support the full UTF-8 character set.

When real-time Text-over-IP is used in conjunction with other modalities, like voice, the presentation MUST clearly indicate this to the user in an area outside the display region for send and received text.

Identification information for other parties in the conversation, like URL?s, user-friendly names from an address book, or CLI in the case of conversations with text telephones, SHOULD be displayed throughout the entire conversation in a region outside the sent and received text area.

10.3 Call control

Call (Session) Control procedures MUST use the SIP protocol. Text sessions MUST be identified in accordance with requirements described earlier.

Text services SHOULD be part of a Total Conversation environment in which voice, text and video sessions can be added, modified or deleted individually.

To enable interworking with Textphones in telephone and cellular (mobile) networks, terminals MUST be able to access Gateways automatically when a PSTN or cellular (mobile) E.164-based telephone number is used as the called address.

Users MUST be able to establish text sessions to emergency service providers using the widely recognised emergency numbers in use in the country of operation of the terminal eg. ?911? in USA.

The ability to transfer Location information SHALL be provided if the information is available from the terminal.

10.4 Device control

ToIP will support the text protocol stack described earlier and will require the use of RFC 2793 [5]. RFC 2793 defines the use of ITU-T T.140 [4] over RTP. T.140 is a text presentation protocol that is also used in the ITU-T H.series multimedia systems including some videoconferencing systems. It is also used by ITU-T V.18 [10], the Textphone interworking specification, and by the GSM and 3GPP text conversation specifications.

ToIP will be a full-duplex service. Small displays may require the users to indicate (via text indications at the user level) that a user wishes to communicate in the half-duplex mode. This will require a signal to inform the other user to proceed eg. ?GA? as traditionally used by many half-duplex TTY users.

10.5 Alerting

The form of Alerting indication(s) provided to the user should be selectable to suit particular users. Alerting indications MAY include Sound, Tactile (eg. vibrational), Visual (on-screen symbols; separate flashing light), Motion (eg. movement of something).

The ability to send an Alerting signal to an external interface SHOULD be provided. This will allow Alerting devices that are specific to users requirements to be attached.

As many as possible of the following alternatives for alerting SHALL be provided:

* Internal flash.
* Two-pole connector for external alerting systems triggered by contact between the two poles when a ring signal is generated.
* Bluetooth serial profile with AT command interface, sending the "RING" message, intended for a Bluetooth alerting receiver with flash, vibration or sound action.
* SIP connected alerting device, that get its stimuli by being registered on the same sip address as the terminal.

10.6 External interfaces

Terminals for ToIP SHOULD provide external interfaces for the following functions:

* Text input.
* Text display.
* Terminal control.

* Session control.

10.7 Power

As terminals could remain active for very long periods of time, the electrical power requirements of all the terminals SHOULD be as low as possible.

If the terminal is to be used for calling Emergency services or where the mains power supply is unreliable, back-up power systems SHOULD be provided for the terminal and all equipment used to provide the ToIP service. This can be implemented in many different ways eg. via the line powering option on some Ethernet interfaces, or by using a ?no break? power supply (a battery back-up system with inverters that can recreate a limited amount of mains power).

11. Security Considerations

There are no additional security requirements other than described earlier.

12. Authors? Addresses

The following people provided substantial technical and writing contributions to this document, listed alphabetically:

Barry Dingle
ACIF, 32 Walker Street
North Sydney, NSW 2060 Australia
Tel +61 (0)2 9959 9111
Fax +61 (0)2 9954 6136
TTY +61 (0)2 9923 1911
Mob +61 (0)41 911 7578
email barry.dingle@bigfoot.com.au

Guido Gybels
RNID, 19-23 Featherstone Street
London EC1Y 8SL, UK
Tel +44(0)20 7294 3713
Txt +44(0)20 7296 8019
Fax +44(0)20 7296 8069
EMail: guido.gybels@rnid.org.uk
Gunnar Hellstrom
Omnitor AB
Renathvagen 2
SE 121 37 Johanneshov
Sweden
Phone: +46 708 204 288 / +46 8 556 002 03
Fax: +46 8 556 002 06
Email: gunnar.hellstrom@omnitor.se

Paul E. Jones

Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
Phone: +1 919 392 6948
E-mail: paulej@packetizer.com

Radhika R. Roy
AT&T
Room C1-2B03
200 Laurel Avenue S.
Middletown, NJ 07748
USA
Phone: +1 732 420 1580
Fax: +1 732 368 1302
Email: rroy@att.com

Henry Sinnreich
MCI
400 International Parkway
Richardson, Texas 75081
Email: henry.sinnreich@mci.com

Gregg C Vanderheiden
University of Wisconsin-Madison
Trace R & D Center
1550 Engineering Dr (Rm 2107)
Madison, Wi 53706
USA
gv@trace.wisc.edu
Phone +1 608 262-6966
FAX +1 608 262-8848

Arnoud A. T. van Wijk
Viataal (Dutch Institute for the Deaf)
Research & Development
Afdeling RDS
Theerestraat 42
5271 GD Sint-Michielsgestel
The Netherlands.
Email: a.vwijk@viataal.nl

13. Full Copyright Statement

14. References

14.1 Normative

1. Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.
2. Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997
3. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, ?SIP: Session Initiation Protocol,? RFC 3621, IETF, June 2002.
4. ITU-T Recommendation T.140, ?Protocol for Multimedia Application Text Conversation (February 1998) and Addendum 1 (February 2000).
5. G. Hellström, ?RTP Payload for Text Conversation, RFC 2793, May 2000.
6. G. Camarillo, H. Schulzrinne, and E. Burger, ?The Source and Sink Attributes for the Session Description Protocol,? IETF, August 2003 - Work in Progress.
7. G.Camarillo,?Framework for Transcoding with the Session Initiation Protocol? IETF august 2003 - Work in progress.
8. G. Camarillo, H. Schulzrinne, E. Burger, and A. Wijk, ?Transcoding Service Invocation in SIP using Third Party Call Control,? IETF, August 2003 - Work in Progress.
9. G. Camarillo, ?The SIP Conference Bridge Transcoding Model,? IETF, August 2003 - Work in Progress.
10. ITU-T Recommendation V.18, ?Operational and Interworking Requirements for DCEs operating in Text Telephone Mode,? November 2000.
11. "XHTML 1.0: The Extensible HyperText Markup Language: A Reformulation of HTML 4 in XML 1.0", W3C Recommendation. Available at http://www.w3.org/TR/xhtml1.
12. Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.
13. TIA/EIA/825 ?A Frequency Shift Keyed Modem for Use on the Public Switched Telephone Network.? (The specification for 45.45 and 50 bit/s TTY modems.)
14. Bell-103 300 bit/s modem.

15. TIA/EIA/IS-823-A ?TTY/TDD Extension to TIA/EIA-136-410 Enhanced Full Rate Speech Codec (must used in conjunction with TIA/EIA/IS-840)?
16. TIA/EIA/IS-127-2 ?Enhanced Variable Rate Codec, Speech Service Option 3 for Wideband Spread Spectrum Digital Systems. Addendum 2.?
17. 3GPP TS26.226 ?Cellular Text Telephone Modem Description? (CTM).
18. I. Butcher, S. Lass, D. Petrie, H. Sinnreich, and C. Stredicke, ?SIP Telephony Device Requirements, Configuration and Data,? IETF, February 2004- Work in Progress.

14.2 Informative

I. A relay service allows the users to transcode between different modalities or languages. In the context of this document, relay services will often refer to text relays that transcode text into voice and vice-versa. See for example http://www.typetalk.org.
II. International Telecommunication Union (ITU), ?300 bits per second duplex modem standardized for use in the general switched telephone network?. ITU-T Recommendation V.21, November 1988.
III. International Telecommunication Union (ITU), ?600/1200-baud modem standardized for use in the general switched telephone network?. ITU-T Recommendation V.23, November 1988.
IV. Third Generation Partnership Project (3GPP), ?Technical Specification Group Services and System Aspects; Cellular Text Telephone Modem; General Description (Release 5)?. 3GPP TS 26.226 V5.0.0, March 2001"SIP Telephony Device Requirements, Configuration and Data" by manyfolks,IETF, October 2003.

SIPPING S. Olson
Internet-Draft O. Levin
Expires: August 16, 2004 Microsoft Corporation
February 16, 2004

Extended-REFER framework and other REFER extensions
draft-olson-sipping-refer-extensions-01

Status of this Memo

Copyright Notice

Abstract

This document proposes to generalize and extend REFER method for facilitating various advanced functionalities within SIP systems. The new extensions are explicitly signaled by inclusion of new option tags in the Require header of REFER.

Table of Contents

1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

To simplify discussions of the REFER method and its extensions, three new terms are being used throughout the document:

- REFER-Issuer: the UA issuing the REFER request
- REFER-Recipient: the UA receiving the REFER request
- REFER-Target: the UA designated in the Refer-To URI

2. Introduction

The REFER [3] extension to SIP [2] defines a basis for remote call control that is useful for implementing features such as call transfer. In fact, originally the REFER was created by the SIP WG as a generic request to ask another UA to perform an operation on your behalf, which is much more powerful than just the phone-like transfer feature.)

REFER has a few limitations with respect to implementing more advanced features in conjunction with the dialog info event package [7]. These limitations derive in part from the limited information available in the message/sipfrag [4] content of the associated NOTIFY. Another limiting factor is the requirement to compress the semantics of the referred request in the Refer-To URI by encoding the desired headers as parameters of that URI. Finally, there are situations where the party issuing the REFER request does not need the NOTIFY associated with the REFER, perhaps because that agent is already subscribed to the appropriate package outside of the REFER request. This represents additional unnecessary traffic and state in the REFER-Issuer and REFER-Recipient.

This document proposes to generalize and extend REFER method for facilitating various advanced functionalities within SIP systems. The new extensions are explicitly signaled by inclusion of new option tags in the Require header of REFER.

3. Requirements

Forking prevention: Prevent forking of a REFER request and allow targeting of that REFER request to a specific device or class of device.

Watching period: Allow the REFER-Issuer to watch the progress of the operation triggered by REFER for the whole duration of the requested operation. For example: Beyond the end of the INVITE transaction and for the duration of the remotely established dialog.

Richness of the retrieved information: Expose rich information about the requested operation. For example: Expose the dialog information, caller preferences, and user defined headers of the dialog established at the REFER-Recipient as a result of the REFER.

Operation efficiency: Reduce the number of messages exchanged to perform a REFER operation. For example, suppress the implicit subscription when the information is known by other means.

Operation abstraction: Reduce the amount of detailed knowledge about the remote party that is required from the REFER-Issuer in order to perform a REFER. For example, for remote-cc applications as described in [11].

4. Overview

4.1 The Extended REFER Framework

The extended REFER behavior is signaled by inclusion of "extended-refer" option tag in the Require header of REFER. If the REFER-Recipient does not understand or does not support the "extended-refer" functionality defined in this specification, it MUST return a 420 (Unsupported Extension) response to the REFER request (as per baseline SIP behavior [2]). The REFER-Issuer SHOULD re-issue the REFER request using URI escaping and only the Refer-To: URI to convey the same information if possible.

Support for this extension can be queried in advance using a standard OPTIONS request.

This extension defines the following normative functionality for extended REFER method which differs from the basic REFER specification:

- Refer-To header contains cid which points to the information placed in the REFER body in MIME form.
- The actual format and the meaning of this information are specified by the value of the Content-Type header of REFER.
- Additional application documents MAY introduce more complicate behavior logic and MAY require to use multipart MIME body in order to implement this logic. In these cases, the Content-Type header can not provide enough information to specify this logic. In these cases, the application template logic MUST be signaled by additional means such as by inclusion of a new dedicated "disposition type" in the Content-Disposition header of REFER. It is expected that new disposition types will be defined and registered with IANA per application for being used with extended REFER.
- The REFER-Issuer MAY specify the request for subscription to any event package by including its MIME type in an Accept header. REFER-Recipient SHOULD maintain the subscription till the operation, requested in REFER, is completed.
- REFER-Issuer MAY suppress the implicit refer subscription (by using the norefersub extension) and subscribe to any event package(s) of its interest independently from issuing the REFER.

4.2 Additional REFER Extensions and Behaviors

Additional REFER extensions and behaviors are defined by this document. Although they are orthogonal to the extended-refer framework, many times they will be used together in advanced SIP applications:

- "norefersub" feature tag suppresses implicit REFER subscription. In this case no dialog is established upon issuing REFER and REFER-Issuer MUST ensure that no forking will be applied to REFER in the network.
- "refer-response" feature tag allows for injecting responses into remote dialogs.
- "isfocus" feature parameter being used with REFER allows for conveying conferencing information to remote parties.

For examples of the extended REFER framework usage and additional REFER extensions, please, refer to "SIP Remote CC" [11] and "Multiple REFER" [10].

The following sections discuss the need and specify the mechanism for each of the extended REFER features.

5. Preventing Forking of REFER Requests

The REFER specification allows for the possibility of forking a REFER request which is sent outside of an existing dialog. In many situation, especially in conjunction with the extended-refer mechanism, forking a REFER may result in absolutely incorrect behavior. This is especially true when the REFER is intended to target a specific device, perhaps with specific capabilities.

The REFER-Issuer can ensure that REFER doesnÆt get forked by specifying the REFER-Recipient as GRUU according to mechanism defined in [12]. No extension is required.

6. Replacing Refer-To URI Syntax with a MIME Body

In the original REFER specification, much of the semantics of the REFER request is encapsulated in the Refer-To URI. This URI will commonly encode the method, From, To, Call-ID, Accept-Disposition, Accept-Contact [9], and other headers all in a properly escaped URI. Such a URI can become long, difficult to debug, and prone to URI escaping errors in SIP implementations. The situation becomes more complex if the method is itself a REFER complete with a Refer-To which must contain URI-escaped characters. This double escaping obfuscates things even more and increases the chances of improperly escaping/unescaping of the Refer-To URI.

This specification makes use of the currently unused REFER body to encapsulate all the information about the requested operation (and that is placed and potentially escaped in the refer-To URI according to the original specification). The possible information includes the method, headers, and body of the request which is desired to be created by the REFER-Recipient. One obvious candidate for this is the message/sipfrag MIME type. This MIME type can express all of these: the method, Request-URI, headers, and body.

Use of additional REFER MIME bodies MAY be defined in separate application specifications.

According to this specification, a cid URI [6] is placed in the Refer-To header to reference the MIME content in the body of the REFER.

In the example below, the body of the extended REFER is of type message/sipfrag.

According to the basic REFER specification:

```
REFER sip:b@tradewind.com SIP/2.0
Via: SIP/2.0/TCP issuer.tradewind.com;branch=z9hG4bK-a-1
From: <sip:a@tradewind.com>;tag=1a
To: <sip:b@tradewind.com>
Call-ID: 1@issuer.tradewind.com
CSeq: 234234 REFER
Max-Forwards: 70
Refer-To: <sip:c@tradewind.com;method=INVITE;response=200?Call-ID=1%40target.tradew
   From=b%40tradewind.com;tag=2b&To=c%40tradewind.com;tag=1c>
Accept: application/dialog-info+xml;q=0.5, message/sipfrag;q=0.1
Contact: sip:a@issuer.tradewind.com
```



```
Content-Length: 0
```

According to this extended REFER specification:

```
REFER sip:b@tradewind.com SIP/2.0
Via: SIP/2.0/TCP issuer.tradewind.com;branch=z9hG4bK-a-1
From: <sip:a@tradewind.com>;tag=1a
To: <sip:b@tradewind.com>
Call-ID: 1@issuer.tradewind.com
Supported: gruu
CSeq: 234234 REFER
Max-Forwards: 70
Refer-To: cid:1239103912039@issuer.tradewind.com
Accept: application/dialog-info+xml;q=0.5, message/sipfrag;q=0.1
Require: extended-refer
Contact: sip:a@issuer.tradewind.com
Content-Type: message/sipfrag
Content-Id: <1239103912039@issuer.tradewind.com>
Content-Length: ...

SIP/2.0 200 OK
Call-ID: 1@target.tradewind.com
From: b@tradewind.com;tag=2b
To: c@tradewind.com;tag=1c
```

If the REFER-Recipient does not understand or doesnÆt support the extended-refer option tag, it MUST return a 420 (Unsupported Extension) response to the REFER request (as per baseline SIP behavior [2]). The REFER-Issuer SHOULD re-issue the REFER request using URI escaping and only the Refer-To: URI to convey the same information if possible.

7. Using Arbitrary Event Packages with REFER

The REFER specification [3] mandates the use of the message/sipfrag [4] MIME type for NOTIFYs of the refer event package. These NOTIFYs are sent as part of the implicit subscription created by the REFER. The purpose of the NOTIFY is to communicate the state of the transaction between the REFER-Recipient and the REFER-Target that is created as a result of the REFER

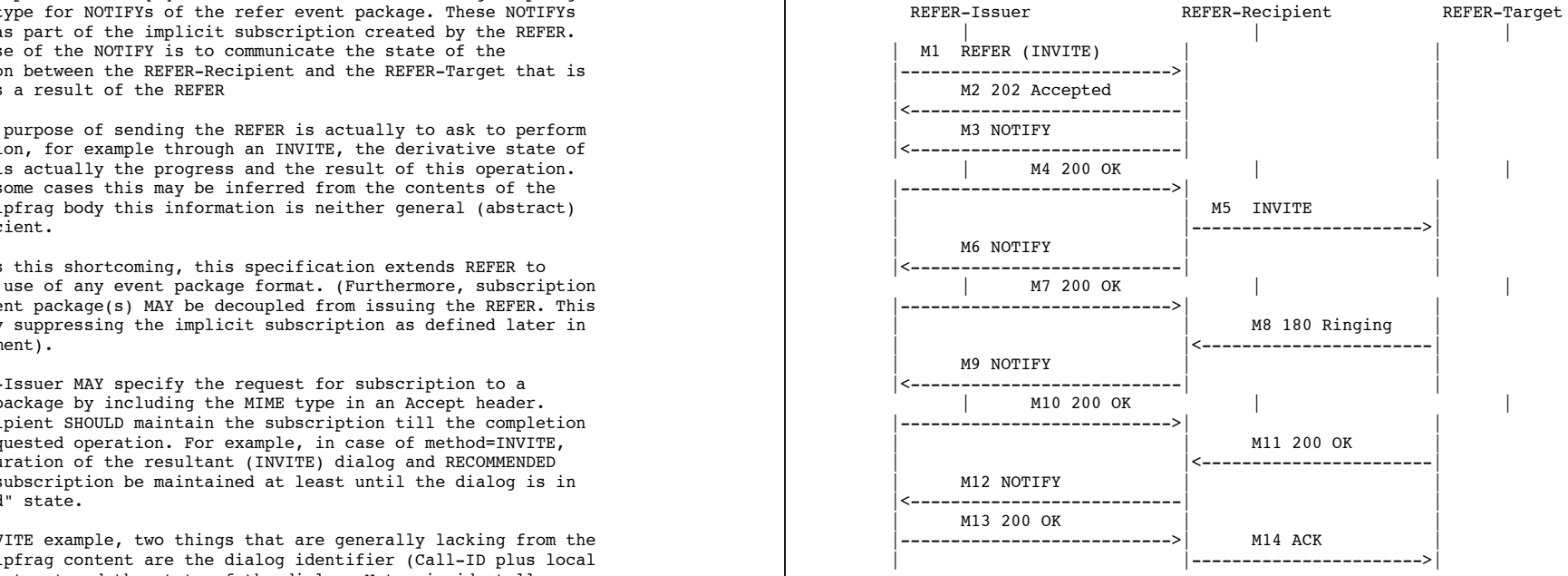
Where the purpose of sending the REFER is actually to ask to perform an operation, for example through an INVITE, the derivative state of interest is actually the progress and the result of this operation. While in some cases this may be inferred from the contents of the message/sipfrag body this information is neither general (abstract) nor sufficient.

