UPPSALA
UNIVERSITET

IT 11 078
Examensarbete 15 hp
Oktober 2011

EIGC integration with a web-browser

Voice communication through a web-browser

Johan Jons

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

EIGC integration with a web-browser

Johan Jons

Voice over IP communication services are constantly increasing as more and more devices are able to implement IP stacks and consequently connect to the Internet. Ericsson has developed one such service called EIGC which functions as a plug-in that can be used by videogame developers. By integrating the module into their game, the players will be able to participate in various voice communication services such as person-to-person calls and proximity-based voice chat.
Ericsson is interested in porting EIGC to different platforms, such as a web-browser; it could for example be incorporated into browser-based games or as a voice-chat hosted on the web.

Handledare: Johan Liljeqvist
Ämnesgranskare: Olle Gällmo
Examinator: Anders Jansson
IT 11 078
Tryckt av: Reprocentralen ITC

Table of contents

Chapter 1 – Introduction

This chapter aims to give the reader a basic understanding of the problem.

1.1 - Background

Imentum Systems AB, henceforth simply referred to as Imentum, is collaborating with Ericsson in developing a Voice over IP ("VoIP[1]") service. The service is to provide three key voice-communication features:

- Person-to-person: Support for establishing direct communication between clients.
- Proximity-based: Clients are able to interact with each other based on their current location. Voice will be propagated depending on the relative direction, distance and orientation between the clients. A proximity session is initiated by simply having avatars, which represents the players approach each other.
- Conference: Enables clients to form teams in which they can all speak to each other. Also gives support for lecture scenarios, where a moderator can allows individuals to speak in turn.

The service is called Ericsson In-Game Communication ("EIGC") and uses the Session Initiation protocol ("SIP[2]") to set up sessions in which voice is transferred using the Real-Time protocol ("RTP[3]"), all relayed through Ericssons IP Multimedia Subsystem ("IMS[4]").

1.2 – Problem definition

The first goal of the assignment is to find out if and how it is possible to implement the EIGC service into a web-browser environment, as well as developing a prototype for testing the service.
Developing applications to be launched through a web-browser means that you do not only have to take operating-system into consideration but also the fact that different web-browsers not always support the same technology. This gives a lot of combinations that needs to be tested; for example, Chrome behaves differently on Windows and Linux, and therefore one solution might not be compatible in all scenarios. However, a universal solution is preferable but not the main focus of this assignment.
Another important feature is to have all components implemented on the client-side rather than having a server performing all audio/connection processing. Using a server as a middleman by letting it do the processing work would make the implementation easier but it would not scale very well since all traffic would be relayed through it. As this solution is meant to be used in web-browser games and alike, typically your average developer might not be able to afford dedicated EIGC servers in addition to their own game servers.
Additionally, the user interaction is an important issue as well. The goal is to avoid bulky installation issues in order to use the client. At most some sort of plug-in initialization should be required of the user in order to use the solution.

Once finding out what web technology to base the solution on, the next goal is to find out how this can implemented using an alternative solution. Imentum has already developed an EIGC solution; however they are looking into an open-source framework called Doubango,

which provides approximately the same services as their EIGC solution they have developed. The second goal is therefore to investigate and implement the web-browser solution using Doubango.

1.3 – Motivation

Having a web interface makes it possible to integrate it into web-based videogames, which is a growing market as of today. A web-service is also very accessible and eliminates the need of manually installing a game client locally. For example, a user can simply run the EIGC service in a web-browser to communicate with users playing an EIGC-supported videogame instead of having to download several gigabytes of game client data.
Doubango is a platform-independent framework which makes it very suitable for future implementations on additional devices besides Windows, which is the prioritized platform. Another very desirable feature of Doubango is its small size – in total, the Doubango implementation takes up approximately 3-4 megabytes, while the EIGC solution uses 9 megabytes.

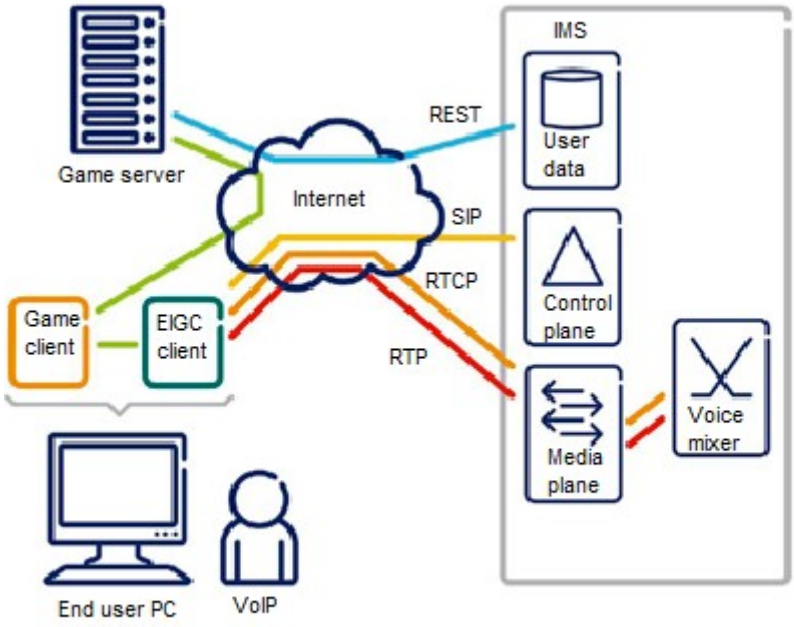
1.4 – Where is EIGC being used today

The EIGC solution has been implemented in the virtual universe called Entropia Universe[5], developed by the Swedish company MindArk[6]. Entropia is a massively multiplayer online role-playing game ("MMORPG") in which players buy in-game virtual currency for real-world money. This means that all virtual goods bought in the game have a corresponding real-world monetary value. Entropia is also registered as a bank; players can both deposit and withdraw money, meaning it is possible to sell virtual objects for a living. For example, in November 2010 a player sold a virtual resort for a total of $635,000.00 USD. [1]
EIGC is the optimal voice chat solution for Entropia since it adds security, anonymity and privacy as all traffic is directed through Ericsson IMS rather than being directly relayed between clients. This means that it is not possible to know who you are physically talking to, only their avatars identity. In a virtual world that deals with real-world monetary transactions, this is highly desirable.
EIGC is also being developed for use on smartphones running the Android operating system.

