Internet Engineering Task Force (IETF)
Request for Comments: 5850
Category: Informational
ISSN: 2070-1721

R. Mahy
Unaffiliated
R. Sparks
Tekelec
J. Rosenberg
jdrosen.net
D. Petrie
SIPez
A. Johnston, Ed.
Avaya

May 2010

A Call Control and Multi-Party Usage Framework for the Session Initiation Protocol (SIP)

Abstract

This document defines a framework and the requirements for call control and multi-party usage of the Session Initiation Protocol (SIP). To enable discussion of multi-party features and applications, we define an abstract call model for describing the media relationships required by many of these. The model and actions described here are specifically chosen to be independent of the SIP signaling and/or mixing approach chosen to actually set up the media relationships. In addition to its dialog manipulation aspect, this framework includes requirements for communicating related information and events such as conference and session state and session history. This framework also describes other goals that embody the spirit of SIP applications as used on the Internet such as the definition of primitives (not services), invoker and participant oriented primitives, signaling and mixing model independence, and others.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5850.

Table of Contents

1. Motivation and Background 4
2. Key Concepts 6
2.1. Conversation Space Model 7
2.2. Relationship between Conversation Space, SIP Dialogs, and SIP Sessions 8
2.3. Signaling Models 9
2.4. Mixing Models 10
2.4.1. Tightly Coupled 11
2.4.2. Loosely Coupled 12
2.5. Conveying Information and Events 13
2.6. Componentization and Decomposition 15
2.6.1. Media Intermediaries 15
2.6.2. Text-to-Speech and Automatic Speech Recognition . . . 17
2.6.3. VoiceXML 17
2.7. Use of URIs 18
2.7.1. Naming Users in SIP 19
2.7.2. Naming Services with SIP URIs 20
2.8. Invoker Independence 22
2.9. Billing Issues 23

3. Catalog of Call Control Actions and Sample Features 23
3.1. Remote Call Control Actions on Early Dialogs 24
3.1.1. Remote Answer 24
3.1.2. Remote Forward or Put 24
3.1.3. Remote Busy or Error Out 24
3.2. Remote Call Control Actions on Single Dialogs 24
3.2.1. Remote Dial 24
3.2.2. Remote On and Off Hold 25
3.2.3. Remote Hangup 25
3.3. Call Control Actions on Multiple Dialogs 25
3.3.1. Transfer 25
3.3.2. Take 26
3.3.3. Add 27
3.3.4. Local Join 28
3.3.5. Insert 28
3.3.6. Split 29
3.3.7. Near-Fork 29
3.3.8. Far-Fork 29
4. Security Considerations 30
Appendix A. Example Features 32
Appendix A.1. Attended Transfer 32
Appendix A.2. Auto Answer 32
Appendix A.3. Automatic Callback 32
Appendix A.4. Barge-In 32
Appendix A.5. Blind Transfer 32
Appendix A.6. Call Forwarding 33
Appendix A.7. Call Monitoring 33
Appendix A.8. Call Park 33
Appendix A.9. Call Pickup 33
Appendix A.10. Call Return 34
Appendix A.11. Call Waiting 34
Appendix A.12. Click-to-Dial 34
Appendix A.13. Conference Call 34
Appendix A.14. Consultative Transfer 34
Appendix A.15. Distinctive Ring 35
Appendix A.16. Do Not Disturb 35
Appendix A.17. Find-Me 35
Appendix A.18. Hotline 35
Appendix A.19. IM Conference Alerts 35
Appendix A.20. Inbound Call Screening 35
Appendix A.21. Intercom 36
Appendix A.22. Message Waiting 36
Appendix A.23. Music on Hold 36
Appendix A.24. Outbound Call Screening 36
Appendix A.25. Pre-Paid Calling 37
Appendix A.26. Presence-Enabled Conferencing 37
Appendix A.27. Single Line Extension/Multiple Line Appearance . . 37
Appendix A.28. Speakerphone Paging 38

Appendix A.29. Speed Dial 38
Appendix A.30. Voice Message Screening 38
Appendix A.31. Voice Portal 39
Appendix A.32. Voicemail 40
Appendix A.33. Whispered Call Waiting 40
Appendix B. Acknowledgments 40
5. Informative References 40

1. Motivation and Background

The Session Initiation Protocol (SIP) [RFC3261] was defined for the initiation, maintenance, and termination of sessions or calls between one or more users. However, despite its origins as a large-scale multi-party conferencing protocol, SIP is used today primarily for point-to-point calls. This two-party configuration is the focus of the SIP specification and most of its extensions.

This document defines a framework and the requirements for call control and multi-party usage of SIP. Most multi-party operations manipulate SIP dialogs (also known as call legs) or SIP conference media policy to cause participants in a conversation to perceive specific media relationships. In other protocols that deal with the concept of calls, this manipulation is known as call control. In addition to its dialog or policy manipulation aspect, call control also includes communicating information and events related to manipulating calls, including information and events dealing with session state and history, conference state, user state, and even message state.

Based on input from the SIP community, the authors compiled the following set of goals for SIP call control and multi-party applications:

- Define primitives, not services. Allow for a handful of robust yet simple mechanisms that can be combined to deliver features and services. Throughout this document, we refer to these simple mechanisms as "primitives". Primitives should be sufficiently robust so that when they are combined with each other, they can be used to build lots of services. However, the goal is not to define a provably complete set of primitives. Note that while the IETF will NOT standardize behavior or services, it may define example services for informational purposes, as in service examples [RFC5359].

- Be participant oriented. The primitives should be designed to provide services that are oriented around the experience of the participants. The authors observe that end users of features and services usually don't care how a media relationship is set up.

Their ultimate experience is only based on the resulting media and other externally visible characteristics.

- Be signaling model independent. Support both a central-control and a peer-to-peer feature invocation model (and combinations of the two). Baseline SIP already supports a centralized control model described in 3pcc (third party call control) [RFC3725], and the SIP community has expressed a great deal of interest in peer-to-peer or distributed call control using primitives such as those defined in REFER [RFC3515], Replaces [RFC3891], and Join [RFC3911].

- Be mixing model independent. The bulk of interesting multi-party applications involve mixing or combining media from multiple participants. This mixing can be performed by one or more of the participants or by a centralized mixing resource. The experience of the participants should not depend on the mixing model used. While most examples in this document refer to audio mixing, the framework applies to any media type. In this context, a "mixer" refers to combining media of the same type in an appropriate, media-specific way. This is consistent with the model described in the SIP conferencing framework.

- Be invoker oriented. Only the user who invokes a feature or a service needs to know exactly which service is invoked or why. This is good because it allows new services to be created without requiring new primitives from all of the participants; and it allows for much simpler feature authorization policies, for example, when participation spans organizational boundaries. As discussed in Section 2.7, this also avoids exponential state explosion when combining features. The invoker only has to manage a user interface or application programming interface (API) to prevent local feature interactions. All the other participants simply need to manage the feature interactions of a much smaller number of primitives.

- Primitives make full use of URIs (uniform resource identifiers). URIs are a very powerful mechanism for describing users and services. They represent a plentiful resource that can be extremely expressive and easily routed, translated, and manipulated -- even across organizational boundaries. URIs can contain special parameters and informational header fields that need only be relevant to the owner of the namespace (domain) of the URI. Just as a user who selects an http: URL need not understand the significance and organization of the web site it references, a user may encounter a SIP URI that translates into an email-style group alias, which plays a pre-recorded message or runs some complex call-handling logic. Note that while this may

seem paradoxical to the previous goal, both goals can be satisfied by the same model.

- Make use of SIP header fields and SIP event packages to provide SIP entities with information about their environment. These should include information about the status/handling of dialogs on other user agents (UAs), information about the history of other contacts attempted prior to the current contact, the status of participants, the status of conferences, user presence information, and the status of messages.

- Encourage service decomposition, and design to make use of standard components using well-defined, simple interfaces. Sample components include a SIP mixer, recording service, announcement server, and voice-dialog server. (This is not an exhaustive list).

- Include authentication, authorization, policy, logging, and accounting mechanisms to allow these primitives to be used safely among mutually untrusted participants. Some of these mechanisms may be used to assist in billing, but no specific billing system will be endorsed.

- Permit graceful fallback to baseline SIP. Definitions for new SIP call control extensions/primitives must describe a graceful way to fallback to baseline SIP behavior. Support for one primitive must not imply support for another primitive.

- Don't reinvent traditional models, such as the model used for the H.450 family of protocols, JTAPI (Java Telephony Application Programming Interface), or the CSTA (Computer-supported telecommunications applications) call model, as these other models do not share the design goals presented in this document.