To address this shortcoming, this specification extends REFER to allow the use of any event package format. (Furthermore, subscription to the event package(s) MAY be decoupled from issuing the REFER. This is done by suppressing the implicit subscription as defined later in this document).

The REFER-Issuer MAY specify the request for subscription to a specific package by including the MIME type in an Accept header. REFER-Recipient SHOULD maintain the subscription till the completion of the requested operation. For example, in case of method=INVITE, for the duration of the resultant (INVITE) dialog and RECOMMENDED that the subscription be maintained at least until the dialog is in "confirmed" state.

In the INVITE example, two things that are generally lacking from the message/sipfrag content are the dialog identifier (Call-ID plus local and remote tags) and the state of the dialog. Not coincidentally, this is the same information available in the application/ dialog-info+xml MIME type [7] used for the dialog event package.

Figure 1: Example of using application/dialog-info+xml



Message flow:

M1: The REFER-Issuer creates a REFER, specifying support for extended-refer and expressing its interest in the application/ dialog-info+xml MIME type.

```
REFER sip:b@tradewind.com SIP/2.0
Via: SIP/2.0/TCP issuer.tradewind.com;branch=z9hG4bK-a-1
From: <sip:a@tradewind.com>;tag=1ab
To: <sip:b@tradewind.com>
Call-ID: 1@issuer.tradewind.com
Supported: gruu
CSeq: 234234 REFER
Max-Forwards: 70
Refer-To: cid:1239103912039@issuer.tradewind.com
Accept: application/dialog-info+xml
Require: extended-refer
Contact: sip:a@issuer.tradewind.com
Content-Type: message/sipfrag
Content-Id: <1239103912039@issuer.tradewind.com>
Content-Length: ...
```

M5: The REFER-Recipient creates an appropriate INVITE based on the REFER and sends it to the REFER-Target.

```
INVITE sip:c@tradewind.com SIP/2.0
Via: SIP/2.0/TCP recipient.tradewind.com;branch=z9hG4bK-b-1
From: <sip:b@tradewind.com>;tag=1bc
To: <sip:c@tradewind.com>
Call-ID: 1@recipient.tradewind.com
CSeq: 1234567 INVITE
Max-Forwards: 70
Contact: sip:b@recipient.tradewind.com
Content-Length: ...

<SDP for audio call>
```

M6: The REFER-Recipient sends a NOTIFY triggered by the INVITE being sent to the REFER-Target. The body of the NOTIFY contains the dialog identifier and current state ("trying").


```
NOTIFY sip:a@issuer.tradewind.com SIP/2.0
Via: SIP/2.0/TCP recipient.tradewind.com;branch=z9hG4bK-b-2
From: <sip:a@tradewind.com>;tag=1ab
To: <sip:b@tradewind.com>;tag=1ba
Call-ID: 1@issuer.tradewind.com
CSeq: 1278784 NOTIFY
Max-Forwards: 70
Event: refer;id=234234
Subscription-State: active;expires=3600
Contact: sip:b@recipient.tradewind.com
Content-Type: application/dialog-info+xml
Content-Length: ...

<?xml version="1.0" ?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
             version="1"
             state="partial"
       entity="sip:b@tradewind.com">
    <dialog id="2" call-id="1@recipient.tradewind.com"
     local-tag="1bc"
     direction="initiator">
     <state>trying</state>
    </dialog>
</dialog-info>
```

M9: The REFER-Recipient sends a NOTIFY triggered by the 180 Ringing received from the REFER-Target. The body of the NOTIFY contains the dialog identifier and current state ("early").

```
NOTIFY sip:a@issuer.tradewind.com SIP/2.0
Via: SIP/2.0/TCP recipient.tradewind.com;branch=z9hG4bK-b-3
From: <sip:a@tradewind.com>;tag=1ab
To: <sip:b@tradewind.com>;tag=1ba
Call-ID: 1@issuer.tradewind.com
CSeq: 1278785 NOTIFY
Max-Forwards: 70
Event: refer;id=234234
Subscription-State: active;expires=3600
Contact: sip:b@recipient.tradewind.com
Content-Type: application/dialog-info+xml
Content-Length: ...

<?xml version="1.0" ?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
             version="2"
             state="partial"
       entity="sip:b@tradewind.com">
    <dialog id="2" call-id="1@recipient.tradewind.com"
     local-tag="1bc" remote-tag="1cb"
     direction="initiator">
     <state>early</state>
    </dialog>
</dialog-info>
```

M12: The REFER-Recipient sends a NOTIFY triggered by the 200 OK received from the REFER-Target. The body of the NOTIFY contains the dialog identifier and current state ("confirmed").


```
NOTIFY sip:a@issuer.tradewind.com SIP/2.0
Via: SIP/2.0/TCP recipient.tradewind.com;branch=z9hG4bK-b-4
From: <sip:a@tradewind.com>;tag=1ab
To: <sip:b@tradewind.com>;tag=1ba
Call-ID: 1@issuer.tradewind.com
CSeq: 1278786 NOTIFY
Max-Forwards: 70
Event: refer;id=234234
Subscription-State: active;expires=3600
Contact: sip:b@recipient.tradewind.com
Content-Type: application/dialog-info+xml
Content-Length: ...

<?xml version="1.0" ?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
             version="3"
             state="partial"
       entity="sip:b@tradewind.com">
    <dialog id="2" call-id="1@recipient.tradewind.com"
     local-tag="1bc" remote-tag="1cb"
     direction="initiator">
     <state>confirmed</state>
    </dialog>
</dialog-info>
```

8. Suppressing the REFER Implicit Subscription

The REFER specification mandates that every REFER creates an implicit subscription between the REFER-Issuer and the REFER-Recipient. This subscription results in at least one NOTIFY being sent from the REFER-Recipient to the REFER-Issuer. The REFER-Recipient may choose to cancel the implicit subscription with this NOTIFY. The REFER-Issuer may choose to cancel this implicit subscription with an explicit SUBSCRIBE (Expires: 0) after receipt of the initial NOTIFY or by sending a 481 response to this initial NOTIFY request.

The purpose of requiring the implicit subscription and initial NOTIFY is to allow for the situation where the REFER request gets forked and the REFER-Issuer needs a way to see the multiple dialogs that may be established as a result of the forked REFER. This is the same approach used to handle forking of SUBSCRIBE [5] requests. Where the REFER-Issuer explicitly specifies that forking not occur, the requirement that an implicit subscription be established is unnecessary.

Another purpose of the NOTIFY is to inform the REFER-Issuer of the progress of the SIP transaction that results from the REFER at the REFER-Recipient. In the case where the REFER-Issuer is already aware of the progress of the requested operation, such as when the REFER-Issuer has an explicit subscription to the dialog event package at the REFER-Recipient, the implicit subscription and resultant NOTIFY traffic related to the REFER is superfluous and unnecessary network overhead.

To avoid this unnecessary overhead, this document defines a new option tag, norefersub, which specifies that an implicit subscription for event package refer should not be created as a result of accepting this REFER request. Consequently, no dialog is created as the result of sending REFER with Require header containing the norefersub option tag.

This MUST be used by the REFER-Issuer only when the REFER-Issuer can be certain that the REFER request will not be forked. The REFER-Recipient MUST signal support for this extension by inserting a Supported: norefersub header in the 2xx response to the REFER request.

Example of extended REFER which suppresses the implicit subscription


```
REFER sip:b@tradewind.com SIP/2.0
Via: SIP/2.0/TCP issuer.tradewind.com;branch=z9hG4bK-a-1
From: <sip:a@tradewind.com>;tag=1a
To: <sip:b@tradewind.com>
Call-ID: 1@issuer.tradewind.com
Supported: gruu
CSeq: 234234 REFER
Max-Forwards: 70
Refer-To: <sip:c@tradewind.com;method=INVITE>
Require: extended-refer; norefersub
Accept-Contact: *;audio;require
Contact: sip:a@issuer.tradewind.com
Content-Type: message/sipfrag
Content-Id: <1239103912039@issuer.tradewind.com>
Content-Length: ...
```

9. Applying REFER to SIP Response Codes

The original REFER is defined to trigger the sending of a request from the REFER-Recipient to the REFER-Target. The intention is most often to initiate a dialog from the REFER-Recipient to the REFER-Target. This is an excellent way to generate an action at the REFER-Recipient based on an event or action that takes places at the REFER-Issuer. The classic example is call transfer as a result of a user at the REFER-Issuer taking some action.

With the use of the dialog event package, it is possible for one UA to monitor events at another UA related to a dialog, such as the receipt of an INVITE to establish a new dialog. What is lacking is a way for the watcher to indicate what should be the response to such an INVITE request. For example, the dialog watcher would like the recipient of the session initiation request to accept the initiation (send a 200 OK response to the INVITE request). One motivating scenario for this is a set of co-operating User Agents (devices) that belong to the same user. The user, while using one SIP device, wishes to answer a call that is being received on another of that user's SIP devices. This gives the user a single UI focus for control while allowing multiple devices with differing capabilities.

To enable such a scenario, this document defines an extension to the SIP(S) URI syntax as defined in SIP [2]. The extension is analogous to the "method" uri-parameter that currently exists to communicate a method for use in the Refer-To header. A new uri-parameter, "response", is proposed that is used in conjunction with the "method" uri-parameter and associated call-id, local tag, and remote tag to request that the REFER-Recipient send a response within the identified SIP transaction to the REFER-Target.

TBD: Generalize the discussion of the "response" uri-parameter to be used with methods other than REFER.

The REFER-Issuer MUST specify a "method" parameter in addition to the "response" parameter. The REFER-Issuer MUST also include the appropriate local-uri, local-tag, remote-uri, and remote-tag encoded as From and To headers in the Refer-To URI (or using a message/sipfrag body when used in conjunction with the extended-refer mechanism). Note that in order to satisfy this requirement, the REFER-Issuer must have access to this information. In particular, it is assumed that the REFER-Issuer receives the local-uri and remote-uri in the NOTIFY for the dialog event package from the REFER-Recipient. These elements are optional in the XML schema. It is anticipated that User Agents that support these REFER extensions will also include these optional elements in the application/dialog-info+xml payload (as privacy concerns allow).

To ensure the REFER-Recipient conformant with RFC3515 does not misinterpret this as a REFER to send a request of the specified method, the REFER-Issuer MUST also include a Require: refer-response header in the REFER request. REFER-Recipients which do not understand this extension will return a 420 response. The REFER-Target does not need to understand this extension for this to work. Support for this extension can be queried in advance using a standard OPTIONS request.

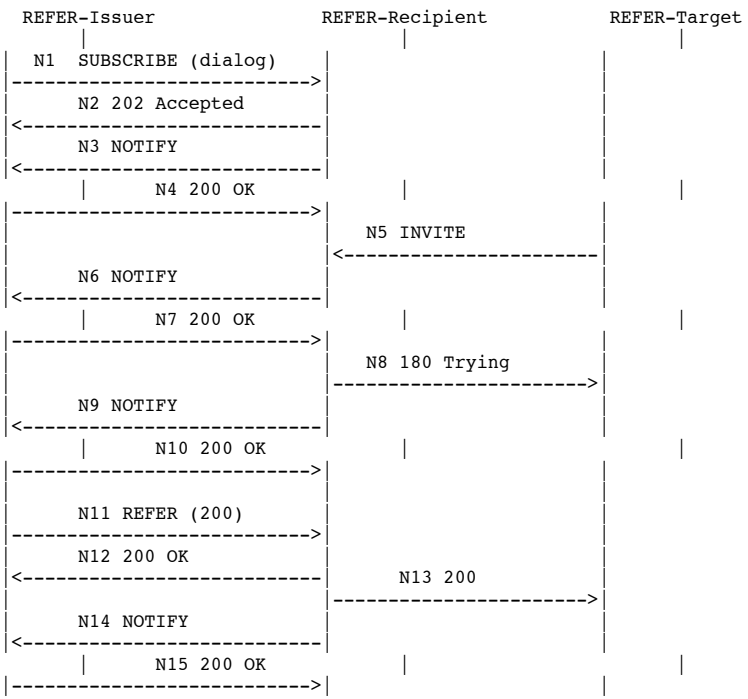
The REFER-Issuer MUST request the use of the application/dialog-info+xml MIME type in NOTIFYs associated with a REFER request which uses the "refer-response" extension.

Note that, although it is RECOMMENDED to use the "refer-response" extension in conjunction to the "extended-refer" framework, the extensions are orthogonal to each other. The extended REFER framework does not include the "refer-response" behavior by default. The "refer-response" extension can be used both with the original REFER mechanism and with the extended REFER framework.

```
uri-parameters    =  *( ";" uri-parameter)
uri-parameter     =  transport-param / user-param / method-param
    / ttl-param / maddr-param / lr-param / response-param / other-param
response-param    = "response=" 1*3DIGIT
```

An example call flow follows:

Figure 2: Example of using the "response" uri-parameter in the Refer-To header



Message flow:

N1: The REFER-Issuer subscribes to the dialog event package at the REFER-Recipient.


```
SUBSCRIBE sip:b@tradewind.com SIP/2.0
Via: SIP/2.0/TCP issuer.tradewind.com;branch=z9hG4bK-a-1
From: <sip:a@tradewind.com>;tag=1ab
To: <sip:b@tradewind.com>
Call-ID: 1@issuer.tradewind.com
CSeq: 234234 SUBSCRIBE
Max-Forwards: 70
Event: dialog
Accept: application/dialog-info+xml
Contact: sip:a@issuer.tradewind.com
Content-Length: 0
```

N5: The REFER-Recipient receives an INVITE from the REFER-Target to start a new call.

```
INVITE sip:b@tradewind.com SIP/2.0
Via: SIP/2.0/TCP target.tradewind.com;branch=z9hG4bK-c-1
From: <sip:c@tradewind.com>;tag=1cb
To: <sip:b@tradewind.com>
Call-ID: 1@target.tradewind.com
CSeq: 1234567 INVITE
Max-Forwards: 70
Contact: sip:c@target.tradewind.com
Content-Length: ...

<SDP for an audio call>
```

N6: The REFER-Recipient sends a NOTIFY triggered by the INVITE received from the REFER-Target. The body of the NOTIFY contains the dialog identifier and current state ("trying").

```
NOTIFY sip:a@issuer.tradewind.com SIP/2.0
Via: SIP/2.0/TCP recipient.tradewind.com;branch=z9hG4bK-b-2
From: <sip:a@tradewind.com>;tag=1ab
To: <sip:b@tradewind.com>;tag=1ba
Call-ID: 1@issuer.tradewind.com
CSeq: 454545 NOTIFY
Max-Forwards: 70
Event: dialog;id=234234
Subscription-State: active;expires=3600
Contact: sip:b@recipient.tradewind.com
Content-Type: application/dialog-info+xml
Content-Length: ...

<?xml version="1.0" ?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
             version="1"
             state="partial"
       entity="sip:b@tradewind.com">
     <dialog id="2" call-id="1@target.tradewind.com"
      remote-tag="1cb"
      direction="recipient">
      <local-uri>b@tradewind.com</local-uri>
      <remote-uri>c@tradewind.com</remote-uri>
      <state>trying</state>
     </dialog>
</dialog-info>
```

N8: The REFER-Recipient sends a 180 Ringing response to the REFER-Target.

```
SIP/2.0 180 Ringing
Via: SIP/2.0/TCP target.tradewind.com;branch=z9hG4bK-b-3
From: <sip:c@tradewind.com>;tag=1cb
To: <sip:b@tradewind.com>;tag=1bc
Call-ID: 1@target.tradewind.com
CSeq: 1234567 INVITE
Contact: sip:b@recipient.tradewind.com
Content-Length: 0
```

N9: The REFER-Recipient sends a NOTIFY triggered by the 180 Ringing sent to the REFER-Target. The body of the NOTIFY contains the dialog identifier and current state ("early").


```
NOTIFY sip:a@issuer.tradewind.com SIP/2.0
Via: SIP/2.0/TCP recipient.tradewind.com;branch=z9hG4bK-b-4
From: <sip:a@tradewind.com>;tag=1ab
To: <sip:b@tradewind.com>;tag=1ba
Call-ID: 1@issuer.tradewind.com
CSeq: 454546 NOTIFY
Max-Forwards: 70
Event: dialog;id=234234
Subscription-State: active;expires=3600
Contact: sip:b@recipient.tradewind.com
Content-Type: application/dialog-info+xml
Content-Length: ...

<?xml version="1.0" ?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
             version="2"
             state="partial"
       entity="sip:b@tradewind.com">
     <dialog id="2" call-id="1@target.tradewind.com"
      local-tag="1bc" remote-tag="1cb"
      direction="recipient">
      <local-uri>b@tradewind.com</local-uri>
      <remote-uri>c@tradewind.com</remote-uri>
      <state event="1xx-tag">early</state>
     </dialog>
</dialog-info>
```

N11: The REFER-Issuer creates a REFER, specifying that the REFER-Recipient should send a 200 OK to accept the session invitation. The From and To headers of the 200 OK are encoded in the Refer-To URI. The local and remote tags for this are determined from the information provided in the NOTIFY for the dialog package. This allows the REFER-Issuer to specify a particular dialog. Combined with the "method" parameter, this identifies a specific transaction within the dialog.

```
REFER sip:b@tradewind.com SIP/2.0
Via: SIP/2.0/TCP issuer.tradewind.com;branch=z9hG4bK-a-2
From: <sip:a@tradewind.com>;tag=1ab
To: <sip:b@tradewind.com>
Call-ID: 2@issuer.tradewind.com
CSeq: 818181 REFER
Max-Forwards: 70
Accept: application/dialog-info+xml;q=0.5, message/sipfrag;q=0.1
Require: refer-response
Refer-To: <sip:c@tradewind.com;method=INVITE;response=200?Call-ID=1%40target.tradew
  To=b%40tradewind.com;tag=1bc&From=c%40tradewind.com;tag=1cb>
Contact: sip:a@issuer.tradewind.com
Content-Length: 0
```

N13: The REFER-Recipient sends a 200 OK to the REFER-Target constructed using the information in the Refer-To header.