Chapter 2 – How EIGC works

2.1 – The big picture

Below follows a conceptual view of the architecture of the EIGC solution.



[7]

Starting on the client side, there are two main components; the game client itself, and the EIGC client. The game client is typically a videogame developed by third-party developers who want to integrate the EIGC service. It is communicating with a game server which enables interaction between several users by receiving and relaying data between them. This is typically the way most videogames implements their multiplayer features.
The EIGC client is the module which will communicate with the IMS. The first task is to setup a voice session, such as a person-to-person chat with another user or to send and receive proximity media streams. In order to do this a protocol called SIP is used to establish and manage the session. Through SIP messages the client can for example register itself with the IMS, join proximity session or start a person-to-person call with another user.

The actual audio streams are transferred using RTP, which is a protocol well-suited for VoIP applications. RTCP (see chapter 2.2.2) is also used to manage the RTP stream by gathering statistics such as packet loss, but more importantly to specify from what clients you want RTP streams in proximity and team chat. The client will typically receive multiple RTP streams in a session as each stream corresponds to a unique client. For example, a user communicating with three other users will receive three audio streams.
Rendering audio is done through OpenAL[8] which is a cross-platform API designed for efficient rendering of 3D positional audio. The audio streams are mixed together with the relative positioning of each client and then played out by the OpenAL engine.
It is important to point out that EIGC is a separate process from the game client and therefore handles functionality like audio playback and connectivity independently of the game client. The important information needed from the game is the statistics of the users, such as positioning, direction and orientation relative between the users to allow realistic 3D audio mixing, and also with whom the users want to communicate with.

2.2 – How the protocols work

In computer networks, a *protocol* is a standardized way of communicating over a network. Different protocols provides different features, making some more suitable at a task than others. The internet today implements the TCP/IP suite which contains a set of protocols commonly used to communicate over the network. A more complete description of the standardized protocols can be found in *Computer Networking*[9] by James F. Kurose and Keith W. Ross.

2.2.1 – UDP: Along with the Transport Control Protocol ("TCP[9]"), the User Datagram Protocol ("UDP[10]") is one of the oldest protocols and is the foundation upon which many other protocols build. In contrast to TCP, UDP is a connectionless and lightweight protocol aiming at providing as little overhead as possible and therefore only implements the basic features needed to provide routing of packets between end-points without any retransmission or throughput control. This makes it attractive for time-critical and real-time applications such as videogames and VoIP applications which are dependent on fast routing.
The protocols presented below are all built on-top of UDP packets.

2.2.2 – RTP/RTCP: Building upon UDP, the Real-Time Protocol ("RTP[11]") seeks to add additional control to UDP packets by implementing features such as sequencing and timestamps. This allows an application to compensate for packets loss and jitter *(variations in the delay)* giving the possibility for more accurate playback of sound and is therefore commonly used in VoIP solutions. It is often used in conjunction with the Real-Time Control Protocol ("RTCP") which provides feedback and statistics. As per the standard, RTP sockets are bound to an even port number, while the RTCP socket is bound to the next highest uneven-port number.

RTCP packets can also be extended with application-specific data. In this solution, an important set of functionality has been added to RTCP; it is used to manage proximity and conference based chat sessions by carrying service data in its payload.

Some of the commonly used extensions are:

- Join: A request to join an existing proximity or conference service.
- Add: This is used to request RTP streams from other user(s) in the service.
- Delete: This request is used to stop receiving RTP streams from the specified user(s).
- Alive: Every fifth second an Alive message is sent to the server to notify it that the client(s) is still connected. Likewise, the server sends out Alive messages to connected participant(s) to inform them the service is running as well.
- Leave: Used for signalling the server that you are leaving the service.

When requesting audio streams you need to specify the ID of the requested user and the stream priority. The ID is called the *SSRC* and is unique to each client.
For example, when a user has twenty avatars around him; the closer an avatar is to the user, the more important it is that the user can hear him, and therefore he has a higher priority. The user would then send an RTCP request to add RTP streams from the twenty avatars but the user's bandwidth only allows for streaming from ten and therefore he will only receive streams from the ten avatars closest to him.