Note that the flexibility in this model does have some disadvantages in terms of interoperability. It is possible to build a call control feature in SIP using different combinations of primitives. For a discussion of the issues associated with this, see [BLISS-PROBLEM].

2. Key Concepts

This section introduces a number of key concepts that will be used to describe and explain various call control operations and services in the remainder of this document. This includes the conversation space model, signaling and mixing models, common components, and the use of URIs.

2.1. Conversation Space Model

This document introduces the concept of an abstract "conversation space" as a set of participants who believe they are all communicating among one another. Each conversation space contains one or more participants.

Participants are SIP UAs that send original media to or terminate and receive media from other members of the conversation space. Logically, every participant in the conversation space has access to all the media generated in that space (this is strictly true if all participants share a common media type). A SIP UA that does not contribute or consume any media is NOT a participant, nor is a UA that merely forwards, transcodes, mixes, or selects media originating elsewhere in the conversation space.

> Note that a conversation space consists of zero or more SIP calls or SIP conferences. A conversation space is similar to the definition of a "call" in some other call models.

Participants may represent human users or non-human users (referred to as robots or automatons in this document). Some participants may be hidden within a conversation space. Some examples of hidden participants include: robots that generate tones, images, or announcements during a conference to announce users arriving and departing, a human call center supervisor monitoring a conversation between a trainee and a customer, and robots that record media for training or archival purposes.

Participants may also be active or passive. Active participants are expected to be intelligent enough to leave a conversation space when they no longer desire to participate. (An attentive human participant is obviously active.) Some robotic participants (such as a voice-messaging system, an instant-messaging agent, or a voice-dialog system) may be active participants if they can leave the conversation space when there is no human interaction. Other robots (for example, our tone-generating robot from the previous example) are passive participants. A human participant "on hold" is passive.

An example diagram of a conversation space can be shown as a "bubble" or ovals, or as a "set" in curly or square bracket notation. Each set, oval, or bubble represents a conversation space. Hidden participants are shown in lowercase letters. Examples are given in Figure 1.

Note that while the term "conversation" usually applies to oral exchange of information, we apply the conversation space model to any media exchange between participants.

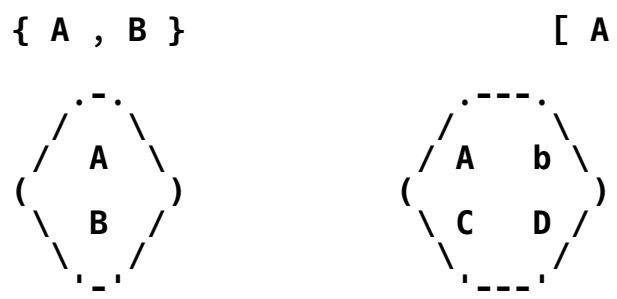

Figure 1. Conversation Spaces

2.2. Relationship between Conversation Space, SIP Dialogs, and SIP Sessions

In [RFC3261], a call is "an informal term that refers to some communication between peers, generally set up for the purposes of a multimedia conversation". The concept of a conversation space is needed because the SIP definition of call is not sufficiently precise for the purpose of describing the user experience of multi-party features.

Do any other definitions convey the correct meaning? SIP and SDP (Session Description Protocol) [RFC4566] both define a conference as "a multimedia session identified by a common session description". A session is defined as "a set of multimedia senders and receivers and the data streams flowing from senders to receivers". The definition of "call" in some call models is more similar to our definition of a conversation space.

Some examples of the relationship between conversation spaces, SIP dialogs, and SIP sessions are listed below. In each example, a human user will perceive that there is a single call.

- A simple two-party call is a single conversation space, a single session, and a single dialog.
- A locally mixed three-way call is two sessions and two dialogs. It is also a single conversation space.
- A simple dial-in audio conference is a single conversation space, but is represented by as many dialogs and sessions as there are human participants.
- A multicast conference is a single conversation space, a single session, and as many dialogs as participants.

2.3. Signaling Models

Obviously, to make changes to a conversation space, you must be able to use SIP signaling to cause these changes. Specifically, there must be a way to manipulate SIP dialogs (call legs) to move participants into and out of conversation spaces. Although this is not as obvious, there also must be a way to manipulate SIP dialogs to include non-participant UAs that are otherwise involved in a conversation space (e.g., back-to-back user agents or B2BUAs, third party call control (3pcc) controllers, mixers, transcoders, translators, or relays).

Implementations may setup the media relationships described in the conversation space model using a centralized control model. One common way to implement this using SIP is known as third party call control (3pcc) and is described in 3pcc [RFC3725]. The 3pcc approach relies on only the following three primitive operations:

- Create a new dialog (INVITE)
- Modify a dialog (reINVITE)
- Destroy a dialog (BYE)

The main advantage of the 3pcc approach is that it only requires very basic SIP support from end systems to support call control features. As such, third party call control is a natural way to handle protocol conversion and mid-call features. It also has the advantage and disadvantage that new features can/must be implemented in one place only (the controller), and it neither requires enhanced client functionality nor takes advantage of it.

In addition, a peer-to-peer approach is discussed at length in this document. The primary drawback of the peer-to-peer model is additional complexity in the end system and authentication and management models. The benefits of the peer-to-peer model include:

- state remains at the edges,
- call signaling need only go through participants involved (there are no additional points of failure), and
- peers may take advantage of end-to-end message integrity or encryption

The peer-to-peer approach relies on additional "primitive" operations, some of which are identified here.

- Replace an existing dialog
- Join a new dialog with an existing dialog
- Locally perform media forking (multi-unicast)
- Ask another user agent (UA) to send a request on your behalf

The peer-to-peer approach also only results in a single SIP dialog, directly between the two UAs. The 3pcc approach results in two SIP dialogs, between each UA and the controller. As a result, the SIP features and extensions that will be used during the dialog are limited to the those understood by the controller. As a result, in a situation where both the UAs support an advanced SIP feature but the controller does not, the feature will not be able to be used.

Many of the features, primitives, and actions described in this document also require some type of media mixing, combining, or selection as described in the next section.

2.4. Mixing Models

SIP permits a variety of mixing models, which are discussed here briefly. This topic is discussed more thoroughly in the SIP conferencing framework [RFC4353] and [RFC4579]. SIP supports both tightly coupled and loosely coupled conferencing, although more sophisticated behavior is available in tightly coupled conferences. In a tightly coupled conference, a single SIP user agent (called the focus) has a direct dialog relationship with each participant (and may control non-participant user agents as well). The focus can authoritatively publish information about the character and participants in a conference. In a loosely coupled conference, there are no coordinated signaling relationships among the participants.

For brevity, only the two most popular conferencing models are significantly discussed in this document (local and centralized mixing). Applications of the conversation spaces model to loosely coupled multicast and distributed full unicast mesh conferences are left as an exercise for the reader. Note that a distributed full mesh conference can be used for basic conferences, but does not easily allow for more complex conferencing actions like splitting, merging, and sidebars.

Call control features should be designed to allow a mixer (local or centralized) to decide when to reduce a conference back to a two-party call, or drop all the participants (for example, if only two automatons are communicating). The actual heuristics used to release calls are beyond the scope of this document, but may depend on properties in the conversation space, such as the number of active, passive, or hidden participants and the send-only, receive-only, or send-and-receive orientation of various participants.

2.4.1. Tightly Coupled

Tightly coupled conferences utilize a central point for signaling and authentication known as a focus [RFC4353]. The actual media can be centrally mixed or distributed.

2.4.1.1. (Single) End System Mixing

The first model we call "end system mixing". In this model, user A calls user B, and they have a conversation. At some point later, A decides to conference in user C. To do this, A calls C, using a completely separate SIP call. This call uses a different Call-ID, different tags, etc. There is no call set up directly between B and C. No SIP extension or external signaling is needed. A merely decides to locally join two dialogs.

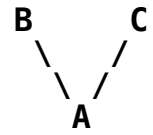

Figure 2. End System Mixing Example

In Figure 2, A receives media streams from both B and C, and mixes them. A sends a stream containing A's and C's streams to B, and a stream containing A's and B's streams to C. Basically, user A handles both signaling and media mixing.

2.4.1.2. Centralized Mixing

In a centralized mixing model, all participants have a pairwise SIP and media relationship with the mixer. Common applications of centralized mixing include ad hoc conferences and scheduled dial-in or dial-out conferences. In Figure 3 below, the mixer M receives and sends media to participants A, B, C, D, and E.