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP target.tradewind.com;branch=z9hG4bK-c-1
From: <sip:c@tradewind.com>;tag=1cb
To: <sip:b@tradewind.com>;tag=1bc
Call-ID: 1@target.tradewind.com
Supported: refer-response
CSeq: 1234567 INVITE
Contact: sip:b@recipient.tradewind.com
Content-Length: 0
```

N14: The REFER-Recipient sends a NOTIFY triggered by the 200 OK sent to the REFER-Target. The body of the NOTIFY contains the dialog identifier and current state ("confirmed").


```
NOTIFY sip:a@issuer.tradewind.com SIP/2.0
Via: SIP/2.0/TCP recipient.tradewind.com;branch=z9hG4bK-b-4
From: <sip:a@tradewind.com>;tag=1ab
To: <sip:b@tradewind.com>;tag=1ba
Call-ID: 1@issuer.tradewind.com
CSeq: 454547 NOTIFY
Max-Forwards: 70
Event: dialog;id=234234
Subscription-State: active;expires=3600
Contact: sip:b@recipient.tradewind.com
Content-Type: application/dialog-info+xml
Content-Length: ...

<?xml version="1.0" ?>
<dialog-info xmlns="urn:ietf:params:xml:ns:dialog-info"
             version="3"
             state="partial"
      entity="sip:b@tradewind.com">
    <dialog id="2" call-id="1@target.tradewind.com"
     local-tag="1bc" remote-tag="1cb"
     direction="recipient">
     <local-uri>b@tradewind.com</local-uri>
     <remote-uri>c@tradewind.com</remote-uri>
     <state event="2xx">confirmed</state>
    </dialog>
</dialog-info>
```

10. Adding callid and tag Parameters to Refer-To Header

TBD: Motivation and examples.

11. Using of isfocus Feature Parameter with REFER

This specification allows for using of isfocus feature parameter defined in [8] with REFER.

TBD: Motivation and examples.

12. REFER with a Referred-By Header with aib

REFER with a Referred-By header with an authenticated identity body (aib) with multipart MIME.

TBD: Mechanism and examples.

13. IANA Considerations

13.1 extended-refer Option Tag Registration

This document defines a new option tag, extended-refer, which specifies that the recipient of the REFER request is expected to understand and act upon the extended REFER framework as specified in this document. This option tag is only meaningful for the REFER request defined in RFC3515.

13.2 norefersub Option Tag Registration

This document defines a new option tag, norefersub, which specifies that an implicit subscription for event package refer should not be created as a result of accepting this REFER request. This option tag is only meaningful for the REFER request defined in RFC3515.

13.3 refer-response Option Tag Registration

This document defines a new option tag, refer-response, which specifies that the recipient of the REFER request is expected to issue a response for the SIP transaction requested within the REFER. This option tag is only meaningful for the REFER request defined in RFC3515.

14. Security Considerations

Security considerations regarding inclusion of sensitive information inside the REFER body in MIME format will be addressed in the next version of this document.

15. Acknowledgements

The authors would like to thank Rohan Mahy, Gonzallo Camarillo, and Fran?ois Audet for their insightful comments and inputs. The authors would also like to thank Sriram Parameswar for his ideas being originally presented in draft-parameswar-sipping-norefersub-00 and incorporated in this document.

References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[3] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[4] Sparks, R., "Internet Media Type message/sipfrag", RFC 3420, November 2002.

[5] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[6] Levinson, E., "Content-ID and Message-ID Uniform Resource Locators", RFC 2392, August 1998.

[7] Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-03 (work in progress), October 2003.

[8] Rosenberg, J., "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", draft-ietf-sip-callee-caps-03 (work in progress), January 2004.

[9] Rosenberg, J., Schulzrinne, H. and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", draft-ietf-sip-callerprefs-10 (work in progress), October 2003.

[10] Camarillo, G., "Refering to Multiple Resources in the Session Initiation Protocol (SIP)", draft-camarillo-sipping-multiple-refer-00 (work in progress), February 2004.

[11] Mahy, R., "Remote Call Control in SIP using the REFER method and the session-oriented dialog package", draft-mahy-sip-remote-cc-01 (work in progress), February 2004.

[12] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-00 (work in progress), January 2004.

Authors' Addresses

Sean Olson
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA

Phone: +1-425-707-2846
EMail: seanol@microsoft.com

Orit Levin
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
USA

Phone: +1-425-722-2225
EMail: oritl@microsoft.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING
Internet-Draft
Expires: August 16, 2004

K. Ono
S. Tachimoto
NTT Corporation
Feb 16, 2004

End-to-middle Security in the Session Initiation Protocol(SIP)
draft-ono-sipping-end2middle-security-01

Status of this Memo

Copyright Notice

Abstract

A Session Initiation Protocol (SIP) User Agent (UA) does not always trust all proxy servers in a request path enough to allow them inspect the message bodies and/or headers contained in a message. The UA might want to protect the message bodies and/or headers from all proxy servers except the particular proxy that provides services that depend on the ability to inspect them. In this situation, SIP needs a mechanism for securing information passed between the UA and an intermediary proxy, also called "end-to-middle security", which can work with end-to-end security. This document proposes mechanisms to achieve end-to-middle security, mainly data confidentiality for end-to-middle communication.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [1].

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [2] supports hop-by-hop security using TLS [3] and end-to-end security using S/MIME [4]. However, a UA sometimes wants to protect the message bodies and/or headers from all proxy servers except a selected proxy server, which provides some sort of service based on their content. Such a proxy is not always adjacent to the UA. These situations require security between the UA and the intermediary proxy server for the message bodies and/or message headers. We call this "end-to-middle security", where by "end" we mean a UA and by "middle" we mean a specific proxy.

End-to-middle security is useful in a network where trusted and partially trusted proxy servers both exist in a message path. The partially trusted proxy servers are trusted only to view headers related to routing. The trusted proxy servers are trusted to view the message bodies and/or headers to provide services based on their content. For a UA requiring such services from intermediaries, end-to-end confidentiality will currently have to be disabled to take advantage of them. This problem is pointed out in Section 23 of [2].

Some examples of services that a proxy provides using the content of message bodies and/or headers follow. One example is firewall traversal. A midcom agent co-located with a proxy server controls a firewall entity. The agent needs to view certain Session Description Protocol (SDP) attributes in a message body or the same kind of data in a SIP header. Another example is the archiving of instant messaging traffic, where the archiving function co-located with a proxy server logs the message bodies in the MESSAGE method. A similar example is the archiving of all SIP headers and bodies traffic after being checked by the proxy server. These services might be deployed for financial or health care applications, where archiving communications is required by policies, as well as other applications.

This document describes proposed mechanisms to achieve data confidentiality and the integrity of end-to-middle security to meet the requirements discussed in [9]. The major requirement is to be compatible with the end-to-end encryption that the S/MIME mechanism provides. Therefore, the proposed mechanisms are based on S/MIME. The mechanisms consists of the creation of encrypted data that is not readable by other proxy servers and the indication by the UA of the data that the UA requests the proxy server to read. In addition, it also includes a mechanism for the discovery of intermediate proxy servers.

2. Overview of Proposed Mechanisms

First, UAs MUST support the creation of CMS EnvelopedData body for multiple recipients for end-to-middle confidentiality. For compatibility with end-to-end security, the data needs to be encrypted for UAs and selected proxy servers. Second, UAs SHOULD support an indication mechanism to specify content in S/MIME that needs to be disclosed to a selected proxy server. Proxy servers MUST support to inspect the indicated content in S/MIME CMS bodies. Third, UAs MAY support a discovery mechanism to find which proxy in a signaling path needs to inspect and/or validate what data. In some cases, UAs will be statically configured with the intermediary proxy's policies and so they would not need to use this discovery mechanism. Proxy servers SHOULD provide the discovery mechanism to notify their policies of UAs.

2.1 Creation of S/MIME CMS Bodies for UAs and Proxy servers

Since end-to-middle security needs to be compatible with end-to-end security, a creation mechanism for S/MIME CMS body is required. For end-to-end data integrity, UAs use S/MIME CMS SignedData body that can be validated by any entity. Therefore no new CMS SignedData creation mechanism is required.

For data confidentiality, UAs use S/MIME CMS EnvelopedData body, whose recipients are specified. There are two ways of creating this data. The first way is to create an S/MIME CMS EnvelopedData body that contains encrypted content that is addressed to multiple recipients, such as a UA and a selected proxy server. UA MUST create an CMS EnvelopedData body can contain multiple recipients for encrypted data as specified in [7]. The structure contains data encrypted with a content-encryption-key (CEK) and the CEK is then encrypted with different key-encryption-keys (KEKs), that are actually the public keys for each recipient. For example, one would be for the recipient UA and another would be for the selected proxy server in the end-to-middle case.

The second way is to create multiple S/MIME CMS EnvelopedData bodies, one for each recipient. For example, one for UA and one for a selected proxy server, and make them part of a multipart MIME body. UAs SHOULD use this method when keying materials, such keys for use by Secure RTP (SRTP)[14], are included in the SDP. One CMS EnvelopedData body contains SDP that includes keying materials of an SRTP stream only for the UA, and the other EnvelopedData body contains an SDP that does not include the keying materials of an SRTP stream that are for the UA and a selected proxy server that needs to view SDP (i.e.: for a firewall control service).

2.2 Indicating the Target Content

When proxy servers receive a message, the proxy server MUST inspect the "Content-Disposition" MIME headers to determine whether to process the S/MIME bodies and if so, which one. UA MUST support a new parameter called "required-entity" in the "Content-Disposition" MIME header that indicates required proxy servers to view the content of the MIME body. There is less of an impact on proxy servers that do not support end-to-middle security because these proxy servers do not inspect the "Content-Disposition" MIME header anyway.

Note: There is an altenative option that use a new SIP header. The proposed mechanism requires more cost on proxy servers to determine the necessity of MIME body handling than using a new SIP header. However, the proxy can view the indicated MIME body more effectively than using a new SIP header. In addition, using a new SIP header could be negative impact on intermediary proxy servers that do not support end-to-middle security, causing unnecessary processing load. We feel that this MIME header parameter mechanism is not as simple, but it is equally effective.

2.3 Discovery of Proxy Server's Policies

A discovery mechanism for proxy server's policies is needed when UAs do not know the policies of the proxy server in a signaling path and the proxy server has its own policy for providing some services. When the proxy server receives a request in which it cannot view some data that must be read in order to proceed or the proxy server receives a request whose sending policy cannot be accepted, the proxy MUST send a response with an error code. If the request is in plain text, the error code SHOULD be 403 (Forbidden) accompanied with a required Content-Type, such as "application/sdp". If the request is in plain text and the digital signature of it is required for an integrity check, the error code SHOULD be 403 (Forbidden) accompanied with a required Content-Type that is "multipart/signed". If the request contains encrypted data, the error code SHOULD be 493 (Undecipherable), accompanied with a proxy's public key certificate and required Content-Type.

Open Issues: Which header is the appropriate one to use to set the required Content-Types in a response? When nested Content-Types are required such as "Content-Type: multipart/signed" for "Content-Type: application/pkcs7-mime;smime-type=enveloped-data", how will it be described?

When the UA receives one of the above error codes, the UA needs to authenticate the proxy server. Therefore, the error code SHOULD

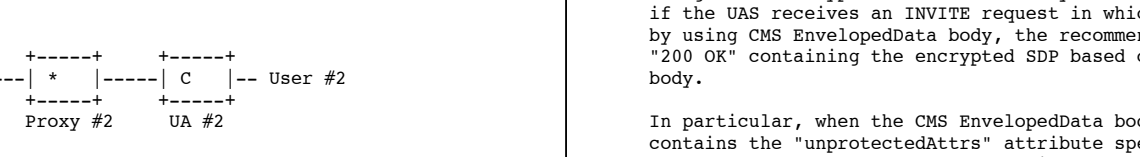
contain the digital signature of the proxy server.

In the worst case, this discovery mechanism requires two messages for each proxy server in the signaling path to establish a session between the UAs. In addition, it requires validation procedures using the digital signatures for all proxy servers. Since this causes a increase in the delay before session establishment, it is recommended that UAs learn in advance the policies of as many proxy servers as they can.

Open Issue: How does this mechanism apply in the case when a proxy server needs to inspect the message body contained in the request in order to provide a service for a UAS? This might be happen where there a firewall in the network on the UAS's side.

3. Behavior of UAs and Proxy Servers

We describe here some examples of the behavior of UAs and proxy servers in a model in which trusted and partially trusted proxy servers are mixed along a message path. In the following example, User #1 does not know the services or security policies of Proxy #1. User#1 sends an INVITE request including encrypted SDP for end-to-end security as shown in Figure 1. Proxy #1 may reject the request because its inability to offer a firewall traversal service. Or Proxy #1 may delete the encrypted data from the body based on a security policy that prevents it from sending unknown data.



C: Content that UA #1 allows the entity to inspect
*: Content that UA #1 prevents the entity from inspecting

Figure 1: Deployment example

3.1 UAC Behavior

When a UAC sends a request and requires end-to-end and end-to-middle confidentiality of the message bodies and/or headers, it uses S/MIME to encrypt them. In the above examples, UA #1 MUST use CMS EnvelopedData body for UA #2 and Proxy #1. At the SIP layer, UA #1 MUST require Proxy #1 to decrypt selected content and to view the content by using the "required-entity" parameter in the Content-Disposition header. Proxy #1 then provides some services based on the decrypted content.

When the UAC needs Proxy #1 to inspect the message bodies and/or headers in the response, it SHOULD request the UAS to encrypt the response by using the same CEK as for the request. The UAC uses the "unprotectedAttrs" attribute to stipulate reuse of the CEK and indicate its identifier as described in [10] [11].

When the UAC sends a request and needs end-to-end and end-to-middle integrity for the message bodies and/or headers, it uses S/MIME to attach a digital signature. In the above examples, it uses the CMS SignedData body of the contents. At the SIP layer, UA #1 requires Proxy #1 to validate the integrity of the selected content by

employing the "required-entity" parameter.

When the UAC receives a response that uses S/MIME, it decrypts and/or validates the S/MIME bodies as usual. If it receives a response that uses CMS EnvelopedData body with the "KEKRecipientInfo" type of "RecepientInfo" attribute, it should decrypt the "RecipientInfo" by using the same CEK as for the sending request.

3.2 UAS Behavior

When a UAS sends a response for the request with this mechanism, using the same type of S/MIME CMS body is recommended. For example, if the UAS receives an INVITE request in which the SDP is encrypted by using CMS EnvelopedData body, the recommended response would be a "200 OK" containing the encrypted SDP based on CMS EnvelopedData body.

In particular, when the CMS EnvelopedData body of the request contains the "unprotectedAttrs" attribute specifying reuse of the CEK, the UAS SHOULD encrypt a CEK with the CEK that was used in the request, instead of the public key of the UAC. The UAS SHOULD use the CMS EnvelopedData body to contain the encrypted SDP in the "200 OK" response. In addition, the UAS SHOULD set the same proxy server in the "required-entity" parameter of the "Content-Dispositon" MIME header in the request.

If the UAS encrypted the SDP with a CEK that was itself encrypted with the CEK in the request, the proxy server selected by the UAC can view the SDP in the "200 OK" response.

> Note: In the case when the response does not contain a message body, even if the request contains a message body and was encrypted by using CMS EnvelopedData body, the UAS does not need to use the CMS EnvelopedData body.

When the UAS receives a request that uses S/MIME, it decrypts and/or validates the S/MIME bodies as usual.

When the UAS sends a response for the request without this mechanism and needs end-to-end and end-to-middle confidentiality of the message bodies and/or headers , it MUST use CMS EnvelopedData to encrypt them. When the UAC sends a request and needs end-to-end and end-to-middle integrity of the message bodies and/or headers, it MUST use CMS SignedData to attach a digital signature. This is the same way the UAC normally performs with this mechanism.

3.3 Proxy Behavior

When a proxy supporting this mechanism receives a message, the proxy server MUST inspect the "Content-Disposition" MIME header and the "required-entity" parameter of that. If the MIME header includes the processing server's own name, the proxy server MUST inspect the specified content.

When the specified content is CMS EnvelopedData body, the proxy server MUST inspect it and try to decrypt the "recipientInfo" attribute. If the proxy server fails to decrypt that, it SHOULD cancel the subsequent procedure and respond with a 493 (Undecipherable) response if it is a request, or any existing dialog MAY be terminated. If the proxy server succeeds in this decryption, it MUST inspect the "unprotectedAttrs" data of the CMS EnvelopedData body. If the attribute gives the key's identifier, the proxy server MUST keep the CEK with its identifier during the dialog. When it receives subsequent messages in the dialog, it MUST try to decrypt the "KEKRecipientInfo" type of "recepientInfo" attribute by using this CEK.

When the specified content is CMS SignedData body, the proxy server MUST inspect it and validate the digital signature. If the verification is failed, the proxy server SHOULD reject the subsequent procedure and respond with a 403 (Forbidden) response if the message is a request, or any existing dialog MAY be terminated.

When the proxy server forwards the request, it modifies the routing headers normally. It does not need to modify the S/MIME body.

If a proxy does not support this mechanism and receives a message with the "required-parameter" parameter in the "Content-Disposition" header, the proxy MUST ignore the header and perform as usual.

4. Summary of Content-Disposition Header Field Use

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC-2234 [13]. The new parameter "required-entity" is defined in "required-param" as one of "disp-param".

