Project : Interactive Voice and Video in Browsers

This page last changed on Sep 19, 2010 by fluffy.

By David Mathieson, Matthew Penning, Niall McDonnell, John O'Reilly, Joe Hildebrand

Abstract

This document reflects early work to help design web standards for supporting interactive communications from inside web browsers.

1. Introduction

Audio and video is becoming more and more part of the browser, but to enable Unified Communications there are still some elements that need to be added. Below we list some of the Unified Communications use cases to help illustrate the current needs, and follow that up with proposals on how this could be accomplished.

---

2. Use Cases

2.1 Browser WebPhone

2.1.1 Use case: Call control

A browser WebPhone should have the ability to establish and control calls with other IP endpoints. Those IP endpoints include phones, soft clients, gateways and IP enabled PBXs. Call control should include the ability to make and receive calls; terminate, hold and resume calls; transfer calls; participate in conference calls.

These call control abilities should not require a plug-in, nor should they require additional ports to be opened on the client device.

2.1.2 Use case: Open media stream

A browser WebPhone should be able to open a secured or unsecured two-way media stream to another phone device. This should not require a browser plug-in.

2.1.3 Use case: Incoming call event

A browser WebPhone should be able to receive call control events in real-time. Such events might include new incoming call, call held, call ended, call transfer requested etc.

Real-time call control events should reliably pass through firewalls and proxy servers and should not require a browser plug-in or additional ports.

2.1.4 Use case: Visual notification of incoming call

Incoming call notifications should be visible to the user, even if the browser window is minimized. A user must be able to work in different applications without fear of missing incoming calls.

2.1.5 Use case: Cross tab co-ordination, cross-process co-ordination

Modern web applications are often spread across multiple tabs of a browser. A browser webphone that is part of a multi-tab web application should behave as a single phone with each tab behaving as a separate line.

2.1.5.1 Call control

A WebPhone user should have full call control capabilities on each tab of the same application - hold/ resume, conference, transfer, end call etc.

Incoming call notifications should come into each tab of a web application.

2.1.5.2 Multiple Tabs of the same application = Multiple lines

If, while on a call in the first tab of an application, a user takes a call in a second tab, the call on the first tab should automatically go on hold.

If the user resumes the call on the first tab, the active call on the second tab should automatically go on hold.

The author of a web application that uses a browser Web Phone should not need to write the logic to manage a phone across multiple tabs.

2.1.5.3 Security

A browser WebPhone instance should not be shared between different web applications. The browser should recognize the distinction between two pages of the same web application and pages of different web applications.

2.1.5.4 Cross process

CTI applications are common in businesses. For example, a call center might application might automatically route a call to the next available agent.

A browser WebPhone should expose a secure CTI interface so that other applications, including other web applications, can control it.

2.1.6 Use case: Device selection

A browser WebPhone user should be able to select microphone, speakers and camera from the list of supported devices on the machine.

2.1.7 Use case: Unattended control

A browser WebPhone user will often work in another application while on a phone or video call. The user must be able to quickly access call control features such as hold, mute and end call, even when not focused on the WebPhone application.

One suggestion is that the WebPhone be able to supply control from the system tray.

2.1.8 Use case: No UI

Users are accustomed to closing the main window of a softphone or a chat client when not engaged in a call or chat session. The softphone or chat application continues to run in the background, but without a UI, thereby reducing visual clutter for the user. When an incoming call or chat is received, the communication session window again becomes visible.

It should be possible to hide a browser WebPhone application, while keeping the browser application running.

2.1.9 Use case: Ability to share desktop as part of a call

A browser WebPhone user should be able to share a document or share their desktop with the person on the other end of the call, without requiring a browser plug-in.

2.1.10 Use case: Ring tones

A browser WebPhone needs the ability to select ring tones and control volume of ringtones.

2.2 Video voicemail

2.2.1 Use case: Play Video/Audio stream recorded via Phone

A browser based video/voicemail application should allow a user to playback a recorded video/audio message that was recorded from an incoming Phone device. This should be enabled without the need for third party plugins, and should allow the user to seek forward/backward within the message without requiring the entire stream to be downloaded.

