SIPPING
Internet-Draft
Expires: January 17, 2005

J. Rosenberg
dynamicsoft

July 19, 2004

# A Framework for Application Interaction in the Session Initiation Protocol (SIP)
## draft-ietf-sipping-app-interaction-framework-02

Status of this Memo

By submitting this Internet-Draft, I certify that any applicable patent or other IPR claims of which I am aware have been disclosed, and any of which I become aware will be disclosed, in accordance with RFC 3668.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on January 17, 2005.

Copyright Notice

Copyright (C) The Internet Society (2004). All Rights Reserved.

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 . . . . . 4
2. Definitions . . . . . 5
3. A Model for Application Interaction . . . . . 8
3.1 Functional vs. Stimulus . . . . . 9
3.2 Real-Time vs. Non-Real Time . . . . . 10
3.3 Client-Local vs. Client-Remote . . . . . 10
3.4 Presentation Capable vs. Presentation Free . . . . . 11
4. Interaction Scenarios on Telephones . . . . . 13
4.1 Client Remote . . . . . 13
4.2 Client Local . . . . . 13
4.3 Flip-Flop . . . . . 14
5. Framework Overview . . . . . 16
6. Deployment Topologies . . . . . 19
6.1 Third Party Application . . . . . 19
6.2 Co-Resident Application . . . . . 19
6.3 Third Party Application and User Device Proxy . . . . . 20
6.4 Proxy Application . . . . . 22
7. Application Behavior . . . . . 23
7.1 Client Local Interfaces . . . . . 23
7.1.1 Discovering Capabilities . . . . . 23
7.1.2 Pushing an Initial Interface Component . . . . . 23
7.1.3 Updating an Interface Component . . . . . 25
7.1.4 Terminating an Interface Component . . . . . 26
7.2 Client Remote Interfaces . . . . . 26
7.2.1 Originating and Terminating Applications . . . . . 27
7.2.2 Intermediary Applications . . . . . 27
8. User Agent Behavior . . . . . 28
8.1 Advertising Capabilities . . . . . 28
8.2 Receiving User Interface Components . . . . . 28
8.3 Mapping User Input to User Interface Components . . . . . 30
8.4 Receiving Updates to User Interface Components . . . . . 30
8.5 Terminating a User Interface Component . . . . . 31
9. Inter-Application Feature Interaction . . . . . 32
9.1 Client Local UI . . . . . 32
9.2 Client-Remote UI . . . . . 33
10. Intra Application Feature Interaction . . . . . 34
11. Example Call Flow . . . . . 35
12. Security Considerations . . . . . 40
13. IANA Considerations . . . . . 41
13.1 SIP Option Tag . . . . . 41
13.2 Header Field Parameter . . . . . 41

14. Contributors . . . . . 42
15. References . . . . . 43
15.1 Normative References . . . . . 43
15.2 Informative References . . . . . 43
Author's Address . . . . . 44
Intellectual Property and Copyright Statements . . . . . 45

## 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 [11] 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 device 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 user device. 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 [11] 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 Device Proxy: A software or hardware system that a user indirectly interacts through in order to communicate with the application. This indirection can be through a network. An example is a gateway from IP to the Public Switched Telephone Network (PSTN). It acts a user device proxy, acting on behalf of the user on the circuit network.

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 [13].

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, making a call on behalf of 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