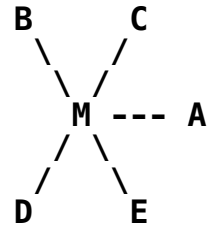

Figure 3. Centralized Mixing Example

2.4.1.3. Centralized Signaling, Distributed Media

In this conferencing model, there is a centralized controller, as in the dial-in and dial-out cases. However, the centralized server handles signaling only. The media is still sent directly between participants, using either multicast or multi-unicast. Participants perform their own mixing. Multi-unicast is when a user sends multiple packets (one for each recipient, addressed to that recipient). This is referred to as a "Decentralized Multipoint Conference" in [H.323]. Full mesh media with centralized mixing is another approach.

2.4.2. Loosely Coupled

In these models, there is no point of central control of SIP signaling. As in the "Centralized Signaling, Distributed Media" case above, all endpoints send media to all other endpoints. Consequently, every endpoint mixes their own media from all the other sources and sends their own media to every other participant.

2.4.2.1. Large-Scale Multicast Conferences

Large-scale multicast conferences were the original motivation for both the Session Description Protocol (SDP) [RFC4566] and SIP. In a large-scale multicast conference, one or more multicast addresses are allocated to the conference. Each participant joins those multicast groups and sends their media to those groups. Signaling is not sent to the multicast groups. The sole purpose of the signaling is to inform participants of which multicast groups to join. Large-scale multicast conferences are usually pre-arranged, with specific start and stop times. However, multicast conferences do not need to be pre-arranged, so long as a mechanism exists to dynamically obtain a multicast address.

2.4.2.2. Full Distributed Unicast Conferencing

In this conferencing model, each participant has both a pairwise media relationship and a pairwise signaling relationship with every other participant (a full mesh). This model requires a mechanism to maintain a consistent view of distributed state across the group. This is a classic, hard problem in computer science. Also, this model does not scale well for large numbers of participants. For <n> participants, the number of media and signaling relationships is approximately n-squared. As a result, this model is not generally available in commercial implementations; to the contrary, it is primarily the topic of research or experimental implementations. Note that this model assumes peer-to-peer signaling.

2.5. Conveying Information and Events

Participants should have access to information about the other participants in a conversation space so that this information can be rendered to a human user or processed by an automaton. Although some of this information may be available from the Request-URI or To, From, Contact, or other SIP header fields, another mechanism of reporting this information is necessary.

Many applications are driven by knowledge about the progress of calls and conferences. In general, these types of events allow for the construction of distributed applications, where the application requires information on dialog and conference state, but is not necessarily a co-resident with an endpoint user agent or conference server. For example, a focus involved in a conversation space may wish to provide URIs for conference status and/or conference/floor control.

The SIP Events architecture [RFC3265] defines general mechanisms for subscription to and notification of events within SIP networks. It introduces the notion of a package that is a specific "instantiation" of the events mechanism for a well-defined set of events.

Event packages are needed to provide the status of a user's dialogs, the status of conferences and their participants, user-presence information, the status of registrations, and the status of a user's messages. While this is not an exhaustive list, these are sufficient to enable the sample features described in this document.

The conference event package [RFC4575] allows users to subscribe to information about an entire tightly coupled SIP conference. Notifications convey information about the participants such as the SIP URI identifying each user, their status in the space (active, declined, departed), URIs to invoke other features (such as sidebar

conversations), links to other relevant information (such as floor-control policies), and if floor-control policies are in place, the user's floor-control status. For conversation spaces created from cascaded conferences, conversation state can be gathered from relevant foci and merged into a cohesive set of state.

The dialog package [RFC4235] provides information about all the dialogs the target user is maintaining, in which conversations the user is participating, and how these are correlated. Likewise, the registration package [RFC3680] provides notifications when contacts have changed for a specific address-of-record (AOR). The combination of these allows a user agent to learn about all conversations occurring for the entire registered contact set for an address-of-record.

Note that user presence in SIP [RFC3856] has a close relationship with these latter two event packages. It is fundamental to the presence model that the information used to obtain user presence is constructed from any number of different input sources. Examples of other such sources include calendaring information and uploads of presence documents. These two packages can be considered another mechanism that allows a presence agent to determine the presence state of the user. Specifically, a user presence server can act as a subscriber for the dialog and registration packages to obtain additional information that can be used to construct a presence document.

The multi-party architecture may also need to provide a mechanism to get information about the status/handling of a dialog (for example, information about the history of other contacts attempted prior to the current contact). Finally, the architecture should provide ample opportunities to present informational URIs that relate to calls, conversations, or dialogs in some way. For example, consider the SIP Call-Info header or Contact header fields returned in a 300-class response. Frequently, additional information about a call or dialog can be fetched via non-SIP URIs. For example, consider a web page for package tracking when calling a delivery company or a web page with related documentation when joining a dial-in conference. The use of URIs in the multi-party framework is discussed in more detail in Section 3.7.

Finally, the interaction of SIP with stimulus-signaling-based applications, which allow a user agent to interact with an application without knowledge of the semantics of that application, is discussed in the SIP application interaction framework [RFC5629]. Stimulus signaling can occur with a user interface running locally with the client, or with a remote user interface, through media streams. Stimulus signaling encompasses a wide range of mechanisms,

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 that framework.

2.6. Componentization and Decomposition

This framework proposes a decomposed component architecture with a very loose coupling of services and components. This means that a service (such as a conferencing server or an auto-attendant) need not be implemented as an actual server. Rather, these services can be built by combining a few basic components in straightforward or arbitrarily complex ways.

Since the components are easily deployed on separate boxes, by separate vendors, or even with separate providers, we achieve a separation of function that allows each piece to be developed in complete isolation. We can also reuse existing components for new applications. This allows rapid service creation, and the ability for services to be distributed across organizational domains anywhere in the Internet.

For many of these components, it is also desirable to discover their capabilities, for example, querying the ability of a mixer to host a 10-dialog conference or to reserve resources for a specific time. These actions could be provided in the form of URIs, provided there is an a priori means of understanding their semantics. For example, if there is a published dictionary of operations, a way to query the service for the available operations and the associated URIs, the URI can be the interface for providing these service operations. This concept is described in more detail in the context of dialog operations in Section 3.

2.6.1. Media Intermediaries

Media intermediaries are not participants in any conversation space, although an entity that is also a media translator may also have a co-located participant component (for example, a mixer that also announces the arrival of a new participant; the announcement portion is a participant, but the mixer itself is not). Media intermediaries should be as transparent as possible to the end users -- offering a useful, fundamental service without getting in the way of new features implemented by participants. Some common media intermediaries are described below.

2.6.1.1. Mixer

A SIP mixer is a component that combines media from all dialogs in the same conversation in a media-specific way. For example, the default combining for an audio conference might be an N-1 configuration, while a text mixer might interleave text messages on a per-line basis. More details about how to manipulate the media policy used by mixers is discussed in [XCON-CCMP].

2.6.1.2. Transcoder

A transcoder translates media from one encoding or format to another (for example, GSM (Global System for Mobile communications) voice to G.711, MPEG2 to H.261, or text/html to text/plain), or from one media type to another (for example, text to speech). A more thorough discussion of transcoding is described in the SIP transcoding services invocation [RFC5369].

2.6.1.3. Media Relay

A media relay terminates media and simply forwards it to a new destination without changing the content in any way. Sometimes, media relays are used to provide source IP address anonymity, to facilitate middlebox traversal, or to provide a trusted entity where media can be forcefully disconnected.

2.6.1.4. Queue Server

A queue server is a location where calls can be entered into one of several FIFO (first-in, first-out) queues. A queue server would subscribe to the presence of groups or individuals who are interested in its queues. When detecting that a user is available to service a queue, the server redirects or transfers the last call in the relevant queue to the available user. On a queue-by-queue basis, authorized users could also subscribe to the call state (dialog information) of calls within a queue. Authorized users could use this information to effectively pluck (take) a call out of the queue (for example, by sending an INVITE with a Replaces header to one of the user agents in the queue).

2.6.1.5. Parking Place

A parking place is a location where calls can be terminated temporarily and then retrieved later. While a call is "parked", it can receive media "on hold" such as music, announcements, or advertisements. Such a service could be further decomposed such that announcements or music are handled by a separate component.

2.6.1.6. Announcements and Voice Dialogs

An announcement server is a server that can play digitized media (frequently audio), such as music or recorded speech. These servers are typically accessible via SIP, HTTP (Hyper Text Transport Protocol), or RTSP (Real-Time Streaming Protocol). An analogous service is a recording service that stores digitized media. A convention for specifying announcements in SIP URIs is described in [RFC4240]. Likewise, the same server could easily provide a service that records digitized media.

A "voice dialog" is a model of spoken interactive behavior between a human and an automaton that can include synthesized speech, digitized audio, recognition of spoken and DTMF key input, a recording of spoken input, and interaction with call control. Voice dialogs frequently consist of forms or menus. Forms present information and gather input; menus offer choices of what to do next.

