Network Working Group	I.B.C. Baz Castillo
Internet-Draft	XtraTelecom S.A.
Intended status: Informational	S.I.C. Ibarra Corretge
Expires: April 27, 2012	AG Projects
	J.L.M.V. Millan Villegas
	XtraTelecom S.A.
	October 25, 2011

Open In-The-Wire Protocol for RTC-Web
draft-sipdoc-rtcweb-open-wire-protocol-00

Abstract

RTC-Web clients communicate with a server in order to request or manage realtime communications with other users. This document exposes four hypothetical and different RTC-Web scenarios and analyzes the requirements of the in-the-wire protocol in each of them.
The aim of this document is to make RTC-Web properly fit in the nature of the Web.

Status of this Memo

Copyright Notice

Table of Contents

1. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

In contrast to protocols such as SIP *[RFC3261]* or XMPP *[RFC6120]*, RTC-Web *[RTC-Web]* does not define a protocol for establishing media sessions between peers. Instead RTC-Web defines the media protocols (RTP/SRTP/ICE) to be used by web browsers. It also states how the web browser natively handles media streams (including media security and validation concerns), defines the requirements for the communication between the RTC-Web stack in the browser and the web application (via a JavaScript API to be defined by W3C) and MAY suggest some kind of media

negotiation protocol to be carried in-the-wire between RTC-Web clients and servers.
That said, RTC-Web does not mandate any user identifier syntax (in the way SIP defines the SIP URI), nor an authentication mechanism, in-the-wire messages format or the way messages are exchanged between RTC-Web clients and servers. There are many different ways by which those targets can be achieved nowadays on the Web.
All this flexibility makes the whole picture of an RTC-Web scenario and the scope in which RTC-Web is involved hard to understand. This document tries to identify each component in the RTC-Web architecture and clarify which components can be left up to web developers and which others should be mandated by RTC-Web specifications.

3. Definitions

The following terms have special significance in the context of RTC-Web.

JavaScript WebRTC API: This is the communication layer between the native RTC-Web stack in the browser and JavaScript. It includes JavaScript functions to manage media sessions along with JavaScript callbacks which will be called by the RTC-Web stack when some media related event takes place. This API MUST be exposed by every RTC-Web compliant browser. It's being defined by W3C.

RTC-Web Server: The server that RTC-Web clients communicate with for initiating and managing media sessions with remote peers. This is usually an HTTP *[RFC2616]* server or a WebSocket *[I-D.ietf-hybi-thewebsocketprotocol]* server behaving as a centralized point for signaling messages exchanged between peers (and MAY accomplish other tasks such as authentication, authorization, peer lookup procedures and protocol conversion). A given RTC-Web server will implement the In-The-Wire Protocol (defined below) chosen by the website developer.

In-The-Wire Protocol: This is the communication layer between users' web browser and the RTC-Web Server. It involves a signaling protocol to be carried over HTTP or WebSocket (the protocols JavaScript can interact with). The messages exchanged over this protocol contain call control information and media negotiation information (both Call Control Protocol and Media Negotiation Protocol explained next).

Call Control Protocol: This involves the format and semantics of the messages exchanged between the RTC-Web client (web browser) and the RTC-Web Server for routing and other purposes such as authorization, authentication, registration and others. Examples of a Call Control Protocol could be SIP or XMPP carried over HTTP or WebSocket, whose messages contain information about the originator and recipient of

the message, credentials, type of message, etc. However any other custom protocol (such as HTTP POST or some JSON-based protocol over WebSocket) can accomplish this task. Call Control Protocol messages MAY carry media negotiation information (in the same way a SIP INVITE request contains an SDP body).

Media Negotiation Protocol: When a RTC-Web client wants to establish a media session with a remote peer, it sends a Call Control Protocol request over the wire to the RTC-Web Server indicating the destination of the request (along with other fields). Such a request also carries media information exposed by the originator. In the SIP protocol example, the Media Negotiation Protocol is represented by the SDP offer/answer body conveyed by INVITE and 200 OK messages (simplified). In the case of RTC-Web, such media negotiation MAY be carried as ROAP *[I-D.jennings-rtcweb-signaling]* Offer/Answer JSON objects.