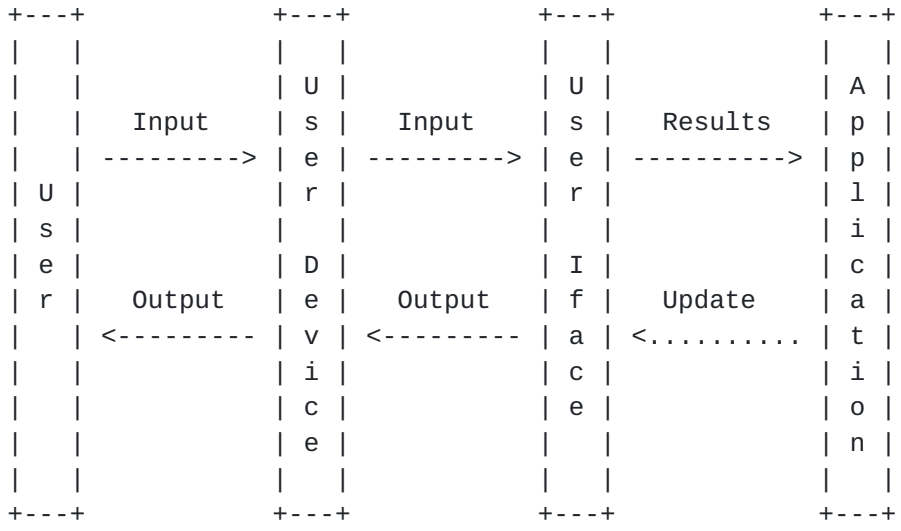



Figure 1: Model for Real-Time Interactions

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 as a user interface event to the application. The application may be able to modify the user interface based on this event. 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 as a meaningful event, 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 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 [14] 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 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 [4], 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.

## 4. Interaction Scenarios on Telephones

In this section, we applied the model of Section 3 to telephones.

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 Device 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.

### 4.1 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) [16]. 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 [17], 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.

### 4.2 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.

For simpler phones without a display, the user interface can be described by a Keypad Markup Language request document [7]. As the user enters digits in the keypad, they are passed to the user interface, which generates user interface events that can be transported to the application.

### 4.3 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 [7]. 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.

## 5. 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 or 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) [7].

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 [6] 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.

## 6. Deployment Topologies

This section presents some of the network topologies in which this framework can be instantiated.

### 6.1 Third Party Application

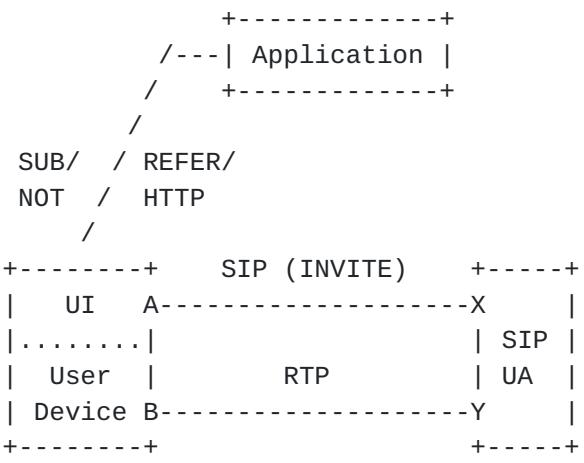



Figure 2: Third Party Topology

In this topology, the application that is interested in interacting with the users exists outside of the SIP dialog between the user agents. In that case, the application learns about the initiation and termination of the dialog, along with the dialog identifiers, through some out of band means. One such possibility is the dialog event package [15]. Dialog information is only revealed to trusted parties, so the application would need to be trusted by one of the users in order to obtain this information.

At any point during the dialog, the application can instantiate user interface components on the user device of the caller or callee. It can do this either using SUBSCRIBE or REFER, depending on the type of user interface (presentation capable or presentation free).

### 6.2 Co-Resident Application

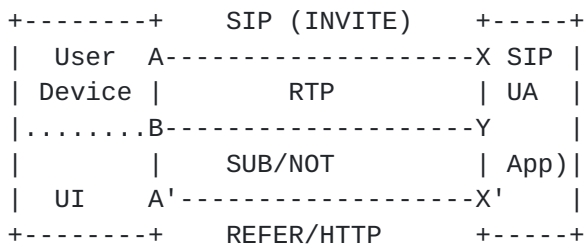



Figure 3: Co-Resident Topology

In this deployment topology, the application is co-resident with one of the user agents (the one on the right in the picture above). This application can install client-local user interface components on the other user agent, which is acting as the user device. These components can be installed using either SUBSCRIBE, for presentation free user interfaces, or REFER, for presentation capable ones.

