PROJECTE DE FI DE CARRERA

TÍTOL: Development of advanced multimedia services in P2P architectures

AUTOR: Alberto José González Cela

DIRECTOR: Antoni Oller Arcas

DATA: 9 de febrer de 2007

Títol: Development of advanced multimedia services in P2P architectures

Autor: Alberto José González Cela

Director: Antoni Oller Arcas

Data: 9 de febrer de 2007

Resum

La transmissió de fluxos multimèdia en temps real (streaming) s'ha convertit en un tema punter i de gran interès al món de la recerca d'Internet, especialment quan ens referim a aplicacions de transmissió d'àudio i vídeo en directe a través de xarxes peer-to-peer (P2P). Generalment, aquestes aplicacions han de fer front a molts problemes en el seu disseny i implementació deguts a la dinamicitat i heterogeneïtat que per natura caracteritzen les xarxes P2P. En aquest projecte, s'introdueixen noves característiques que les aplicacions de transmissió multimèdia P2P actuals no contemplen.

Els requisits de connexió i maquinari són diferents per a la transmissió de fluxos de baixa i alta capacitat, no obstant, tots els nodes s'acostumen a considerar idèntics, cosa que no representa una aproximació gaire encertada tenint en compte un medi tan heterogeni. A més a més, amb la finalitat d'aconseguir distinció entre nodes, es fa necessari la introducció d'un mecanisme que permeti l'intercanvi de les capacitats específiques de cada node, incloent-hi les de transcodificació de fluxos. Un altre aspecte a destacar és el fet que aquestes aplicacions són difícils d'ampliar, incorporar nous serveis o modificar les dades que porten precarregades com ara la llista de canals de televisió disponibles, cosa que impossibilita garantir la disponibilitat de la font tot el temps. Per altra banda, els serveis interactius tampoc s'han desenvolupat gaire.

Aquest projecte proposa el disseny i implementació d'una plataforma de difusió multimèdia P2P cooperativa i interactiva que permet superar els problemes esmentats. La plataforma integra diferents mecanismes que permeten la distribució en temps real de continguts multimèdia en diferents qualitats incloent fluxos d'alta capacitat (com per exemple HD). Aquesta plataforma és una solució novedosa basada en JXTA, DONET i ALM (Arbres Multicast a nivell d'Aplicació) que proporciona un sistema ampliable segons noves necessitats i facilita la inserció de nous serveis de valor afegit. La plataforma proposada es fonamenta en la creació d'una arquitectura de 2 capes lògiques superposades: una capa lògica JXTA, encarregada bàsicament de la senyalització i intercanvi de metadades, i una capa de transmissió basada en sockets UDP unicast. D'aquesta manera, la diferència entre la capa de transmissió i la capa física es pot veure reduïda a partir de la informació obtinguda de la capa JXTA, la qual es va actualitzant al llarg del temps.

Title: Development of advanced multimedia services in P2P architectures

Author: Alberto José González Cela

Director: Antoni Oller Arcas

Date: February, 9th 2007

Abstract

Media streaming has become a hot topic in Internet research, especially when it is related to live streaming audio and video applications and peer-to-peer (P2P) networks. In general, they have to cope with challenging problems in their design and implementation, due to the dynamic and heterogeneous nature of P2P networks. In this project, some new features that other P2P media streaming applications do not consider or barely specify are introduced.

The connection and hardware requirements are different if low bitrate or high bitrate streams are transmitted. However, all peers are usually considered identical, which is not a good approach considering a heterogeneous environment. Furthermore, in order to achieve distinction among peers, it is necessary to provide a mechanism that enables the exchange of the specific capabilities (including transcoding) of each peer. Moreover, these applications have difficulties to extend its features by adding new services or when trying to modify any preloaded data such as the channel list provided to the user, making impossible to guarantee the source availability all the time. Finally, interactive services have not been deeply discussed.

This project proposes the design and implementation of a collaborative and interactive media streaming platform that allows coping with the issues before mentioned. The platform integrates different mechanisms that permit real-time distribution of multimedia contents with different qualities including high bitrate media streaming (e.g. HD). This platform is a novel hybrid solution based on JXTA, DONET and ALM (Application Layer Multicast), providing a flexible and extensible platform according to new requirements and future value-added services. In addition, this platform allows creating an architecture with two overlay layers: a JXTA based logical layer, which is in charge of signalling and metadata exchange, and a UDP socket based layer, which is in charge of the multimedia transmission of unicast traffic. Consequently, the mismatch between transmission and physical layers can be reduced by means of the data obtained from the JXTA layer, which is updated through time.

INDEX

INDEX OF FIGURES

INDEX OF TABLES

CHAPTER 1. INTRODUCTION

Recently, Peer-to-Peer (P2P) networks have gained popularity thanks to the deployment of file-sharing applications. However, nowadays real-time media streaming applications arouse a great interest both to commercial level and academic research. Live streaming introduces new challenging problems [1], [2] different to ordinary file-sharing. In general, media streaming solutions have different features that determine the operation of the applications. For example: the large volume of media data along with stringent timing constraints, the dynamic and heterogeneous nature of P2P networks and the unpredictable behaviour of peers.

It is possible to emphasize some important issues closely related to P2P media streaming which are mainly associated to two aspects: collaboration between peers and interactivity. With regard to collaboration between peers, the following issues have to be taken into account: dynamic peer discovery, peer relationship maintenance and exchange of capabilities (transcoding information). Related to interactivity, the issues are the membership service and the messaging exchange.

From the operative point of view, when a peer starts a streaming P2P application, it has no information about other peers, which are also present in the virtual network. This implies that an efficient discovery mechanism must be initiated in order to find other peers and start receiving the desired media stream from the best ones.

Another key point is the maintenance of the relationship among peers and their update. Due to the heterogeneous environment of P2P networks and the unpredictable behaviour of peers (e.g. constant appearance and disappearance) stable and dynamic membership and presence mechanisms must be incorporated to know which peers are up, especially to ensure the state of the provider peers (partners) of a specific peer. Therefore, a peer must update its relationships to replace those partners that are down and, consequently, to not decrease the quality of the received stream.

The process of discovery is especially important when referring to live streaming applications because depending on the number and specific features of the found peers, the desired stream will be received under some conditions. In the worst case, it will not be received or it will be received with losses or delay, causing the rejection of the final user because of unacceptable visual quality.

P2P networks are composed by thousands of different peers, each of them with specific features, such as bandwidth, decoding capabilities or storage capacity. It is a main issue to select the providers that better fulfil the capabilities and requirements of a peer that wants to start receiving a media stream; therefore, a dynamic mechanism to advertise the concrete capabilities of each peer must exist. Once delivered a stream at a specific bitrate and codec, maybe not all of the peers in the network are able to receive this stream due to insufficient downlink bandwidth. This problem is solved if the streaming platform enables to advertise peers with adaptation or transcoding functions. For example, if a source is sending a High Definition stream of 20 Mbps (MPEG2-TS, 1280x720), maybe most of the peers with current Internet access (e.g. ADSL or Cable Modem) can not receive this stream delivered at that bitrate, but if a peer has transcoding capability, possibly this stream can be adapted to a smaller bitrate.