JavaScript RTC-Web Library: This is a website-agnostic JavaScript library. Web developers (end-users) can include it within their websites. The library defines a custom In-The-Wire Protocol by providing an API with functions for generating requests and responses (to be sent in-the-wire) along with callbacks for processing incoming request/responses and state change notifications. This JavaScript library is coded by an RTC-Web expert and is supposed to hide the complexity of the JavaScript WebRTC API and the management of the In-The-Wire Signaling to the end-user. This is the API the end-user should use and care about (and here is where the famous term "20 lines of code" applies).

JavaScript WebSite Library: This is the custom JavaScript library that a web developer (the end-user) provides in his website. In the context of RTC-Web, this custom library is supposed to make usage of functions and features present in a JavaScript RTC-Web Library (defined above). This is the library in which the end-user writes "20 lines of code" for integrating RTC-Web capabilities within his website.

4. Overview of an RTC-Web Communication

```
{
  "request": {
    "call-id": "0skilqwp",
    "transaction-id": 1001,
    "request-type": "call",
    "source-user": "Bob",
    "destination-user": "Alice",
    "subject": null,
    "cookie": "kj87kjsdhf",
    "media": _ROAP_OFFER_OBJECT_
  }
}

{
  "response": {
    "call-id": "0skilqwp",
    "transaction-id": 1001,
    "status": "accepted",
    "source-user": "Alice",
    "destination-user": "Bob",
    "cookie": "t112mnkszz",
    "media": _ROAP_ANSWER_OBJECT_
  }
}

{
  "request": {
    "call-id": "0skilqwp",
    "transaction-id": 1001,
    "request-type": "ack",
    "source-user": "Bob",
    "destination-user": "Alice",
    "cookie": "kj87kjsdhf",
    "media": _ROAP_OK_MESSAGE_
  }
}

{
  "request": {
    "call-id": "0skilqwp",
    "transaction-id": 1002,
    "request-type": "hangup",
    "source-user": "Alice",
    "destination-user": "Bob",
    "cookie": "t112mnkszz"
  }
}
```

Here a hypothetical and very simple RTC-Web scenario is described:

*A user visits a web page "mysite.com" using his browser and logs-in the web by introducing his username (Bob) and password. The authentication is achieved by sending an HTTP POST request with the user's credentials. The web server validates the credentials and replies with a HTTP 200 response containing a Cookie "kj87kjsdhf" to be used within the same session. The user is redirected to a new page from which the browser retrieves two JavaScript libraries:

rtcquery.js (a JavaScript RTC-Web Library): This is a GPL JavaScript library implementing a custom In-The-Wire Protocol for RTC-Web based on JSON and WebSocket. This library is becoming the most successful and extended RTC-Web library and there are O'Reilly books about it.

mysite.js (a JavaScript WebSite Library): This is the JavaScript code created by the web developer of "mysite.com". It includes website specific functions and makes use of the "rtcquery.js" library to incorporate RTC-Web capabilities to the web page (by adding "20 lines of code").

*Once all the JavaScript code (both "rtcquery.js" and "mysite.js") is loaded by the browser, it opens a WebSocket connection with the web server, which is also a WebSocket server listening on the same port (it could be a different server though).

*The web developer of "mysite.com" has implemented the In-The-Wire Protocol defined in "rtcquery.js" into his WebSocket server using PHP. For this task, the developer has studied the documentation available in the website of the rtcQuery.js project.

*The custom In-The-Wire Protocol states that each request sent over the WebSocket connection is a JSON object containing:

A mandatory Call Control Protocol section: This includes fields such as "call-id" (a common string for all the requests/responses within the same call), "transaction-id" (an integer for correlating a request and its associated responses), "request-type" (for example "call"), "source-user" (current user), "destination-user" (the remote peer), "subject" (some description of the call) and "cookie" (the Cookie sent by web server, which is used by the client to identify itself and get authorization by the WebSocket server).

An optional Media Negotiation Protocol section: This is conveyed by attaching a ROAP *[I-D.jennings-rtcweb-signaling]* Offer or OK JSON object.