Spoken dialogs are a basic building block of applications that use voice. Consider, for example, that a voicemail system, the conference-id and passcode collection system for a conferencing system, and complicated voice-portal applications all require a voice-dialog component.

2.6.2. Text-to-Speech and Automatic Speech Recognition

Text-to-speech (TTS) is a service that converts text into digitized audio. TTS is frequently integrated into other applications, but when separated as a component, it provides greater opportunity for broad reuse. Automatic Speech Recognition (ASR) is a service that attempts to decipher digitized speech based on a proposed grammar. Like TTS, ASR services can be embedded, or exposed so that many applications can take advantage of such services. A standardized (decomposed) interface to access standalone TTS and ASR services is currently being developed as described in [RFC4313].

2.6.3. VoiceXML

VoiceXML is a W3C (World Wide Web Consortium) recommendation that was designed to give authors control over the spoken dialog between users and applications. The application and user take turns speaking: the application prompts the user, and the user in turn responds. Its major goal is to bring the advantages of web-based development and content delivery to interactive voice-response applications. We believe that VoiceXML represents the ideal partner for SIP in the development of distributed IVR (interactive voice response) servers. VoiceXML is an XML-based scripting language for describing IVR services at an abstract level. VoiceXML supports DTMF recognition,

speech recognition, text-to-speech, and the playing out of recorded media files. The results of the data collected from the user are passed to a controlling entity through an HTTP POST operation. The controller can then return another script, or terminate the interaction with the IVR server.

A VoiceXML server also need not be implemented as a monolithic server. Figure 4 shows a diagram of a VoiceXML browser that is split into media and non-media handling parts. The VoiceXML interpreter handles SIP dialog state and state within a VoiceXML document, and sends requests to the media component over another protocol.

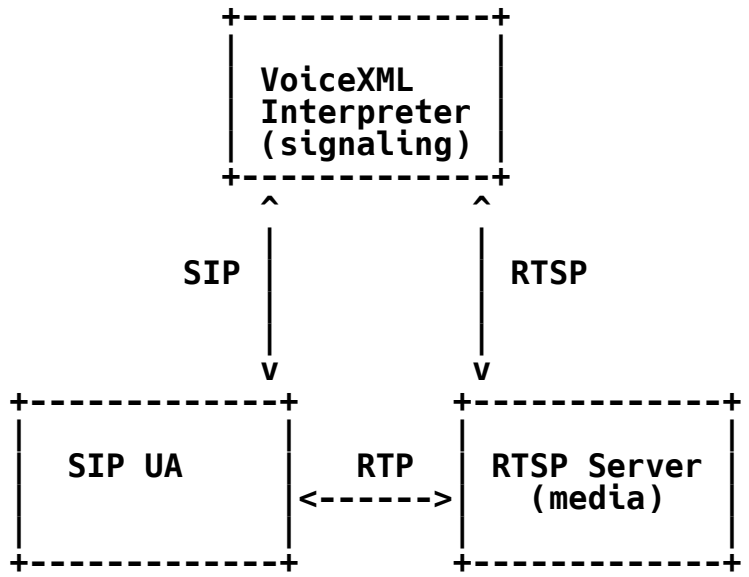

Figure 4. Decomposed VoiceXML Server

2.7. Use of URIs

All naming in SIP uses URIs. URIs in SIP are used in a plethora of contexts: the Request-URI; Contact, To, From, and *-Info header fields; application/uri bodies; and embedded in email, web pages, instant messages, and ENUM records. The Request-URI identifies the user or service for which the call is destined.

SIP URIs embedded in informational SIP header fields, SIP bodies, and non-SIP content can also specify methods, special parameters, header fields, and even bodies. For example:

sip:bob@b.example.com;method=REFER?Refer-To=http://example.com/~alice

Throughout this document, we discuss call control primitive operations. One of the biggest problems is defining how these operations may be invoked. There are a number of ways to do this. One way is to define the primitives in the protocol itself such that SIP methods (for example, REFER) or SIP header fields (for example, Replaces) indicate a specific call control action. Another way to invoke call control primitives is to define a specific Request-URI naming convention. Either these conventions must be shared between the client (the invoker) and the server, or published by or on behalf of the server. The former involves defining URI construction techniques (e.g., URI parameters and/or token conventions) as proposed in [RFC4240]. The latter technique usually involves discovering the URI via a SIP event package, a web page, a business card, or an instant message. Yet, another means to acquire the URIs is to define a dictionary of primitives with well-defined semantics and provide a means to query the named primitives and corresponding URIs that may be invoked on the service or dialogs.

2.7.1. Naming Users in SIP

An address-of-record, or public SIP address, is a SIP (or Secure SIP (SIPS)) URI that points to a domain with a location service that can map the URI to set of Contact URIs where the user might be available. Typically, the Contact URIs are populated via registration.

```
Address-of-Record                Contacts

sip:bob@biloxi.example.com -> sip:bob@babylon.biloxi.example.com:5060
                              sip:bbrown@mailbox.provider.example.net
                              sip:+1.408.555.6789@mobile.example.net
```

Callee Capabilities [RFC3840] define a set of additional parameters to the Contact header field that define the characteristics of the user agent at the specified URI. For example, there is a mobility parameter that indicates whether the UA is fixed or mobile. When a user agent registers, it places these parameters in the Contact header fields to characterize the URIs it is registering. This allows a proxy for that domain to have information about the contact addresses for that user.

When a caller sends a request, it can optionally request Caller Preferences [RFC3841] by including the Accept-Contact, Request-Disposition, and Reject-Contact header fields that request certain handling by the proxy in the target domain. These header fields contain preferences that describe the set of desired URIs to which the caller would like their request routed. The proxy in the target domain matches these preferences with the Contact characteristics originally registered by the target user. The target user can also

choose to run arbitrarily complex "Find-me" feature logic on a proxy in the target domain.

There is a strong asymmetry in how preferences for callers and callees can be presented to the network. While a caller takes an active role by initiating the request, the callee takes a passive role in waiting for requests. This motivates the use of callee-supplied scripts and caller preferences included in the call request. This asymmetry is also reflected in the appropriate relationship between caller and callee preferences. A server for a callee should respect the wishes of the caller to avoid certain locations, while the preferences among locations has to be the callee's choice, as it determines where, for example, the phone rings and whether the callee incurs mobile telephone charges for incoming calls.

SIP User Agent implementations are encouraged to make intelligent decisions based on the type of participants (active/passive, hidden, human/robot) in a conversation space. This information is conveyed via the dialog package or in a SIP header field parameter communicated using an appropriate SIP header field. For example, a music on hold service may take the sensible approach that if there are two or more unhidden participants, it should not provide hold music; or that it will not send hold music to robots.

Multiple participants in the same conversation space may represent the same human user. For example, the user may use one participant device for video, chat, and whiteboard media on a PC and another for audio media on a SIP phone. In this case, the address-of-record is the same for both user agents, but the Contacts are different. In this case, there is really only one human participant. In addition, human users may add robot participants that act on their behalf (for example, a call recording service or a calendar announcement reminder). Call control features in SIP should continue to function as expected in such an environment.

2.7.2. Naming Services with SIP URIs

A critical piece of defining a session-level service that can be accessed by SIP is defining the naming of the resources within that service. This point cannot be overstated.

In the context of SIP control of application components, we take advantage of the fact that the left-hand side of a standard SIP URI is a user part. Most services may be thought of as user automatons that participate in SIP sessions. It naturally follows that the user part should be utilized as a service indicator.

For example, media servers commonly offer multiple services at a single host address. Use of the user part as a service indicator enables service consumers to direct their requests without ambiguity. It has the added benefit of enabling media services to register their availability with SIP Registrars just as any "real" SIP user would. This maintains consistency and provides enhanced flexibility in the deployment of media services in the network.

There has been much discussion about the potential for confusion if media-service URIs are not readily distinguishable from other types of SIP UAs. The use of a service namespace provides a mechanism to unambiguously identify standard interfaces while not constraining the development of private or experimental services.

In SIP, the Request-URI identifies the user or service for which the call is destined. The great advantage of using URIs (specifically, the SIP Request-URI) as a service identifier comes because of the combination of two facts. First, unlike in the PSTN (Public Switched Telephone Network), where the namespace (dialable telephone numbers) is limited, URIs come from an infinite space. They are plentiful, and they are free. Secondly, the primary function of SIP is call routing through manipulations of the Request-URI. In the traditional SIP application, this URI represents a person. However, the URI can also represent a service, as we propose here. This means we can apply the routing services SIP provides to the routing of calls to services. The result -- the problem of service invocation and service location becomes a routing problem, for which SIP provides a scalable and flexible solution. Since there is such a vast namespace of services, we can explicitly name each service in a finely granular way. This allows the distribution of services across the network. For further discussion about services and SIP URIs, see RFC 3087 [RFC3087].

