XEP-0327: Rayo

Abstract: This specification defines an XMPP protocol extension for the third-party control of telephone calls and other similar media sessions. The protocol includes support for session management/signaling, as well as advanced media resources such as speech recognizers, speech synthesizers and audio/video recorders. The protocol serves a different purpose from that of first-party protocols such as Jingle or SIP, and is compatible with those protocols.

Authors: Ben Langfeld, Jose de Castro

Status: Experimental
Type: Standards Track
Version: 0.2
Last Updated: 2013-06-10

WARNING: This Standards-Track document is Experimental. Publication as an XMPP Extension Protocol does not imply approval of this proposal by the XMPP Standards Foundation. Implementation of the protocol described herein is encouraged in exploratory implementations, but production systems are advised to carefully consider whether it is appropriate to deploy implementations of this protocol before it advances to a status of Draft.

Table of Contents

1. Introduction

Rayo is a protocol to allow third-party remote control over media sessions, audio/video mixers and a variety of advanced media resources such as speech recognizers, speech synthesizers and audio/video recorders. These capabilities can be combined to create a wide variety of applications such as menu-based phone systems, in-game conferencing and anonymous dating services. Unlike Jingle or even SIP, a Rayo client is not concerned with being a party to either the session negotiation or the media stream itself.

- A Rayo server takes on the role of negotiating a media session between itself and some other endpoint, or between two external endpoints, by way of an implementation-specific means, be that Jingle, SIP, the public-switched telephone network, or anything else. The server may even bridge multiple networks.
- The server then presents the Rayo protocol as an interface to a Rayo client, allowing it to monitor and/or exercise third-party control over the established media sessions.
- The client has the option to accept/reject/answer inbound session requests, request the creation of outbound sessions and monitor their progress, execute media operations such as speech synthesis, speech recognition & recording, and to end sessions.

The relationship between the calling parties, the Rayo server and the Rayo client looks something like this:

```
    [caller] ----SIP---- [rayo server] ( -----Jingle---- [callee] ) optional
                              |
                              |
                         rayo client
```

This document defines the core Rayo protocol, and contains provisions for its extension by further specifications.

2. How it works

In order to understand the nature of a Rayo interaction, here we show a simple example of a control session.

Example 1. New call announces itself to a potential controlling party

```
<presence from='9f00061@call.shakespeare.lit'
          to='juliet@capulet.lit/balcony'>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='urn:xmpp:rayo:call:1'
     ver='QgayPKawpkPSDYmwT/WM94uAlu0='/>
  <offer xmlns='urn:xmpp:rayo:1'
         to='tel:+18003211212'
         from='tel:+13058881212'/>
</presence>
```

In this example, a call from 'tel:+13058881212' has reached the Rayo server 'shakespeare.lit' by calling 'tel:+18003211212', and been assigned an ID '9f00061'. The server has determined that 'juliet@capulet.lit' is a valid candidate to be the client to whom the server delegates control of the call, and so has directed an offer event to her 'balcony' resource.

The client, 'juliet@capulet.lit', then decides that it is able to handle the incoming call, and so accepts it from the server, thus gaining exclusive control and indicating to the calling party that the call will be processed and that it should ring.

Example 2. Potential controlling party attempts to become definitive controlling party by sending the call an accept command

```
<iq from='juliet@capulet.lit/balcony'
    to='9f00061@call.shakespeare.lit'
    type='set'
    id='hd721'>
  <accept xmlns='urn:xmpp:rayo:1'/>
</iq>
```

Example 3. Call acknowledges accept command to the (now) definitive controlling party

```
<iq from='9f00061@call.shakespeare.lit'
    to='juliet@capulet.lit/balcony'
    type='result'
    id='hd721'/>
```

Following confirmation from the server that the attempt to gain control of the call was successful, the client proceeds to answer the call, opening up the media stream between the caller and the server.