```
Content-Disposition   =  "Content-Disposition" HCOLON
                         disp-type *( SEMI disp-param )
disp-type             =  "render" / "session" / "icon" / "alert"
                         / disp-extension-token
disp-param            =  handling-param / required-param / generic-param
handling-param        =  "handling" EQUAL
                         ( "optional" / "required" / other-handling )
other-handling        =  token
disp-extension-token  =  token
required-param        =  "required-entity" EQUAL
                         proxy-uri *(COMMA proxy-uri)
proxy-uri             = ( name-addr / addr-spec )
```

5. Examples

The following examples illustrate the use of the mechanism defined in the previous section.

5.1 Request Example

In the following example, a UA needs SDP in an INVITE message to be confidential and the UA allows a proxy server to view the SDP in an INVITE request. In addition, the UA needs to protect the policy of the proxy server. In the example encrypted message below, the text with the box of asterisks ("*") is encrypted:

```
 INVITE alice@atlanta.example.com --> ss1.atlanta.example.com


INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
Route: <sip:ss1.atlanta.example.com;lr>
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Contact: <sip:alice@client.atlanta.example.com;transport=tcp>
Date: Fri, 20 June 2003 13:02:03 GMT
Content-Type: multipart/signed;protocol="application/pkcs7-signature";
         micalg=sha1;boundary=boundary1
Content-Length: ...

--boundary1

Content-Type: application/pkcs7-mime;smime-type=enveloped-data;
              name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: session;filename=smime.p7m;handling=required;
                     required-entity=ss1.atlanta.example.com
Content-Length: ...

*****************************************************************
* (encryptedContentInfo)                                        *
* Content-Type: application/sdp                                 *
* Content-Length: ...                                           *
*                                                               *
* v=0                                                           *
* o=alice 2890844526 2890844526 IN IP4 client.atlanta.example.com*
* s=-                                                           *
* c=IN IP4 192.0.2.101                                          *
```



```
* t=0 0                                                         *
* m=audio 49172 RTP/AVP 0                                       *
* a=rtpmap:0 PCMU/8000                                          *
*                                                               *
* (recipientInfos)                                              *
* RecepientInfo[0] for ss1.atlanta.example.com public key       *
* RecepientInfo[1] for bob's public key                         *
*                                                               *
* (unprotectedAttr)                                             *
* CEKReference                                                  *
*****************************************************************

--boundary1--
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
filename=smime.p7s;handling=required

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary1--
```

5.2 Response Example

In the following example, a UA sends a response with this mechanism. In the example encrypted message below, the text boxed in asterisks ("*") is encrypted:

```
 200 OK bob@biloxi.example.com --> ss2.biloxi.example.com


SIP/2.0 200 OK
Via: SIP/2.0/TCP
ss2.biloxi.example.com:5060;branch=z9hG4bK721e418c4.1
;received=192.0.2.222
Via: SIP/2.0/TCP ss1.atlanta.example.com:5060;branch=z9hG4bK2d4790.1
;received=192.0.2.111
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
;received=192.0.2.101
Record-Route: <sip:ss2.biloxi.example.com;lr>,
<sip:ss1.atlanta.example.com;lr>
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=314159
```

```
Call-ID: 384827629822018851l@atlanta.example.com
CSeq: 2 INVITE
Contact: <sip:bob@client.biloxi.example.com;transport=tcp>
Content-Type:multipart/signed;protocol="application/pkcs7-signature";
             micalg=sha1;boundary=boundary41
Content-Length: ...

--boundary41

Content-Type: application/pkcs7-mime;
              smime-type=enveloped-data;name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: session;
filename=smime.p7m;handling=required;
Content-Length: ...

*****************************************************************
* (encryptedContentInfo)                                        *
* Content-Type: application/sdp                                 *
* Content-Length: ...                                           *
*                                                               *
* v=0                                                           *
* o=bob 2890844527 2890844527 IN IP4 client.biloxi.example.com  *
* s=-                                                           *
* c=IN IP4 192.0.2.201                                          *
* t=0 0                                                         *
* m=audio 3456 RTP/AVP 0                                        *
* a=rtpmap:0 PCMU/8000                                          *
*                                                               *
* (recipientInfos)                                              *
* RecepientInfo[0] for KEKidentifier                            *
*****************************************************************

--boundary41--

Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;handling=required

hhhHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary41--
```


6. Security Considerations

This specification is about applying S/MIME-secured messages for use in end-to-middle security. It is also about applying the CEK reuse method defined in [10]. This requires the same security consideration as those of [4] and [10].

7. IANA Considerations

This document requires a new parameter in "Content-Disposition" header fields, namely "required-entity".

8. Acknowledgments

Thanks to Rohan Mahy and Cullen Jennings for their initial support of this concept, and to Jon Peterson for his helpful comments.

References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[3] Allen, C. and T. Dierks, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[4] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1992.

[5] Srisuresh, P., Kuthan, J., Rosenberg, J., Brim, S., Molitor, A. and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, August 2002.

[6] Campbell, Ed., B., Rosenberg, J., Schulzrinne, H., Huitema, C. and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[7] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[8] Rosenberg, J., "Requirements for Session Policy for the Session Initiation Protocol (SIP)", draft-rosenberg-sipping-session-policy-req-00 (work in progress), December 2002.

[9] Ono, K. and S. Tachimoto, "Requirements for end-to-middle security in the Session Initiation Protocol (SIP)", draft-ietf-sipping-e2m-sec-reqs-01 (work in progress), February 2004.

[10] Farrell, S. and S. Turner, "Reuse of CMS Content Encryption Keys", RFC 3185, October 2001.

[11] Ono, K. and S. Tachimoto, "Key reuse in S/MIME for SIP", draft-ono-sipping-keyreuse-smime-00 (work in progress), February 2004.

[12] Sparks, R., "Internet Media Type message/sipfrag", RFC 3420, November 2002.

[13] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[14] Andreasen, F., Baugher, M. and D. Wing, "Session Description Protocol Security Descriptions for Media Streams", draft-ietf-mmusic-sdescriptions-03.txt (work in progress), February 2004.

Authors' Addresses

Kumiko Ono
Network Service Systems Laboratories
NTT Corporation
9-11, Midori-Cho 3-Chome
Musashino-shi, Tokyo 180-8585
Japan

EMail: ono.kumiko@lab.ntt.co.jp

Shinya Tachimoto
Network Service Systems Laboratories
NTT Corporation
9-11, Midori-Cho 3-Chome
Musashino-shi, Tokyo 180-8585
Japan

EMail: tachimoto.shinya@lab.ntt.co.jp

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING K. Ono
Internet-Draft S. Tachimoto
Expires: August 8, 2004 NTT Corporation
Feb 8, 2004

Key reuse in Secure MIME for the Session Initiation Protocol(SIP)
draft-ono-sipping-smime-keyreuse-00

Status of this Memo

Copyright Notice

Abstract

SIP uses Secure MIME (S/MIME) Cryptographic Message Syntax (CMS) EnvelopedData to protect SIP messages for confidentiality. While SIP can be encrypted with different keying materials for each message, it usually requires a public key operation for each message and the computational cost of such operations are relatively expensive. This draft proposes a method of bidirectional key exchange to reuse keying materials for S/MIME-secured messages in a dialog and use a symmetric key mechanism instead of an asymmetric key mechanism such as a public key operation. The proposed mechanism also achieves the sharing of keying material among multiple entities in a simple way.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [1].

Table of Contents

1. Introduction

The SIP [2] supports S/MIME [3] CMS [4] EnvelopedData for confidentiality. The CMS EnvelopedData contains content encrypted with a content encryption key (CEK) and the CEKs are encrypted with key encryption keys (KEKs), which are usually public keys of recipients. The confidential service is currently used for ensuring end-to-end security, and it is now being considered for use in end-to-middle security as described in [5]. In SIP, several messages are transmitted among User Agents (UAs) via proxy servers in a dialog. While separate keying materials can be used for each recipient and each message, public key operations and asymmetric key mechanisms are required for each recipient and each message.

As for end-to-end confidentiality, a User Agent Client (UAC) needs to send a User Agent Server (UAS) a request with its own public key certificate (PKC) that is a relatively large amount of data in order to make sure that the UAC can receive a response properly using the CMS EnvelopedData. If multiple UAs join a dialog, all UAs need to send other UAs a request with its own PKC and send other UAs subsequent messages with multiple KEKs for other UAs. These operations increase the data size of the initial request by using the originator's PKC and the number of KEKs in subsequent messages.

As for end-to-middle confidentiality that combines with end-to-end one, a UAC needs to send a UAS a request with its own PKC and a selected proxy server's one in order to make sure that the UAC and the proxy server can receive a response properly using the CMS EnvelopedData. The UAS also need to create the response explicitly using the two PKCs. This complicates the specification of end-to-middle confidentiality.

This draft proposes a method to reuse keying materials for subsequent messages in SIP. This reuse method is based on [6]. Since the reuse mechanisms allow UAs to avoid public key operations for each message, UAs can create CMS EnvelopedData with low computational cost. In addition, the reuse mechanism also achieves the sharing of keying materials among multiple entities including proxy servers in a simple way. It can also reduce the data size of the initial request, the number of KEKs in subsequent messages, and the complication of the end-to-middle security's specification.

2. Overview of proposed solution

This proposed solution has three phases based on [6]. The first phase is preparing for a CEK to be reused as the KEK in a subsequent message. The second phase is reusing the KEK derived from a CEK in subsequent messages, while the CEK is updated for each message. The third phase is ending the reuse when a KEK is updated or the lifetime for key reuse ends. The mechanism at the third one needs some additional considerations for SIP.

2.1 Preparation for reuse

A method of preparation is to include a key identifier of a CEK in the CMS EnvelopedData in order to reuse the CEK, a symmetric key, as a KEK of the EnvelopedData in a subsequent message as described in [6]. A "CEKReference" of "unprotectedAttrs" attributes contains the key identifier of the symmetric key and the attribute expresses a sender's preferences to reuse the CEK as the KEK in subsequent messages.

As a prerequisite for a UAC to send a request using the CMS EnvelopedData to a UAS, the UAC needs to know the public key of the UAS in order to use this public key as the KEK. The UAC creates a symmetric key to be used as the CEK. If a UAC needs to reuse the CEK and/or needs to share the CEK among multiple UASs, it MUST use a "CEKReference" attribute in a request message to stipulate reuse of the CEK in subsequent messages and indicate its identifier. When a UAS needs to reuse the CEK, the UAS MUST use a "CEKReference" attribute in a response message to request a UAC to reuse the CEK as the KEK of subsequent messages.

2.2 Reuse CEK as KEK

The following describes the method for KEK reuse, where the KEK is derived from a CEK. After a UAS receives EnvelopedData that contains a "CEKReference" attribute in a request message, the UAS creates an EnvelopedData with the CEK received from the UAC as the KEK and does not set a "CEKReference" attribute. Not setting a "CEKReference" attribute indicates that the KEK can be reused as the KEK of subsequent messages.

When a UAC requests to reuse the CEK, the UAC does not know if a UAS can support this key reuse mechanism. If the UAS supports this key reuse mechanism, the UAS SHOULD use a symmetric key received from the UAC as the KEK to encrypt a new CEK of a response message. The CMS EnvelopedData that the UAS creates contains a "KEKRecipientInfo" type of "recipientInfo" attribute. When receiving the response, the UAC will be able to determine that the UAS supports key reuse and uses

it. If the UAs support and decide to use this reuse mechanism, the UAC does not need to send its own PKC. This is because the UAS can create the CMS EnvelopedData with a new CEK and the KEK derived from a CEK previously received from the UAC.

If the UAS does not support this reuse mechanism, or for some reason cannot use it based on a policy, the UAS MUST use the UAC's public key as a KEK to encrypt a CEK in response. When receiving the EnvelopedData, the UAC will be able to determine that the UAS did not accept the request to enable key reuse. The UAC will need to send its own PKC in a request so that the UAS can create a response with a valid the CMS EnvelopedData.

Open issue: For end-to-middle security, how does a UA know whether a proxy server support this key reuse mechanism or not ? One option is that the proxy server adds a digital signature in a response when it uses the key reuse mechanism.

2.3 Lifetime of key reuse

The reused CEK is available until the KEK is updated or the maximum lifetime ends. The originator and recipients SHOULD maintain the "CEKReference" attribute until the reused CEK is expired.

In [6], the maximum lifetime of the CEK is indicated in a "CEKMaxDecrypts" attribute in the "unprotectedAttrs" field of EnvelopedData. If "CEKMaxDecrypts" is missing, or has the value "1", then each CEK will be reused once as the KEK for the next message.

Generally, reusing the same key many times is weak from a security viewpoint. When a UA wants to stop reusing the same KEK, the UA can update the KEK. The UA MUST follow the method of the preparation for reuse as described above.

In SIP, a UA can know whether a recipient UA receives and reuses the CEK, when the UA receives a subsequent message. However, a UA does not always receive a subsequent message to a provisional response and an ACK request. The UA SHOULD not update the KEK that is derived from the CEK in such messages even when the "CEKMaxDecrypts" value is one. That results in the situation that the number in "CEKMaxDecrypts" does not work correctly in SIP. Therefore, the maximum lifetime of key reuse in SIP equals to the time until the dialog ends . The reused CEK is available on several messages until the dialog ends at the maximum lifetime of key reuse. If the message that indicates the reuse of the CEK does not create a dialog, the reuse is only available in a transaction.

3. Examples

The following examples illustrate the use of the mechanism described in the previous section.

3.1 The reused CEK Lifetime in a dialog

When a UA needs to protect Session Description Protocol (SDP) in a message for end-to-end confidentiality, the messages that include the offer/answer procedures use the CMS EnvelopedData. The CEK is reused in a dialog as illustrated in Figure 1.

```
UAC -> UAS: INVITE
            E-CEK_1(Content), E-pub_key.UAS(CEK_1),CEK_1_id
UAC <- UAS: 200 OK
            E-CEK_2(Content), E-CEK_1(CEK_2)

UAC -> UAS: re-INVITE
            E-CEK_3(Content), E-CEK_1(CEK_3)
UAC <- UAS: 200 OK
            E-CEK_4(Content), E-CEK_1(CEK_4)
```

Figure 1: Example of key reuse in a dialog for end-to-end confidentiality

```
E-CEK_n(Content)  : Content encrypted using CEK_n
E-pub_key.x(CEK_n): CEK_n encrypted using x's public key
E-CEK_n(CEK_m)    : CEK_m encrypted using CEK_n
CEK_n_id          : Key identifier of CEK_n in "CEKReference"
```

When a UA needs to protect SDP in a message for end-to-middle confidentiality that combines with end-to-end one, the messages for the offer/answer procedures use the CMS EnvelopedData. The CEK is reused in a dialog as illustrated in Figure 2.

```
UAC -> Proxy: INVITE
              E-CEK_1(Content), E-pub_key.UAS(CEK_1), E-pub_key.proxy(CEK_1),CEK_1_
Proxy -> UAS: INVITE
              E-CEK_1(Content), E-pub_key.UAS(CEK_1), E-pub_key.proxy(CEK_1),CEK_1_

Proxy <- UAS: 200 OK
              E-CEK_2(Content), E-CEK_1(CEK_2)
UAC <- Proxy: 200 OK
              E-CEK_2(Content), E-CEK_1(CEK_2)
```

```
UAC -> Proxy: re-INVITE
              E-CEK_3(Content), E-CEK_1(CEK_3)
Proxy -> UAS: re-INVITE
              E-CEK_3(Content), E-CEK_1(CEK_3)

Proxy <- UAS: 200 OK
             E-CEK_4(Content), E-CEK_1(CEK_4)
UAC <- Proxy: 200 OK
             E-CEK_4(Content), E-CEK_1(CEK_4)
```

Figure 2: Example of key reuse in a dialog for end-to-middle confidentiality

3.2 The reused CEK Lifetime when used in the case of a subsequent message

When a UA needs to protect some SIP headers for end-to-end confidentiality, all messages in a dialog use the CMS EnvelopedData. The CEK is reused in a subsequent message as illustrated in Figure 3. When sending a provisional response like 180, the CEK is updated and named as CEK_2. CEK_2 may not be received by a UAC. Therefore, when sending the final response like 200, UAS must use CEK_1 as the KEK again.

```
UAC -> UAS: INVITE
             E-CEK_1(Content), E-pub_key.UAS(CEK_1),CEK_1_id
UAC <- UAS: 180
             E-CEK_2(Content), E-CEK_1(CEK_2),CEK_2_id
UAC <- UAS: 200 OK
             E-CEK_3(Content), E-CEK_1(CEK_3),CEK_3_id
UAC -> UAS: ACK
             E-CEK_4(Content), E-CEK_3(CEK_4),CEK_4_id