*In-The-Wire Protocol responses are similar to the requests, but instead of "request-type" and "subject", they contain a "status" field indicating the nature of the response (which can be "accepted", "rejected" or "not-online") and instead of a ROAP Offer they MAY contain a ROAP Answer.

*The new page rendered by the browser includes a big section with "online" users (those that are available for audio/video sessions). The user wants to make an audio call with Alice (who is online) and clicks a "Call" button next to Alice's buddy.

*Then the JavaScript code makes a call (using the JavaScript WebRTC API) to the browser RTC-Web stack in order to ask for a ROAP Offer object with just "audio" capability (note: PeerConnection stuff ommited for brevity). The RTC-Web stack performs some internal operations to discover the browser IP, gets some available UDP port for sending RTP, chooses an audio codec from the list of available codecs in the browser, and returns the ROAP Offer JSON object.

*After that, the JavaScript code constructs an In-The-Wire Protocol request (a JSON object) as follows:

*The JavaScript code sends the request to the web server via the existing WebSocket connection.

*The server inspects the "cookie" and the "source-user" fields, and authorizes the request since such Cookie value is associated to an existing web session owned by Bob.

*Then the server checks for Alice's status. Alice is online and connected via WebSocket to the server, so using such connection the server delivers the request to Alice (previously removing the "cookie" field).

*Alice's browser receives the request and some JavaScript callback (defined by the JavaScript code when a WebSocket messsage is received) is called. Alice is prompted to accept or reject the incoming call request from Bob. Alice presses "Accept".

*The JavaScript code in Alice's browser makes a call (using the JavaScript WebRTC API) to the browser RTC-Web stack and gets a ROAP Answer object with just "audio" capability. Then it constructs an In-The-Wire Protocol response as follows (note that it includes her Cookie) and sends it to the server via the WebSocket connection:

*The server inspects the "cookie" and "source-user" and validates the response. Then it inspects the "destination-user" and routes the response object to Bob.

*Upon receipt of the response, the JavaScript code in Bob's browser automatically sends a request acknowledging the response to the server:

*The request is routed to Alice's browser and its ROAP OK message automatically delivered to the RTC-Web stack by the JavaScript code.

*At this point, both Bob and Alice's browsers perform ICE connectivity checks and finally establish a RTP audio session. The success of the media session establishment is notified to the users via some JavaScript pop-up.

*After a while Alice decides to terminate the call by pressing a "Hangup" button. The JavaScript code asks the RTC-Web stack in the browser to finish sending RTP and sends a JSON request to Bob via the WebSocket connection:

*Bob receives the request. His JavaScript code calls the RTC-Web stack to finish the media session (by passing it some "session-id" identifier retrieved from the initial ROAP Offer).

*Session is now terminated. Bob did not get a date with Alice, but has been enjoying a RTC experience so he is satisfied.

Lets inspect the RTC-Web components as defined by this document in the given scenario:

RTC-Web Server: A WebSocket server running in the same port as the web server.

In-The-Wire Protocol: Custom JSON messages over WebSocket transport.

Call Control Protocol: All the fields in the JSON message (but the "media" field).

Media Negotiation Protocol: The media information is located in the "media" parameter of the JSON message. It's carried as ROAP Offer/Answer JSON object.

JavaScript RTC-Web Library: "rtcquery.js" library made GPL by the rtcQuery.js project.

JavaScript WebSite Library: "mysite.js" custom library (makes use of "rtcquery.js").