Example 4. Controlling party answers the call

```
<iq from='juliet@capulet.lit/balcony'
    to='9f00061@call.shakespeare.lit'
    type='set'
    id='43jo3'>
  <answer xmlns='urn:xmpp:rayo:1'/>
</iq>
```

Example 5. Call acknowledges answer command to controlling party

```
<iq from='9f00061@call.shakespeare.lit'
    to='juliet@capulet.lit/balcony'
    type='result'
    id='43jo3'/>
```

Once the client has confirmation that the call has been answered, it triggers the start of a media output component in order to play a message to the caller using a Text-to-speech (TTS) engine.

```
<iq from='juliet@capulet.lit/balcony'
    to='9f00061@call.shakespeare.lit'
    type='set'
    id='j9d3j'>
  <output xmlns='urn:xmpp:rayo:output:1'
          voice='allison'>
    <document content-type="text/plain">
      <![CDATA[
        You have no new messages. Goodbye!
      ]]>
    </document>
  </output>
</iq>
```

Example 7. Call acknowledges request for new output component and provides its ID

```
<iq from='9f00061@call.shakespeare.lit'
    to='juliet@capulet.lit/balcony'
    type='result'
    id='j9d3j'>
  <ref xmlns='urn:xmpp:rayo:1' uri='xmpp:9f00061@call.shakespeare.lit/fgh4590'/>
</iq>
```

After confirmation that the output component was successfully created, the client then awaits notification of its completion.

Example 8. Output component announces its completion, giving the reason

```
<presence from='9f00061@call.shakespeare.lit/fgh4590'
          to='juliet@capulet.lit/balcony'
          type='unavailable'>
 <complete xmlns='urn:xmpp:rayo:ext:1'>
   <finish xmlns='urn:xmpp:rayo:output:complete:1' />
 </complete>
</presence>
```

The client then decides it has no further operations to perform on the call, and that the call should end. It instructs the server to hang up the call gracefully.

Example 9. Controlling party hangs up the call

```
<iq from='juliet@capulet.lit/balcony'
    to='9f00061@call.shakespeare.lit'
    type='set'
    id='f3wh8'>
  <hangup xmlns='urn:xmpp:rayo:1'/>
</iq>
```

Example 10. Call acknowledges hangup command to controlling party

```
<iq from='9f00061@call.shakespeare.lit'
    to='juliet@capulet.lit/balcony'
    type='result'
    id='f3wh8'/>
```

Example 11. Controlling party receives notification of the call being terminated

```
<presence from='9f00061@call.shakespeare.lit'
          to='juliet@capulet.lit/balcony'
          type='unavailable'>
  <end xmlns='urn:xmpp:rayo:1'>
    <hangup-command/>
  </end>
</presence>
```

3. Requirements

The protocol defined herein is designed to provide the following features:

1. **Call Control:** Incoming calls are "offered" to clients at which point they can be answered, rejected, redirected to another destination, etc. Outbound calls may also be made and monitored. Every attempt is made to be shield the Rayo client from the low level telephony protocol (e.g. SIP, Jingle, PSTN, etc).
2. **Audio File Playback:** A compatible Rayo server will fetch a file from a a specified URL and play the containing audio to the caller.
3. **Speech Synthesis / TTS:** In cases where dynamic data must be spoken, a Speech Synthesis engine may be used to play computer generated speech to the caller.
4. **Touch-tone Events / DTMF:** Rayo surfaces real-time event when the caller presses keys on their touch-tone keypad.
5. **Speech Recognition:** Enables the phone application to take spoken queues allowing for sophisticated voice-driven menus and directory services.
6. **Call Recording:** Can be used to capture the caller's voice (e.g. Voicemail) or both sides of the call for auditing and compliance purposes.
7. **Mixing:** Typically referred to as an audio "conference"; calls can be joined together so that the participants can hear each other in real-time.

Many third-party call control protocols have preceeded Rayo (see Asterisk's AGI/AMI, FreeSWITCH's eventsocket, Microsoft's TAPI, Java's JTAPI, Novell/AT&T's TSAPI, CSTA, etc). None of these protocols is ideal, and all have one or more of the following drawbacks:

- **Totally ground-up wire protocol** requiring implementation all the way down to the socket.