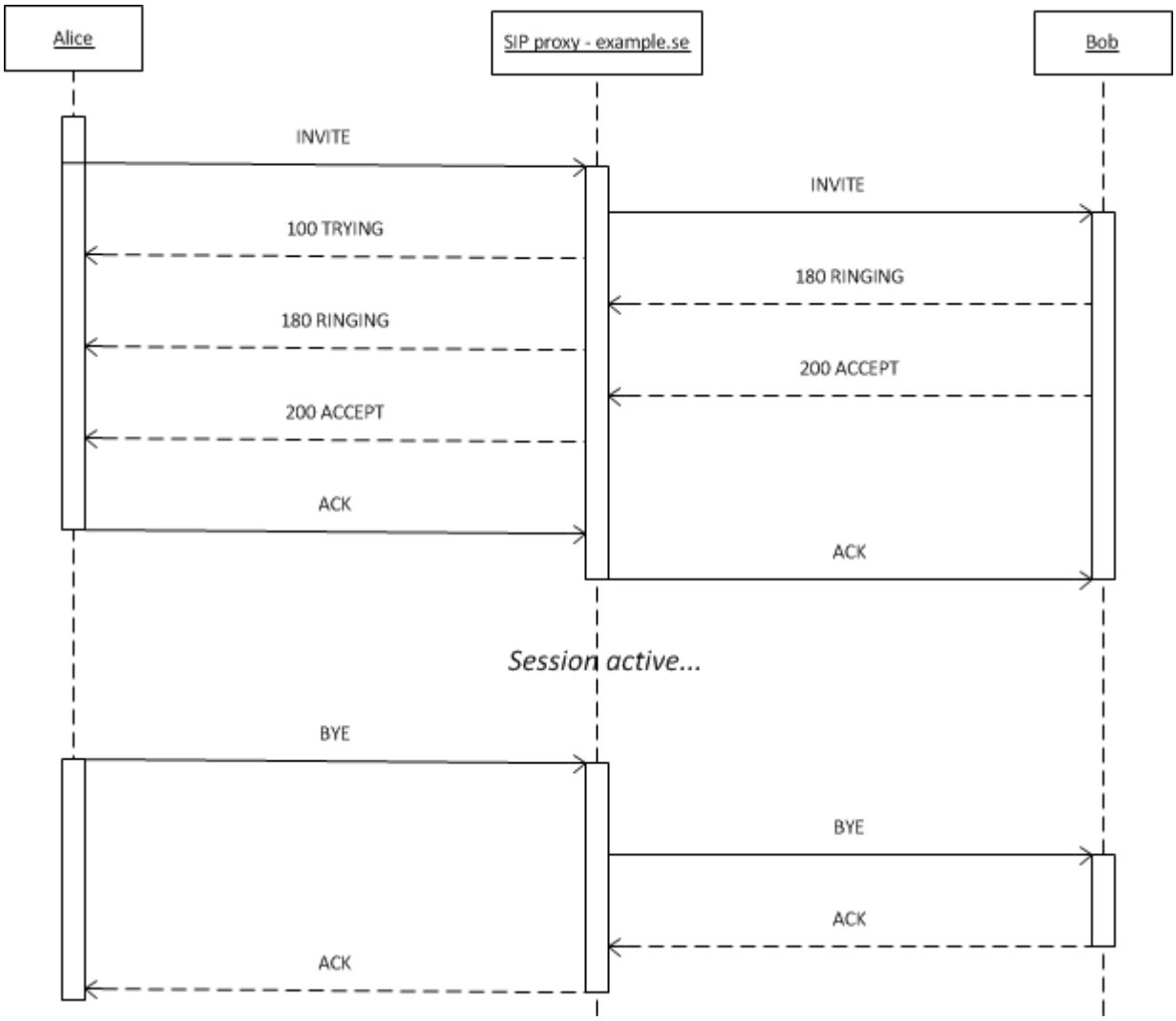
2.2.3 – SIP/SDP: The Session Initiation Protocol ("SIP[12]") is used to establish and manage media sessions between clients. Different messages can be passed through the system, such as a request to call a user or to register the current user with the SIP service. SIP solutions commonly deploy entities such as a *registrar server* (A server which registers users to the SIP service) and *proxy server* (A server which acts as an intermediate node in SIP communication).
SIP messages come in two kinds;

- Requests; sent from client to the server, which then routes it to the recipient. The first field of a request message states the type of the request and the second field states where it should be sent. Typical request methods are:
  - INVITE: A request to initiate a call. In this solution it can be a point-to-point, proximity or team chat call.
  - ACK: Confirms the response from a previous INVITE message.
  - BYE: Terminates a call.
  - REGISTER: Registers the client with the service.
- Responses; sent from server to client. The first field contains a numeric response code indicating how the request was handled. The response codes are divided into different value ranges depending on their nature. Typical response codes are:
  - 1xx: Provisional messages that indicate progress of a request, such as 'ringing' (180).
  - 2xx: Indicates success. For example 'OK' corresponds to 200.
  - 3xx: Indicates redirection: The request user cannot be reached directly.
  - 4xx: Indicates failure on the client side, such as a bad request.
  - 5xx: Server failed to handle a valid request
  - 6xx: Indicates global failure: Such as the requesting user is currently busy.

SIP INVITE requests can also carry additional information in a body section to modify the meaning of the message. The Session Description Protocol ("SDP") is used to describe the format of the media stream, such as codec's used and bandwidth limit, and is carried in the body of an INVITE message when negotiating how the RTP streams are to be set up.
An important extension to the SDP standard is the addition of the "SSRC" field; it advertises the users SSRC to the server and is the users voice-chat ID in the Ericsson IMS – it is the same SSRC used in RTP/RTCP packets, see 2.2.2.

2.2.4 – Example: Session establishment and termination

The following picture shows the SIP call flow of setting up a session and later terminating it.



The above scenario is an example of when the registered user Alice is trying to establish a multimedia session with another registered user, Bob. A SIP proxy is used to relay SIP messages between the clients.

- Alice sends an INVITE to Bob, indicating that she wants to establish a session. The proxy relays the INVITE message to Bob while returning a TRYING message to Alice. The INVITE message also carries session information such as possible media protocols and formats to be used.
- Bobs SIP client sends a RINGING message indicating that Bob has received the INVITE request and is no waiting for input from Bob.
- Bob accepts the INVITE request and sends an ACCEPT message back to Alice. The ACCEPT message also contains a description of the media streams Bob is willing to accept.

- Alice sends an ACK to signal she has received the ACCEPT message and accepts the media stream format. This is the final part in negotiating a session. The session between the clients is now active.
- Alice sends a BYE message indicating she wants to terminate the SIP session. Bob receives the message and replies with a final ACK. The SIP session has now been gracefully closed.