Flexibility in supported codecs is important here to minimize the amount of video processing required on the mail server. In particular, H.264 is a common video codec used in telephony - if it were to become royalty free, having H.264 support in browsers would be ideal.

2.2.2 Use case: Record Video/Audio

A browser based video/voicemail application should allow users to compose and reply to video/voicemail within the browser without requiring external plugins. The browser should support capturing and sending an audio/video stream to a media server to support this.

2.2.3 Use case: Record Video/Audio over low bandwidth connection

A browser based video/voicemail application should support recording video/audio over low bandwidth connections. Video and/or audio quality can be degraded if necessary to support this, by different codecs or lower bitrate versions of codecs.

2.2.4 Use case: Visual notification of new voice/video message

Notification of new mail should be visible to the user, even if the browser window is minimized. A user must be able to work in different applications without fear of missing important messages.

2.2.5 Use case: Monitor message recording

In a browser based voice/video mail application, users would like the ability to listen/watch incoming video/voice mail being recorded via a phone. This is analogous to listening to your answering machine while someone is leaving a message.

To support this would require notification that a message recording has been initiated, and the ability to request a fork of the incoming stream to monitor it.

2.2.6 Use case: Device selection

A voice/video mail user should be able to select microphone, speakers and camera from the list of supported devices on the machine.

2.2.7 Use case: Better quality video/audio when higher bandwidth available

In situations where high bandwidth is available, codecs to support improved audio/video quality are desirable. Some video codecs also support variable bit rate encoding, in which case dynamic bit rate selection would be desirable.

2.2.8 Use case: Secure message playback

Many enterprise customers require secure, encrypted, voicemail. Playback of secure messages is currently difficult in a browser environment since there isn't a good way to do decryption using standard mechanisms (such as JavaScript). To facilitate this, a browser mechanism for decrypting audio/video streams is needed that does not require a plugin.

2.3 Online Meetings

2.3.1 Ability to initiate voice/video conference

2.3.2 Ability to share documents to meeting participants

2.3.3 Ability to share desktop

2.4 Notifications

As well as the incoming call and voicemail notifications mentioned in section 2.1 and 2.2 it is also necessary to be able to receive and provide visual indication of the following UC notifications:

- IM
- Location Update
- Presence Update
- Meeting Reminder
- 'Social' updates (e.g. Microblog, Activity Stream, Watchlist)

For incoming IMs the same toast experience needs to exist for IMs as for voice/video calls. The browser can be minimized during an IM conversation and the user needs to be notified of new IMs and of new IM conversations.

Regarding presence updates it should also be possible to:

- share presence updates between tabs or browser instances
- share presence state with other applications, both browser and non browser based.

2.5 OS Integration (cover both mobile and desktop)

Users are accustomed to running Unified Communications applications such as Cisco Unified Personal Communicator, Microsoft's Office Communicator and the Skype client, in a "system tray mode" whereby the client's window is not visible but the application continues to run and can be interacted with through a system tray icon. In this way the phone does not take up screen real estate but it is still available to receive incoming calls.

A browser WebPhone, on a desktop or mobile browser, should have the ability to run at system start-up, to run without requiring that the main browser window be visible, and to create a system tray entry that will allow the user interact with the phone.

---

3. Enabling Technologies

3.1 Media

Current Deficiencies

- HTML5 <audio> and <video> tags currently only address 1-way progressive download of media
- Voice codec (SPEEX, G.711, G.729) support in browsers is limited
- Inconsistency in video codecs supported by current browsers
- No mechanism to get access to devices or get local IP address
- No mechanism to encrypt/decrypt media streams
- No mechanism to get list of support codecs (although there is a way to try codecs in order inside an audio or video tag)

Proposal

- Browsers to support some common subset of royalty free codecs

Need ability for browsers to support many codecs but have a minimal subset that is known to work. The minimal subset needs to be easily shippable by all browser vendors with likely implies a royalty free codec. Recommend minimal subset consists of iLBC (RFC 3951), G.711, and standardized wideband codec out of IETF CODEC WG. Given the rapid advances in royalty free video codecs, such as webm effort, we expect to soon see a royalty free video codec from a widely recognized SDO.