5. More Use Cases

5.1. RTC-Web in Facebook

```
POST /ajax/chat/send.php?__a=1 HTTP/1.1
Host: www.facebook.com
Connection: keep-alive
X-SVN-Rev: 460802
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://www.facebook.com/?sfrm=1
Content-Length: XXX
Cookie: datr=ZjyfTjurvmsqvYVBDcXF8u; c_user=104442654509775; L=2;
  lu=RgQytVtJJBoSvWUNOYzs0oQg; sct=1319153603;
  xs=60%3A1c179a7dfb7f08278477b20e778bd391; p=112;
  presence=631L212REp_5f1B08654409875F4EriF0CEstateFDutF131910363 \
  96EvisF1HsndF1ODiFA21B02609687525A2C_5dEfFA21B02602687525A2Euct \
  319103618FD55F1G318103604PEuoFD1B02602687525FDexpF1319103680370 \
  lF_5b1_5dEolF0CCEalFD1B02602687525FDiF0umF0CCCC;
  wd=1366x675
Pragma: no-cache
Cache-Control: no-cache

msg_id=1319103647568%3A3629978310&client_time=13191036460048&
to=100002772687525&num_tabs=1&pvs_time&msg_text=hello&
to_offline=false&to_idle=false&window_id=2877189837&
sidebar_launched=true&sidebar_enabled=true&sidebar_capable=true&
sidebar_should_show=false&sidebar_visible=false&
post_form_id=449eb730c851e127f21d8a88b6a00667&fb_dtsg=AQC3StlW&lsd&
post_form_id_source=AsyncRequest&__user=100002624409995
```

Facebook integrates IM (instant messaging) into its web application. For that the JavaScript code performs HTTP long polling for retrieving incoming IM messages in realtime, and sends an HTTP POST request when a user sends a message to another user. Such HTTP POST requests look as follows:
The above request contains some parameters in both the Cookie header and the body. The Cookie header seems to contain information about the identity of the user sending the IM message. The Cookie value is probably also used for authentication in the server. The "msg_text" parameter in the body contains the IM text message itself while the "to" parameter in the body seems to contain the destination user (a long integer probably representing the user ID). The meaning of other parameters in both the Cookie and the body are up to Facebook, this is, they are specific to the application. It seems obvious that it's not possible to standarize all these parameters.

```
POST /ajax/call/call.php?__a=1 HTTP/1.1
Host: www.facebook.com
Connection: keep-alive
X-SVN-Rev: 460802
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://www.facebook.com/?sfrm=1
Content-Length: XXX
Cookie: datr=ZjyfTjurvmsqvYVBDcXF8u; c_user=104442654509775; L=2;
  lu=RgQytVtJJBoSvWUNOYzs0oQg; sct=1319153603;
  xs=60%3A1c179a7dfb7f08278477b20e778bd391; p=112;
  presence=631L212REp_5f1B08654409875F4EriF0CEstateFDutF131910363 \
  96EvisF1HsndF1ODiFA21B02609687525A2C_5dEfFA21B02602687525A2Euct \
  319103618FD55F1G318103604PEuoFD1B02602687525FDexpF1319103680370 \
  lF_5b1_5dEolF0CCEalFD1B02602687525FDiF0umF0CCCC;
  wd=1366x675
Pragma: no-cache
Cache-Control: no-cache

call_id=1319103647568%3A3629978310&client_time=1319103646048&
to=100002772687525&num_tabs=1&pvs_time&to_offline=false&
to_idle=false&window_id=2877189837&sidebar_launched=true&
sidebar_enabled=true&sidebar_capable=true&
sidebar_should_show=false&sidebar_visible=false&
post_form_id=449eb730c851e127f21d8a88b6a00667&fb_dtsg=AQC3StlW&
lsd&post_form_id_source=AsyncRequest&__user=1000026244099995&
media=_ROAP_OFFER_OBJECT_
```

Assuming that Facebook is willing to integrate RTC-Web within the web application, it makes sense that Facebook would be interested in reusing the same protocol and message format it's already using for IM (which is also realtime communication). So when a user clicks some "Call" button within his Facebook contact list, it is expected that the JavaScript code could generate an HTTP POST as follows:
The new HTTP POST request differs in the request URI (which now points to "/ajax/call/call.php"). The body includes a "media" parameter whose value is a ROAP Offer JSON object (properly encoded if necessary). Given this HTTP POST request, lets inspect the RTC-Web components as defined by this document:

RTC-Web Server: Facebook uses their HTTP servers as RTC-Web servers.

In-The-Wire Protocol: Facebook uses HTTP protocol and a common HTTP POST request.

Call Control Protocol: The information about the call originator is mainly included in the Cookie header, while other topics as the destination user of the call are located in the body (the "to" parameter).