If the application resides in the called party, it is called a terminating application. If it resides in the calling party, it is called an originating application.

This kind of topology is common in protocol converter and gateway applications.

## 6.3 Third Party Application and User Device Proxy

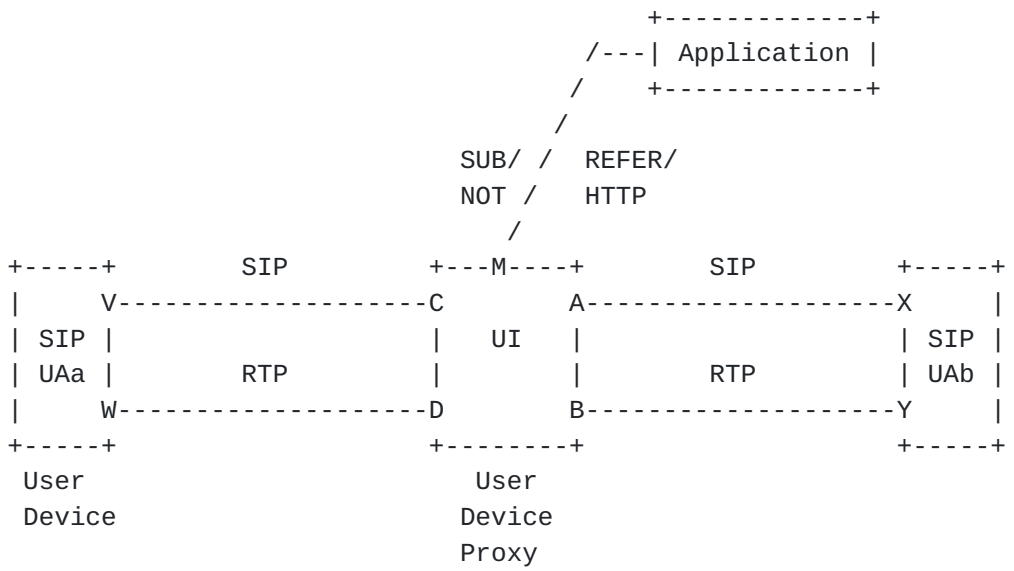



Figure 4: User Device Proxy Topology

In this deployment topology, there is a third party application as in

Section 6.1. However, instead of installing a user interface component on the end user device, the component is installed in an intermediate device, known as a User Device Proxy. From the perspective of the actual user device (on the left), the User Device Proxy is a client remote user interface. As such, media, typically transported using RTP (including RFC 2833 for carrying user input), is sent from the user device to the client remote user interface on the User Device Proxy. As far as the application is concerned, it is installing what it thinks is a client local user interface on the user device, but it happens to be on a user device proxy which looks like the user device to the application.

The user device proxy will need to terminate and re-originate both signaling (SIP) and media traffic towards the actual peer in the conversation. The User Device Proxy is a media relay in the terminology of RFC 3550 [16]. The User Device Proxy will need to monitor the media streams associated with each dialog, in order to convert user input received in the media stream to events reported to the user interface. This can pose a challenge in multi-media systems, where it may be unclear on which media stream the user input is being sent. As discussed in RFC 3264 [18], if a user agent has a single media source and is supporting multiple streams, it is supposed to send that source to all streams. In cases where there are multiple sources, the mapping is a matter of local policy. In the absence of a way to explicitly identify or request which sources map to which streams, the user device proxy will need to do the best job it can. This specification RECOMMENDS that the User Device Proxy monitor the first stream (defined in terms of ordering of media sessions within a session description). As such, user agents SHOULD send their user input on the first stream, absent a policy to direct it otherwise.