Consider a conferencing service, where we have separated the names of ad hoc conferences from scheduled conferences, we can program proxies to route calls for ad hoc conferences to one set of servers and calls for scheduled ones to another, possibly even in a different provider. In fact, since each conference itself is given a URI, we can distribute conferences across servers, and easily guarantee that calls for the same conference always get routed to the same server. This is in stark contrast to conferences in the telephone network, where the equivalent of the URI -- the phone number -- is scarce. An entire conferencing provider generally has one or two numbers. Conference IDs must be obtained through IVR interactions with the caller or through a human attendant. This makes it difficult to distribute conferences across servers all over the network, since the PSTN routing only knows about the dialed number.

For more examples, consider the URI conventions of RFC 4240 [RFC4240] for media servers and RFC 4458 [RFC4458] for voicemail and IVR systems.

In practical applications, it is important that an invoker does not necessarily apply semantic rules to various URIs it did not create. Instead, it should allow any arbitrary string to be provisioned, and map the string to the desired behavior. The administrator of a service may choose to provision specific conventions or mnemonic strings, but the application should not require it. In any large installation, the system owner is likely to have preexisting rules for mnemonic URIs, and any attempt by an application to define its own rules may create a conflict. Implementations should allow an arbitrary mix of URIs from these schemes, or any other scheme that renders valid SIP URIs, rather than enforce only one particular scheme.

As we have shown, SIP URIs represent an ideal, flexible mechanism for describing and naming service resources, regardless of whether the resources are queues, conferences, voice dialogs, announcements, voicemail treatments, or phone features.

2.8. Invoker Independence

With functional signaling, only the invoker of features in SIP needs to know exactly which feature they are invoking. One of the primary benefits of this approach is that combinations of functional features work in SIP call control without requiring complex feature-interaction matrices. For example, let us examine the combination of a "transfer" of a call that is "conferenced".

Alice calls Bob. Alice silently "conferences in" her robotic assistant Albert as a hidden party. Bob transfers Alice to Carol. If Bob asks Alice to Replace her leg with a new one to Carol, then both Alice and Albert should be communicating with Carol (transparently).

Using the peer-to-peer model, this combination of features works fine if A is doing local mixing (Alice replaces Bob's dialog with Carol's), or if A is using a central mixer (the mixer replaces Bob's dialog with Carol's). A clever implementation using the 3pcc model can generate similar results.

New extensions to the SIP Call Control Framework should attempt to preserve this property.

2.9. Billing Issues

Billing in the PSTN is typically based on who initiated a call. At the moment, billing in a SIP network is neither consistent with itself nor with the PSTN. (A billing model for SIP should allow for both PSTN-style billing and non-PSTN billing.) The example below demonstrates one such inconsistency.

Alice places a call to Bob. Alice then blind transfers Bob to Carol through a PSTN gateway. In current usage of REFER, Bob may be billed for a call he did not initiate (his UA originated the outgoing dialog, however). This is not necessarily a terrible thing, but it demonstrates a security concern (Bob must have appropriate local policy to prevent fraud). Also, Alice may wish to pay for Bob's session with Carol. There should be a way to signal this in SIP.

Likewise, a Replacement call may maintain the same billing relationship as a Replaced call, so if Alice first calls Carol, then asks Bob to Replace this call, Alice may continue to receive a bill.

Further work in SIP billing should define a way to set or discover the direction of billing.

3. Catalog of Call Control Actions and Sample Features

Call control actions can be categorized by the dialogs upon which they operate. The actions may involve a single or multiple dialogs. These dialogs can be early or established. Multiple dialogs may be related in a conversation space to form a conference or other interesting media topologies.

It should be noted that it is desirable to provide a means by which a party can discover the actions that may be performed on a dialog. The interested party may be independent or related to the dialogs. One means of accomplishing this is through the ability to define and obtain URIs for these actions, as described in Section 2.7.2.

Below are listed several call control "actions" that establish or modify dialogs and relate the participants in a conversation space. The names of the actions listed are for descriptive purposes only (they are not normative). This list of actions is not meant to be exhaustive.

In the examples, all actions are initiated by the user "Alice" represented by UA "A".

3.1. Remote Call Control Actions on Early Dialogs

The following are a set of actions that may be performed on a single early dialog. These actions can be thought of as a set of remote control operations. For example, an automaton might perform the operation on behalf of a user. Alternatively, a user might use the remote control in the form of an application to perform the action on the early dialog of a UA that may be out of reach. All of these actions correspond to telling the UA how to respond to a request to establish an early dialog. These actions provide useful functionality for PDA-, PC-, and server-based applications that desire the ability to control a UA. A proposed mechanism for this type of functionality is described in remote call control [FEATURE-REF].

3.1.1. Remote Answer

A dialog is in some early dialog state such as 180 Ringing. It may be desirable to tell the UA to answer the dialog. That is, tell it to send a 200 OK response to establish the dialog.

3.1.2. Remote Forward or Put

It may be desirable to tell the UA to respond with a 3xx class response to forward an early dialog to another UA.

3.1.3. Remote Busy or Error Out

It may be desirable to instruct the UA to send an error response such as 486 Busy Here.

3.2. Remote Call Control Actions on Single Dialogs

There is another useful set of actions that operate on a single established dialog. These operations are useful in building productivity applications for aiding users in controlling their phones. For example, a Customer Relationship Management (CRM) application that sets up calls for a user eliminating the need for the user to actually enter an address. These operations can also be thought of as remote control actions. A proposed mechanism for this type of functionality is described in remote call control [FEATURE-REF].

3.2.1. Remote Dial

This action instructs the UA to initiate a dialog. This action can be performed using the REFER method.

3.2.2. Remote On and Off Hold

This action instructs the UA to put an established dialog on hold. Though this operation can conceptually be performed with the REFER method, there are no semantics defined as to what the referred party should do with the SDP. There is no way to distinguish between the desire to go on or off hold on a per-media stream basis.

3.2.3. Remote Hangup

This action instructs the UA to terminate an early or established dialog. A REFER request with the following Refer-To URI and Target-Dialog header field [RFC4538] performs this action. Note: this example does not show the full set of header fields.

```
REFER sip:carol@client.chicago.net SIP/2.0
Refer-To: sip:bob@babylon.biloxi.example.com;method=BYE
Target-Dialog: 13413098;local-tag=879738;remote-tag=023214
```

3.3. Call Control Actions on Multiple Dialogs

These actions apply to a set of related dialogs.

3.3.1. Transfer

This section describes how call transfer can be achieved using centralized (3pcc) and peer-to-peer (REFER) approaches.

The conversation space changes as follows:

```
 before            after
{ A , B }  -->   { C , B }
```

A replaces itself with C.

To make this happen using the peer-to-peer approach, "A" would send two SIP requests. A shorthand for those requests is shown below:

```
REFER B  Refer-To:C
BYE B
```

To make this happen using the 3pcc approach instead, the controller sends requests represented by the shorthand below:

```
INVITE C (w/SDP of B)
reINVITE B (w/SDP of C)
BYE A
```

Features enabled by this action:

- blind transfer
- transfer to a central mixer (some type of conference or forking)
- transfer to park server (park)
- transfer to music on hold or announcement server
- transfer to a "queue"
- transfer to a service (such as voice-dialog service)
- transition from local mixer to central mixer

This action is frequently referred to as "completing an attended transfer". It is described in more detail in [RFC5589].

Note that if a transfer requires URI hiding or privacy, then the 3pcc approach can more easily implement this. For example, if the URI of C needs to be hidden from B, then the use of 3pcc helps accomplish this.

3.3.2. Take

The conversation space changes as follows:

{ B , C } --> { B , A }

A forcibly replaces C with itself. In most uses of this primitive, A is just "un-replacing" itself.

Using the peer-to-peer approach, "A" sends:

```
 INVITE B  Replaces: <dialog between B and C>
```

Using the 3pcc approach (all requests sent from controller):

```
 INVITE A (w/SDP of B)
 reINVITE B (w/SDP of A)
 BYE C
```

Features enabled by this action:

- transferee completes an attended transfer
- retrieve from central mixer (not recommended)
- retrieve from music on hold or park
- retrieve from queue
- call center take
- voice portal resuming ownership of a call it originated
- answering-machine style screening (pickup)
- pickup of a ringing call (i.e., early dialog)

Note that pick up of a ringing call has perhaps some interesting additional requirements. First of all, it is an early dialog as opposed to an established dialog. Secondly, the party that is to pick up the call may only wish to do so only while it is an early dialog. That is in the race condition where the ringing UA accepts just before it receives signaling from the party wishing to take the call, the taking party wishes to yield or cancel the take. The goal is to avoid yanking an answered call from the called party.