UAC -> UAS: BYE
             E-CEK_5(Content), E-CEK_3(CEK_5),CEK_5_id
UAC <- UAS: 200 OK
             E-CEK_6(Content), E-CEK_4(CEK_6)
```

Figure 3: Example of key reuse in a subsequent message

4. Security Considerations

TBD.

5. IANA Considerations

This document introduces no additional considerations for IANA.

References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[3] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1992.

[4] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[5] Ono, K. and S. Tachimoto, "Requirements for end-to-middle security in the Session Initiation Protocol (SIP)", draft-ietf-sipping-e2m-sec-reqs-00 (work in progress), October 2003.

[6] Farrell, S. and S. Turner, "Reuse of CMS Content Encryption Keys", RFC 3185, October 2001.

Authors' Addresses

Kumiko Ono
Network Service Systems Laboratories
NTT Corporation
9-11, Midori-Cho 3-Chome
Musashino-shi, Tokyo 180-8585
Japan

EMail: ono.kumiko@lab.ntt.co.jp

Shinya Tachimoto
Network Service Systems Laboratories
NTT Corporation
9-11, Midori-Cho 3-Chome
Musashino-shi, Tokyo 180-8585
Japan

EMail: tachimoto.shinya@lab.ntt.co.jp

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

SIPPING JF Rey
Internet Draft O Rousseau
Expires: July 2004 Alcatel
J. Elwell
Siemens

February 2004

Interworking between Session Initiation Protocol (SIP) and QSIG for call transfer

draft-rey-sipping-qsig2sip-transfer-00.txt

Status of this Memo

Abstract

This document specifies call transfer interworking between the Session Initiation Protocol (SIP) and QSIG within corporate telecommunication networks (also known as enterprise networks). SIP is an Internet application-layer control (signalling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include, in particular, telephone calls. QSIG is a signalling protocol for creating, modifying and terminating circuit-switched calls, in particular telephone calls, within Private Integrated Services Networks (PISNs). QSIG is specified in a number of ECMA Standards and published also as ISO/IEC standards.

Table of Contents

1. Introduction

This document specifies signalling interworking between QSIG and the Session Initiation Protocol (SIP) in support of call transfer within a corporate telecommunication network (CN) (also known as enterprise network).

QSIG is a signalling protocol that operates between Private Integrated Services eXchanges (PINX) within a Private Integrated Services Network (PISN). A PISN provides circuit-switched basic services and supplementary services to its users. QSIG is specified in ECMA Standards, in particular [2] (call control in support of basic services), [3] (generic functional protocol for the support of supplementary services) and a number of Standards specifying individual supplementary services. Transfer services are specified in [5], [7] and the QSIG signalling protocol in support of these services is specified in [6], [8]. In particular, this signalling protocol signals information about call transfer to the users involved.

NOTE. The name QSIG was derived from the fact that it is used for signalling at the Q reference point. The Q reference point is a point of demarcation between two PINXs.

SIP is an application layer IP protocol for establishing, terminating and modifying multimedia sessions. Telephone calls are considered as a type of multimedia session where just audio is exchanged. SIP is defined in [1].

As the support of telephony within corporate networks evolves from circuit-switched technology to Internet technology, the two technologies will co-exist in many networks for a period, perhaps several years. Therefore there is a need to be able to establish, modify, terminate and transfer sessions involving participants in the SIP network and participants in the QSIG network. Such calls are supported by gateways that perform interworking between SIP and QSIG.

This document specifies SIP-QSIG signalling interworking for transfer services between a PISN employing QSIG and a corporate IP network employing SIP.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [4] and indicate requirement levels for compliant SIP implementations.

In the interests of keeping the normative text and the diagrams as simple as possible, the QSIG messages in this document implicitly follow QSIG signalling rules of [2] and [3]. For instance, sending a QSIG DISCONNECT message on a call where a QSIG DISCONNECT has already been sent is implicitly forbidden and therefore not mentioned as such in this document.

The figures in this document are provided as examples. They are not normative. In the interests of keeping the diagrams simple, some SIP messages (ACK, PRACK, final responses to BYE and NOTIFY) are not shown.

The following notation is used for call transfer information within QSIG messages:

- xxx.inv - invoke application protocol data unit (APDU) of operation xxx.
- xxx.res - return result APDU of operation xxx.
- xxx.err - return error APDU of operation xxx.

The following abbreviations are used:

- ctActive stands for callTransferActive.
- ctComplete stands for callTransferComplete.

The drawings use the following conventions :

- D1 and D2 are SIP dialogs. CR1 and CR2 are QSIG call references. By convention, D1 is mapped to CR1 and D2 to CR2.
- A SIP message is prefixed by (Dx-y), when it belongs to SIP dialog Dx and is part of SIP transaction y.
- The method or response code of the SIP messages is displayed, as well as the name of SIP header fields that play a role in the interworking functions. Some examples display an "Identity:" information field. It indicates that the local identity information field should be mapped to a real SIP identity information field as described in section 8.2.

3. Definitions

For the purposes of this specification, the following definitions apply.

3.1 External definitions

The definitions in [2] and [1] apply as appropriate.

3.2 Other definitions

3.2.1 Call transfer

The act of enabling a user to transform two of that user's calls (at least one of which must be answered) into a new call between the two other users in the two calls.

Note : Call transfer is very similar to the "attended transfer" described in [17].

A Call transfer before answer is a Call transfer that occurs before User C answers the call initiated by User A.

3.2.2 Single Step Call transfer

The act of enabling a served user (User A) to transfer an active call (with User B) to a user (User C) which has no call established either to User A or to User B. On successful completion of Single Step Call transfer User B and User C can communicate with each other and User A will no longer be involved in a call with User B or User C.

Note: Single-Step Call transfer is very similar to the "basic transfer" described in [17].

3.2.3 Call transfer by join

The effecting of transfer by joining together the connections of the calls to User B and User C at User A's PINX.

3.2.4 Call transfer by rerouteing

The effecting of transfer by establishing a new connection to replace all or part of the connections of the calls to User B and User C.

3.2.5 Corporate telecommunication Network (CN)

Sets of privately-owned or carrier-provided equipment that are located at geographically dispersed locations and are interconnected to provide telecommunication services to a defined group of users.

NOTE 1. A CN can comprise a PISN, a private IP network (intranet) or a combination of the two.

NOTE 2. Also known as enterprise network.

3.2.6 Gateway

An entity that performs interworking between a PISN using QSIG and an IP network using SIP.

3.2.7 IP network

A network, unless otherwise stated a corporate network, offering connectionless packet-mode services based on the Internet Protocol (IP) as the network layer protocol.

3.2.8 Private Integrated Services Network (PISN)

A CN or part of a CN that employs circuit-switched technology.

3.2.9 Private Integrated services Network eXchange (PINX)

A PISN nodal entity comprising switching and call handling functions and supporting QSIG signalling in accordance with [2].

3.2.10 User A

The served user, i.e. the user requesting Call transfer or Single-Step Call transfer.

3.2.11 User B

A user who is in communication with User A and who will be transferred to User C.

NOTE. This definitions differs from [5], in order to use similar conventions for QSIG Call transfer and QSIG Single-Step Call transfer.

3.2.12 User C

The user to whom the call is transferred.

4. Acronyms

APDU Application Protocol Data Unit
IP Internet Protocol
PINX Private Integrated services Network eXchange

PISN Private Integrated Services Network
SIP Session Initiation Protocol
UA User Agent
UAC User Agent Client
UAS User Agent Server

5. Background and architecture for SIP-QSIG interworking

The background and architecture of [11] applies. In addition, the interworking function in the protocol model handles interworking for call transfer services. This involves interworking between the QSIG call transfer protocol specified in [6],[8] and SIP [1], including the use of UPDATE [10] and REFER [9] SIP methods, and Replaces [12], and Referred-By [13] SIP header fields.

6. Call transfers in QSIG

For the purposes of QSIG, transfers are classified into one of the following types:

- call transfer by join: defined in 3.2.3 ;
- call transfer by rerouteing: defined in 3.2.4 ; and
- single-step call transfer: defined in section 3.2.2.

QSIG Call transfer by rerouteing is out of scope of this document because of its lesser deployment.

QSIG signalling for transfers is based on [3] and comprises the following remote operations:

- ssctInitiate - this confirmed operation is sent by User A's PINX to User B's PINX. It initiates a single-step call transfer. It comprises a "rerouteingNumber" of User C. It comprises an optional "transferredAddress" of User B, an optional "transferringAddress" of User A, and a optional boolean "awaitConnect" that indicates if the operation return result is expected when the call is connected or when it is in alerting state.

- callTransferActive - this unconfirmed operation is sent to User B. It indicates that answer has taken place following an alerting transfer. It comprises a "connectedAddress" that identifies the other User that answered the transferred call.

- callTransferComplete - this unconfirmed operation is sent to User B and User C. It indicates that a transfer has been effected. It comprises an alerting indication referred later in this document as

"callStatus". It comprises a "redirectionNumber" that identifies the other User in the transferred call.

- ssctSetup - this confirmed operation is sent by User B to User C. It initiates a new call between User B and User C for the purpose of single step call transfer. It comprises an optional "transferringAddress" that indicates who initiated the transfer.

- callTransferUpdate - this optional unconfirmed operation is sent to User B and User C. It informs each other of all details about the transferred users that are known to the network.

- subaddressTransfer - this optional unconfirmed operation is sent to User B or to User C. It informs each other of the other party's public ISDN subaddress. It comprises a "connectedSubaddress" that identifies the public ISDN subaddress.

7. Call transfer in SIP

SIP has the concept of requesting a UA to refer (establish a dialog to) a third party [9]. SIP also has the concept of replacing a dialog by another one [12], and [13] provides a mechanism so that information about the initiator of the refer request is sent to the third party. The informational [17] gives examples on how to support call transfers thanks to these SIP extensions.

8. Transfer interworking

8.1 Scope of the interworking functions

8.1.1 QSIG side

The interworking functions provided in the following sections encompass QSIG Call transfer by join and QSIG Single-Step Call transfer. QSIG Call transfer by rerouteing is out of scope of this document because of its lesser deployment. The functions take into account that User A, B and C can be in the IP network or in the PISN.

8.1.2 SIP side

The interworking functions rely on SIP mechanisms and are therefore scoped by them. Thus, the interworking functions of this document make the following assumptions :

- the gateway and the SIP User Agents use globally routable SIP addresses, or use SIP addresses in an environment where they are

routable, or will use other future mechanisms that allow global routing,

- it is RECOMMENDED that the gateway and the SIP User Agents have and apply security policies regarding mutual interactions and exchange of information.

The required implementations for these assumptions are defined in the SIP documents, and are therefore out of scope of this document.

8.1.3 Discussion over transfer interworking functions

This section is not normative and aims at giving explanations concerning the implementation choices of this document.

The QSIG Single-Step Call transfer mechanism is very similar to the informational "basic transfer" described in [17]. The QSIG Call transfer is also very similar to the informational "attended transfer" described in [17]. The latter uses the REFER method and the Replaces header. Yet, it is not possible to use this mechanism in all the interworking situations. For instance, if two gateways are used, there is no opportunity to optimize the signalling path by using the REFER method in the IP network. Figure 1 gives an example of such a situation where transfer by join has been performed at user A's PINX. The gateway to user B is unaware that user C is also in the IP network (and even if it were aware, it has no information for building a Replaces header). Similarly the gateway to user C is unaware that user B is in the IP network.

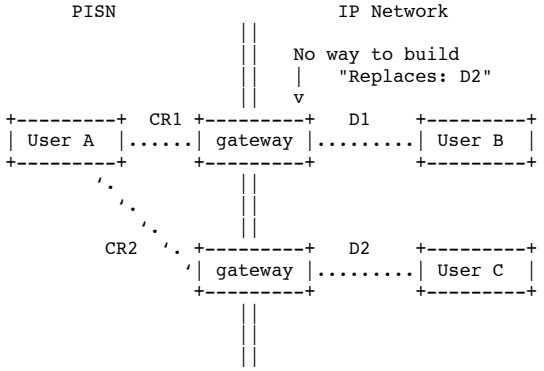


Figure 1: Example of a call transfer by join that involves two gateways

Of course, an optimization is possible when only one gateway is used, so that the gateway can send a SIP REFER request to User B. This optimization is implementation dependant and is out of scope of this document.

When the transfer takes place in the PISN, updating the display (name, address, and dialog state of the new remote party) of the SIP User Agents is needed. An existing dialog between a gateway and a SIP User Agent could be "refreshed" thanks to a SIP INVITE transaction or thanks to a SIP UPDATE transaction. Unfortunately, there is no clearly defined mechanism in SIP to update the display of the SIP User Agents with these transactions.If the situation is clarified, this document will be updated to take it into account. The present solution is based on the gateway sending an INVITE with new dialog identifiers and a Replaces header field that matches the existing dialog. How the QSIG and SIP fields that indicate identities are derived is described in section 8.2.

Another issue is the lack of SIP mechanism to indicate to User B in the IP network that User C in the PISN is in alerting state. In-band media information (e.g. ringback tone, announcements) may help to inform User B that the call to User C is not connected yet.

Another issue is that the gateway does not always know whether the new call triggered by a single step call transfer requested from the PISN must be placed in the IP network or in the PISN. The preferred solution is to reach User C in the IP network if the address derivation is possible. Upon failure of address derivation or upon failure of the REFER request, the call is placed in the PISN. Figure 2 illustrates this situation.

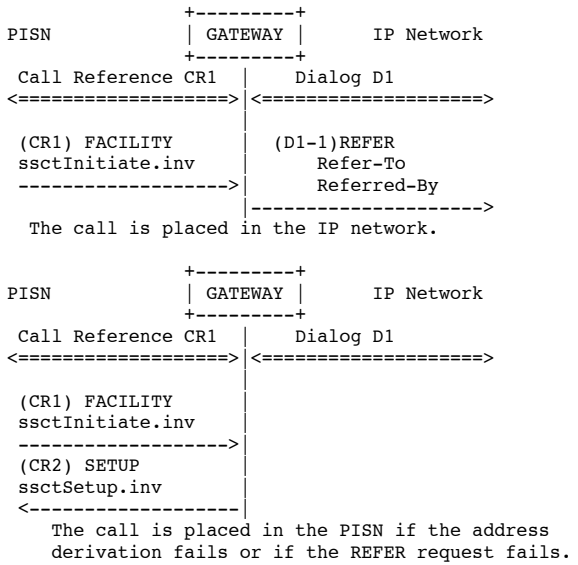


Figure 2: Example of call-flows on receipt of a ssctInitiate invoke APDU.

There is also an ambiguity when a SIP REFER requests arrives at a gateway because the gateway might not know if User C (the transfer target) is in the PISN or in the IP network. Therefore the gateway does not know whether to issue an INVITE request or map the REFER request to a QSIG ssctInitiate invoke APDU. If there is an embedded Replaces header field as a parameter of the Refer-To URI in the REFER request, the gateway knows that User C can be reached through the IP network. If there is no Replaces header field, and if a PISN number can be derived from the Refer-to URI, the gateway sends a QSIG ssctInitiate invoke APDU. If the derivation fails or upon failure of the QSIG ssctInitiate invoke APDU,.a SIP INVITE message is sent accordingly to the REFER request. Figure 3 illustrates this situation.

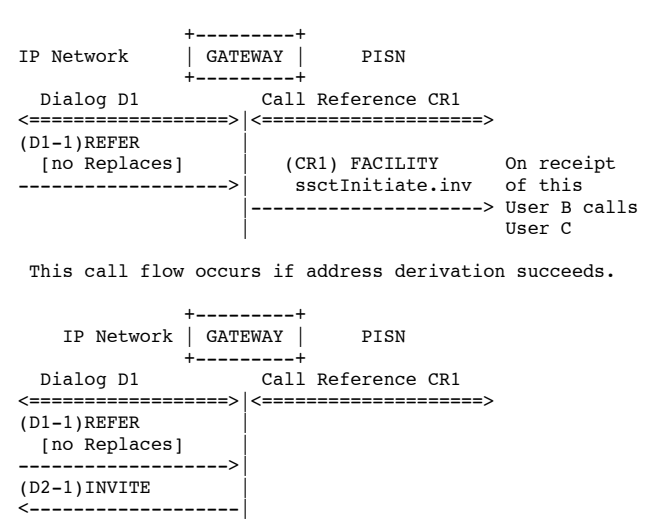


Figure 3: Example of call-flows on receipt of a REFER request.

The interworking of call transfer before answer is an open issue. A Call transfer before answer cannot be implemented with a Replaces header field, because Replaces cannot match an early dialog at the UAS. Therefore "attended transfer" as defined in [17] cannot happen until User C answers the call. [17] suggests that a fallback to "unattended transfer" is used if attended transfer cannot be done.

8.2 Mapping of numbers and URIs

Most of the interworking functions require the mapping of identifiers, e.g., the identifier User A, User B and User C. In QSIG users are identified by numbers. In SIP users are identified by URIs. Mapping of identifiers is described in detail in [18].

In some cases it may not be possible for a gateway to map a SIP URI to a QSIG number or vice versa. If it is not possible to derive an identifier that is essential for generating a signalling element relating to transfer, unless otherwise stated the call should be allowed to continue without that signalling element. In that case, the gateway should use a QSIG indication that the number is not provided for interoperability reasons.

The identity of the remote party must be derived as described in section 9.1 and 9.2 of [11].

Section 10 defines how to proceed in case of privacy or trust issues.

8.3 UAC Processing

8.3.1 Receipt of a FACILITY message with callTransferComplete invoke APDU

On receipt of a QSIG FACILITY message that contains a callTransferComplete invoke APDU (as a result of transfer by join in the QSIG network), the gateway SHALL send a SIP INVITE request that initiates a new dialog to replace the existing one. The identity information field in the INVITE request SHALL be derived from the redirectionNumber of the callTransferComplete invoke APDU. The INVITE request SHALL use [12] to indicate which dialog is replaced. The INVITE request SHOULD use [13] to indicate the identity of the Referrer and copy the URI of the PISN party of the replaced dialog into the Referred-By header field.

On receipt of an error final response to the INVITE request, no QSIG message is sent. There is no way for the gateway to notify the PINX that the transfer failed at that stage.

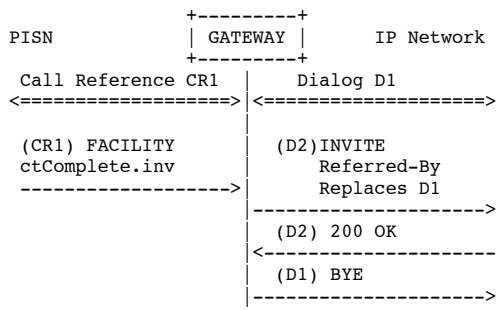


Figure 4: Example of message flows on receipt of a FACILITY

OPEN ISSUE. If the existing dialog is not confirmed (because the call is not answered yet), the INVITE request can be rejected because the Replaces header field does not match. Instead, the gateway MAY cancel the existing dialog and send a SIP INVITE request as above, except that it does not contain a Replaces header.

If the URI of the identity information field in the INVITE request cannot be derived from the QSIG number in the callTransferComplete invoke APDU, the gateway SHALL take no action.

NOTE. If a SIP mechanism that describes how to update identity information in an existing dialog is defined later, this document will be updated to take it into account.

8.3.2 Receipt of a FACILITY message with callTransferUpdate invoke APDU

On receipt of a QSIG FACILITY message that contains a callTransferUpdate invoke APDU (as a result of the identity of the remote party being updated in the QSIG network), the gateway SHALL look up the optional redirectionNumber. If the information updates the previous settings, the gateway SHALL act as if it received a callTransferComplete invoke APDU (cf. 8.3.1).

8.3.3 Receipt of a FACILITY message with ssctInitiate invoke APDU

On receipt of a QSIG FACILITY message that contains a ssctInitiate invoke APDU (as a result of a single step call transfer request in the QSIG network), the gateway SHALL send a SIP REFER request inside the current dialog as described in [9]. The URI in the Refer-To header field SHALL be derived from the rerouteingNumber of the ssctInitiate invoke APDU. The REFER request SHOULD use [13] to indicate the identity of the transferor and derive the URI in the Referred-By header field from the transferringAddress of the ssctInitiate invoke APDU.

On receipt of a SIP NOTIFY request indicating:
- a provisional alerting response, the gateway SHOULD send a QSIG FACILITY message that contains a ssctInitiate return result APDU if the awaitConnect boolean value in the invoke APDU is FALSE.
- a successful final response, the gateway SHALL send a FACILITY message with the ssctInitiate return result APDU if the awaitConnect boolean value in the invoke APDU is TRUE.
- an error final response, the gateway SHALL send a FACILITY message with a ssctInitiate return error APDU, unless a ssctInitiate return result has already been sent.

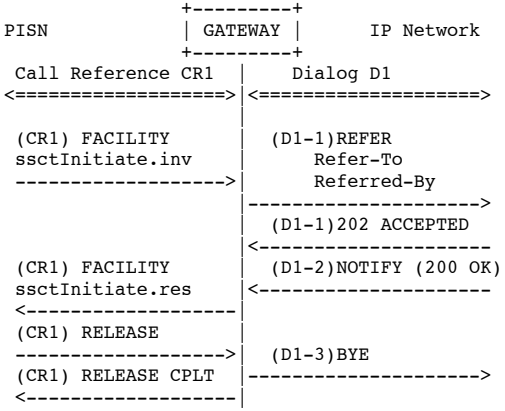


Figure 5: Example of message flows on receipt of a FACILITY message with ssctInitiate invoke APDU

If the URI in the Refer-To header field cannot be derived, the gateway SHALL send a FACILITY message that contains a ssctInitiate return error APDU.

8.3.4 Receipt of a SETUP message with ssctSetup invoke APDU

On receipt of a QSIG SETUP message that contains a ssctSetup invoke APDU (as a result of single step call transfer in the QSIG network), the gateway SHALL send a SIP INVITE that initiates a new dialog with the transfer target. The INVITE request SHALL be generated in accordance with [11]. In addition, it SHOULD use [13] to indicate the identity of the transferor and derive the URI in the Referred-By header field from the transferringAddress of the ssctSetup invoke APDU.

On receipt of responses to the INVITE request, the gateway SHALL act as described in [11].

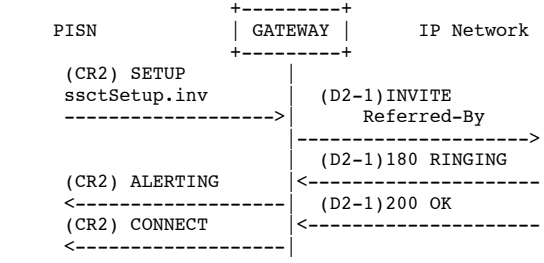


Figure 6: Example of message flows on receipt of a SETUP message with ssctSetup invoke APDU

8.3.5 Receipt of a FACILITY message with subaddressTransfer invoke APDU

On receipt of a QSIG FACILITY message that contains a subaddressTransfer invoke APDU (as a result of the remote party being in the public ISDN and having a subaddress), the gateway MAY derive a SIP or tel URI from the connectedSubaddress and act as if it received a callTransferComplete invoke APDU (cf. 8.3.1).

8.4 UAS Processing

8.4.1 Receipt of a SIP REFER request

On receipt of a SIP REFER request that is inside a dialog, the gateway can act in various ways depending on the contents of the Refer-To header field. Only Refer-To URIs with a method=INVITE parameter are in the scope of this document.

In this section, an embedded Replaces header is a Replaces header field as a parameter of a Refer-To URI

8.4.1.1 No embedded Replaces header field in the Refer-To URI.

On receipt of a SIP REFER request that is inside a dialog and that has no embedded Replaces header field in the Refer-To URI, the gateway SHALL accept the REFER request.

If the Refer-To URI can be derived, the gateway SHALL send a QSIG FACILITY message that contains a ssctInitiate invoke APDU. The gateway SHOULD derive the transferringAddress of the ssctInitiate invoke APDU from the Referred-By header field if present. The gateway SHALL derive the rerouteingNumber from the URI in the Refer-To header field and SHOULD derive the transferredAddress from the identity of the local PISN user in the existing dialog.

On receipt of a QSIG FACILITY message that comprises a ssctInitiate return result APDU, the gateway SHALL send a SIP NOTIFY request as described in [9]. The final status in the NOTIFY body is "200 OK".

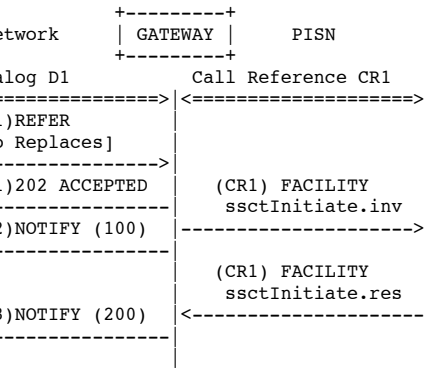


Figure 7: Example of message flows on receipt of a SIP REFER request (no Replaces header)

If the derivation of the rerouteingNumber fails, or upon reception of a QSIG FACILITY message that comprises a ssctInitiate return error APDU, the gateway SHALL send a SIP INVITE message as described in [9] and in [12]. The gateway SHALL then comply to [9] and in [12] and send SIP NOTIFY messages that indicate the result of the INVITE transaction.

On receipt of a "180 RINGING" provisional response to the INVITE request, the gateway SHOULD send a QSIG FACILITY message that contains a callTransferComplete invoke APDU with a callStatus of "alerting".

On receipt of a successful final response to the INVITE request, the gateway SHALL send a QSIG FACILITY message that contains :
- a callTransferActive invoke APDU, if a callTransferComplete invoke APDU has already been sent, or
- a callTransferComplete invoke APDU with a callStatus of "answered".

The gateway SHALL derive the redirectionNumber of the callTransferComplete invoke APDU and the connectedAddress of the callTransferActive invoke APDU from the identity information representing the called party transported in the responses to the INVITE request, or from the URI in the Refer-To header field if no

identity information is available. If the derivation fails, the gateway SHALL signal that numbers and addresses are not available due to interworking issues.

On receipt of an error final response to the INVITE request and if a callTransferComplete invoke APDU was invoked before, the gateway SHOULD send a FACILITY message that contains a callTransferActive invoke APDU. It indicates that the QSIG call between transferror and transferree is active again.

Figure 8: Example of message flows on receipt of a SIP REFER request (no Replaces header) when address derivation fails

8.4.1.2 Matching embedded Replaces header in the Refer-To URI.

On receipt of a SIP REFER request that is inside a dialog and that has an embedded Replaces header field in the Refer-To URI, the gateway SHALL compare the Refer-To URI with the local Contact URI of its internal list of SIP dialogs. If there is no match, the gateway SHALL act as described in section 8.4.1.3.

If there is a match, the gateway SHALL accept the REFER request and look up its internal list of SIP dialogs. If one of the dialogs matches the Replaces header field contents as described in [12], the gateway SHALL send a QSIG FACILITY message that contains a callTransferComplete invoke APDU with the QSIG call reference that maps to the SIP dialog of the REFER message. The gateway SHALL copy

the number of the PISN party of the matched dialog to the redirectionNumber of the callTransferComplete invoke APDU.

If the matching dialog is:
- not established then the callStatus of the callTransferComplete invoke APDU SHALL be "alerting", and the gateway SHALL send a SIP NOTIFY request with a body that indicates the latest provisional response code sent on the matching dialog,
- established then the callStatus of the callTransferComplete invoke APDU SHALL be "answered", and the gateway SHALL send a NOTIFY request with a body that indicates "200 OK".

The gateway SHALL also send a FACILITY message that contains a callTransferComplete invoke APDU with the QSIG call reference that maps the matching SIP dialog. The gateway SHALL derive the redirectionNumber of the callTransferComplete invoke APDU from the number of the PISN party in the dialog to which the REFER request belongs.

The gateway MAY then release the two SIP dialogs. The gateway SHALL then act as a transit PINX as defined in [3].

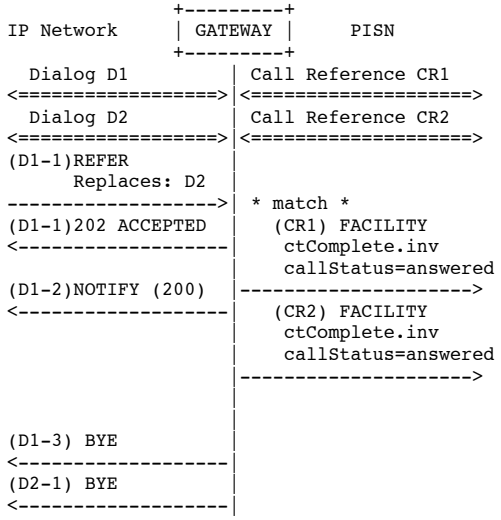


Figure 9: Example of message flows on receipt of a SIP REFER request (matching Replaces header field)

8.4.1.3 Non-matching embedded Replaces header in the Refer-To URI.

On receipt of a SIP REFER request that is inside a dialog and that has an embedded Replaces header in the Refer-To URI, the gateway SHALL compare the Refer-To URI with the local Contact URI of its internal list of SIP dialogs.

If there is a match but none of the internal list of dialogs matches the Replaces header field contents as described in [12], the gateway SHALL reject the REFER request with a SIP response code of "502 Bad Gateway".

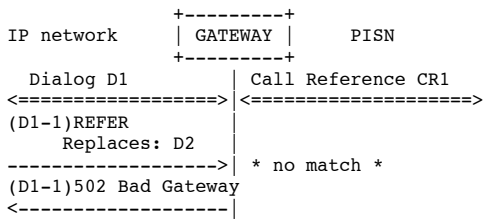


Figure 10: Example of message flows on receipt of a SIP REFER request (rejection on non-matching Replaces header field)

If the Refer-To URI does not match any local Contact URI, the gateway SHALL accept the REFER request and send a SIP INVITE message as described in [9] and in [12]. The gateway SHALL then comply to [9] and in [12] and send SIP NOTIFY messages that indicate the result of the INVITE transaction.