### 6.4 Proxy Application

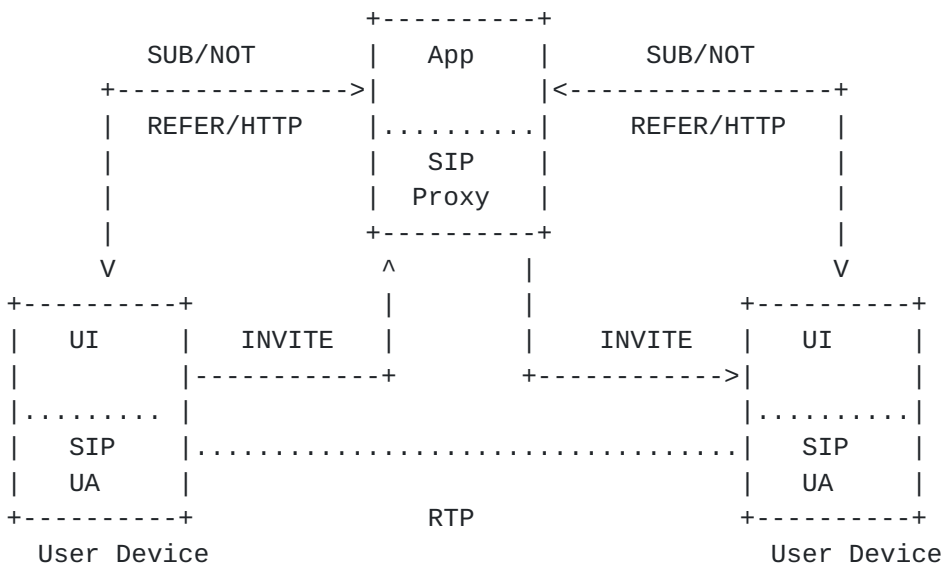



Figure 5: Proxy Application Topology

In this topology, the application is co-resident with a transaction stateful, record-routing proxy server on the call path between two user devices. The application uses SUBSCRIBE or REFER to install user interface components on one or both user devices.

This topology is common in routing applications, such as a web-assisted call routing application.

## 7. 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.

### 7.1 Client Local Interfaces

One key component of this framework is support for client local user interfaces.

#### 7.1.1 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 [7], and the Supported header field indicates that its Contact URI is a GRUU [8], 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 7.1.2. 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, the Supported header field indicates support for the "refer-context" extension described below, and the Contact header field contains UA capabilities [5] 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 7.1.2. The specific markups that are supported are indicated in the Accept header field.

A third party application that is not present on the call path will not be privy to these headers in the dialog requests that pass by. As such, it will need to obtain this capability information in other ways. One way is through the registration event package [19], which can contain user agent capability information provided in REGISTER requests [5].

#### 7.1.2 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 request confirming the receipt of a dialog-creating response. This could be an ACK for a 200 OK, or a PRACK for a provisional response [2]. 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 dialog-creating response 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) [8] 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.

Furthermore, it is essential for the REFER request to be correlated with the dialog to which the user interface component will be associated. This is necessary for authorization and for terminating the user interface components when the dialog terminates. To provide this context, this specification defines the "context" header field parameter as an extension to the Refer-To heder field. The grammar for this header field parameter is:

```
refer-to-ctxt     = "context" EQUAL DQUOTE local-tag "," remote-tag
                    "," callid DQUOTE    ; callid defined in RFC 3261
                    ;; NOTE: any DQUOTEs inside callid MUST be escaped
                    ;; using quoted pair
local-tag         = token
remote-tag        = token

Refer-To          = ("Refer-To" / "r") HCOLON ( name-addr / addr-spec ) *
     (SEMI (generic-param / refer-to-ctxt))
```

The application MUST include the context header field parameter in the REFER request. The remote-tag MUST be set to the remote tag of the dialog as seen by the user device. The local-tag MUST be set to the local tag of the dialog as seen by the user device. The callid MUST be set to the Call-ID of the dialog as seen by the device. Since the callid grammar allows it to contain double quotes, any such double quotes MUST be represented with a quoted pair.

Since the "context" parameter in the Refer-To header field must be understood by the UA to process the request, this specification defines a new SIP option tag, "refer-context". A REFER request generated by an application MUST include a Require header field with this option tag value. Fortunately, the application will know ahead of time whether this extension is supported, as discussed in Section 7.1.1.