Regarding interactivity among peers, the existing P2P applications do not promote value-added interactive services among connected peers, currently known as social networks. When talking about terrestrial TV and radio broadcasting the interaction with the viewers or listeners is a hot topic. A possible way to interact with the audience is to make available an infrastructure to receive SMS, which can be expensive and complex. It is also usual to install devices dedicated to calculate the *share* gathered from a TV programme and determine the success of the programme according to the obtained data. P2P applications offer the possibility to create interactive experiences for the users by providing cheap and simple mechanisms to interact with the connected user. For instance, peers can chat at playing time. Using P2PTV, the broadcasters can get the feedback of online users at real time, the same way that TV broadcasters do in current TV channels with SMS, but with the difference that IP infrastructure is much cost effective. Statistical information such as number of current/average viewers, time of playback or the user profile can be directly obtained thanks to an efficient membership service.

Moreover, it must be mentioned that popular streaming P2P applications focused on delivering TV streams on the Internet, such as PPLive [3] or SopCast [4], have a default static list of P2PTV channels, which identify the well known media streaming sources. In spite of the fact that these lists can be updated, this process is inflexible because when a change in the contents is required, retrieval or addition of channels, the whole application must be updated online or, even, it is mandatory to download a new release of the whole application. This feature causes that these applications can not be easily extended.

A problem derived from providing a static list of channels is source availability. The fact that the list of channels is prefixed does not guarantee that they are fully available. In this sense, it is when a user wants to view a channel that the system checks if the source is available or not. In the case of being unavailable, the user has wasted its time trying to view this channel.

1.1. The Proposal: CIMS-Live

Taking into account this environment and with the idea to solve the issues before mentioned, this project proposes the design and implementation of a media streaming platform called: *CIMS-Live* (Collaborative and Interactive Media Streaming Platform). This platform is a hybrid solution based on the P2P Java API called JXTA [5], DONET (Data-driven Overlay Network) [6] and ALM (Application Layer Multicast) structure [7], [8]. Thanks to this platform, the publication and discovery of media streaming channels and peers can be fully decentralized and automated. It also allows delivering any kind of multimedia stream thanks to the cooperation among peers. This infrastructure makes possible to create an extensible application which allows incorporating new services. Furthermore, the flexible publication mechanisms provided by JXTA allow announcing the specific capabilities of the different peers present in the network and, consequently, finding the best peers that will become providers of critical services such as media streaming.

1.2. Document Overview

This document will first introduce in CHAPTER 2 the specification of the functionalities that CIMS-Live will provide to its users. In CHAPTER 3, the architecture of CIMS-Live is presented. Next, in CHAPTER 4, the design of the system is presented and the different elements that constitute the application are detailed. In CHAPTER 5, two P2P media streaming mechanisms used in the solution are described, which allow real-time distribution of multimedia content with different qualities. After that, in CHAPTER 6, a developed prototype and its development environment are described in order to check the different programmed functionalities and to validate the concepts on which the application is based. In CHAPTER 7 it can be seen the planning and the cost estimation of the project. Finally, CHAPTER 8 concludes this project suggesting future works and presenting the inferred conclusions.

1.3. Related Works

In recent years, there have been significant researches into a variety of issues related to P2P media streaming. However, it is necessary to mention that there are some popular media streaming applications whose internal operation is barely known; therefore, it is difficult to analyze the algorithms and mechanisms used. On the other hand, there are academic developments associated to universities or communities of developers that have been well specified and published. One example of this last type is STARCast [9], which is a JXTA based platform that offers streaming services.

STARCast uses an ALM structure for streaming tasks. But, opposite to CIMS-Live, it treats all the streams in the same manner, that is, it does not distinguish between low and high bitrate streams. In addition, there is no detailed information about its concrete operations.

Moreover, there are different strategies related to the selection of the partners and to the construction of an efficient overlay network for multimedia streaming. Next, some techniques applied or associated to this study will be mentioned.

Borrowing ideas from IP multicast technology, tree-based protocols can be considered simple, efficient and scalable. Specifically, the main goal of single tree protocols is to build a scalable multicast tree with high efficiency. A representative example of this is ZIGZAG [8], which is a P2P technique that allows the media server to distribute content to many clients, by organizing them into an appropriate tree rooted in the server. Basically, it is ALM tree that has height logarithmic with the number of clients. ZIGZAG deals with the problem of one source towards multiple destinations with consideration of network condition. The objectives are to minimize the end-to-end (E2E) delay, to manage user dynamicity and to keep the overhead traffic as small as possible to achieve scalability. ZIGZAG is a typical example of a Source-Driven scheme.

Another well-known example is DONet. DONet proposes a Data-driven Overlay Network for live P2P media streaming. The core operations in DONet are very simple and do not need any kind of complex tree structure for data transmission. Actually, every node periodically exchanges data availability information with a set of partners, and retrieves unavailable data from one or more partners, or supplies available data to partners. Authors show through analysis that DONet is scalable with bounded delay and also address a set of practical challenges for realizing DONet. An efficient member and Gossip based partnership management algorithm is proposed, together with an intelligent scheduling algorithm that achieves real-time and continuous distribution of streaming contents. DONET is currently implemented in a commercial application called CoolStreaming [6].

AnySee [10] is a P2P live streaming system based on an inter-overlay optimization scheme and where the resources can join multiple overlays. This system creates an overlay network of the peers in the application according to location-aware topology matching (LTM), but it supposes a prominent management effort.

1.4. Context

This project is interested in solving the problem of streaming live bandwidth-intensive media (from low to high bitrates) in P2P networks, that is, streaming from a single source to a large number of receivers. It is obvious that P2P techniques provide a scalable and robust solution to streaming tasks in comparison with pure centralized strategies which are based on a simple strategy where it is dedicated an individual connection to stream the content to each receiver. Centralized strategies consume tremendous bandwidth and require great processing capabilities, so it is nearly impossible for a service provider to serve quality streaming to large audiences while generating profits.

Nowadays, with the improvement of network bandwidth, multimedia services based on streaming have grown in popularity. IP multicast could be the best way to deliver multimedia streams but its deployment on the Internet is very limited due to practical issues of routers.

There are two main approximations to solve media streaming delivery on P2P networks. The first one is based on the construction of an Application Layer Multicast (ALM) structure that can be a tree rooted at source or a mesh. The other one is based on the segmentation of a media stream into blocks. Current

P2PTV applications implement mechanisms based on one of these two schemes.

Once constructed the multimedia streaming service, it can be provided interactive services to the users connected to the streaming platform. It is easy to develop new social networks that promote the relationship among peers thanks to the virtual connections created.