- **Inconsistent** - evolved, rather than designed.
- **Lacking in scalability** - client/server sometimes tied one-to-one, servers rarely clustered, advanced message routing not possible.
- **Poor security** - lack of wire-level encryption, lack of or sub-standard authentication mechanisms, lack of or limited authorization mechanisms, lack of or poor sandboxing between multiple tenants on one system.
- **Inextensible** - The specification of extensions to the core protocol is either impossible or very difficult.

Rayo has been designed with these failings in mind, and intends to address many concerns not addressed by these earlier attempts. The following considerations were made:

- **Simple client library implementation** - XMPP client libraries exist in all modern languages, and many are of a high standard of quality and maturity.
- **Cross-platform standard** - The protocol must not expose any platform specifics and all elements should be candidates for implementation on any suitable platform. Addiditionally, the protocol must be ratified as a standard following a community discussion.
- **Asynchronous interface** - The protocol should present an asynchronous interface for the purposes of performance and flexibility in performing parallel operations.
- **Consistent** - The protocol must provide a considered, unobtrusive, logically and philisophically consistent interface.
- **Federated** - The protocol must support communication between client and server entities on separately owned, operated and addressed networks.
- **Flexible routing** - The protocol must lend itself to routing across wide networks such as the internet, and to potential complex routing such as proxying or redirection. Additionally, the client and server should each be aware of the presence of the other and be able to use such information to make routing decisions.
- **Extensible** - The protocol must provide for the possibility of extra functionality being added by future specifications or an individual implementation.
- **Secure** - The protocol should include appropriate measures for authentication and authorization of participants, as well as preventing third-parties from intercepting control messages.

Many of the features in the above list are available to Rayo at no specification or implementation cost, since they are core to XMPP itself and thus Rayo inherits these 'for free'.

Additionally, the protocol is required to abstract away the complexity of the back-end negotiation, especially the details of the transport protocols such as SIP or Jingle, but to map conceptually to such protocols.

4. Terminology

4.1 Glossary

Third-party call control (3PCC)
: The observation and/or control of a live media session by an entity which is not a direct party to the session.

Command
: Commands instruct the receiving entity to perform some atomic action. Commands may be executed against a given call, component or mixer and can be considered completed as soon as they receive a response. Some commands create components, and return a reference to the component in their response.

Component
: Components extend the Rayo protocol by providing additional media and call control functionality. Components are created by an appropriate command, which returns a reference to the component. Components are executed asynchronously, and have a lifecycle attached to them, with the ability to trigger events or have commands issued to it. Once a component is stopped or comes to an end naturally, it will issue a special <complete/> event, indicating that it has ceased executing and deliver any required data.

Potential controlling party (PCP)
: An XMPP entity to which an offer to control an incoming call may be sent.

Definitive controlling party (DCP)
: The XMPP entity which gains a lock on control of a session, either by requesting the session's creation, or being the first respondent to an offer.

Security Zone
: A security zone is the conceptual border around a call which defines which parties may interact with the call's media or signaling. A security zone MUST contain the rayo server's internal implementation, the media server to which the call is joined, the DCP, and any JID whose bare form is the same as the DCP. A server MAY relax this definition further, for example to consider all JIDs at the same domain to be in the same security zone.

4.2 Conventions

In examples, the following JIDs are used:

- **juliet@capulet.lit/balcony, romeo@montague.lit/orchard** - Potential controlling parties
- **shakespeare.lit** - The root domain of the Rayo service

5. Concepts and Approach

A complete Rayo deployment has several elements and interacting entities which must be understood.

5.1 Actors

5.1.1 Server

A Rayo server is an entity which is capable of receiving and initiating calls and being party to their media stream, while exposing a Rayo interface to a client in order to permit control over its calls. The Rayo server may handle calls in any way supported by the implementation, such as SIP, Jingle, etc, and should expose a full XMPP domain at the root level of the service deployment (eg shakespeare.lit).

The Rayo server is responsible for keeping track of valid clients, routing calls to the correct potential controlling parties, performing authorization measures on received stanzas, etc.

For the purposes of this specification, complex server-side deployments such as clusters, proxies, gateways, protocol translators, etc are not considered. Further details of such concepts may be found in their (present or future) relevant specifications.

5.1.2 Client(s)

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.