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 [7] 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>
```

Any of the SIP identity assertion mechanisms that have been defined, such as [10] and [12] are applicable to these requests as well.

### 7.1.3 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.

### 7.1.4 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 capable 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 application 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".

## 7.2 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.

### 7.2.1 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. It is 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.

### 7.2.2 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 9.

# 8. User Agent Behavior

## 8.1 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, along with the "context" extension defined here. It MUST include, in all dialog initiating requests and responses, an Allow header field that includes the REFER method and and the Supported header field that includes the value "refer-context". Furthermore, the UA MUST support the SIP user agent capabilities specification [5]. 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 [3] method. It MUST support the KPML [7] 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-request+xml" MIME type.

For either presentation free or presentation capable user interfaces, the user agent MUST support the GRUU [8] 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.

## 8.2 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, authenticated identities can be obtained through other means, such as [10].

A user agent MAY have pre-defined authorization policies which permit applications which have authenticated themselves with a particular identity, to push user interface components. If such a set of policies are present, it is checked first. If the application is authorized, processing proceeds.

If the application has authenticated itself, but it is not explicitly authorized or blocked, this specification RECOMMENDS that the application be automatically authorized if it can prove that it was either on the call path, or is trusted by one of the elements on the call path. An application proves this to the user agent 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 [7]. In the case of REFER, those identifiers are present in the "context" parameter of the Refer-To header field.

Because of the dialog identifiers serve as a tool for authorization, a user agent compliant to this framework SHOULD 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 or trusted by 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 the dialog was not established with a sips URI, or the user agent did not choose cryptographically random dialog identifiers, then the application MUST NOT automatically be authorized, even if it presented valid dialog identifiers. 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 and the dialog it is associated with. 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 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.

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.

## 8.3 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.

Otherwise, for client local user interfaces, the user input is assumed to be associated with all user interface components. For client remote user interfaces, the user device converts the user input to media, typically conveyed using RFC 2833, and sends this to the client remote user interface. This user interface then needs to map user input from potentially many media streams into user interface events. The process for doing this is described in Section 6.3.

## 8.4 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.

### 8.5 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.

## 9. 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.

### 9.1 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 [17]. 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 KPML request document is used to suppress.

### 9.2 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.

## 10. 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.

## 11. 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.

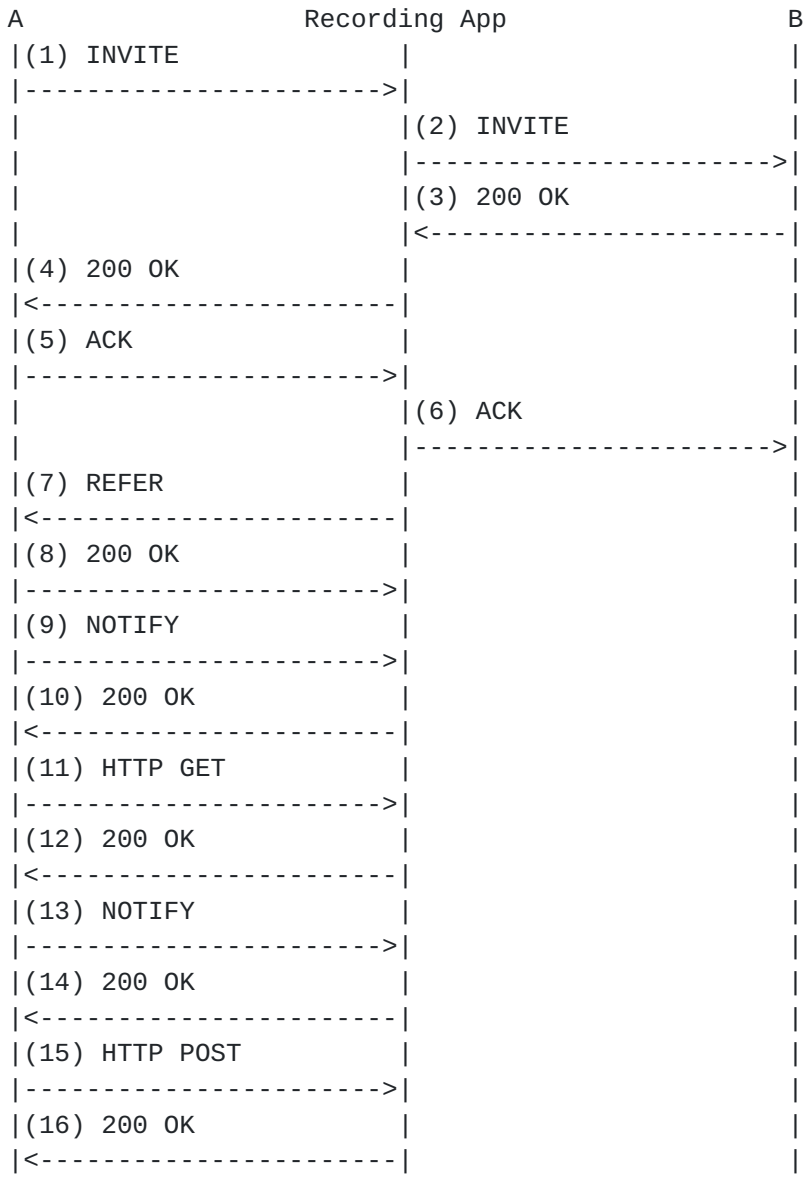



Figure 8

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 that the GRUU extension and the REFER context extension are understood, Allow indicating that REFER is understood, and a Contact header field that includes the "schemes" header field parameter.