1.5. Objectives

As it has been mentioned on the context section, this project expects to solve the problem of delivering multimedia streams on a P2P network and the creation of interactive applications. This is why it is proposed a platform that allows the cooperation among peers in order to carry out heavy streaming tasks and also offers interactive services such as chats which promote the creation of social networks.

In order to achieve this, the main objective of this project is to study different strategies and technologies which will allow delivering any kind of multimedia stream using P2P mechanisms. These strategies are: ALM structures and buffer segmentation.

One objective prior to starting with the development of the study is to detail a prototype, specifying the functionalities it offers.

Moreover, it will be implemented a software prototype that will allow joining a P2P network, discovering media streaming channels and starting the playback of a stream using the studied strategies. The resulting application will be named CIMS-Live.

Once studied the different strategies and implemented the prototype, some conclusions will be inferred.

CHAPTER 2. SPECIFICATION

This chapter details the basic functionalities defined for the proposed P2P application named CIMS-Live.

CIMS-Live is a cooperative and interactive application. When it is mentioned that it is proposed a collaborative platform, it is focused on a P2P strategy that allows the different users to cooperate in order to provide multimedia streaming services, which would suppose a heavy task for a single user. The other key aspect is the creation of a platform that provides interactive services such as chat and videoconference in order to promote the relationship among peers.

CIMS-Live is extensible, so new functionalities can be added to the application when needed.

2.1. Functionalities

The functionalities of CIMS-Live are focused on the following logic elements.

- **Users**. Those peers which are present on the logical network of the application (CIMS-Live-Net) or join one or more groups in the network but are not on the list of contacts of a user. Other users can be found when joining the application or when joining a specific group. CIMS-Live is prepared to distinguish among 3 different types of users when being part of a streaming task. There are specified 3 user profiles.
  - **Simple Peer**. User that wants to play a channel.
  - **Source**. Owner of a multimedia group (channel), it is the original multimedia provider.
  - **Transcoder**. A user that receives a multimedia stream from the platform and can adapt it to another stream with some specific features according to concrete needs of other users.
- **Contacts**. Those peers that a user adds to its contacts list.
- **Groups and channels**. Associations of users according to specific affinities or interests. This project will be focused on a specific type of group which is *P2PTV or P2PRadio channel*. These groups are specific groups that peers can join in order to receive media streams. It is the same concept as “channel” when talking about terrestrial or satellite broadcasting.

The proposed functionalities are listed next.

2.1.1. Media Streaming

Any peer of the application can discover media streaming channels and can ask the system for starting its playback.

It is also possible to create media streaming channels to deliver any kind of media through the platform if the requirements of the user are enough. This case implies that the user becomes a streaming source into the channel group (see section 3.3 for deep review).

2.1.2. Videoconference

Any user with a webcam can start a videoconference with one of its contacts or more than one (multiconference).

Thanks to the shared calendar functionality (see section 2.1.3) videoconferences can be scheduled for a specific date and hour.

2.1.3. Shared Calendar

Those users that join this service can share a calendar among their contacts. The users can edit tasks and events, share the information of their list of contacts and arrange meetings or videoconferences.

2.1.4. Chat

All users of the application have a general chat (1:N chat) both in the application group and in every specific group or channel. All the users in a group can chat with the rest of the connected users in a single window.

It is also possible to create a user-to-user chat (1:1 chat) between two peers when they are added in their respective lists of contacts. Then, they can start private chat.

In addition, all the peers can specify their current state: away, busy, online, offline. The state has global view in the system.

2.1.5. File Sharing

All the peers can exchange files among their contacts (point-to-point) but it is also possible to share files among all the peers in a distributed scenario. So, peers can look for specific files that other peers have made available for download. The desired file is downloaded from multiple peers, in a typical P2P scenario.

2.1.6. Group / Channel Creation

Any user can create a group or media channel. These groups allow the peers to associate according to specific affinities and interests.

2.1.7. Advertising (Yellow Pages)

The P2P application allows the peers to introduce their own advertisements in the platform in a well classified manner. All the peers can discover these advertisements.

Moreover, the searches can be filtered according to specific interests of the user. These interests can be introduced in the application when configuring the preferences of the user.

2.1.8. Monitoring and Statistics (Backoffice)

The users of the application can know the state of the network according to the statistical data gathered by the application. A user can monitor the following Quality of Service (QoS) parameters.

- Packet loss
- Delay between two users
- Received and transmitted bytes
- Available bandwidth

Another functionality related to monitoring is to enable the request of these statistics by other peers on the network.

Monitoring is also important when talking about media streaming channels because it is possible to create statistics in order to know which the real share of a programme is according to some parameters such as number of viewers, peak time of users connected and so on. These important statistics can determine the success of a broadcaster and can be achieved in an easier way when using an IP scenario in comparison with terrestrial or satellite TV broadcasting. Some monitored parameters can be translated to other more abstract parameters very useful for streaming providers. An example of this would be to obtain the number of viewers of a stream when knowing the number of opened connections for transmission. Note that this parameter has no sense in a P2P scenario because the source is not the only peer that delivers a multimedia stream. In a P2P network, this parameter would be mapped to the number of peers joining a specific channel.

2.2. Technical Specification

The application interacts with different modules (Fig. 2.1) in order to offer the functionalities above mentioned (section 2.1.). The services that offers the proposed application are contained in a module called “Services”, which includes:

- Multimedia
- File Sharing
- Monitoring
- Shared Calendar
- Yellow Pages

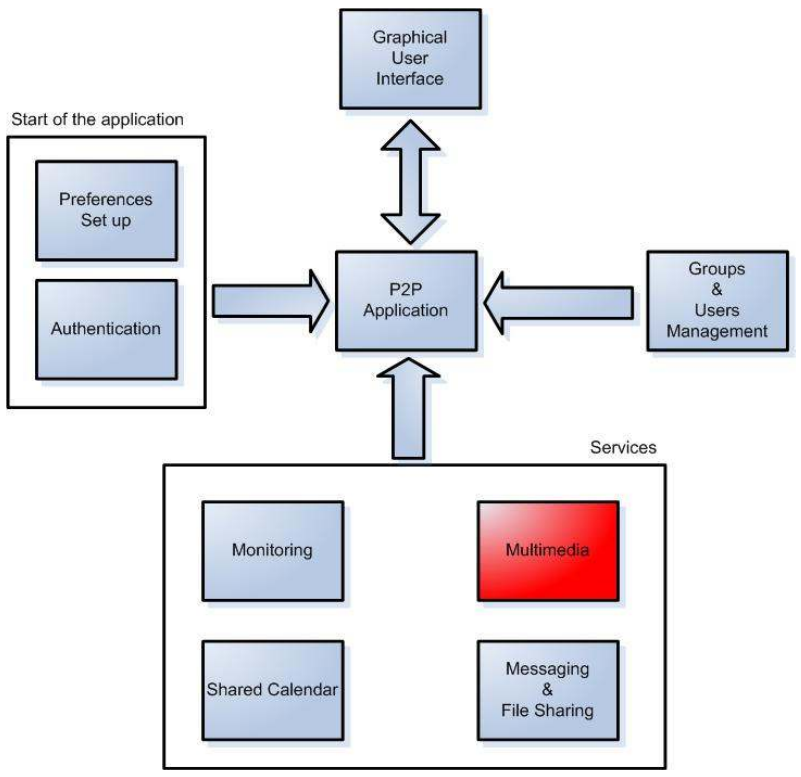


Fig. 2.1 Modules of the application

Fig. 2.2 shows the main services offered to a user by the application.