This action is described in Replaces [RFC3891] and in [RFC5589].

3.3.3. Add

Note that the following four actions are described in [RFC4579].

This is merely adding a participant to a SIP conference. The conversation space changes as follows:

{ A , B } --> { A , B , C }

A adds C to the conversation.

Using the peer-to-peer approach, adding a party using local mixing requires no signaling. To transition from a two-party call or a locally mixed conference to central mixing, A could send the following requests:

```
REFER B  Refer-To: conference-URI
INVITE conference-URI
BYE B
```

To add a party to a conference:

```
REFER C  Refer-To: conference-URI
                or
REFER conference-URI  Refer-To: C
```

Using the 3pcc approach to transition to centrally mixed, the controller would send:

```
INVITE mixer leg 1 (w/SDP of A)
INVITE mixer leg 2 (w/SDP of B)
INVITE C (late SDP)
reINVITE A (w/SDP of mixer leg 1)
reINVITE B (w/SDP of mixer leg 2)
INVITE mixer leg3 (w/SDP of C)
```

To add a party to a SIP conference:

```
 INVITE C (late SDP)
 INVITE conference-URI (w/SDP of C)
```

Features enabled:

- standard conference feature
- call recording
- answering-machine style screening (screening)

3.3.4. Local Join

The conversation space changes like this:

```
{ A , B } , { A , C }  -->  { A , B , C }

        or like this

{ A , B } , { C , D }  -->  { A , B , C , D }
```

A takes two conversation spaces and joins them together into a single space.

Using the peer-to-peer approach, A can mix locally, or REFER the participants of both conversation spaces to the same central mixer (as in Section 3.3.5).

For the 3pcc approach, the call flows for inserting participants, and joining and splitting conversation spaces are tedious yet straightforward, so these are left as an exercise for the reader.

Features enabled:

- standard conference feature
- leaving a sidebar to rejoin a larger conference

3.3.5. Insert

The conversation space changes like this:

```
{ B , C } --> { A , B , C }
```

A inserts itself into a conversation space.

A proposed mechanism for signaling this using the peer-to-peer approach is to send a new header field in an INVITE with "joining" [RFC3911] semantics. For example:

INVITE B Join: <dialog id of B and C>

If B accepted the INVITE, B would accept responsibility to set up the dialogs and mixing necessary (for example, to mix locally or to transfer the participants to a central mixer).

Features enabled:

- barge-in
- call center monitoring
- call recording

3.3.6. Split

{ A , B , C , D } --> { A , B } , { C , D }

If using a central conference with peer-to-peer

```
REFER C  Refer-To: conference-URI (new URI)
REFER D  Refer-To: conference-URI (new URI)
BYE C
BYE D
```

Features enabled:

- sidebar conversations during a larger conference

3.3.7. Near-Fork

A participates in two conversation spaces simultaneously:

{ A, B } --> { B , A } & { A , C }

A is a participant in two conversation spaces such that A sends the same media to both spaces, and renders media from both spaces, presumably by mixing or rendering the media from both. We can define that A is the "anchor" point for both forks, each of which is a separate conversation space.

This action is purely local implementation (it requires no special signaling). Local features such as switching calls between the background and foreground are possible using this media relationship.

3.3.8. Far-Fork

The conversation space diagram.

{ A, B } --> { A , B } & { B , C }

A requests B to be the "anchor" of two conversation spaces.

This is easily set up by creating a conference with two sub-conferences and setting the media policy appropriately such that B is a participant in both. Media forking can also be set up using 3pcc, as described in Section 5.1 of RFC 3264 [RFC3264] (an offer/answer model for SDP). The session descriptions for forking are quite complex. Controllers should verify that endpoints can handle forked media, for example, using prior configuration.

Features enabled:

- barge-in
- voice-portal services
- whisper
- key word detection
- sending DTMF somewhere else

4. Security Considerations

Call control primitives provide a powerful set of features that can be dangerous in the hands of an attacker. To complicate matters, call control primitives are likely to be automatically authorized without direct human oversight.

The class of attacks that are possible using these tools includes the ability to eavesdrop on calls, disconnect calls, redirect calls, render irritating content (including ringing) at a user agent, cause an action that has billing consequences, subvert billing (theft-of-service), and obtain private information. Call control extensions must take extra care to describe how these attacks will be prevented.

We can also make some general observations about authorization and trust with respect to call control. The security model is dramatically dependent on the signaling model chosen (see Section 2.3)

Let us first examine the security model used in the 3pcc approach. All signaling goes through the controller, which is a trusted entity. Traditional SIP authentication and hop-by-hop encryption and message integrity work fine in this environment, but end-to-end encryption and message integrity may not be possible.

When using the peer-to-peer approach, call control actions and primitives can be legitimately initiated by a) an existing participant in the conversation space, b) a former participant in the conversation space, or c) an entity trusted by one of the participants. For example, a participant always initiates a

transfer; a retrieve from park (a take) is initiated on behalf of a former participant, and a barge-in (insert or far-fork) is initiated by a trusted entity (an operator, for example).

Authenticating requests by an existing participant or a trusted entity can be done with baseline SIP mechanisms. In the case of features initiated by a former participant, these should be protected against replay attacks, e.g., by using a unique name or identifier per invocation. The Replaces header field exhibits this behavior as a by-product of its operation (once a Replaces operation is successful, the dialog being Replaced no longer exists). These credentials may, for example, need to be passed transitively or fetched in an event body.

To authorize call control primitives that trigger special behavior (such as an INVITE with Replaces or Join semantics), the receiving user agent may have trouble finding appropriate credentials with which to challenge or authorize the request, as the sender may be completely unknown to the receiver, except through the introduction of a third party. These credentials need to be passed transitively in some way or fetched in an event body, for example.

Standard SIP privacy and anonymity mechanisms such as [RFC3323] and [RFC3325] used during SIP session establishment apply equally well to SIP call control operations. SIP call control mechanisms should address privacy and anonymity issues associated with that operation. For example, privacy during a transfer operation using REFER is discussed in Section 7.2 of [RFC5589]

Appendix A. Example Features

Primitives are defined in terms of their ability to provide features. These example features should require an amply robust set of services to demonstrate a useful set of primitives. They are described here briefly. Note that the descriptions of these features are non-normative. Note also that this document describes a mixture of both features originating in the world of telephones and features that are clearly Internet oriented.

Appendix A.1. Attended Transfer

In Attended Transfer [RFC5589], the transferring party establishes a session with the transfer target before completing the transfer.

Appendix A.2. Auto Answer

In Auto Answer, calls to a certain address or URI answer immediately via a speakerphone. The Answer-Mode header field [RFC5373] can be used for this feature.

Appendix A.3. Automatic Callback

In Automatic Callback [RFC5359], Alice calls Bob, but Bob is busy. Alice would like Bob to call her automatically when he is available. When Bob hangs up, Alice's phone rings. When Alice answers, Bob's phone rings. Bob answers and they talk.

Appendix A.4. Barge-In

In Barge-in, Carol interrupts Alice who has an in-progress call with Bob. In some variations, Alice forcibly joins a new conversation with Carol, in other variations, all three parties are placed in the same conversation (basically a three-way conference). Barge-in works the same as call monitoring except that it must indicate that the send media stream be mixed so that all of the other parties can hear the stream from the UA that is barging in.

Appendix A.5. Blind Transfer

In Blind Transfer [RFC5589], Alice is in a conversation with Bob. Alice asks Bob to contact Carol, but makes no attempt to contact Carol independently. In many implementations, Alice does not verify Bob's success or failure in contacting Carol.

Appendix A.6. Call Forwarding

In call forwarding [RFC5359], before a dialog is accepted, it is redirected to another location, for example, because the originally intended recipient is busy, does not answer, is disconnected from the network, or has configured all requests to go elsewhere.

Appendix A.7. Call Monitoring

Call monitoring is a Join operation [RFC3911]. For example, a call center supervisor joins an in-progress call for monitoring purposes. The monitoring UA sends a Join to the dialog to which it wants to listen. It is able to discover the dialog via the dialog state on the monitored UA. The monitoring UA sends SDP in the INVITE that indicates receive-only media. As the UA is only monitoring, it does not matter whether the UA indicates it wishes the send stream to be mixed or point to point.