```
INVITE sips:B@example.com SIP/2.0
Via: SIP/2.0/TLS host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>
Call-ID: faif9ahhs9dd8==-sd98ajzz@host.example.com
CSeq: 1 INVITE
Max-Forwards: 70
Supported: gruu, refer-context
Allow: INVITE, OPTIONS, BYE, CANCEL, ACK, REFER
Contact: <sips:bad998asd8asd0000a@example.com>;schemes="http,sip,sips"
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 sips:B@pc.example.com SIP/2.0
Record-Route: <sips:app.example.com;lr>
Via: SIP/2.0/TLS app.example.com;branch=z9hG4bK97sh
Via: SIP/2.0/TLS host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>
Call-ID: faif9ahhs9dd8==-sd98ajzz@host.example.com
CSeq: 1 INVITE
Max-Forwards: 69
Supported: gruu, refer-context
Allow: INVITE, OPTIONS, BYE, CANCEL, ACK, REFER
Contact: <sips:bad998asd8asd0000a@example.com>;schemes="http,sip,sips"
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: <ssip:app.example.com;lr>
Via: SIP/2.0/TLS app.example.com;branch=z9hG4bK97sh
Via: SIP/2.0/TLS host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>;tag=7777
Call-ID: faif9ahhs9dd8==-sd98ajzz@host.example.com
CSeq: 1 INVITE
Contact: <sips: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: <sips:app.example.com;lr>
Via: SIP/2.0/TLS host.example.com;branch=z9hG4bK9zz8
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>;tag=7777
Call-ID: faif9ahhs9dd8==-sd98ajzz@host.example.com
CSeq: 1 INVITE
Contact: <sips:B@pc.example.com>
Content-Length: ...
Content-Type: application/sdp

--SDP not shown--
```

The caller generates an ACK (message 5).

```
ACK sips:B@pc.example.com
Route: <sips:app.example.com;lr>
Via: SIP/2.0/TLS host.example.com;branch=z9hG4bK9zz9
From: Caller <sip:A@example.com>;tag=kkaz-
To: Callee <sip:B@example.com>;tag=7777
Call-ID: faif9ahhs9dd8==-sd98ajzz@host.example.com
CSeq: 1 ACK
```

The ACK is forwarded to the called party (message 6).