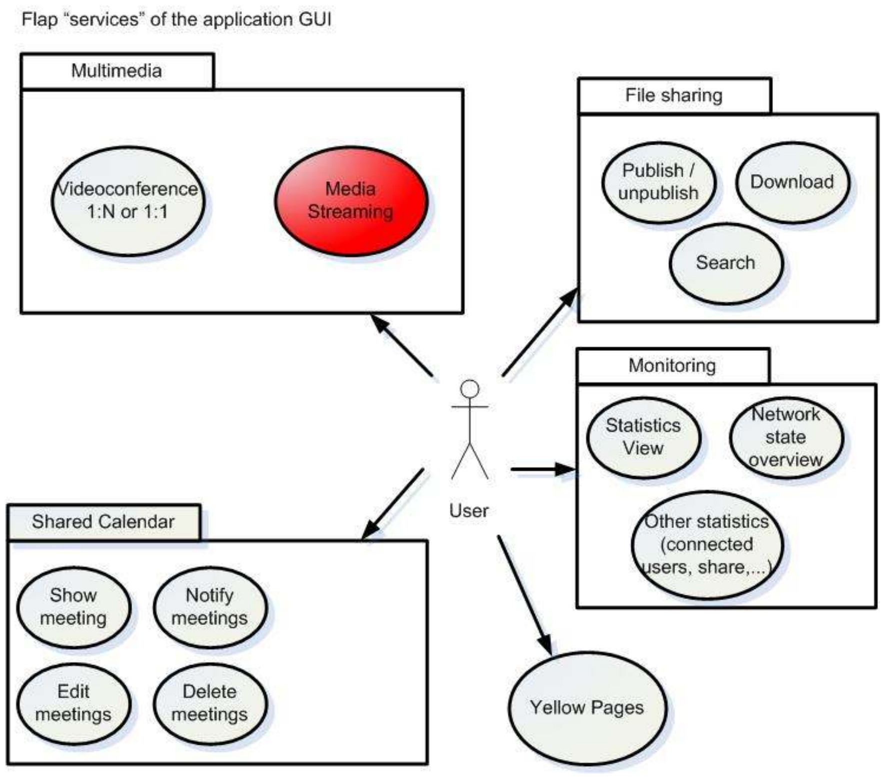


Fig. 2.2 Services Module

In this section are shown some use case diagrams that allow observing the functionalities of the application in detail. Just the most important one is considered in this section: media streaming, included in the multimedia module.

Other technical specifications can be further seen in ANNEX A.

2.2.1. Media Streaming

The Media Streaming functionality offers to the user the actions shown in Fig. 2.3.

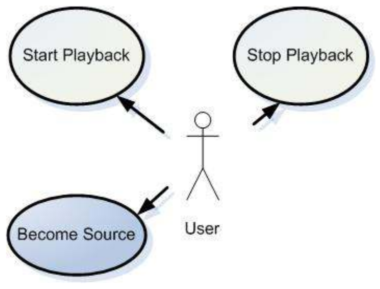


Fig. 2.3 Multimedia streaming options

A user can ask the application for starting a channel playback or stopping it when the user wants (Table 2.1, Table 2.2).

Table 2.1 Start playback

Name	Start playback.
Description	Ask for starting the playback of a media stream delivered when joining a specific media P2P channel.
Actors	User.
Preconditions	The peer has discovered the channel group.
Normal flow	1. User clicks on the panel of P2P channels. 2. The application shows the list of P2P channels. 3. User presses right-mouse button over the desired channel and selects “Start playback”.
Postconditions	The desired media stream starts its playback.

Table 2.2 Stop playback

Name	Stop playback.
Description	Ask for stopping the playback of a media stream delivered when joining a specific media P2P channel.
Actors	User.
Preconditions	The peer has discovered the channel group and is currently playing the desired media stream.
Normal flow	1. User clicks on the panel of P2P channels. 2. The application shows the list of P2P channels. 3. User presses right-mouse button over the desired channel and selects “Stop playback”.
Postconditions	The media stream is stopped. The user terminates the visualization.

Another key functionality that offers the application is the possibility of becoming a streaming source when having enough capabilities (Table 2.3). If the minimum requirements are satisfied, the application shows a simple panel which allows the user to configure the channel that is going to be published and the stream that is going to be delivered.

Table 2.3 Become a source

Name	Become a source.
Description	Any peer can become a media streaming source.
Actors	User.
Preconditions	The user has enough capabilities to start a media streaming process: bandwidth, processing, devices,...
Normal flow	1. User clicks on the panel of P2P channels. 2. User clicks on “Publish Channel” button. 3. The application shows the panel which allows the peer to introduce the required parameters in order to become a streaming source. 4. User clicks “Publish” button when is finished.
Postconditions	No other tasks can be realized during the stream delivery process with an exception, the peer can consult the statistics generated by the application.

CHAPTER 3. ARCHITECTURE

The previous chapter specified the functionalities that CIMS-Live can offer to its users. In this chapter it is commented the architecture that allows constructing the logical platform created by the application. The architecture of the platform can be separated into three well-defined layers that will be deeply described in this chapter. These layers are the following ones.

- **Physical layer**. Contains all the nodes of the network physically separated.
- **Transmission layer**. Overlay network disposed to carry out streaming transmission tasks in an efficient way thanks to the data gathered from the JXTA layer.
- **JXTA layer**. This is the most abstract overlay layer. It is in charge of organizing the users into groups. In addition, it is used for signalling tasks. This layer will enable to decrease the mismatch between the physical layer and the transmission layer in order to optimize streaming tasks.

3.1. System Overview

In this section it is going to be explained which are the mechanisms that allow constructing the collaborative and interactive P2P media streaming platform and how the peers are organized into different overlay networks.

3.1.1. JXTA Based Application

The proposed platform is based on the P2P Java API provided by the JXTA project [31], [18], which offers automatic mechanisms for publishing and discovering peers, groups of peers, and every desired resource. Thanks to this API it is easy to develop a full featured P2P application where peers are self organized into logical groups. The self-grouped peers can discover each others in an easy way, and it is simple to create membership and presence mechanisms. Next, Fig. 3.1 shows the JXTA architecture which allows creating P2P applications.