Appendix A.8. Call Park

In Call Park [RFC5359], a participant parks a call (essentially puts the call on hold), and then retrieves it at a later time (typically from another location). Call park requires the ability to put a dialog some place, advertise it to users in a pickup group, and to uniquely identify it in a means that can be communicated (including human voice). The dialog can be held locally on the UA parking the dialog or alternatively transferred to the park service for the pickup group. The parked dialog then needs to be labeled (e.g., orbit 12) in a way that can be communicated to the party that is to pick up the call. The UAs in the pickup group discover the parked dialog(s) via the dialog package from the park service. If the dialog is parked locally, the park service merely aggregates the parked call states from the set of UAs in the pickup group.

Appendix A.9. Call Pickup

There are two different features that are called Call Pickup [RFC5359]. The first is the pickup of a parked dialog. The UA from which the dialog is to be picked up subscribes to the dialog state of the park service or the UA that has locally parked the dialog. Dialogs that are parked should be labeled with an identifier. The labels are used by the UA to allow the user to indicate which dialog is to be picked up. The UA picking up the call invoked the URI in the call state that is labeled as replace-remote.

The other call pickup feature involves picking up an early dialog (typically ringing). A party picks up a call that was ringing at another location. One variation allows the caller to choose which

location, another variation just picks up any call in that user's "pickup group". This feature uses some of the same primitives as the pickup of a parked call. The call state of the UA ringing phone is advertised using the dialog package. The UA that is to pick up the early dialog subscribes either directly to the ringing UA or to a service aggregating the states for UAs in the pickup group. The call state identifies early dialogs. The UA uses the call state(s) to help the user choose which early dialog is to be picked up. The UA then invokes the URI in the call state labeled as replace-remote.

Appendix A.10. Call Return

In Call Return, Alice calls Bob. Bob misses the call or is disconnected before he is finished talking to Alice. Bob invokes Call return, which calls Alice, even if Alice did not provide her real identity or location to Bob.

Appendix A.11. Call Waiting

In Call Waiting, Alice is in a call, then receives another call. Alice can place the first call on hold, and talk with the other caller. She can typically switch back and forth between the callers.

Appendix A.12. Click-to-Dial

In Click-to-Dial [RFC5359], Alice looks in her company directory for Bob. When she finds Bob, she clicks on a URI to call him. Her phone rings (or possibly answers automatically), and when she answers, Bob's phone rings. The application or server that hosts the Click-to-Dial application captures the URI to be dialed and can set up the call using 3pcc or can send a REFER request to the UA that is to dial the address. As users sometimes change their mind or wish to give up listing to a ringing or voicemail answered phone, this application illustrates the need to also have the ability to remotely hangup a call.

Appendix A.13. Conference Call

In a Conference Call [RFC4579], there are three or more active, visible participants in the same conversation space.

Appendix A.14. Consultative Transfer

In Consultative Transfer [RFC5589], the transferring party establishes a session with the target and mixes both sessions together so that all three parties can participate, then disconnects leaving the transferee and transfer target with an active session.

Appendix A.15. Distinctive Ring

In Distinctive Ring, incoming calls have different ring cadences or sample sounds depending on the From party, the To party, or other factors. The target UA either makes a local decision based on information in an incoming INVITE (To, From, Contact, Request-URI) or trusts an Alert-Info header field [RFC3261] provided by the caller or inserted by a trusted proxy. In the latter case, the UA fetches the content described in the URI (typically via HTTP) and renders it to the user.

Appendix A.16. Do Not Disturb

In Do Not Disturb, Alice selects the Do Not Disturb option. Calls to her either ring briefly or not at all and are forwarded elsewhere. Some variations allow specially authorized callers to override this feature and ring Alice anyway. Do Not Disturb is best implemented in SIP using presence [RFC3856].

Appendix A.17. Find-Me

In Find-Me, Alice sets up complicated rules for how she can be reached (possibly using CPL (Call Processing Language) [RFC3880], presence [RFC3856], or other factors). When Bob calls Alice, his call is eventually routed to a temporary Contact where Alice happens to be available.

Appendix A.18. Hotline

In Hotline, Alice picks up a phone and is immediately connected to the technical support hotline, for example. Hotline is also sometimes known as a Ringdown line.

Appendix A.19. IM Conference Alerts

In IM Conference Alerts, a user receives a notification as an instant message whenever someone joins a conference in which they are already a participant.

Appendix A.20. Inbound Call Screening

In Inbound Call Screening, Alice doesn't want to receive calls from Matt. Inbound Screening prevents Matt from disturbing Alice. In some variations, this works even if Matt hides his identity.

Appendix A.21. Intercom

In Intercom, Alice typically presses a button on a phone that immediately connects to another user or phone and causes that phone to play her voice over its speaker. Some variations immediately set up two-way communications, other variations require another button to be pressed to enable a two-way conversation. The UA initiates a dialog using INVITE and the Answer-Mode: Auto header field as described in [RFC5373]. The called UA accepts the INVITE with a 200 OK and automatically enables the speakerphone.

Alternatively, this can be a local decision for the UA to auto answer based upon called-party identification.

Appendix A.22. Message Waiting

In Message Waiting [RFC3842], Bob calls Alice when she has stepped away from her phone. When she returns, a visible or audible indicator conveys that someone has left her a voicemail message. The message waiting indication may also convey how many messages are waiting, from whom, at what time, and other useful pieces of information.

Appendix A.23. Music on Hold

In Music on Hold [RFC5359], when Alice places a call with Bob on hold, it replaces its audio with streaming content such as music, announcements, or advertisements. Music on hold can be implemented a number of ways. One way is to transfer the held call to a holding service. When the UA wishes to take the call off hold, it basically performs a take on the call from the holding service. This involves subscribing to call state on the holding service and then invoking the URI in the call state labeled as replace-remote.

Alternatively, music on hold can be performed as a local mixing operation. The UA holding the call can mix in the music from the music service via RTP (i.e., an additional dialog) or RTSP or other streaming media source. This approach is simpler (i.e., the held dialog does not move so there is less chance of loosing them) from a protocol perspective, however it does use more LAN bandwidth and resources on the UA.

Appendix A.24. Outbound Call Screening

In Outbound Call Screening, Alice is paged and unknowingly calls a PSTN pay-service telephone number in the Caribbean, but local policy blocks her call, and possibly informs her why.

Appendix A.25. Pre-Paid Calling

In Pre-paid Calling, Alice pays for a certain currency or unit amount of calling value. When she places a call, she provides her account number somehow. If her account runs out of calling value during a call, her call is disconnected or redirected to a service where she can purchase more calling value.

For prepaid calling, the user's media always passes through a device that is trusted by the pre-paid provider. This may be the other endpoint (for example, a PSTN gateway). In either case, an intermediary proxy or B2BUA can periodically verify the amount of time available on the pre-paid account, and use the session-timer extension to cause the trusted endpoint (gateway) or intermediary (media relay) to send a reINVITE before that time runs out. During the reINVITE, the SIP intermediary can re-verify the account and insert another session-timer header field.

Note that while most pre-paid systems on the PSTN use an IVR to collect the account number and destination, this isn't strictly necessary for a SIP-originated prepaid call. SIP requests and SIP URIs are sufficiently expressive to convey the final destination, the provider of the prepaid service, the location from which the user is calling, and the prepaid account they want to use. If a pre-paid IVR is used, the mechanism described below (Voice Portals) can be combined as well.

Appendix A.26. Presence-Enabled Conferencing

In Presence-Enabled Conferencing, Alice wants to set up a conference call with Bob and Cathy when they all happen to be available (rather than scheduling a predefined time). The server providing the application monitors their status, and calls all three when they are all "online", not idle, and not in another call. This could be implemented using conferencing [RFC4579] and presence [RFC3264] primitives.

Appendix A.27. Single Line Extension/Multiple Line Appearance

In Single Line Extension/Multiple Line Appearances, groups of phones are all treated as "extensions" of a single line or AOR. A call for one rings them all. As soon as one answers, the others stop ringing. If any extension is actively in a conversation, another extension can "pick up" and immediately join the conversation. This emulates the behavior of a home telephone line with multiple phones. Incoming calls ring all the extensions through basic parallel forking. Each extension subscribes to dialog events from each other extension. While one user has an active call, any other UA extension can insert

itself into that conversation (it already knows the dialog information) in the same way as barge-in.

When implemented using SIP, this feature is known as Shared Appearances of an AOR [BLISS-SHARED]. Extensions to the dialog package are used to convey appearance numbers (line numbers).

Appendix A.28. Speakerphone Paging

In Speakerphone Paging, Alice calls the paging address and speaks. Her voice is played on the speaker of every idle phone in a preconfigured group of phones. Speakerphone paging can be implemented using either multicast or through a simple multipoint mixer. In the multicast solution, the paging UA sends a multicast INVITE with send-only media in the SDP (see also [RFC3264]). The automatic answer and enabling of the speakerphone is a locally configured decision on the paged UAs. The paging UA sends RTP via the multicast address indicated in the SDP.