- Mechanism to install codecs not supported out of the box by a browser

This will allow codec development and improvement to occur independent of the browser, and make it possible to adapt to more existing back end media servers that may not yet support the standard codecs.

- API to support device selection, notifications, control and media origination/termination
- API to get access to local IP address
- API to adjust video brightness
- API to for encryption and decryption, integrity, and identity of media stream
- API for QOS
- API to enable/disable bitrate adjustment
- Support for RTP(s) media streams

API

getSupportedCodecs() - for an audio or video tag this would return a list of all supported codecs

This is desirable so we can check formats up front without trying to play the different types within a video tag. This will help to provide better information to users and for logging purposes.

The current mechanism would also, it seems force a load of the media stream before providing an error message, and also not provide a nice way to capture that event and react to it differently in JavaScript:

```
<video controls>
  <source src="foo.ogg" type="video/ogg">
  <source src="foo.mp4">
  Your browser does not support the <code>video</code> element.
</video>
```

setBrightness(level) - for a video tag this would set the brightness adjustment level to help in viewing video that may be too light or dark

Since we are going to be dealing with video created in different many environments, being able to adjust the brightness is desirable.

setAutoBitrateAdjust() - enable/disable auto bitrate adjustment based on bandwidth (when available)

Might need some other adjustment parameters, but this is a start (i.e. set thresholds and so forth)

3.2 Phone API

Current Deficiencies

- Call control not currently addressed in HTML5 standards.

Proposal

- Embed call control stack (SIP/Jingle) in browser (for information, Cisco's call control stack image is approximate 1MB for unoptimized devices and closer to 100KB for space optimized devices).
- Registration/login is done on a per user basis. No explicit device configuration is required in the API.
- Addition of <phone> and <call> elements and associated JavaScript APIs to the HTML standards providing call control functionality in the browser.

API

The following is an outline of the JavaScript API for the <phone> element. Note that this set of methods is a subset of the full API, but these represent a workable set of operations for a phone that supports a single simultaneous call. Multiple call support entails, amongst other things, extending the API to identify the call to which the operation applies:-

Methods

`login(username, password)` Log the phone in

`logout()` Log the phone out

`call(number)` Make a call to the specified number

`answer()` Answer an incoming call

`endCall()` End the existing call

`hold()` Hold the current call

`resume()` Resume a currently held call

`sendDTMF(digit)` Send a DTMF digit

`mute(audio|video|all)` Mute the current call

`getCall()` Return details of the current call:-

- exists
  - true|false
- callId
- callState
  - Created|Dialing|Progressing|Alerting|Connected|Held|Disconnected|Failed
- callType
  - Incoming|Outgoing
- mediaState
  - NotStarted|Ok|TempFail|PermFail
- callingPartyNumber
- callingPartyName
- calledPartyNumber
- calledPartyName
- capabilities
  - HoldCall,ResumeCall,SendDtmf,Terminate
- callMuted
  - true|false

*Events**In addition to the above methods, the following callback handlers can be registered with the phone to receive asynchronous events:*

`registerPhoneStateChangeCallback(function(state))` Listen for phone status change updates.
callback passes state parameter: Ready|Registering|Idle|Connected|Busy

`registerCallStateChangeCallback(function(callInfo))` Listen for changes to the call's state
Callback passes callInfo parameter which is the same as **`getCall()`** method above

`registerLoginFailureCallback(function(reason))` Register for login failures
Callback passes reason parameter: Ok|Unknown|NoDevicesFound|NoRegistrarConfigured|
ConfigFetchError|AuthenticationFailure

Code Samples

The code example below illustrates how a softphone could be used via HTML-only. The code consists of a top-level <phone> element which contains attributes related to the configuration and display of the

softphone. It contains a sub-element that's based on the proposed stream API that is used to render the media. By making this a subelement of <phone>, it should be possible (and certainly desirable) for the implementation to automatically 'wire up' the media and signaling. Similarly, the <phone> element contains an inner <device> element that could be used for user device selection. This is a description of the attributes for <phone>:-

type - the signaling implementation type - either sip or jingle (possibly with sub-types)
registrar - the IP address of the SIP registrar
ringer - the ID of the DOM element on the page that represents the audio to be played when the phone rings
dialpad - the ID of the DOM element that represents the phone's dialpad (if needed)
conversation - the ID of the DOM element that's displayed when a conversation is active
incoming - the ID of the DOM element that's displayed when an incoming call is received
onCallStateChange - event handler for call state change events. There will be other events related to registration and media state changes, errors etc.