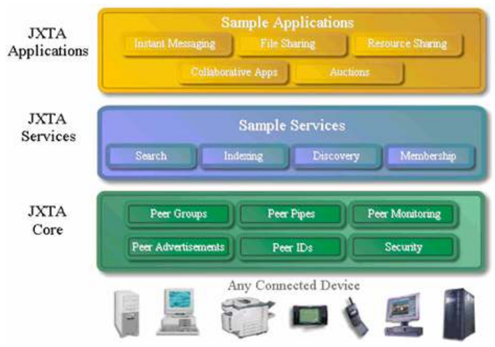


Fig. 3.1 JXTA Architecture

In addition, a goal feature offered by the JXTA platform is that the more time the application is running, the more global knowledge of the logical network or group can be achieved thanks to the discovery requests and events generated or caught by the peers. According to the information gathered from the discovery, membership and presence processes, a peer can optimize its transmission layer by updating its relationships with other discovered peers.

The powerful mechanism for publishing advertisements of JXTA, also makes easy to advertise the specific capabilities that each peer has, even it can be done periodically if they change in time. Peers can work attending to three profiles: *source*, *transcoder* and *simple peer*. In section 3.3 these profiles will be further described.

3.1.2. Multimedia Streaming and Interactivity

As it has been mentioned, the platform makes possible to deliver media streams, which supposes a key point. In this project it is proposed to adopt two different strategies depending on the kind of stream that is going to be delivered.

- **Low bitrate streams**: streams delivered at less than 2Mbps (<2Mbps)
- **High bitrate steams**: streams delivered at more than 2Mbps (>2Mbps)

This distinction is done because the connection and hardware requirements are not the same in both situations. For further detail, see CHAPTER 5.

On the one hand, it is used a similar mechanism to the one suggested by DONET in order to deliver low bitrate streams, which is based on buffer segmentation and multiple-peer retrieval of data blocks. Nevertheless, it is used the information exchanged in the JXTA network to configure and update the scheduling algorithm according to required information such as the Buffer Map of each peer. Thanks to the JXTA layer, each peer can dynamically discover other peers and optimize its transmission layer with the obtained information. On the other hand, ALM structures are used to deliver high bitrate streams because it is not required any complex process of the media streams.

Moreover, a peer can obtain information of proximity to other peers by exchanging some messages at application level in order to discover what the delay between two peers is. Once obtained the delay and its variation (jitter), a peer can establish relationships with the best found peers.

Another new feature presented by this platform is that it enables everyone to publish a stream, that is, to be a streaming source and make it available to everybody. It is also allowed publishing into a specific streaming group (channel) those peers with specific capability of transcoding in order to adapt the media stream to other limited devices.

Finally, new interactive services such as chat, file sharing, opinion poll or survey can be added to this platform. These services promote social networks.

3.2. P2P Architecture

This section shows the proposed pure P2P architecture based on the Java API named JXTA.

The architecture can be mapped to three different logical layers as can be seen in Fig. 3.2.

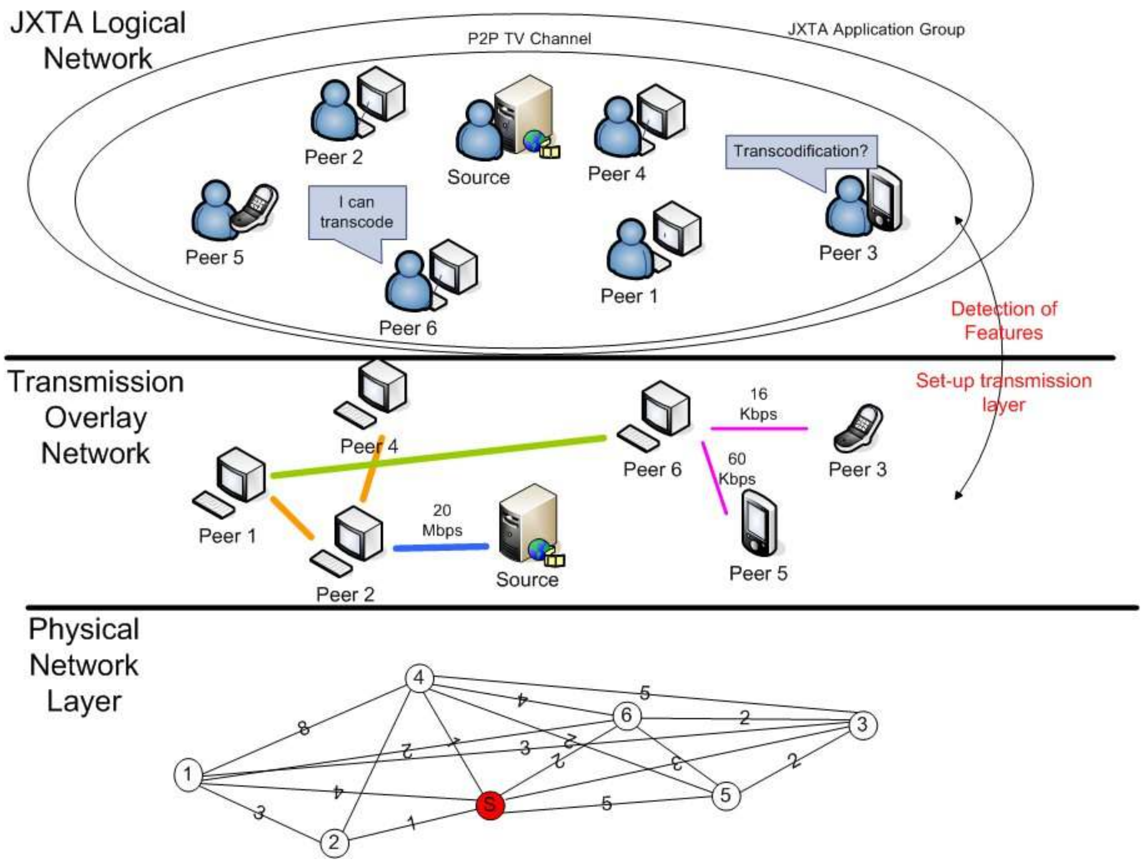


Fig. 3.2 Layered Architecture

Fig. 3.2 depicts the layered architecture, structured as follows.

- **Physical Network Layer.** It represents the physical topology of the different peers separated by physical links. The number placed over the link represents a symbolic cost estimated by a function of hops/delay between two end points.
- **Transmission Overlay Network.** This layer consists of a group of unicast links between peers. It is responsible of transmitting the multimedia packets.
- **JXTA Logical Network**. This is the most abstract level which contains all the self-organized peers. This layer is in charge of signalling and discovery tasks.

All the peers are nodes physically separated by a certain number of hops and a variable delay, which supposes that a data packet sent from one node to another can go through many different links and cross many nodes until it arrives to destiny. The physical topology is a layer joined by lots of heterogeneous peers.