2.4 – Ericsson IMS server

IMS[13] is a standardised architecture for offering multimedia services over IP for all forms of telecommunication. It is a solution which supports a multitude of devices, making it possible for users on different platforms to establish multimedia sessions between each other. The only requirement for connecting to the IMS service is to run IP and SIP protocol stacks. Other technologies such as analogue telephones can connect to the IMS with the help of gateways that use SIP for them. It also aims to give third-party developers a reliable platform on which they can build their multimedia services.
The Ericsson IMS acts as a proxy for all communication between the clients. It provides a SIP registrar and proxy through which clients can register and set up multimedia sessions between each other. The audio streams are also sent to the IMS, by which they are then are relayed to the clients requesting them. This means it is not possible for the end user to directly see who he is communicating with except for the username provided by the service, because all packets are routed to the IMS rather than directly to the recipient. This is a desired feature for many applications which require a secure communication line which cannot be intercepted.

2.5 – Audio with OpenAL

The EIGC solution utilises an open-source, platform-independent audio API called OpenAL[8] for rendering of sound. The library was specifically designed to excel at 3D audio mixing. There are two objects that have to be initialised when running OpenAL:

- ALCDevice: On a PC system there may exist numerous audio-rendering devices. One of them has to be specified for use in OpenAL.
- ALCcontext: Once a device has been selected, a context for that device is created as well.

In addition, there are three major components used when managing sound in OpenAL:

- Listener: A listener is a unique object in each OpenAL instance. The listener represents the users location.
- Sources: A source is used to identify where a sound originates from.
- Buffers: Each source has a number of buffers attached to it, where each buffer contains sound samples. The buffers are queued and played out in order. The client has to continuously feed the queue with buffers containing new data to be played.

Each client the user is communicating with is represented by a source in the OpenAL plug-in. The audio data retrieved from the RTP packets will be placed in a buffer of the source representing the client who sent the data. As more packets are received, additional buffers are queued and eventually de-queued once the contents have been played.

Each packet contains 20ms of voice data. In optimal conditions, packets will arrive frequent enough to not cause an interruption; if that occurs however, silence is attached to the queue while waiting for additional packets. Silence is represented in binary by a trail of zeroes. Due to jitter, RTP packets may also arrive out of order or too late to be of use in which silence is also attached to the queue.

Besides having sound buffers, each source also has several important properties used to provide a realistic 3D audio mixing. Some of them are:

- Position(x,y,z): The current position of the source.
- Direction(x,y,z): The direction the source is facing.
- Velocity(x,y,z): The movement of the source, if any.
- Cone properties: If the sound of the source is projected within a specific angle.
- Reference distance: The distance from which the volume of the source will rapidly decline.
- Max distance: The furthest point at which the source can be heard.

Even services such as person-to-person and team-chat use 3D mixing consistency in audio playback; however, they use the origin as the positional attributes for all clients, thereby removing any 3D effect. Those services are meant to emulate popular VoIP clients such as Ventrilo[14].

Chapter 3 – Web browser research

3.1 – What is a web browser

A web browser is a software application whose primary function is to retrieve and display web pages on the World Wide Web (WWW[15]). Web pages are typically written in the HyperText Markup Language (HTML[16]). Transfer of HTML code can be done using several different network protocols; the predominant today is the HyperText Transfer Protocol (HTTP[17]). The web browser locates web pages based on the Uniform Resource Locator (URL[18]) provided by the user and then retrieves and renders them.
Most web browsers aren't limited to rendering of web pages; many also allow access to other resources such as downloading files through the File Transfer Protocol (FTP[19]), playing audio/video or accessing local files through the "File" prefix. In addition most web browsers support plug-ins which allows for example Java Applets or Adobe Flash Player to run. Most also have support for JavaScript, which is a scripting language for enabling dynamic content and interactivity on the web, rather than a static website created entirely through HTML. JavaScript alone is not enough to implement EIGC however and therefore other technologies are needed as well.

3.2 – Adobe Flash

One of the technologies looked into is Adobe Flash. Flash content is rendered using Adobe Flash Player, which is a popular multimedia plug-in for web browsers, enabling audio, video and interactivity. The development language for Flash is called ActionScript ("AS").
The power of Flash lies in its widespread usage; Flash may be seen as the industry standard for multimedia features in web browsers. This means it is compatible with nearly all popular platforms and devices on the market which makes it an attractive platform for development.

The problem with Flash however is the lack of open-source frameworks for protocols such as SIP and RTP. At the time of writing no framework is available for use. For network communication, AS implements XML Sockets, which are based on the TCP. As the name implies its primary use is with XML, a mark-up language not unlike HTML. This does not make it particularly suitable for audio streaming.
With the introduction of Flash 10, Adobe introduced a new service called Cirrus (previously codenamed Stratus). Cirrus is a rendezvous service hosted by Adobe which aids in setting up Peer-to-Peer (P2P) sessions between clients connected to the service. P2P enables direct communication between users bypassing the need of having media being relayed through a server beforehand.
In order to do this, Adobe released a protocol called the Real-Time Media Flow Protocol ("RTMFP[20]"). As this is a proprietary protocol and patented by Adobe, no details are available as to how it is constructed.
While this P2P solution attractive in some regards, it does not coincide with Ericsson's vision of having everything going through their IMS server and is therefore not applicable in this assignment.

3.3 – HTML 5

HTML 5 is the new revision of the HTML standard, which is the dominant mark-up language of the web today. One of the biggest changes is the introduction for multimedia features such as video and audio into the syntax of the language. This bypasses the need to use plug-ins such as Flash for multimedia solutions on the web.
HTML 5 is still under development. Some of its features are present in more or less stable releases today, although nothing is final as of yet. Some websites have begun to introduce HTML 5 today; noteworthy users are for example YouTube and JustinTV.
While it is an interesting piece of technology, not being in a release state and general lack of development information makes it a less attractive platform today, although certainly promising for future projects.