Everything here is of course fully scriptable, so for more advanced users it will be more likely that much of this behavior would be achieved via JavaScript instead. Of course, there will be a rich set of events available to allow users to trap things like call state changes, media changes, registration state etc.

The three elements further down the page are user implementations of the various UI elements of the phone - this enables the user to define whatever CSS and behavior that they want. However, the inner <button> elements could be automatically wired up to the phone controls by using the "name" attribute.

```
<phone type="sip" registrar="10.23.45.122" onCallStateChange="myCallStateChangeHandler()"
      ringer="myringer" dialpad="mydialpad"
      conversation="myconversation" incoming="myincoming">

       <stream type="audio/g711">
             Unable to create Media!
       </stream>

       <device type=media></device>
</phone>


<!-- standard HTML5 <audio> element -->
<audio id="myringer" src="ringring.mp3" loop="loop"></audio>

<div id="mydialpad">
      <!-- can use standard <button> elements - value attribute is used to define the dialed digit -->
      <button value="0">0</button>
      <button value="1">1</button>
      <button value="2">2</button>
      <!-- etc etc -->
</div>

<div id="myconversation">
      <div class="mystyle" id="myid"></div> <!-- custom content -->

      <!-- Button values are standard and used to automatically "do the right thing" -->
      <button value="end">End Call</button>
      <button value ="mute">Mute</button>
      <button value="holdtoggle">Hold</button>
      <button value="dialpadtoggle">Dialpad</button>
</div>

<div id="myincoming">
      <div class="myotherstyle" id="myotherid"></div> <!-- custom content -->
      <!-- Button value is standard and used to automatically "do the right thing" -->
      <button value="answer">Answer</button>
```

```
</div>
```

Another "declarative" enhancement would be to be able to initiate a phone call by using HTML tags. The following code has two versions, one with a new <call> element and another using an <a> element with a new 'sipto' URL type.

```
<!-- version with specific <call> element
    <call> element has a 'phone' attribute that specifies the DOM ID of the <phone> to be used
    to place the call.
-->
<ul>
      <li><call phone="myphone" to="sip:fluffy@cisco.com">Cullen Jennings</call></li>
      <li><call phone="myphone" to="sip:jooreill@cisco.com">John O'Reilly</call></li>
      <li><call phone="myphone" to="sip:damathie@cisco.com">Dave Mathieson</call></li>
</ul>

<!-- version that uses <a> element analogous to mailto
    Note: In this version it's not as easy to specify the ID of the <phone> element
-->
<ul>
      <li><a "sipto:fluffy@cisco.com">Cullen Jennings</a></li>
      <li><a "sipto:jooreill@cisco.com">John O'Reilly</a></li>
      <li><a "sipto:damathie@cisco.com">Dave Mathieson</a></li>
</ul>
```

Thoughts on signaling

The idea of a phone built into the browser raises the question of what kind of signaling should be supported. Obviously the main choices are SIP or Jingle, but building these into the browser raises questions over compatibility and interoperability with different endpoints and registrars. However, the functionally we are discussing here works in a widely interoperable way between existing SIP devices and without building in a signaling protocol to the browser, it is hard to imagine how to provide an interface that web developer could easily use to enable interface communications. The phone object like interfaces above all require a signaling interface in the browser.

A more purist 'web' approach would be for signaling to be carried out using HTTP. This work has already been prototyped with some success, but it requires the design and implementation of a new API that mirrors functionality of the existing protocols. Developing a protocol that deals with the range of things required to have RTP interoperability with existing systems is very complex. Existing master slave control protocols that have about the same level of functionality are MGCP (RFC3435), Megaco, and Cisco SCCP protocol. All of these are relatively large and complex protocols.