The physical topology is mapped to a logical overlay network, called JXTA Logical Network, where all the peers that join the application are self organized. Thanks to the JXTA API, these nodes can be advertised, discovered and self organized in a logical group of peers.

When the application starts (Fig. 3.3), the first thing it must be done is to find the peer group created by the application, which contains all the connected peers. Then, if the peer finds the group it will join it and then will look for a Rendezvous peer in the group. If the group has not been discovered, it will create the group. This also occurs with the Rendezvous peer, that is, if the peer does not find the Rendezvous, and it has enough capability, this peer will become Rendezvous. Once connected to the Rendezvous, the peer will start a thread for managing discovery events.

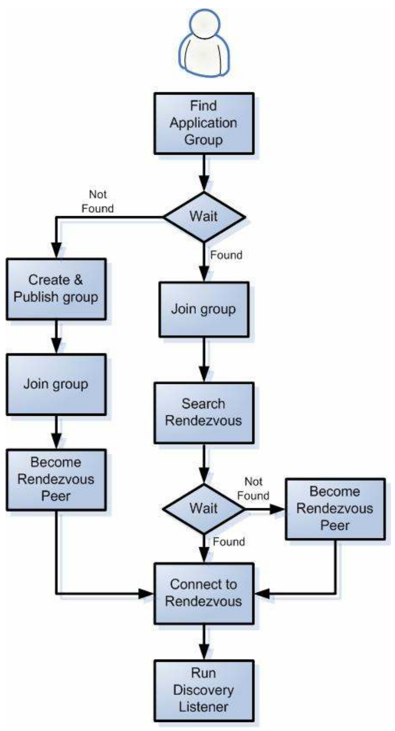


Fig. 3.3 Start of the application

In spite of the fact that JXTA provides mechanisms to send and receive data (Pipes and JXTA Sockets) they are not optimum for high data volume transferences due to the heavy weighted encapsulation of this data into XML messages. Straight afterwards, Fig. 3.4 shows the proportion dedicated for payload and headers (TCP/IP and JXTA) when sending a JXTA message of three different services. It can be seen that the JXTA header can represent the 95% of the sent message.

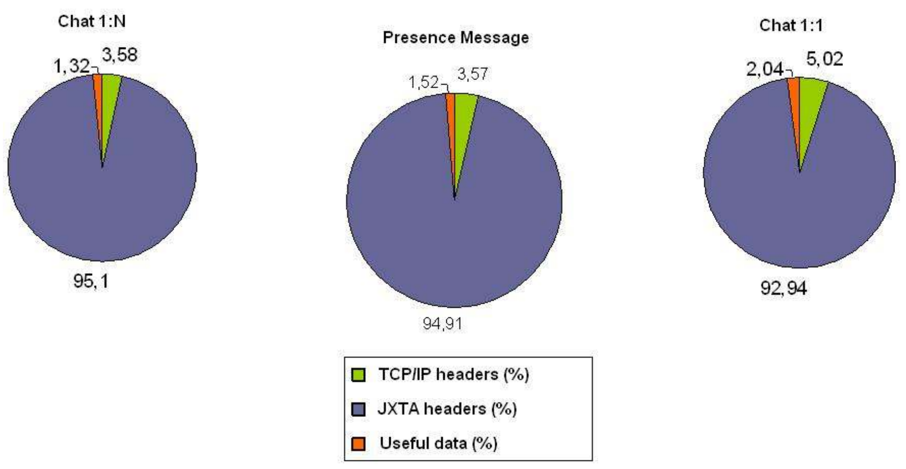


Fig. 3.4 JXTA messaging efficiency

This is the reason why it is used a unicast data transmission layer, specially aimed at media stream delivery. The JXTA layer is the responsible of signalling and other light services provided by the platform such as presence and chat.

Thanks to the dynamic information exchanged among different peers in the JXTA network, a global view of the group can be achieved and the transmission layer can be set-up to obtain the best possible performance. The peers are always listening to the changes produced in the network, that is, the state of the peers and their features. Then, it is proposed a feedback from the top layer to the transmission layer.

Moreover, this architecture enables to optimize the relationships among peers as more time they are up by discovering other new peers.

Another key issue is that when a peer generates a discovery request it waits for some answers and, implicitly, just the closest peers will answer. With this approximation, the mismatch between physical and logical layers can be decreased. But this is not the only parameter to be taken into account when constructing logical links between two peers; it is also considered the delay between them.

3.3. Peer Profiles

It must be specified the different roles carried out by each peer of the P2P system. It must be noticed that all the peers in the application belong to a common group. The application creates a logical group that contains all the peers in the system and all the groups created by the peers of the application. This organization can be seen in Fig. 3.5.

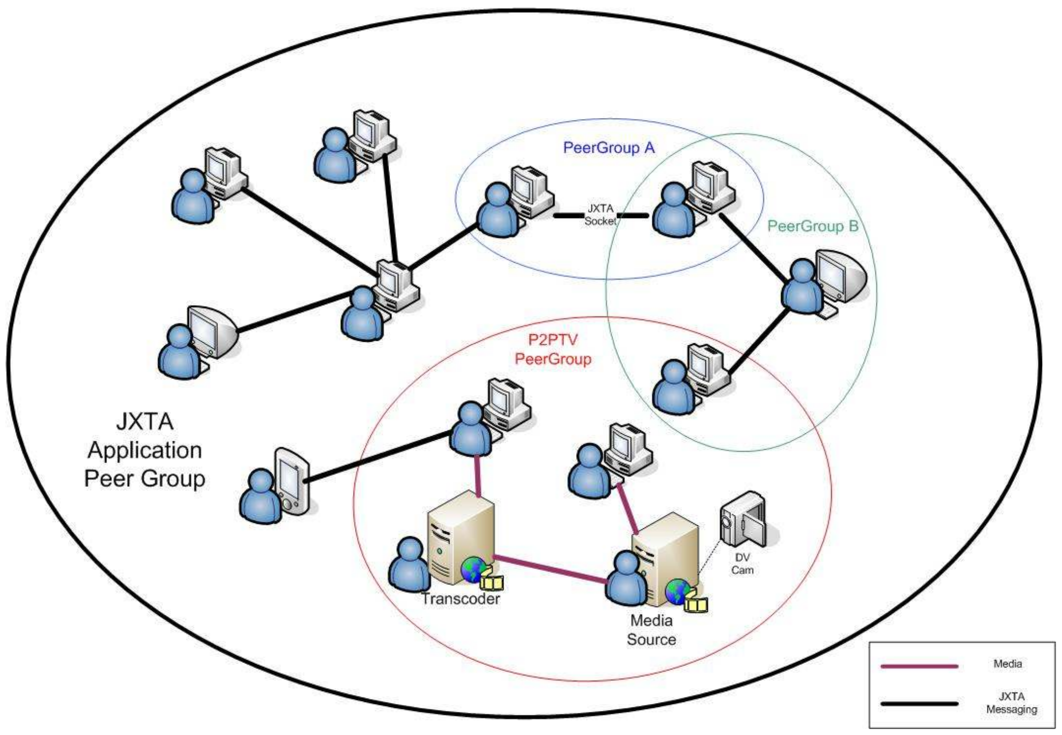


Fig. 3.5 Logical organization of peers

Different types of peers can be found according to its features. Each user connected to the application will become a peer of the logical application group and will be capable to discover and join other peers and groups present in the logical system. Each peer publishes its features: bandwidth, maximum number of connections accepted, delay to the source and hops to the source. In addition, each peer publishes its role into the peer group: source, transcoder or simple peer.