3.4 – ActiveX Controls

ActiveX controls are small programs that can be integrated and run in Microsoft's Internet Explorer. It can be developed in any language which supports Microsoft's Component Object Model ("COM"). In practice it is typically written in C++ or Visual Basic.
An ActiveX control, once started, has unrestricted access to the computers resources. This means there is very little an ActiveX control cannot do which gives it a lot of potential but has also proven to be a major security risk due to malicious ActiveX controls installing viruses. To counter this, ActiveX controls have to be digitally signed by the developer. This certificate will then be presented to the user, who then has the option to allow or deny the ActiveX controller from starting.
The total control given by ActiveX certainly makes it possible to implement the EIGC solution. There are some downsides though which makes other options more appealing; First of all, ActiveX controls are strictly limited to Internet Explorer on the Windows platform; no other web browser or operating system natively supports it. While Internet Explorer is still the dominant web browser on the market, it still only allows you to target a portion of the total audience. Secondly, it relies on the older COM technology which is slowly being phased out of the market.

3.5 – Java Applet

A Java applet is a program compiled to Java byte-code which can then be run in a web browser as a plug-in. The most common programming language is Java, although any language which compiles into Java byte-code is applicable, for example Jython[21]. Java byte-code is the instruction set that a Java Virtual Machine ("JVM[22]") translates to the hardware architecture of the local computer and then executes. An applet therefore requires the Java Runtime Environment ("JRE[23]") to be installed on the local computer.
Applets are run in a sandboxed environment, meaning that by default it cannot access local resources. This can be circumvented by digitally signing the applet and then giving the user the option to either allow or deny local access. In this regard an applet is very much like an ActiveX control.
One attractive feature with applets is that since the program is compiled into Java byte-code, it is portable to any platform able to run a JVM, which is available on all major platforms

today, unlike an ActiveX control which can only be run on Windows. It is also supported by most web browsers.
Another interesting feature is that since applets can be written in Java, the developer will have access all the Java libraries available to standard Java applications. This opens up a lot of possibilities. One of the interesting libraries is the Java Native Interface ("JNI[24]") library which allows the developer to load libraries written in native languages such as C/C++ in addition to the Java libraries. Those libraries are not restricted by the Java sandbox, giving the developer more control over local resources. To utilize the JNI, the applet needs to be digitally signed, much like an ActiveX control.

3.6 – Conclusion

For this solution, a Java applet is the platform of choice. The key feature is the access of local resources, which is an assurance that the solution can be implemented one way or another. An applet functions very much like an ActiveX control except it also adds greater flexibility in the form of portability and the potential use of the Java libraries for the project. Applets can also interact with JavaScript and vice versa, meaning it does not have to be a static object or a stand-alone application on a webpage.
There are also a few options that were not fully explored due to time constraints, most notably Microsoft Silverlight and Oracles JavaFX, although from an early look they did not seem quite as applicable as the other technologies brought up.

Chapter 4 – Doubango

4.1 – Doubango overview

Doubango is a light-weight open-source framework designed specifically for use with IMS services. It is written in ANSI-C and designed to be resource-efficient and portable, enabling use on several platforms.
The framework is composed of several components; some which can be run independently and some which depends on others to function. The following is a brief list of libraries within the Doubango framework that this solution utilizes; it is a subset of the complete framework and therefore does not cover all the libraries.

- tinySAK(tsk): This is the master library which contains the core functionality of Doubango. As ANSI-C is not an object-oriented programming language by default, one of the most important things it implements is the ability to define and create objects. An object in Doubango is a C-struct that contains an object-definition struct which holds information such as size and constructor /destructor method pointers. Objects can then be instantiated and destroyed just like any object-oriented programming language.
  Besides objects Doubango also implements a wide range of commonly used tools such as linked lists, thread- and memory management and encryption for cross-platform use.
  Dependencies: None.
- tinyNET(tnet): tnet contains all network functionality used in Doubango, more specific socket management and NAT-traversal techniques.
  Dependencies: tinySAK.
- tinyHTTP(thttp): This library manages a HTTP/HTTPS stack which allows the use of said protocols.
  Dependencies: tinySAK, tinyNET
- tinyIPSec(tipsec): tipsec enables secure network communication.
  Dependencies: tinySAK
- tinySDP(tsdp): As the name suggests tsdp provides all functionality related to SDP messages, whose functionality is explained in 2.2.3.
  Dependencies: tinySAK
- tinyMEDIA(tmedia): tmedia contains all media-related components such as audio/video engine plug-ins and codecs. One of the important features is the ability to create and include custom audio/video engines and codecs. This is important since OpenAL it not included as an audio engine by default and is required for 3D audio mixing.
  Dependencies: tinySAK, tinyNET, tinySDP
- tinySIP(tsip): tsip contains a fully-functional SIP stack that together with the libraries mentioned above, forms a complete IMS stack, making it possible to connect and communicate with any IMS.
  Dependencies: tinySAK, tinyNET, tinySDP, tinyMEDIA, tinyHTTP, tinyIPSec

4.2 - Object-oriented programming with Doubango

ANSI-C is not an object-oriented programming language by default. Doubango rectifies this by allowing the defining of objects that can be created and destroyed with ease. Each object has properties as would be expected of them to; constructor, destructor, comparator, inheritance, etc. All these are defined by the programmer and the first three mentioned above are mandatory when declaring an object. All memory allocation is handled by Doubango; tsk_object_new(…) is used to create a new object and the TSK_OBJECT_FREE(..) macro is used to destroy objects.