The multipoint solution is accomplished by sending an INVITE to the multipoint mixer. The mixer is configured to automatically answer the dialog. The paging UA then sends REFER requests for each of the UAs that are to become paging speakers (the UA is likely to send out a single REFER that is parallel forked by the proxy server). The UAs performing as paging speakers are configured to automatically answer based upon caller identification (e.g., the To field, URI, or Referred-To header fields).

Finally, as a third option, the user agent can send a mass-invitation request to a conference server, which would create a conference and send INVITEs containing the Answer-Mode: Auto header field to all user agents in the paging group.

Appendix A.29. Speed Dial

In Speed Dial, Alice dials an abbreviated number, enters an alias, or presses a special speed-dial button representing Bob. Her action is interpreted as if she specified the full address of Bob.

Appendix A.30. Voice Message Screening

In Voice Message Screening, Bob calls Alice. Alice is screening her calls, so Bob hears Alice's voicemail greeting. Alice can hear Bob leave his message. If she decides to talk to Bob, she can take the call back from the voicemail system; otherwise, she can let Bob leave a message. This emulates the behavior of a home telephone answering machine.

At first, this is the same as Call Monitoring (Appendix A.7). In this case, the voicemail service is one of the UAs. The UA screening the message monitors the call on the voicemail service, and also subscribes to dialog information. If the user screening their messages decides to answer, they perform a take from the voicemail system (for example, send an INVITE with Replaces to the UA leaving the message).

Appendix A.31. Voice Portal

Voice Portal is service that allows users to access a portal site using spoken dialog interaction. For example, Alice needs to schedule a working dinner with her co-worker Carol. Alice uses a voice portal to check Carol's flight schedule, find a restaurant near her hotel, make a reservation, get directions there, and page Carol with this information. A voice portal is essentially a complex collection of voice dialogs used to access interesting content. One of the most desirable call control features of a Voice Portal is the ability to start a new outgoing call from within the context of the Portal (to make a restaurant reservation, or return a voicemail message, for example). Once the new call is over, the user should be able to return to the Portal by pressing a special key, using some DTMF sequence (e.g., a very long pound or hash tone), or by speaking a key word (e.g., "Main Menu").

In order to accomplish this, the Voice Portal starts with the following media relationship:

{ User , Voice Portal }

The user then asks to make an outgoing call. The Voice Portal asks the user to perform a far-fork. In other words, the Voice Portal wants the following media relationship:

{ Target , User } & { User , Voice Portal }

The Voice Portal is now just listening for a key word or the appropriate DTMF. As soon as the user indicates they are done, the Voice Portal takes the call from the old target, and we are back to the original media relationship.

This feature can also be used by the account number and phone number collection menu in a pre-paid calling service. A user can press a DTMF sequence that presents them with the appropriate menu again.

Appendix A.32. Voicemail

In Voicemail, Alice calls Bob who does not answer or is not available. The call forwards to a voicemail server that plays Bob's greeting and records Alice's message for Bob. An indication is sent to Bob that a new message is waiting, and he retrieves the message at a later date. This feature is implemented using features such as Call Forwarding (Appendix A.6) and the History-Info header field [RFC4244] or voicemail URI convention [RFC4458] and Message Waiting [RFC3842] features.

Appendix A.33. Whispered Call Waiting

In Whispered Call Waiting, Alice is in a conversation with Bob. Carol calls Alice. Either Carol can "whisper" to Alice directly ("Can you get lunch in 15 minutes?"), or an automaton whispers to Alice informing her that Carol is trying to reach her.

Appendix B. Acknowledgments

The authors would like to acknowledge Ben Campbell for his contributions to the document and thank AC Mahendran, John Elwell, and Xavier Marjou for their detailed Working-Group review of the document. The authors would like to thank Magnus Nystrom for his review of the document.

5. Informative References

[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.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC5359] Johnston, A., Sparks, R., Cunningham, C., Donovan, S., and K. Summers, "Session Initiation Protocol Service Examples", BCP 144, RFC 5359, October 2008.

[RFC3725] Rosenberg, J., Peterson, J., Schulzrinne, H., and G. Camarillo, "Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)", BCP 85, RFC 3725, April 2004.

[RFC3515] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[RFC3891] Mahy, R., Biggs, B., and R. Dean, "The Session Initiation Protocol (SIP) "Replaces" Header", RFC 3891, September 2004.

[RFC3911] Mahy, R. and D. Petrie, "The Session Initiation Protocol (SIP) "Join" Header", RFC 3911, October 2004.

[BLISS-PROBLEM] Rosenberg, J., "Basic Level of Interoperability for Session Initiation Protocol (SIP) Services (BLISS) Problem Statement", Work in Progress, March 2009.

[RFC4235] Rosenberg, J., Schulzrinne, H., and R. Mahy, "An INVITE-Initiated Dialog Event Package for the Session Initiation Protocol (SIP)", RFC 4235, November 2005.

[RFC4575] Rosenberg, J., Schulzrinne, H., and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State", RFC 4575, August 2006.

[RFC3680] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", RFC 3680, March 2004.

[RFC3856] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.

[RFC4353] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)", RFC 4353, February 2006.

[RFC5629] Rosenberg, J., "A Framework for Application Interaction in the Session Initiation Protocol (SIP)", RFC 5629, October 2009.

[RFC5369] Camarillo, G., "Framework for Transcoding with the Session Initiation Protocol (SIP)", RFC 5369, October 2008.

[XCON-CCMP] Barnes, M., Boulton, C., Romano, S., and H. Schulzrinne, "Centralized Conferencing Manipulation Protocol", Work in Progress, February 2010.

[RFC5589] Sparks, R., Johnston, A., and D. Petrie, "Session Initiation Protocol (SIP) Call Control - Transfer", BCP 149, RFC 5589, June 2009.

[RFC4579] Johnston, A. and O. Levin, "Session Initiation Protocol (SIP) Call Control - Conferencing for User Agents", BCP 119, RFC 4579, August 2006.

[RFC3840] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[RFC3841] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", RFC 3841, August 2004.

[RFC3087] Campbell, B. and R. Sparks, "Control of Service Context using SIP Request-URI", RFC 3087, April 2001.

[FEATURE-REF] Audet, F., Johnston, A., Mahy, R., and C. Jennings, "Feature Referral in the Session Initiation Protocol (SIP)", Work in Progress, February 2008.

[RFC4240] Burger, E., Van Dyke, J., and A. Spitzer, "Basic Network Media Services with SIP", RFC 4240, December 2005.

[RFC4458] Jennings, C., Audet, F., and J. Elwell, "Session Initiation Protocol (SIP) URIs for Applications such as Voicemail and Interactive Voice Response (IVR)", RFC 4458, April 2006.

[RFC4538] Rosenberg, J., "Request Authorization through Dialog Identification in the Session Initiation Protocol (SIP)", RFC 4538, June 2006.

[RFC3880] Lennox, J., Wu, X., and H. Schulzrinne, "Call Processing Language (CPL): A Language for User Control of Internet Telephony Services", RFC 3880, October 2004.

[RFC5373] Willis, D. and A. Allen, "Requesting Answering Modes for the Session Initiation Protocol (SIP)", RFC 5373, November 2008.

[RFC3842] Mahy, R., "A Message Summary and Message Waiting Indication Event Package for the Session Initiation Protocol (SIP)", RFC 3842, August 2004.

[BLISS-SHARED] Johnston, A., Soroushnejad, M., and V. Venkataramanan, "Shared Appearances of a Session Initiation Protocol (SIP) Address of Record (AOR)", Work in Progress, October 2009.

[RFC4244] Barnes, M., "An Extension to the Session Initiation Protocol (SIP) for Request History Information", RFC 4244, November 2005.

[RFC4313] Oran, D., "Requirements for Distributed Control of Automatic Speech Recognition (ASR), Speaker Identification/Speaker Verification (SI/SV), and Text-to-Speech (TTS) Resources", RFC 4313, December 2005.

[RFC3323] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002.

[RFC3325] 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.

Authors' Addresses

Rohan Mahy
Unaffiliated

EMail: rohan@ekabal.com

Robert Sparks
Tekelec

EMail: rjsparks@nostrum.com

Jonathan Rosenberg
jdrosen.net

EMail: jdrosen@jdrosen.net

Dan Petrie
SIPez

EMail: dan.ietf@sipez.com

Alan Johnston (editor)
Avaya

EMail: alan.b.johnston@gmail.com