Source: the owner peer of a specific channel of media streaming and also the peer that provides the stream to the group. A P2PTV or P2PRadio channel can be mapped to a logical group in JXTA environment. This group is joined by all the peers that want to play a concrete stream, that is, all the peers have a common interest. When a peer wants to become a stream deliverer it must create a JXTA peer group advertisement describing the characteristics of the P2PTV channel: name, theme, description, bit rate, codec and any other required parameter. When other peers discover the advertisements, they get ready to try to receive the media stream.

Transcoder: a peer that can receive a media stream and can transcode (Fig. 3.6) it to another specific media format. Transcoders are peers that can collaborate in the peer group by adapting the delivered streams for other limited peers present in the group such as PDA and mobile phones.

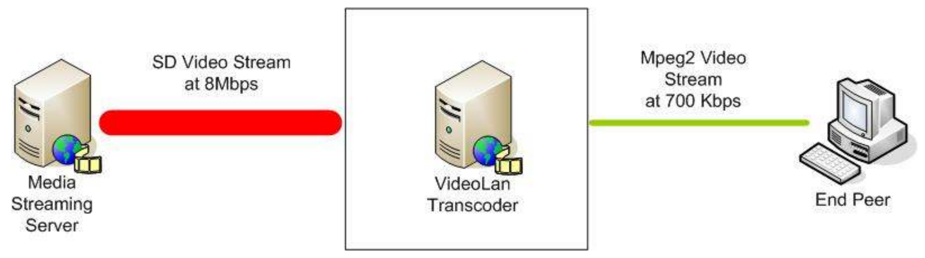


Fig. 3.6 Transcoding process

Simple Peer: this is the default profile of a user in the application group. Each peer can discover and join the different groups in the JXTA world and discover and use the services it offers: media streaming, file sharing, and so on. Also, if the peer has enough capacity, it can become a source peer if it wants to deliver a media stream.

CHAPTER 4. SYSTEM DESIGN

Once defined the proposed architecture for the collaborative and interactive platform, the design of CIMS-Live can be described. CIMS-Live is a modular application that can be extended in an easy manner. The design can be seen in Fig. 4.1.

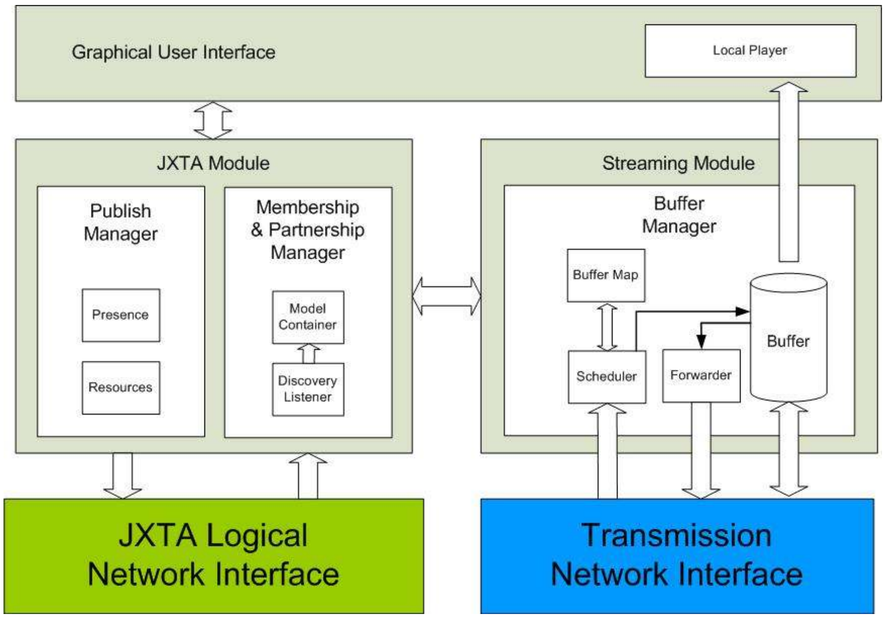


Fig. 4.1 Generic system diagram of a peer

Fig. 4.1 depicts the system diagram of a peer. It is composed by the following modules.

- **Graphical User Interface** (GUI), it is the User front-end.
- **JXTA Module**, which maintains global view of the logical group, establishes and maintains the partnership with other peers, publishes its features and capabilities and discovers resources on the peer group.
- **Streaming Module**, which schedules the transmission of a media stream to other peers in the system according to the information gathered from the JXTA Module and also forwards the stream to the local player embedded into the GUI for its playback.

The system has two key network interfaces.

- **JXTA Logical Network Interface**, which enables all the signalling through the JXTA network. It is based on UDP unicast sockets.
- **Transmission Network Interface,** which allows the user to receive and send streaming packets. The JXTA network interface is based on JXTA Sockets, JXTA multicast Sockets and the JXTA discovery and publishing services.

4.1. Graphical User Interface (GUI)

The view of the application (Fig. 4.2) is constructed using the Java Swing framework [15]. The user can view all the data gathered from the JXTA network referenced to groups, media channels, chat rooms, connected peers, advertising, shared resources and any other new service added to the platform thanks to a generic services directory (yellow pages). The GUI listens to the actions of the user and generates events for its later processing.

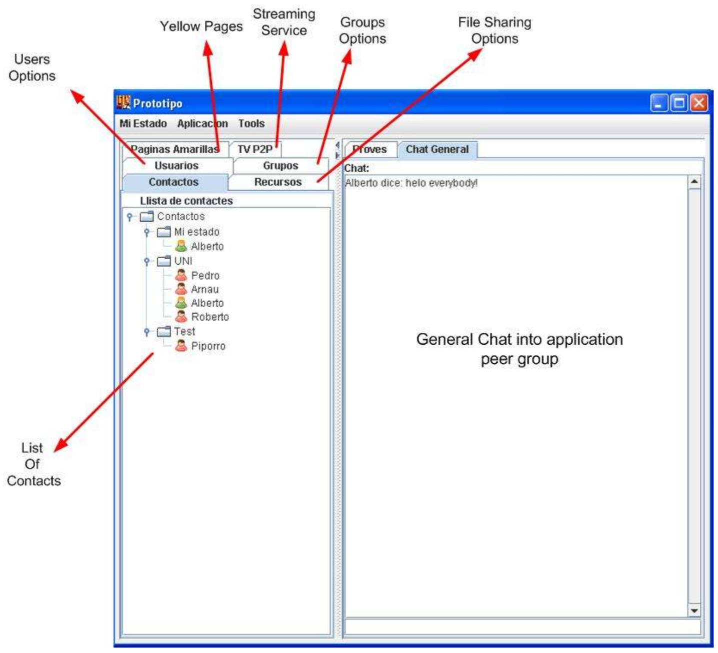


Fig. 4.2 Graphical User Interface

In order to create the visual interface of the user, it was followed the Model-View-Controller (MVC) [11], [19] design pattern, which is mainly used for developing GUIs.