4.3 – SIP with Doubango

One the most important functionalities of Doubango used in this solution is the *SIP stack*. The SIP stack is a defined object used to manage all SIP-related functionality.
A very handy feature is the optional but highly recommended SIP call-back method that can be specified upon creating the stack; all SIP- and media-related events will send status messages to that method. It provides a centralised and structured way of communicating with the Doubango framework.
The SIP stack manages several *SIP sessions*. Each session is tied to one recipient, who is identified by the "Call-ID" or "To" header in SIP messages.

Doubango also provides the means to customise all outgoing SIP messages by adding or modifying the sip headers. Customisation is done at three different levels;

- Stack-level: Adding headers at stack-level will cause all outgoing SIP messages to incorporate that header. As the stack-level is higher than session-level, this will affect each session and consequently affect each request.
- Session-level: By adding headers at session-level, each request belonging to that session will add that header.
- Request-level: Adding headers to requests will only make them valid in that SIP request dialog. Requests – more commonly referred in Doubango as *actions* – are described in 2.2.3.

4.4 – Media with Doubango

Each SIP session is assigned a *media manager,* which in turn contains one or more *media sessions.* A media session can be of several types; for example it can be an *audio*, *video*, *audio/video* or *file* session. In this case, each media session is an audio session, as this is a voice-chat service.
Going further down in the hierarchy, each audio session contains the following important objects;

- RTP manager: This object manages everything related to RTP and RTCP. It also contains a transport manager, used to manage sockets and other connection-related data.
  The RTP manager will also build RTP packets, send and retrieve them. Packets that are retrieved will be disassembled and sent to the Decoder object. The RTP header, its payload and the size of the payload are passed as parameters for decoding.
  As of v1.0.0 of Doubango, RTCP was not implemented. More on how the RTCP issue was solved in chapter 5.2
- Encoder: The Encoder is given raw voice data by the Producer and then encodes it depending on what codec is being used by the media session. It is then sent to the IMS through the bound RTP socket.
  By default, only the AMR codec is compatible with Ericsson IMS. Future plans are to add G719 as well as it is also supported in the Ericsson IMS and it carries significantly better voice quality at the price of larger RTP packets.
- Decoder: The Decoder receives RTP packets from the RTP manager and decodes the payload using the codec specified by the media session and then passes the raw voice data to the Consumer.
- Consumer: Receives raw voice data from the Decoder and then passes it onto its Jitterbuffer object which will calculate and compensate for any potential jitter issues. More about jitter in 2.2.2.
  Once it has been passed to the Jitterbuffer the Consumer can then request audio samples back. This data can then be inserted into the audio engine used and played out.
- Producer: This object will capture audio samples and send it to the Encoder.
- Denoise: Denoise is used for post-processing of sampled audio data. It provides several features such as echo-suppression and Voice Activity Detection (“VAD”).

All the above mentioned objects except the RTP manager can be defined by the programmer, although Doubango comes with a set of default ones – for example, Producer/Consumer on Windows is DirectSound, which is a part of DirectX. It is important to point out that Producer and Consumer, Encoder and Decoder does not have to be the same engine or object. It is for example possible to record audio through DirectSound and play received samples using OpenAL.

5 - Implementation

5.1 – The Java Applet

The idea is to develop a Java wrapper that can be used in an applet. This wrapper acts as a bridge between Java and the Doubango interface; the wrapper exposes several methods that link to their respective in the native code, where the actual processing is done. Therefore in practice the Java part is little more than a shell for communicating between the applet and the native library. The wrapper is required because by default the applet cannot make explicit function-calls to the methods implemented in the interface.

This communication between Java and the interface is achieved through the Java framework called Java Native Interface ("JNI"). Being able to use native code can be beneficial due since the standard Java libraries may not cover all functionality for each platform. Another feature is that native code can create and manipulate Java objects, and vice-versa. It is important to note that the automatic garbage collection in Java does not apply to objects allocated in the native code, so it is up to the developer to de-allocate memory manually to avoid memory leaks.
As Java is run in a virtual machine – the JVM – it runs in a sandbox environment which provides a lot of functionality such as garbage collection at the expense of performance. By manipulating Java objects in native code instead you are not affected by this so for some time-sensitive applications it could be beneficial. Although not used that much in this solution, it is nonetheless an interesting feature.
In order for an applet to use JNI, it has to be digitally signed using the provided tools by the Java SDK and the user must explicitly allow it to run. Native programs have no restriction on what they can do and can therefore pose a potential security risk. It is the very same reason that makes applets so powerful with the use of JNI, however.

An applet is run as a separate object on a webpage. The webpage can communicate with the applet, such as calling methods or reading variables, using JavaScript. In the solution the applet was represented with a GUI as it needed a simple test client, but in practice the applet doesn't even have to be visible at all.
The applet is loaded and run when the user visits the webpage. In order to run Doubango you need to do a one-time download of the related files, since they have to exist on the local computer in order to run it. How this is actually done is up to the developer, but a suggestion is to allow the applet itself to extract the files to the local computer, as it has full access rights.

5.2 – The Doubango interface

The Doubango interface acts as a bridge of communication between the applet and the Doubango framework. The interface provides a high-level API which is used by the applet. Below are some of the calls exposed by the interface:

```
// high-level API method declaration
// return value is 1 for success, 0 <= for error.
int init();
int cleanup();
int createUser(string user, string pw);
int register();
int unregister();

int callP2P(string user);
int acceptCall();
int denyCall();
int endCall();

int startProximity();
int addSSRC(soundSource ss);
int deleteSSRC(soundSource ss);

// misc. options/calls
int setDomain(string domain);
int setRegistrar(string registrar);
// ........
```

The interface performs all communication and coordination with the framework. In addition, it also implements some features that are critical for IMS which Doubango does not do.