```
ACK sips:B@pc.example.com
Via: SIP/2.0/TLS app.example.com;branch=z9hG4bKh7s
Via: SIP/2.0/TLS host.example.com;branch=z9hG4bK9zz9
From: Caller <sip:A@example.com>;tag=kkaz-
```

```
To: Callee <sip:B@example.com>;tag=7777
Call-ID: faif9ahhs9dd8==-sd98ajzz@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 sips:bad998asd8asd0000a@example.com SIP/2.0
Refer-To: https://app.example.com/script.pl
 ;context="kkaz-,7777,faif9ahhs9dd8==-sd98ajzz@host.example.com"
Via: SIP/2.0/TLS app.example.com;branch=z9hG4bK9zh6
Max-Forwards: 70
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: <sips:app.example.com>
```

The REFER is answered by a 200 OK (message 8).

```
SIP/2.0 200 OK
Via: SIP/2.0/TLS 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, refer-context
Allow: INVITE, OPTIONS, BYE, CANCEL, ACK, REFER
Contact: <sips:bad998asd8asd0000a@example.com>;schemes="http,sip,sips"
CSeq: 1 REFER
```

User A sends a NOTIFY (message 9):

```
NOTIFY sips:app.example.com SIP/2.0
Via: SIP/2.0/TLS 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: <sips:bad998asd8asd0000a@example.com>;schemes="http,sip,sips"
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/TLS 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 REFER request contained a "context" Refer-To header field parameter with a valid dialog identifier. Furthermore, all of the signaling was over TLS and the dialog identifiers contain sufficient randomness. As such, the caller, A, automatically 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. Because the client executed the URL in the Refer-To, it generates another NOTIFY to the application, informing it of the successful response (message 13). This is answered with a 200 OK (message 14). When the user clicks on the link (message 15), the results are posted to the server, and an updated display is provided (message 16).

## 12. 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.

This specification defines specific authorization techniques and requirements. Automatic authorization is granted if the application can prove that it is on the call path, or is trusted by an element on the call path. As documented above, this can be accompished by the use of cryptographically random dialog identifiers and the usage of sips for message confidentiality. It is RECOMMENDED that sips be implemented by user agents compliant to this specification. This does not represent a change from the requirements in RFC 3261.

## 13. IANA Considerations

### 13.1 SIP Option Tag

This specification registers a new SIP option tag, as per the guidelines in Section 27.1 of RFC 3261 [1].
Name: refer-context
Description: This option tag is used to identify the REFER extension that defines the "context" parameter of the Refer-To header field.

### 13.2 Header Field Parameter

This specification defines a new header field parameter, as per the registry created by [9]. The required information is as follows:
Header field in which the parameter can appear: Refer-To
Name of the Parameter context
RFC Reference RFC XXXX [[NOTE TO IANA: Please replace XXXX with the RFC number of this specification.]]

## 14. 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

## 15. References

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

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

[4] 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.

[5] Rosenberg, J., "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", draft-ietf-sip-callee-caps-03 (work in progress), January 2004.

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

[7] Burger, E., "A Session Initiation Protocol (SIP) Event Package for Key Press Stimulus (KPML)", draft-ietf-sipping-kpml-03 (work in progress), May 2004.

[8] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-01 (work in progress), February 2004.

[9] Camarillo, G., "The Internet Assigned Number Authority (IANA) Header Field Parameter Registry for the Session Initiation Protocol (SIP)", draft-ietf-sip-parameter-registry-02 (work in progress), June 2004.

### 15.2 Informative References

[10] Peterson, J., "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-02 (work in progress), May 2004.

[11] Day, M., Rosenberg, J. and H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000.

[12] 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.

[13] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-01 (work in progress), October 2003.

[14] 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.

[15] Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-04 (work in progress), February 2004.

[16] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

[17] Schulzrinne, H. and S. Petrack, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals", RFC 2833, May 2000.

[18] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

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

## 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

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights 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; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat 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 implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

## Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Copyright Statement

Copyright (C) The Internet Society (2004). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

## Acknowledgment

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