On receipt of a "180 RINGING" provisional response to the INVITE request, the gateway SHOULD send a QSIG FACILITY message that contains a callTransferComplete invoke APDU with a callStatus of "alerting". The FACILITY message SHALL be sent with the QSIG call reference that maps to the SIP dialog of the REFER message.

On receipt of a successful final response to the INVITE request, the gateway SHALL send a QSIG FACILITY message that contains :
- a callTransferActive invoke APDU, if a callTransferComplete invoke APDU has already been sent, or
- a callTransferComplete invoke APDU with a callStatus of "answered".
The FACILITY message SHALL be sent with the QSIG call reference that maps to the SIP dialog of the REFER message.

The gateway SHALL derive the redirectionNumber of the callTransferComplete invoke APDU and the connectedAddress of the

callTransferActive invoke APDU from identity information representing the SIP called party transported in the responses to the INVITE request or from the URI in the Refer-To header field if no identity information is available.

On receipt of an error final response to the INVITE request and if a callTransferComplete invoke APDU was invoked before, the gateway SHOULD send a FACILITY message that contains a callTransferActive invoke APDU. It indicates that the QSIG call between transferror and transferee is active again.

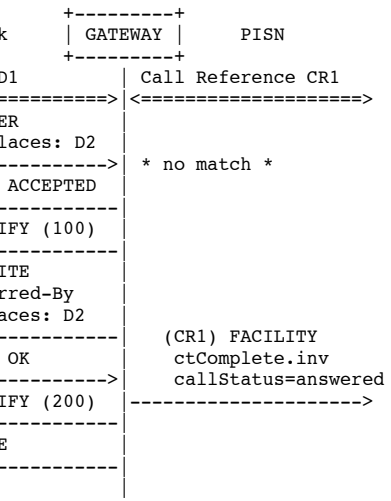


Figure 11: Example of message flows on receipt of a SIP REFER request (accept non-matching Replaces header field)

8.4.2 Receipt of a SIP INVITE request

8.4.2.1 Receipt of an INVITE with no Replaces header

On receipt of a SIP INVITE request that is outside a dialog and that has no Replaces header but has a Referred-by header, the gateway SHALL send an QSIG SETUP message as described in [11]. If the INVITE request comprises a Referred-By header field, the QSIG SETUP message SHOULD contain a ssctSetup invoke APDU with a transferringAddress derived from the Referred-By header described in [13].

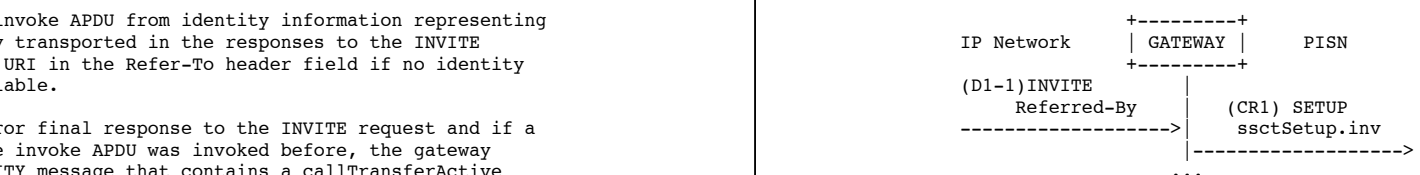


Figure 12: Example of message flows on receipt of a SIP INVITE request (no Replaces header)

8.4.2.2 Receipt of an INVITE with a Replaces header

On receipt of a SIP INVITE request that is outside a dialog and that has a matching Replaces header field [12], the gateway SHALL send an QSIG FACILITY message that contains a callTransferComplete invoke APDU. The FACILITY message is sent with the QSIG call reference that maps to the SIP dialog matched against the Replaces header field. The redirectionNumber in the callTransferComplete invoke APDU SHALL be derived from the SIP URI representing the calling party as described in section "9.2.2 Generating the QSIG Calling party number information element" of [11].

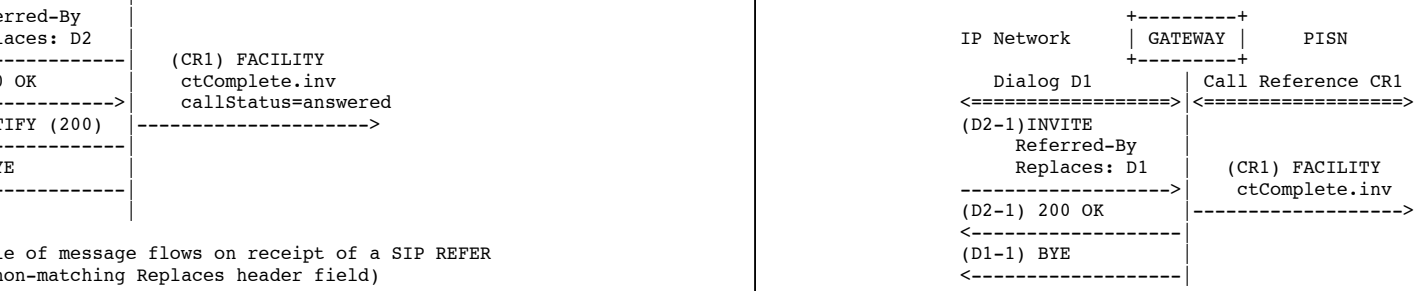


Figure 13: Example of message flows on receipt of a SIP INVITE request (matching Replaces header field)

If the Replaces header field does not match, the gateway SHALL reject the INVITE request as described in [12]. Note that this may happen if two different gateways can be used to reach the QSIG side.

9. Example message sequences

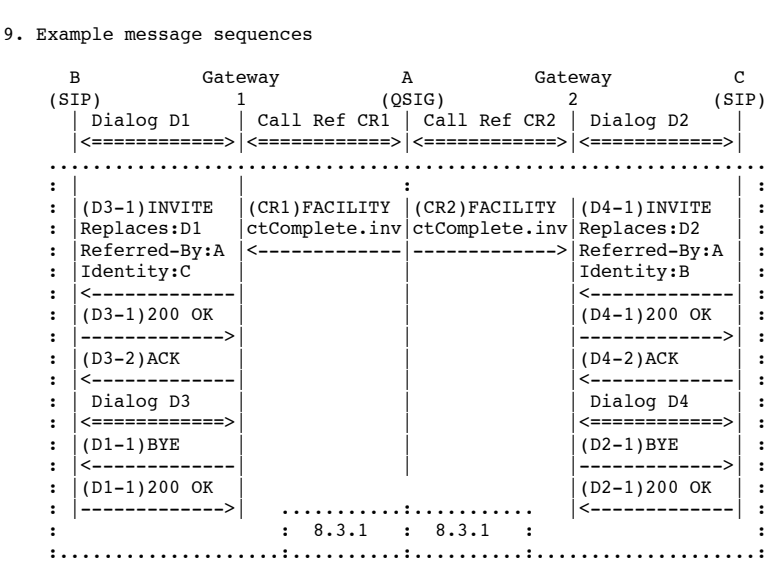


Figure 14: Example of Call transfer by Join where User B and User C are SIP participants

NOTE. If Gateway 1 and Gateway 2 of Figure 14 are the same gateway, some implementation dependant optimization mechanisms using a SIP REFER request may take place.

Figure 15: Example of Call transfer by join where User A is a SIP participant and where one gateway is used.

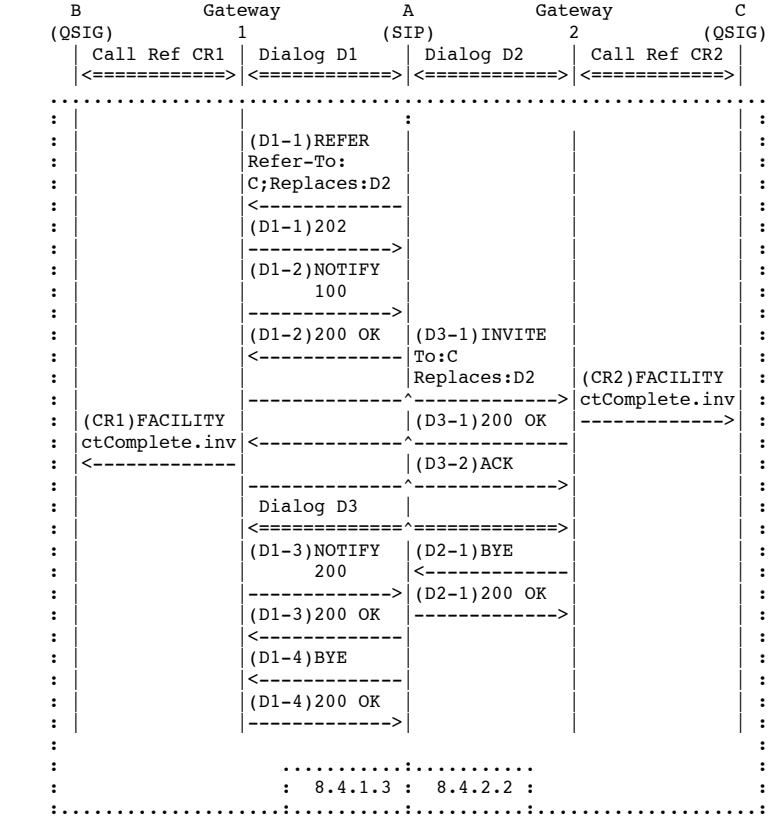


Figure 16: Example of Call transfer by join where User A is a SIP participant and where two gateways are used.

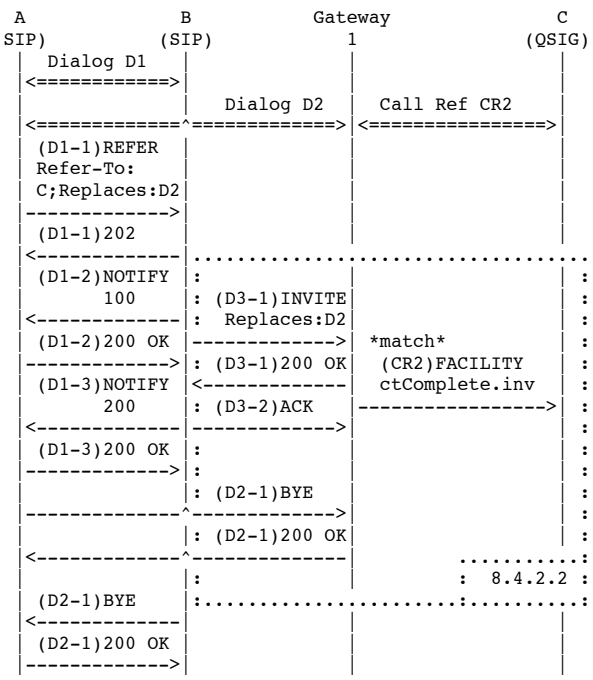


Figure 17: Example of Call transfer by join where User A and User B are SIP participants.

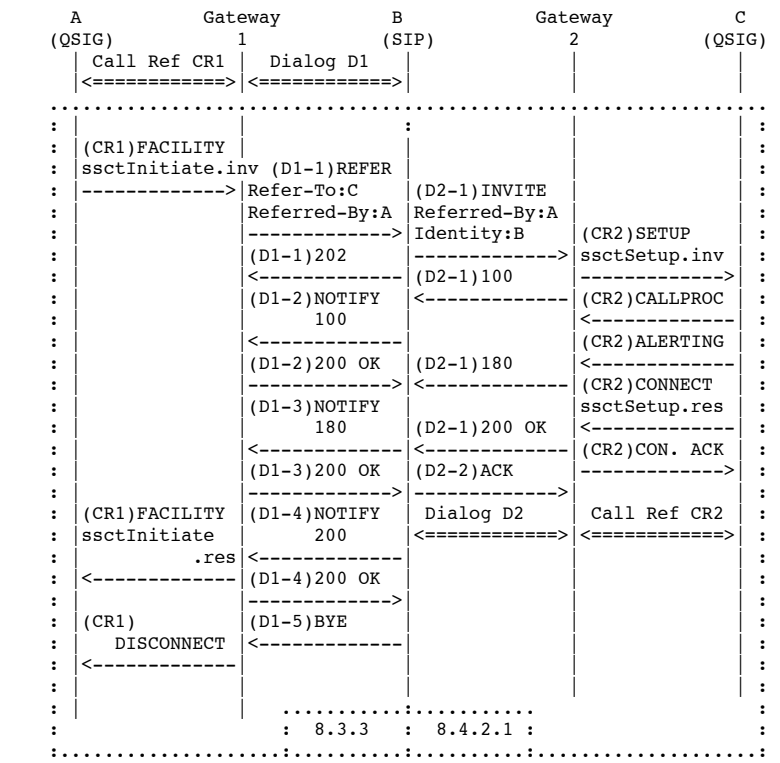


Figure 18: Example of a Single-Step Call transfer where User B is a SIP participant.

Figure 19: Example of an unsuccessful Single-Step Call transfer where User A and User C are SIP participants.

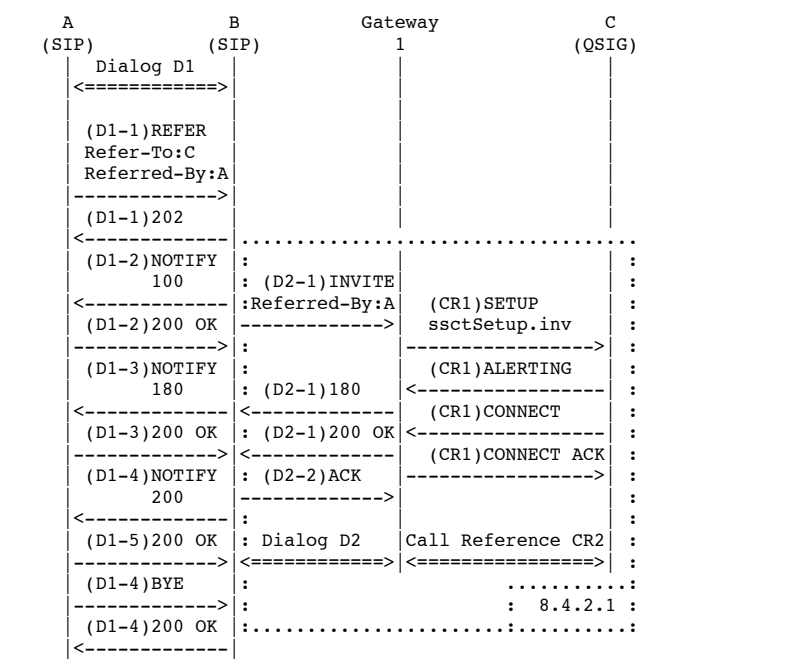


Figure 20: Example of a Single-Step Call transfer where User A and User B are SIP participants.

10. Security Considerations

The security considerations related to the SIP mechanisms used in this document apply.

The security considerations related to the derivation of address and identity between SIP and QSIG described in [11] apply.

The security considerations due to the transfer interworking functions between QSIG and SIP are related to the derivation of the identity of User A, User B and User C.

If identity information issued by the IP network is not trusted, the gateway SHOULD not derive the transferredAddress, transferringAddress

and redirectionNumber of the QSIG invoke APDUs from the SIP header fields. It SHOULD indicate that the numbers are not available due to interworking.

If identity information issued by the IP network is not disclosed for privacy reasons, the gateway SHALL not derive the transferredAddress, transferringAddress and redirectionNumber of the QSIG invoke APDUs from the SIP header fields, except if the trust model applies. It SHALL indicate that the numbers are not available due to screening. If the trust model applies and the Privacy header defined in [14] is used, the gateway MAY derive the identity information and SHALL indicate that the presentation is restricted.

If identity information issued by the PISN is not to be disclosed because of local policy or PISN privacy requests, the gateway SHALL not derive SIP header field URIs from the transferredAddress, transferringAddress and redirectionNumber of the QSIG invoke APDUs, except if the trust model of applies. If the trust model applies, the information MAY be disclosed in conjunction with the Privacy header defined in [14].

Normative References

[1] J. Rosenberg, H. Schulzrinne, et al., "SIP: Session initiation protocol", RFC 3261, June 2002.

[2] International Standard ISO/IEC 11572 "Private Integrated Services - Circuit-mode Bearer Services - Inter-Exchange Signalling Procedures and Protocol" (also published by ECMA as Standard ECMA-143).

[3] International Standard ISO/IEC 11582 "Private Integrated Services Network - Generic Functional Protocol for the Support of Supplementary Services - Inter-Exchange Signalling Procedures and Protocol" (also published by ECMA as Standard ECMA-165).

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[5] International Standard ISO/IEC 13865 "QSIG - Specification, functional model and information flows - Call transfer supplementary services" (also published by ECMA as Standard ECMA-177)

[6] International Standard ISO/IEC 13869 "QSIG - Inter-Exchange Signalling Protocol for the Support of Call Transfer Supplementary Services" (also published by ECMA as Standard ECMA-178)

[7] International Standard ISO/IEC 19459 "QSIG - Specification, Functional Model and Information Flows - Single Step Call Transfer Supplementary Service" (also published by ECMA as Standard ECMA-299)

[8] International Standard ISO/IEC 19460 "QSIG - Inter-Exchange Signalling Protocol - Single Step Call Transfer Supplementary Service" (also published by ECMA as Standard ECMA-300)

[9] R. Sparks, "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[10] J. Rosenberg, "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, September 2002.

[11] F. Derks, J. Elwell, P. Mourot, O. Rousseau, "Interworking between SIP and QSIG", draft-ietf-sipping-qsig2sip-04 (work in progress)

[12] R. Mahy, B. Biggs, R. Dean, " The Session Inititation Protocol (SIP) "Replaces" Header", draft-ietf-sip-replaces-04 (work in progress)

[13] R. Sparks, " The SIP Referred-By Mechanism", draft-ietf-sip-referredby-03 (work in progress)

[14] J. Peterson, "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323

[15] C. Jennings, J. Peterson, M. Watson, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", RFC 3325

[16] J. Rosenberg, "The Session Initiation Protocol UPDATE Method", RFC 3311.

Informative References

[17] R. Sparks, A. Johnston, "Session Initiation Protocol Call Control - Transfer", draft-ietf-sipping-cc-transfer-01 (work in progress)

[18] ECMA Technical Report ECMA-TR/86, "Corporate Telecommunication Networks - User Identification in a SIP/QSIG Environment"

Authors' Addresses

Jean-Francois Rey
Alcatel Business Systems
8, rue de Kervezennec BP 82 802
29228 Brest cedex 2
France
Email: jean-francois.rey@alcatel.fr

Olivier Rousseau
Alcatel Business Systems
32, Avenue Kleber
92700 Colombes
France
email: olivier.rousseau@col.bsf.alcatel.fr

John Elwell
Siemens Communications
Technology Drive
Beeston
Nottingham, UK, NG9 1LA
email: john.elwell@siemens.com

Intellectual Property Statement

Full Copyright Statement

Network Working Group
Internet-Draft
Expires: August 8, 2004

H. Schulzrinne
Columbia U.
B. Rosen
Marconi

February 8, 2004

Emergency Services for Internet Telephony Systems
draft-schulzrinne-sipping-emergency-arch

Status of this Memo

Copyright Notice

Abstract

Summoning emergency help is a core feature of telephone networks. This document describes how the Session Initiation Protocol (SIP) can be used to provide advanced emergency services for voice-over-IP (VoIP). The architecture employs standard SIP features and requires no new protocol mechanisms. DNS is used to map civil and geospatial locations to the appropriate emergency call center.

Table of Contents

1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Overview

Summoning police, the fire department or an ambulance in emergencies is one of the fundamental and most-valued functions of the telephone. As telephone functionality moves from circuit-switched telephony to Internet telephony, its users rightfully expect that this core functionality works at least as well as for the older technology. However, many of the technical advantages of Internet telephony require re-thinking of the traditional emergency calling architecture. This challenge also offers an opportunity to improve the working of emergency calling technology, while potentially lowering its cost and complexity.

It is beyond the scope of this document to enumerate and discuss all the differences between traditional (PSTN) and Internet telephony, but the core differences can be summarized as separation of signaling and media data, the emergence of application-independent carriers, and the potential mobility of all end systems, including landline systems and not just those using radio access technology.

This document focuses on how ECCs can natively handle Internet telephony emergency calls, rather than Describing how circuit-switched ECCs can handle VoIP calls. However, in many cases, ECCs making the transition from circuit-switched interfaces to packet-switched interfaces may be able to use some of the mechanisms described here, in combination with gateways that translate packet-switched calls into legacy interfaces, e.g., to continue to be able to use existing call taker equipment.

Existing emergency call systems are organized nationally; there are currently no international standards. However, Internet telephony does not respect national boundaries, and thus an international standard is required.

Furthermore, VoIP endpoints can be connected through tunneling mechanisms such as virtual private networks (VPNs). This significantly complicates emergency calling, because the location of the caller and the first element that routes emergency calls can be on different continents, with different conventions and processes for handling of emergency calls. The IETF has historically refused to create national variants of its standards. Thus, this document attempts to take into account best practices that have evolved for circuit switched ECCs, but makes no assumptions on particular operating practices currently in use, numbering schemes or organizational structures.

This document assumes that ECC interface is using the Session Initation Protocol (SIP). Use of a single protocol greatly

simplifies the design and operation of the emergency calling infrastructure. Only peer-to-peer protocols such as H.323, ISUP and SIP are suitable for inter-domain communications, ruling out master-slave protocols such as MGCP or H.248/Megaco. The latter protocols can natually be used by the enterprise or carrier placing the call, but any such call would reach the ECC through a media gateway controller, similar to how interdomain VoIP calls would be placed. Other signaling protocols may also use protocol translation to communicate with a SIP-enabled ECC.

Existing emergency services rely exclusively on voice, and conventional TDD text media streams. However, more choices of media offer additional ways to communicate, evaluate and assist callers and call takers to handle emergency calls. For example, instant messaging and video could improve the ability to evaluate the situation and provide appropriate instruction prior to arrival of emergency crews. Thus, the architecture described here supports the creation of sessions of any media type, negotiated between the caller and ECC using existing SIP protocol mechanisms [RFC3264].

While traditionally, emergency services have been summoned by voice calls only, this document does not rule out the use of additional media during an emergency call, both to support callers with disabilities (e.g., through interactive text or video communications) and to provide additional information to the call taker and caller. For example, video from the caller to the ECC may allow the call taker to better assess the emergency situation; a video session from the ECC to the emergency caller may allow the call taker to provide instructions for first aid.

The choice of media and encodings is negotiated on a call-by-call basis using standard SIP mechanisms [RFC3265]. To ensure that at least one common means of communications, this document recommends certain minimal capabilities in Section Section 10 that call taker user agents and ECC-operated proxies should possess.

This document does not prescribe the detailed network architecture for ECCs or collection of ECCs. For example, it does not describe where ECCs may place firewalls or how many SIP proxies they should use.

This document does not introduce any new SIP header fields, request methods, status codes, message bodies, or events. User agents unaware of the recommendations in this draft can place emergency calls, but may not be able to provide the same user interface functionality. The document suggests behavior for proxy servers, in particular outbound proxy servers.

3. Terminology

(emergency) call taker Person that answers an emergency call; typically located in an emergency call center.

ECC (emergency call center) Call center that receives emergency calls and dispatches polic, fire and rescue services. An ECC serves a limited geographic area. In the United States, PSAPs are ECCs.

ESRP (emergency service routing proxy) SIP proxy that routes incoming emergency calls to the appropriate ECC.

PSAP (public safety answering point) The United States and Canadian term for ECC.

SIP proxy see [RFC3261]

SIP UA (user agent) see [RFC3261]

4. Identifying an Emergency Call

Using the PSTN, emergency help can often be summoned at a designated, widely known number, regardless of where the telephone was purchased. However, this number differs between localities, even though it is often the same for a country or region (such as many countries in the European Union). For end systems based on the Session Initiation Protocol (SIP), it is desirable to have a universal identifier, independent of location, to simplify the user experience, allow the automated inclusion of location information and to allow the device and other entities in the call path to perform appropriate processing.

As part of the overall emergency calling architecture, we define a common user identifier, "sos" and "sos" with an emergency service designation, as the contact mechanism for emergency assistance. In addition, two tel URIs, tel:112 and tel:911, are permissible as emergency call identifiers issued by a user agent. We refer to this URI as the "emergency calling URI". The calling user agent sets both the "To" header and the request-URI to the emergency URI, so that entities after the ESRP can still readily determine that this is an emergency call. Details are described in [draft-schulzrinne-sipping-sos].

In addition, user agents SHOULD detect emergency calls following local emergency calling conventions. There are two local conventions, namely those local to the user's SIP domain, e.g., a user's network at work, and those at the caller's current geographic location, e.g., while traveling. The former can be obtained using SIP configuration mechanisms (Section Section 9). Obtaining geographically local emergency numbers is more difficult, particularly if the outbound proxy or DHCP server may be in a different country than the caller. There are several, complementary solutions. First, DHCP can be extended with a pointer to a local SIP configuration, including the dial plan specifying emergency numbers. In addition, we define a new DNS resource record that identifies the country-specific emergency number.

Location information can be provided by the user agent or a proxy. If the user agent provides this information, the user agent needs to be able to determine that a call is indeed an emergency call as it is unlikely to include location information in each call.

5. Location and Its Role in an Emergency Call

5.1 Introduction

Caller location plays a central role in routing emergency calls. For practical reasons, each ECC generally handles only calls for a certain geographic area. Other calls that reach it by accident must be manually re-routed (transferred) to the appropriate ECC, increasing call handling delay and the chance for errors. The area covered by each ECC differs by jurisdiction, where some countries have only a small number of ECCs, while others devolve ECC responsibilities down to the community level.

In most cases, ECCs cover at least a city or town, but there are some areas where ECC coverage areas follow old telephone rate center boundaries and may straddle more than one city.

5.2 Types of Location Information

There are four primary types of location information: civil, postal, geospatial, and cellular cell tower and sector.

Civil: Civil information describes the location of a person or object by a floor and street address that corresponds to a building or other structure.

Postal: Postal addresses are similar to civil addresses, but the may contain post office boxes or street addresses that do not correspond to an actual building. Postal addresses are generally unsuitable for emergency call routing, but may be the only address available to a service provider, derived from billing records.

Geospatial: Geospatial addresses contain longitude, latitude and altitude information.

Cell tower/sector: Cell tower and sectors identify the cell tower and the antenna sector that the mobile device is currently using. (Cell/sector information could also be transmitted as an irregularly shaped polygon of geospatial coordinates reflecting the likely geospatial location of the mobile device, but since these boundaries are not sharp, transmitting the raw information is probaby preferable.)

5.3 Sources of Location Information

There are two principal contributors of location information. In some cases, the calling end system knows its current civil and/or

geospatial location, in others the outbound proxy or other network entity. In some cases, both such entities may have location information, possibly partially contradictory. This document provides no recommendation on how to reconcile conflicting location information or which one is to be used by routing elements. Conflicting location information is particularly harmful if it points to multiple distinct ECCs. If there is no other basis for choice, the ESRP SHOULD determine the appropriate ECC for all location objects and, if there is a conflict, route based on the most accurate one.

To facilitate such policy decisions, location information SHOULD contain information about the source of data, such as GPS, manually entered or based on subscriber address information. In addition, the author of the location information SHOULD be included. TBD: need to add this to (e.g.) PIDF-LO!

End systems and network elements can derive location information from a variety of sources. It is not the goal of this document to exhaustively enumerate them, but we provide a few common examples below:

GPS: Global Positioning System (GPS) information is generally only available where there is a clear view of a large swath of the sky. It is accurate to tens of feet.

Wireless triangulation: Either cell towers or 802.11 access points triangulate based on signal strength or time of arrival.

DHCP: The device obtains location information provided by its DHCP server, derived through one of the other methods enumerated here.

Location beacons: A short range wireless beacon, e.g., using BlueTooth or infrared, announces its location to mobile devices in the vicinity.

Subscriber information: A carrier has address information reflecting the service location for its subscribers.

Manual configuration: A user manually enters civil or geospatial information into a mobile or stationary device.

TBD: there is a need to indicate which location information has been used to avoid possible routing loops, where two proxies pick different location information from the call request, each pointing to the other one.

5.4 Using Location Information for Call Routing

Note that emergency calls may not be routed to the geographically closest ECC, but rather to the most jurisdictionally appropriate one, which may well be further away.

Location information may not be available at call setup time. For example, if a GPS-enabled cell phone is turned on and then immediately places an emergency call, it can take an additional 20-25 seconds before the cell phone acquires a GPS fix and its location. Thus, while it is necessary and expedient to include caller location information in the call setup message, this is not sufficient in all circumstances. In some cases, the initial call setup will proceed based on, for example, cell and sector information and then add location information during the call, rather than delaying the initial call setup by an unacceptable amount of time.

In addition, the location of a mobile caller, e.g., in a vehicle or aircraft, can change significantly during the emergency call.

6. Routing the Call to the ECC

6.1 Routing the First Request

Emergency calls are routed based on location information contained within the call setup request (INVITE). If there is no or imprecise (e.g., cell tower/sector) information, an on-going emergency call may also be transferred to another ECC based on location information.

Each proxy receiving an emergency call request, identified as described in Section Section 4, attempts to route the call to the most appropriate ECC. Similarly, a user agent can also directly route emergency calls if it has location information, either obtained locally or from a redirect response provided by the outbound proxy. There are three types of routing actions: default routing, DNS-based routing and local routing.

ESRPs and user agents using default routing forward all emergency call requests to one designated ESRP, regardless of the location of the caller.

ESRPs and user agents using DNS-based routing employ the mechanism in [-dns-] to route calls to another ESRP that is qualified to handle the emergency call. Thus, DNS acts here as a location service for the proxy.

Finally, an ESRP MAY use a local database to perform location-based call routing. The details of such a database are beyond the scope of this document.

If an emergency call INVITE request does not contain location information and no other location hints (such as subscriber identity) are available, the first ESRP in the call path SHOULD route it to an ECC that is geographically local to that proxy, since no other call routing can be performed.

Jurisdictions organizing ECCs may choose to implement multiple levels of routing based on location. For example, a state or province might deploy an ESRP in front of a collection of ECCs. The information available to a VoIP carrier or enterprise ESRP may be coarse, so that any location within the state or province gets routed to the state/province ESRP, with that ESRP doing the detailed routing to a specific ECC. Thus, each ESRP MUST inspect the INVITE request and determine if more precise request routing is called for.

6.2 Updating Location Information

Location information is needed both for routing the initial INVITE

message in a call as well as possibly later during a call since location information may change or only become available later, after the call has reached an ECC.

This document considers three mechanisms for conveying updated location during a call: UPDATE or re-INVITE, INFO or dialog events.

In the first approach, the caller sends UDPATE [RFC3311], prior to completion of the initial INVITE transaction, or re-INVITE requests to the destination. Care must be taken that these requests are routed to the same destination as the original call-initiating request. This is unlikely to be a problem for a re-INVITE if the Contact header field in the 200 OK indicates the ECC address.

In the second approach, the ECC subscribes to the location of the caller, using the SIP event mechanism and PIDF-LO [I-D.ietf-geopriv-pidf-lo]. This approach has the advantage that authorized third parties can easily get access to call-related location information. This also works better if a network entity has location information, rather than the user agent. With the re-INVITE approach, the user agent would have to obtain this information via redirection.

A third alternative is to use the SIP INFO method, as the location update does not require an offer-answer exchange and does not change call state.

7. Preventing Call Misdirection

We need to prevent that an emergency call reaches a destination other than an ECC. For example, a rogue UA able to intercept SIP requests might be able to impersonate an ECC.

In the absence of a globally recognized certificate that ensures that the owner is a legitimate PSAP, we rely on a chain of trust enforced by the 'sips' URI schema. The 'sips' URI schema forces each SIP hop to route the call only to destinations supporting TLS transport. Each ESRP MUST verify that the next-hop destination chosen as described in Section Section 6 corresponds to the server certificate offered by that destination.

8. Requirements for SIP Proxy Servers

All ESRP SHOULD support RFC 3261 [RFC3261] with UDP, TCP, TLS transports.

For robustness, ESRPs SHOULD NOT use RFC 1918 [RFC1918] addresses, i.e., should not be behind network address translators.

9. Configuration

SIP devices do not require any additional configuration to place emergency calls. They SHOULD use the local outbound proxy, discovered via [RFC3361] or [RFC3319].

However, to acquire local dial plan numbers, the SIP configuration framework [I-D.ietf-sipping-config-framework] can be used. The format for dial plans remains to be defined. A device may retrieve dial plan information for emergency calls from two locations, namely the user's home domain and the local outbound proxy, as described in Section 3.13 of [I-D.ietf-sipping-config-framework].

Since a traveling user cannot rely on a DHCP server in the visited location to have accurate local emergency number information, we also propose a new DNS resource record, EN. Typically, this resource record will be associated with a country-level 'sos.arpa' zone, as most countries either have or are developing country-wide emergency numbers. These number strings are treated as dial strings, not "tel" URIs. TBD: It might be possible to use NAPTR [RFC2915] records to include translations such that 110 becomes sos.police for de.sos.arpa. NAPTR translations are not limited to hostnames or URIs.

In the example below, the German emergency number for police is translated into an 'sos' URI. This only works if there is a designated SIP proxy that can route all emergency calls originating in Germany. There does not appear to be a way to substitute the caller's current home AOR domain, although one could conceivably adopt a convention for including this information. Note that this mechanism would also allow direct routing based on finer-grained location information, e.g., at the city level.