RTCP was not implemented in the v1.0.0 of Doubango. While RTCP is not required for person-to-person, it is for proximity and team chat. Therefore the interface implements an RTCP manager which can create and send RTCP messages to the IMS server.
As a layer above, a proximity manager was also implemented. Its purpose to handle everything related to the proximity service, including the RTCP manager. Besides managing the RTCP flow, it also provides the Doubango OpenAL plug-in with data required for 3D audio mixing.
For example, suppose the user receives from the game server that a new player has joined the proximity session. The proximity manager will then have to create and send an ‘Add’ RTCP message regarding the player in addition to adding the new players position to the OpenAL plug-in.

5.3 – Doubango extensions

This solution takes advantage of the fact that you can easily integrate custom plug-ins into Doubango as described in 4.3.3.
For example, in this solution an OpenAL Consumer object was defined as it is required for 3D audio playback. The object definition of this Consumer looked like this;

```
// OpenAL consumer object
typedef struct tdav_consumer_openAL_s
{
            TDAV_DECLARE_CONSUMER_AUDIO;

            tsk_bool_t running;
            void* tid;              //thread ID

            ALCdevice* device;
            ALCcontext* context;
            tsk_list_t* soundSourceList;

}
tdav_consumer_openAL_t;
```

Amongst OpenAL data such as ALCdevice and ALCcontext, the object definition also contains the 'TDAV_DECLARE_CONSUMER_AUDIO' macro. This declaration means that it inherits the base-class for Audio Consumer objects in Doubango that contains data such as bits per sample, number of channels and sampling rate of the audio session. The values for these variables are inherited from the media session, as it is negotiated during the SIP dialogs. As mentioned in 4.3.1, all objects must also implement mandatory methods such as constructor, destructor and comparator. As an example, the OpenAL Consumer constructor method is shown below.

```
// constructor
static tsk_object_t* tdav_consumer_openAL_ctor(tsk_object_t * self, va_list * app)
{
            tdav_consumer_openAL_t *consumer = self;
            if(consumer){
                        tdav_consumer_audio_init(TDAV_CONSUMER_AUDIO(consumer));

                        consumer->tid = tsk_null;
            }
            return self;
}
```

Besides an object definition, one has also has to declare the plug-in definition, where properties such as the plug-in media type, the object definition and mandatory methods are specified.

```
//plug-in definition
static const tmedia_consumer_plugin_def_t tdav_consumer_openAL_plugin_def_s =
```

```
{
		&tdav_consumer_openAL_def_s,

		tmedia_audio,
		"OpenAL consumer",

		tdav_consumer_openAL_set,
		tdav_consumer_openAL_prepare,
		tdav_consumer_openAL_start,
		tdav_consumer_openAL_consume,
		tdav_consumer_openAL_pause,
		tdav_consumer_openAL_stop
};
const tmedia_consumer_plugin_def_t *tdav_consumer_openAL_plugin_def_t =
&tdav_consumer_openAL_plugin_def_s;
```

The ‘tdav_consumer_openAL_*’ are memory addresses of methods that are mandatory to specify for each Consumer plug-in.

- tdav_consumer_openAL_set: A null-pointer; not used.
- tdav_consumer_openAL_prepare: This method is called to signal the Consumer that a session is being negotiated. Typically all initialisation of the Consumer is done in this method, such as initialising the audio engine.
- tdav_consumer_openAL_start: Called when the session has been established. Typically most Consumers use a separate thread to continuously poll the Jitterbuffer for audio samples every 20ms for playback; that thread is started in this method.
- tdav_consumer_openAL_consume: This call-back occurs every time an RTP packet has been received by the RTP manager and decoded. The parameters given to this method is the RTP header, the decoded payload and it’s length. Here any pre-processing of the audio data can be done before it is inserted in the Jitterbuffer.
- tdav_consumer_openAL_pause: A null-pointer; not used.
- tdav_consumer_openAL_stop: When the session has terminated, this method is called. It stops the running thread and frees all resources allocated by the Consumer object. When it returns, the Consumer object itself is destroyed.

Below is an example showing how the ‘tdav_consumer_openAL_prepare’ method was implemented.

```
//
// prepare OpenAL
//
int tdav_consumer_openAL_prepare(tmedia_consumer_t* self, const tmedia_codec_t* codec)
{
		tdav_consumer_openAL_t* openAL = (tdav_consumer_openAL_t*)self;

		ALfloat listPos[] = {0.0,0.0,0.0};	// default positions
		ALfloat listVel[] = {0.0,0.0,0.0};
		ALfloat listOri[] = {0.0,0.0,1.0};

		if(!openAL){
			TSK_DEBUG_ERROR("OPENAL: Invalid parameter");
			return -1;
		}
```

```
		// needed by doubango
		TDAV_CONSUMER_AUDIO(openAL)->channels = codec->plug-in->audio.channels;
		TDAV_CONSUMER_AUDIO(openAL)->rate = codec->plug-in->rate;

		// get default device
		if (!(openAL->device = alcOpenDevice(NULL))) {
			TSK_DEBUG_ERROR("OPENAL: Unable to open default device");
			return -1;
		}

		// create context
		if (!(openAL->context = alcCreateContext(openAL->device,0))) {
			TSK_DEBUG_ERROR("OPENAL: Unable to create context");
			return -1;
		}

		// activate context
		if (alcMakeContextCurrent(openAL->context) == ALC_FALSE) {
			TSK_DEBUG_ERROR("OPENAL: Unable to activate context");
			return -1;
		}

		// set default lister properties
		alListenerf(AL_GAIN, 1.000);
		alListenerfv(AL_POSITION, listPos);
		alListenerfv(AL_VELOCITY, listVel);
		alListenerfv(AL_DIRECTION, listOri);

		// create the sound source list
		openAL->soundSourceList = tsk_list_create();

		return 0;
}
```

Plug-ins for Producers, Encoders, Decoders and so on works on the same principles as the Consumer example above. In this solution only the OpenAL Consumer needed to be added to Doubango; due to time restraints, the G719 codec could not be added.