One possible solution is to send SIP over HTTP. This would entail the deployment of code on the server to pack and unpack the SIP messaging from HTTP, and to write a SIP stack in JavaScript. The advantage here is that the SIP (or other signaling) stack can be updated as needed without requiring a new browser, and it could be tailored as needed for the environment. So there could be a generic stack that supports basic call functionality, but a more advanced vendor-specific stack made available for a more advanced feature set. Most useful SIP stacks represent in the order of 10 to 100 person years of development effort so the prospect of developing a useful SIP stack in JavaScript is a large and complex project.

3.3 Notifications

- HTML5 desktop notifications as currently implemented in Chrome are suitable for notification of Unified Communication events to the user.
- Should investigate the possibilities of all applications, both web and non-web, sharing the same notification framework. For HTML5, this would mean that the implementation would detect the system-level notification framework and use that if appropriate/available e.g. Growl on Mac.

3.4 Cross-tab co-ordination

Current Deficiencies

HTML5 SharedWorkersprovide a good, general purpose solution to sharing memory between documents and tabs. For example, a voicemail gadget, a phone gadget and an instant messaging gadget may all share access to corporate directory presence. By sharing this information locally, the gadget author reduces the number of server connections, the network traffic and local memory since the shared presence state is not duplicated across the gadgets

A browser WebPhone should implicitly work across tabs of the same web application, with multiple tabs acting as multiple lines. Each developer who integrates a WebPhone into his/her application should not have to rewrite synchronization logic to have the webphone work seamlessly across tabs.

Proposal

A browser should recognize that different pages can be part of the same application - either because they are served from the same site, or because they share an "application key". A browser should then treat the pages of an application that use a webphone, as separate lines of the same device.

Each tab of a web application should have call control access over calls on other tabs of the same web application - the ability to hold/resume, conference, transfer, end call etc.

At the same time, a phone running as a separate application in the same browser can not be allowed to see what calls the other phones is making or access the media in the other virtual phone.

3.5 OS Integration

Current Deficiencies

- Web Applications cannot request that they be started at system start-up.
- Web Applications cannot addd system tray entries
- Web applications cannot request that they run in a "micro" mode whereby they continue to run when the browser's window is closed.

Proposal&

- must enable a web application to be started at system start-up
- must enable a web application to add a system tray entry
- must enable a web application to to run even when the browser window is closed

```
<!-- W3C widget configuration file used as an application configuration file-->
<widget id=http://webphone.org/webphone >

<icon src="http://webphone.org/webphone/systray.png"/>
<!-- addition to W3C widget configuration file -->
<system type="startup" systemTray="true" hideOnClose="true"/>

</widget>


<!-- additional event called when the user clicks the web application's system tray entry. In response to this
 the WebApplication might
offer a menu that will be rendered by the web browser over the status bar. -->

  interface SystemTrayCallback {
     void handleEvent();
  };
```

.

4. References

Device Element - http://www.whatwg.org/specs/web-apps/current-work/multipage/commands.html#devices

Stream API - http://www.whatwg.org/specs/web-apps/current-work/multipage/commands.html#stream-api

Device APIs and Policy Working Group - http://www.w3.org/2009/dap/

HTML Media Capture - http://www.w3.org/TR/2010/WD-html-media-capture-20100720/

Beyond HTML5 - Conversational Voice and Video Implemented in WebKit GTK+ - https://labs.ericsson.com/blog/beyond-html5-conversational-voice-and-video-implemented-webkit-gtk

Firefox 4 Audio Data API - https://wiki.mozilla.org/Audio_Data_API

Other Topics

- Privacy e.g. important that we have good visual indications of when camera/microphone goes on/off
- Security. Todays enterprises do you a "security in depth" approach but unfortunately the primary security is provided by the "crunch outside, soft chewy middle model" that the firewalls provide. Allowing the web browser to send arbitrary UDP packets, such at a data base attack, to any IP address on the inside of the firewall is unacceptable. Forcing the media to go to the web site that downloaded the web pages is also not acceptable due to the loss of audio quality this implies not to mention the bandwidth required.
- Propose that the browser be allowed to send a fixed format "request to send" packet to any location. If that location replies with an "OK to send", then SRTP or RTP media can be sent. The browser would need to enforce this restriction. STUN could be used as the "request to send" and "OK to send" signaling.