```
de.sos.arpa.
;;       order pre flags service        regexp     replacement
IN NAPTR 100 10 "u" "SOS" "/110/sips:sos.police@notfall.de/i" .

bonn.nrw.de.sos.arpa.
;;       order pre flags service        regexp      replacement
IN NAPTR 100 10 "u" "SOS" "/110/sips:sos.police@pol.bonn.de/i" .
```

Example NAPTR records to map dial strings to 'sos' URIs

Figure 1

10. Requirements for SIP User Agents

10.1 Emergency call taker

To increase the likelihood that diverse user equipment can successfully communicate with the ECC, it is recommended that call taker equipment has the following minimal capabilites:

signaling: RFC 3261, with UDP, TCP and TLS (sips) support RFC 3262

audio: RTP and RTCP according to ..., G.711, GSM 06.10, FEC, SRTP

Interactive text

SIP-based instant messaging

10.2 Calling users

A user agent placing an emergency call SHOULD use the "sips" URI schema for all such calls, forcing these calls to use TLS as secure hop-by-hop transport. If a call cannot be established using TLS transport, the user agent SHOULD attempt a call using the "sip" URI.

If a user agent receives a redirect (3xx) response for an emergency call, it MUST include the location information contained in that response in the outgoing call. This differs from regular behavior for redirects, where the message body is not copied into the new call.

A user agent MUST check the Contact URI in redirect responses to see if it is an emergency call, as described in Section X. If so, the behavior in the previous paragraph applies.

End systems that allow human users to initiate an emergency call with a single button press or other similar stimulus SHOULD require callers to confirm their call.

11. Example Call Flows

TBD

12. Security Considerations

12.1 ECC Impersonation

See Section Section 7.

With DNS-based call routing (Section Section 6), an attacker could modify the DNS entries for one or more ECCs, re-routing calls destined for them. Thus, the use of secure DNS is RECOMMENDED.

12.2 Call Content Integrity

Normative References

[I-D.ietf-geopriv-pidf-lo]
Peterson, J., "A Presence-based GEOPRIV Location Object Format", draft-ietf-geopriv-pidf-lo-01 (work in progress), February 2004.

[I-D.ietf-sipping-config-framework]
Petrie, D., "A Framework for SIP User Agent Configuration", draft-ietf-sipping-config-framework-01 (work in progress), October 2003.

[I-D.schulzrinne-geopriv-dhcp-civil]
Schulzrinne, H., "DHCP Option for Civil Location", draft-schulzrinne-geopriv-dhcp-civil-01 (work in progress), February 2003.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2915] Mealling, M. and R. Daniel, "The Naming Authority Pointer (NAPTR) DNS Resource Record", RFC 2915, September 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3265] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[RFC3311] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

[RFC3319] Schulzrinne, H. and B. Volz, "Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers", RFC 3319, July 2003.

[RFC3361] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers", RFC 3361, August 2002.

Informative References

[RFC1918] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G. and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

Authors' Addresses

Henning Schulzrinne
Columbia University
Department of Computer Science
450 Computer Science Building
New York, NY 10027
US

Phone: +1 212 939 7042
EMail: hgs@cs.columbia.edu
URI: http://www.cs.columbia.edu

Brian Rosen
Marconi
2000 Marconi Drive
Warrendale, PA 15086
US

EMail: brian.rosen@marconi.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

Session Initiation Protocol (SIP)
Internet Draft
Document: draft-stucker-sip-guid-00.txt
Expires: July 2004

B. Stucker
Nortel Networks

February 2004

Client Globally Unique ID for SIP

Status of this Memo

Abstract

A number of applications using the Session Initiation Protocol (SIP) protocol require or can be enhanced by being able to uniquely identify a particular user agent (UA) instance in the network. This document describes an extension to SIP that allows clients to generate globally unique identifiers (GUID) for use within SIP based applications, providing an example of their use.

Table of Contents

1. Introduction

Within SIP, there arise situations where it is necessary to ensure that an action is applied to a particular user agent (UA) instance, but the existing mechanisms within SIP are not always reliable. For example, although registrations identify a routable address and port of a particular UA, in an environment that uses dynamically assigned IP addresses (NATs, VPNs, short-lease DHCP networks) there is no ready way of always tying registrations together across time for a particular UA instance. In these environments, the usual IP/port combination that defines a particular routing location of a UA is unreliable over time as an identifier of that UA.

As a result, an identifier that UAs can use as a "finger-printing" mechanism to identify themselves is useful. Whereas the Globally Routable UA URIs (GRUU) draft [4] seeks to address a server-generated identifier for the UA, this draft seeks to define a client-generated approach to a similar problem.

The mechanism defined in this document allows a particular UA instance to construct a globally unique identifier which can be used by SIP services to process requests that require, or are enhanced by, the ability to identify a particular UA instance in the network over a long period of time.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [ii].

3. Creating a GUID

This section covers the details of creating a GUID on the client UA.

3.1 Characteristics of a GUID

The idea of a globally unique ID is hardly a new concept. Designers and developers of all sorts of applications in the physical world and the Internet have required the ability to uniquely identify a particular entity from a larger set. This is especially true when every other property of the entity is subject to substantial changes over time that would render it difficult or impossible to uniquely identify over time.

For example, governments frequently assign us a number (or other identifying string) when we are born because they have a need to identify us as taxpayers throughout our lives. There are several other examples of unique IDs, such as vehicle identification numbers and serial numbers on items we buy from large manufacturers.

A common characteristic of these identification numbers is that they have two basic properties:

- They are unique to the entity they are associated with.
- A central authority coordinates the assignment of IDs to ensure that no two entities are given the same identifier.

Note, that there is no requirement that there be any sort of registry that knows which entity has what identifier. This would be needed if the identifier were to be used for non-repudiation purposes, but that is not always a goal that needs to be fulfilled depending on the application.

Sometimes entities need to be able to be identified uniquely, but to have a central authority assign an identifier would be difficult or impossible. In these situations, it is still possible for the entity to assign itself a unique identifier. This can be achieved by using a mechanism that ensures that the odds of any two entities having the same identifier are statistically insignificant.

An example of this mechanism would be human fingerprints. Fingerprints can be used as a globally unique identifier of who you

are, and the odds of two people having the same fingerprints are statistically insignificant (even twins have a different set). No central authority coordinates the assignment of who gets what fingerprints, and yet they can be used to uniquely identify a particular person. If they are registered with a central authority, they can be used for purposes of non-repudiation. In either case, they are very useful, as other characteristics of people may change wildly over time.

3.2 Construction of a SIP GUID

Constructing an identifier that describes a UA is trivialquite straightforward. SIP TAGs are frequently generated to identify a particular UA session within SIP. Ensuring that the identifier is unique within a small, controlled set of UAs is more difficult, but still manageable by simply assigning them directly to the UA upon creation (like assigning a static IP address to a machine on a LAN). However, making that identifier unique across very large sets could be very difficult by simple assignment through sheer logistics (think about your experiences trying to get a driver's license).

Because a straightforward assignment of a GUID is problematic at best (and impossible at worst) this approach is ruled out in favor of using a standard mechanism: use time and space to your advantage. All SIP GUIDs MUST be generated based off the time that they were generated, and the "space" in which they were generated.

Obviously, generating a SIP GUID that is composed of a three-digit number would not satisfy most reasonable definitions of "unique" within a SIP network. Therefore, SIP GUIDs MUST be at least 128-bits in length.

3.2.1 Time Component

Time can be used to create uniqueness because each instant in time only occurs once. This can be used to constrain the set of all UAs that wish to create a GUID at that instant from the set of all UAs that will ever exist (ie. all of the UAs that wish to create a GUID on February 6th, 2004 at 10:45pm as opposed to all UAs that will ever exist from now to eternity). This means that a component of a GUID should be based on the current local time. It is not necessary that every UA generating a GUID need to have synchronized clocks with every other UA. This is because we're not interested in being able to tell the exact moment a GUID is created. It's used simply as a component of the GUID in order to constrain the larger set.

Many computers and development platforms vary in the scale at which time can be measured. Because we are using time to constrain the set of all UAs that may ever wish to generate a GUID, it is important

that the smallest available unit be used by the UA generating the GUID. Additionally, a large random number from a cryptographically-strong random number generator can be appended to the current time to create a pseudo-timestamp with very fine resolution.

Here's an example:

- A computer's clock can be resolved down to 1 millisecond.
- The computer's random number generator can produce a random integer up to (2^31)-1.
- From this a "pseudo-clock" can then be constructed that resolves time down to the order of a pico-second (10^-12 seconds, or trillionths of a second).

Friday, February 6th, 2004 at 21:30:54 CST can be expressed as 1076124654957 milliseconds since January 1, 1970, 00:00:00 GMT.

A possible random number generated by a cryptographically-strong random number generator: 190182543.

Taken together, it is possible to create a "pseudo-time" of 1076124654957190182543 pico-seconds since January 1, 1970 00:00:00 GMT.

This is a very powerful notion, and if further resolution is required, successive random numbers can be appended to further resolve the "pseudo-clock" to fantastically small instants of time. It is critical, however, that an actual clock source be used as the most-significant digits of the "pseudo-clock".

In the example given, even if 1 billion SIP UAs decided to generate a new GUID at the same time, it is still a 1 in a trillion chance that they come up with the same "pseudo-clock" time.

SIP GUIDs MUST use a "psuedo-clock" that resolves to a minimum of 10^-12 seconds.

3.2.2 Space Component

The other component to a well-formed globally unique identifier that is not assigned by a central authority is to use space (or an approximation of it) as a component. It can obviously be the same time in multiple places, but no two UAs can ultimately occupy the same position in "space".

Because we are dealing with the electronic world, the notion of space is used somewhat conceptually; depending on the application, what

constitutes "space" may vary. The MAC address of the device that the UA instance runs upon would be a good way to denote its position in space, where space is given as the network. However, there are security implications involved with handing out a MAC address at the application level. For one, it can be used to discover the manufacturer of the device, which may help an attacker determine a method of attack.

Therefore, MAC addresses SHOULD NOT be used as an identifier of space for the purposes of a SIP GUID.

Additionally, there may be multiple UA instances executing on the same CPU. For this reason, it is RECOMMENDED that the space component of a SIP GUID be a location in memory that is uniquely held by that UA instance, as well as the IP address of the UA. Taken together with the time component, this still provides a high level of uniqueness within the network. It is extremely unlikely that two UA instances would be stored in the same location in memory, on two computers with the exact same IP address, at the exact same "pseudo-clock" time.

SIP GUIDs MUST contain a space component that provides no fewer than 64 bits of uniqueness.

3.3 Comparing SIP GUIDs

When comparing two SIP GUIDs, their values SHOULD be considered a unique identifier for the UA instance associated with the party that sent the SIP request, including any aliases of the user or entity identified by the sending party.

3.4 The GUID Header

The GUID header identifies a UA GUID. This header denotes the GUID for that UA instance. The GUID header MUST NOT appear in a SIP response, and if present MUST be ignored by the recipient. The GUID header MAY appear in any SIP request type. It is RECOMMENDED that user agents include their GUID in any REGISTER request sent.

3.4.1 Syntax

This document adds the following entry to Table 2 of [1]. Additions to this table are also provided for extension methods defined at the time of publication of this document. This is provided as a courtesy to the reader and is not normative in any way. MESSAGE, SUBSCRIBE and NOTIFY, REFER, INFO, UPDATE, PRACK, and PUBLISH are defined respectively in [6], [7], [5], [8], [9], [10], and [11].

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	MSG
GUID	R		o	o	o	o	o	o	o

Header field	where	SUB	NOT	REF	INF	UPD	PRA	PUB
GUID	R	o	o	o	o	o	o	o

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC-2234 [3].