Media Negotiation Protocol:
The media information is located in the "media" parameter of the body. In this case Facebook uses a ROAP Offer JSON object for carrying such media information.

JavaScript RTC-Web Library: It is expected to be an advanced JavaScript library designed by Facebook which also includes other functions unrelated to RTC-Web.

JavaScript WebSite Library: Merged with the JavaScript RTC-Web Library into a single JavaScript file.

If Facebook would desire to interoperate (federate) with a SIP network it is clear that it would need a signaling protocol gateway which converts the HTTP POST information into a SIP request, and the ROAP Offer into a SDP body.

5.2. SIP over WebSocket

This is an optimal solution for interoperating with SIP without requiring a protocol gateway. In this scenario the web user downloads a JavaScript code from the website and the JavaScript code establishes a WebSocket connection with a SIP proxy (the RTC-Web server) implementing the WebSocket transport *[I-D.ibc-rtcweb-sip-websocket]* (along with other common SIP transports as UDP and TCP). The messages exchanged between the RTC-Web client and server over the WebSocket connection are pure SIP requests and responses, with no modifications (others than the new "WS" transport identificator in the Via header).

```
INVITE sip:bob@example.org SIP/2.0
Via: SIP/2.0/WS invalid.domain;branch=z9hG4bK56sdasks
From: sip:alice@example.org;tag=asdyka899
To: sip:bob@example.org
Call-ID: asidkj3ss
CSeq: 1 INVITE
Max-Forwards: 70
Contact: <sip:alice@invalid.domain;transport=ws>
Supported: path, outbound
Content-Type: application/sdp

_SDP_
```

When the user makes a call from the web it generates a SIP INVITE to be sent over the WebSocket connection, which looks as follows:
For this to work, the JavaScript code must map the ROAP Offer retrieved via the JavaScript WebRTC API into a normal SDP (it's not the aim of this documment to discuss about the complexity such mapping could involve).
When the INVITE arrives to the RTC-Web Server (which behaves as a pure SIP proxy) it just performs standard SIP routing procedures (the same

as if the request would have arrived via UDP or TCP transports), so there is no need for a protocol gateway when interoperating with a pure SIP network out there.
Given this INVITE request, lets inspect the RTC-Web components as defined by this document:

RTC-Web Server: A SIP proxy that also implements the WebSocket transport.

In-The-Wire Protocol: Pure SIP protocol.

Call Control Protocol: Contained in the headers of the SIP request (From, request URI, Contact, Authorization...).

Media Negotiation Protocol: The session description (SDP) carried in the SIP request body.

JavaScript RTC-Web Library: It could be a GPL "jssip.js" library implementing SIP over WebSocket and a SIP stack in JavaScript.

JavaScript WebSite Library: Website specific. It would make use of the "jssip.js" library for adding RTC-Web capabilities "in 20 lines of code".