5.4 – Doubango source modifications

While a basic person-to-person can be implemented straight out-of-the-box with Doubango, there are some tweaks that had to be done to the source code in order to achieve total compliance with the Ericsson IMS. This especially applies to the proximity service which requires several modifications of the framework to run.

Most of the modifications involve minor things such as editing a variable here and there in the source. For example, the SSRC has to be manually added to each SDP message, see 2.2.3.

Other modifications are more critical, such as RTCP. Doubango will bind the RTCP socket, but beyond that it will not use it. Therefore it was necessary to write a method that exposes a handle to the RTCP socket which could then be used in the Doubango interface to implement the RTCP features.
Another important modification was to add a list of Jitterbuffers instead of just one buffer per audio session. Each element in the list corresponds to one SSRC. This enables support for multiple clients in one session; by default, Doubango is only intended for one-to-one

communication which is why it only has one buffer. Since proximity and team chat allows for more than one participant in the chat during the same session, there needs to be one additional buffer for each new client. If there was only one buffer, the voices of several clients would be mixed and play them as one sound source, which would not be correct.
When an RTP packet is received, the payload is decoded and then put into the Jitterbuffer which corresponds to the SSRC specified in the packets header. The OpenAL Consumer then retrieves audio data from the Jitterbuffer, adds it to the corresponding sound source and plays it.
Modification of the source should be avoided to make it easier to integrate new versions of the framework. At the time of writing, a version 2.0.0 of the framework was in the making. Unfortunately it cannot replace the modified v1.0.0 framework directly and therefore has to undergo the same changes as the previous version before it can be applied.

6 – Result and discussion

Porting the EIGC service to the web-browser certainly is possible. While there are alternatives, a Java applet provides an easy integration into the web and since there is no restriction on what a signed applet can do, any service could be ported to the web through applets.
As for Doubango, the framework provides a nearly complete way of communicating with the Ericsson IMS. While basic person-to-person works by default, for full compliance with Ericsson IMS and to implement proximity, some changes have to be done to the source code in the current state. Future versions may be able to provide the user with more options to customise the SIP stack and therefore remove the need to modify the source.

Approximately three-to-four weeks were spent on learning how EIGC works and researching on how to best implement the service in a web-browser. As for the implementation, person-to-person was up and running in a week's time. The big time sink was proximity, where most major components had to be written from scratch. At the end, proximity was working but not well enough. There were bugs that were hard to stomp out, one which specifically caused every fifth or so packet to be removed during the jitter calculations. It caused the sound quality to be barely acceptable, but it did work and the proximity features were in place.
As for the future, when presenting it to Ericsson they did sound pleased with it and using Doubango with a similar web-browser solution is definitely within the realm of possibility.

7 – References

[1]: *7th October, 2011: http://en.wikipedia.org/wiki/Voice_over_Internet_Protocol*
[2]: *7th October, 2001: http://en.wikipedia.org/wiki/Session_Initiation_Protocol*
[3]: *7th October 2011*: *http://en.wikipedia.org/wiki/Real-time_Transport_Protocol*
[4]: *7th October, 2011: http://en.wikipedia.org/wiki/IP_Multimedia_Subsystem*
[5]: *7th October, 2011: http://en.wikipedia.org/wiki/Entropia_Universe*
[6]: *7th October, 2011: http://en.wikipedia.org/wiki/Mindark*
[7]: *7th October, 2011: http://archive.ericsson.net/service/internet/picov/get?DocNo=1/28701-FGC1011257&Lang=EN&rev=A1*
[8]: *8th October, 2011*: *http://en.wikipedia.org/wiki/OpenAL*
[9]: *Computer Networking: A Top-Down Approach* (2010) 5th edition, by Kurose, J.F and Ross, K.W.
[10]: *8th October, 2011*: *http://en.wikipedia.org/wiki/Transmission_Control_Protocol*
[11]: *8th October, 2011*: *http://en.wikipedia.org/wiki/User_Datagram_Protocol*
[12]: *8th October, 2011*: *http://en.wikipedia.org/wiki/Real-time_Transport_Protocol*
[13]: *8th October, 2011: http://en.wikipedia.org/wiki/Session_Initiation_Protocol*
[14]: *8th October, 2011*: *http://en.wikipedia.org/wiki/IP_Multimedia_Subsystem*
[15]: *8th October, 2011*: *http://en.wikipedia.org/wiki/Ventrilo*
[16]: *8th October, 2011*: *http://en.wikipedia.org/wiki/WWW*
[17]: *8th October, 2011*: *http://en.wikipedia.org/wiki/Html*
[18]: *8th October, 2011*: *http://en.wikipedia.org/wiki/HTTP*
[19]: *8th October, 2011*: *http://en.wikipedia.org/wiki/URL*
[20]: *8th October, 2011*: *http://en.wikipedia.org/wiki/FTP*
[21]: *8th October, 2011*: *http://en.wikipedia.org/wiki/Real_Time_Media_Flow_Protocol*
[22]: *8th October, 2011*: *http://en.wikipedia.org/wiki/Jython*
[23]: *9th October, 2011*: *http://en.wikipedia.org/wiki/Java_virtual_machine*
[24]: *9th October, 2011*: *http://en.wikipedia.org/wiki/JRE#Execution_environment*
[25]: *9th October, 2011*: *http://en.wikipedia.org/wiki/JNI*