```
GUID        = "GUID" HCOLON token
```

A SIP request MUST contain no more than one GUID.

Examples:

```
GUID: f7ca930e2412f1bf016eb4940441672d3c26b17

GUID: 10761246549571901825 43+47bfc83e+10.33.15.8
```

3.5 Proxy Behavior

Proxies MUST NOT modify the contents of the GUID header during processing. It MAY be stripped according to the privacy policies of the system should header privacy have been requested by the UA sending the request in accordance with RFC-3323.

4. Security Considerations

The extension defined in this document may impact the security of a particular SIP application. Depending on the use of the GUID in a given application, considerations may need to be made to use a secure transport mechanism such as TLS for sending SIP requests containing a GUID.

5. IANA Considerations

5.1 Registration of the "GUID" SIP header

Name of Header: GUID

Short form: none

Normative description: section 3.4 of this document.

6. References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R. Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[4] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-00 (work in progress), January 2004.

[5] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[6] Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C. and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[7] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[8] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[9] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

[10] Rosenberg, J. and Schulzrinne, H., "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 2362, June 2002.

[11] Niemi, A., "SIMPLE Presence Publication Mechanism", draft-ietf-sip-publish-02 (work in progress), January 2004.

7. Acknowledgements

Thanks to Jennifer Beckman, Mary Barnes, Alberto Villarica, Trip Ingle, and William Janning for comments and helping to create the foundation for this document. Additionally, thanks is given to Jonathan Rosenberg for introduction of the GRUU draft which describes the server-side generation of unique identifiers within SIP.

Author's Addresses

Brian Stucker
Nortel Networks
2380 Performance Drive
Richardson, Texas
Email: bstucker@nortelnetworks.com