5.3. Poker Game

A website "www.poker-game.info" makes use of HTTP Comet technology for carrying realtime information about the game to each participant. The messages exchanged via HTTP Comet between participants and the web server are XML documents conveying updates and actions happening during the game.
Now that website wants to integrate RTC-Web capabilities and enter each participant into an audio multiconference in which every user listens to all the participants and can speak to them.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<game-multiconf>
  <action>call</action>
  <media>_ROAP_OFFER_XML_</media>
</game-multiconf>
```

To accomplish this architecture and still reuse the existing design, once the user logs-in the web his browser receives an incoming audio call from the conference server. Such call request is carried over the HTTP connection (HTTP Comet) as a new XML document which looks as follows:
The ROAP Offer is generated by the conference server, which satisfies all the media requirements of RTC-Web (ICE, SRTP).

The JavaScript code in the client side answers the call (once the user accepted it) and sends a similar XML containing a ROAP Answer via the existing HTTP connection.
Given the above XML document, lets inspect the RTC-Web components as defined by this document:

RTC-Web Server: The web server of "www.poker-game.info".

In-The-Wire Protocol: Custom XML document through HTTP Comet.

Call Control Protocol: It's minimal. There is no information about the originator or recipient of the call (not needed in this scenario). The XML contains an "action" field whose value "call" means "incoming call request from the website".

Media Negotiation Protocol: A ROAP Offer/Answer in XML format.

JavaScript RTC-Web Library: A library created by the developer of "www.poker-game.info". It implements the In-The-Wire Protocol as stated above.

JavaScript WebSite Library: It is merged with the JavaScript RTC-Web Library, so there is a single JavaScript file.

6. Conclusions

This document has shown four hypothetical scenarios of RTC-Web. Each scenario uses its own In-The-Wire Protocol (JSON over WebSocket, HTTP POST, SIP over WebSocket and XML over HTTP Comet) and it's hard to expect that all these scenarios could be constrained to use the same protocol and message format in-the-wire. The needs of each scenario are not the same, neither the custom fields carried in-the-wire (see for example the ammount of custom parameters Facebook includes within the HTTP POST request).
In the Web each website decides how to accomplish the features and capabilities it wants to provide to its users. Mandating the message format in-the-wire seems not to be an option given the nature of the Web. Mandating it would also make RTC-Web integration very hard into existing websites which already implement their custom signaling protocol and message format for realtime communications (as instant messaging), their authentication mechanisms, etc.
RTC-Web will bridge the gap between realtime communication services such as VoIP and the Web, so it must play by the rules present on the Web. These rules include the freedom by which the web developer chooses his preferred option to innovate and offer new services to users. That is the key to success of the Web and should be respected.

7. New Requirements for RTC-Web

Given the conclusions in the previous section, this document states some new requirements for RTC-Web:

1. It MUST be possible for a website developer to design his own In-The-Wire Protocol (including the messages format and transport used for carrying such messages).

2. It MUST be possible for a website developer to choose his favourite JavaScript RTC-Web Library and adapt his web application to make use of it.

3. It MUST be possible for a website developer to design a Media Negotiation Protocol in which the media information is not carried as ROAP Offer/Answer objects (by letting the developer implement the ROAP mapping in JavaScript).

   *NOTE: This text is written assuming that ROAP *[I-D.jennings-rtcweb-signaling]* becomes a standard in RTC-Web.

4. It MUST be possible for a website developer to make his RTC-Web scenario to interoperate with a pure SIP or XMPP/Jingle network without requiring a signaling protocol gateway (by using SIP over WebSocket *[I-D.ibc-rtcweb-sip-websocket]* or XMPP over WebSocket *[I-D.moffitt-xmpp-over-websocket]*).

   *This would be indeed feasible if previous bullets are satisfied.

8. Security Considerations

Not applicable.

9. IANA Considerations

Not applicable.

10. References

10.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[RFC3261]	Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E.

	Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
[RFC2616]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
[RFC6120]	Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
[I-D.jennings-rtcweb-signaling]	Jennings, C and J Rosenberg, "RTCWeb Offer/Answer Protocol (ROAP)", Internet-Draft draft-jennings-rtcweb-signaling-00, October 2011.
[I-D.ietf-hybi-thewebsocketprotocol]	Fette, I and A Melnikov, "The WebSocket protocol", Internet-Draft draft-ietf-hybi-thewebsocketprotocol-17, September 2011.
[I-D.ibc-rtcweb-sip-websocket]	Castillo, I, Millan, J and V Pascual, "WebSocket Transport for Session Initiation Protocol (SIP)", Internet-Draft draft-ibc-rtcweb-sip-websocket-00, September 2011.
[I-D.moffitt-xmpp-over-websocket]	Moffitt, J and E Cestari, "An XMPP Sub-protocol for WebSocket", Internet-Draft draft-moffitt-xmpp-over-websocket-00, December 2010.
[RTC-Web]	IETFW3C, "Real Time Collaboration on the World Wide Web", October 2010.

Authors' Addresses

Inaki Baz Castillo Baz Castillo XtraTelecom S.A. Barakaldo, Basque Country Spain EMail: ibc@aliax.net

Saul Ibarra Corretge Ibarra Corretge AG Projects Amsterdam, Netherlands EMail: saul@ag-projects.com

Jose Luis Millan Villegas Millan Villegas XtraTelecom S.A. Bilbao, Basque Country Spain EMail: jmillan@aliax.net