4.1.1. Model-View-Controller Design Pattern

The MVC is a design pattern used in software engineering. In complex computer applications that present lots of data to the user, one often wishes to separate data (model) and user interface (view), so that changes to the user interface do not impact the data handling, and that the data can be reorganized without changing the user interface. The model-view-controller design pattern solves this problem by decoupling data access and business logic from data presentation and user interaction, by introducing an intermediate component: the controller. The interaction between each module and the tasks carried out by each of them are shown in Fig. 4.3.

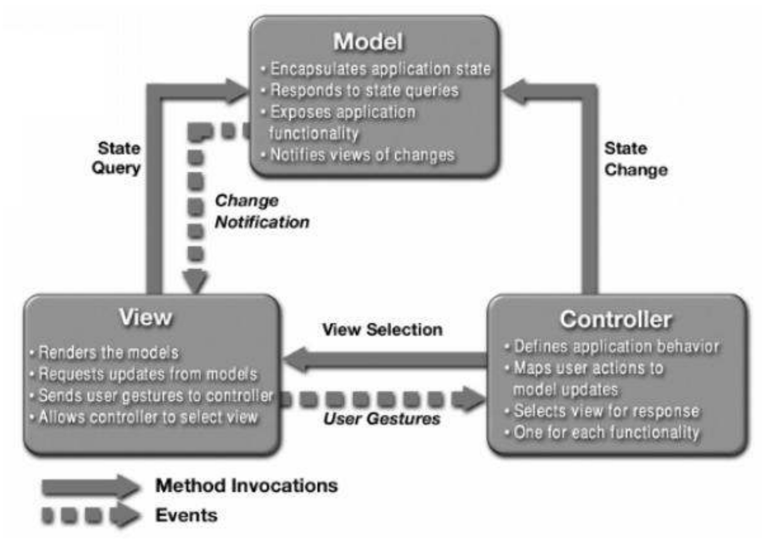

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Fig. 4.3 MVC

MVC is generally used as follows.

1. The user interacts with the user interface in some way, for instance, user presses a button.
2. A controller handles the input event from the user interface, often via a registered handler or callback.
3. The controller accesses the model, possibly updating it in an appropriate way to the action of the user. For example, the controller updates the state of the user.
4. A view uses the model to generate an appropriate user interface, for example, the view produces a screen change on the colour of the user icon. The view gets its own data from the model. The model has no direct knowledge of the view. However, the observer pattern can be used to allow the model to indirectly notify interested parties of a change.
5. The user interface waits for further user interactions, which begins the cycle anew.

Thanks to this design pattern, the GUI can be changed without affecting to the rest of the code. So, it supposes a structure that allows reusing the code.

Other design patterns [19] such as Singleton, Factory, Command and Observer have been also used in order to improve the performance of the application and provide a reusable code.

4.2. JXTA Module

This module is the responsible of interacting with the JXTA network interface. The main task it must carry out is the signalling of the system which will be managed by the partnership and membership submodules. All the signalling is based on JXTA socket messages and on publish and discovery events.

All the information gathered from peers, groups, resources and services is stored in a model container submodule which represents the model layer of the application.

Membership Manager submodule. This submodule is entrusted of managing all the data discovered from the JXTA overlay network thanks to a dedicated thread (Fig. 4.4) that listens to the generated events. These events are linked to new advertisements of peers, groups and other JXTA resources discovered by the application. When any advertisement or event of this kind is found, the GUI is automatically updated.

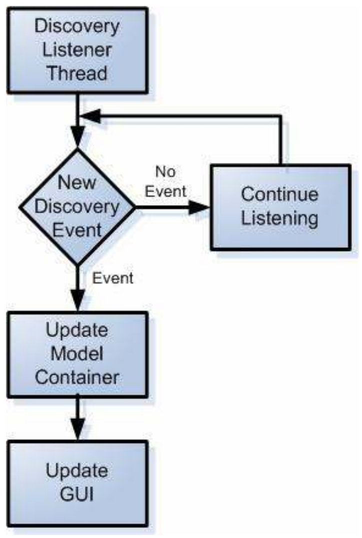


Fig. 4.4 Detection of events and GUI update

The main functions of the membership manager are:

- Manage connections and departures of peers.
- Manage groups: connection and disconnection through JXTA Membership service.
- Update of the model container.

This submodule handles all the data contained on the model container, that is, every resource published in the JXTA network such as peers, groups, resources, publishing, state of peers and Buffer Maps.

Partnership Manager submodule. This submodule is focused on maintaining a selection of the best found peers which a user will interact with in streaming tasks. The selection of these peers is based on different parameters (Table 4.1) according to the kind of stream which is desired and to the specific features and capabilities of the requesting peer. This module includes or rejects the streaming partners of a peer.

Table 4.1 Parameters under consideration

High bitrate stream	Maximum number of connections accepted, time of playback, proximity to the source and end-to-end delay.
Low bitrate stream	Available bandwidth, end-to-end delay, all the Buffer Maps of the known peers and the number of peers with the desired data available.

When referring to low bitrate streaming, this module contains a list of partner peers and their Buffer Maps. It will interact with the Publish Manager submodule in order to publish the Buffer Map periodically through JXTA Sockets. When talking about high bitrate streaming it will contain a list with the main parent, its children and a backup parent. Both cases will manifest the potential suppliers.

Publish Manager submodule. This module is dedicated to any process that requires a publishing operation in JXTA network. It can publish among others:

- Peer Advertisement: contains the basic description of a peer (peer ID).
- Communication point: JXTA socket (Pipe).
- PeerGroup Advertisement: it is the specification of a group of peers. It is used a specific peer group advertisement in order to advertise media streaming channels, basically P2PTV and P2P Radio.
- Digital resources: media files, digital documents, and others.
- Publishing.

This module is also the responsible of periodically propagating the state of a peer through the presence service, its features and the Buffer Map when being part of a low bitrate streaming task. This module is implemented with JXTA multicast and unicast sockets in a specific peer group.

4.3. Streaming Module

This module is the responsible of managing the streaming operations based on the updated information contained in the list of partners provided by the partnership manager. It is basically composed by the following different submodules (Fig. 4.5): buffer, buffer map, scheduler and forwarder.

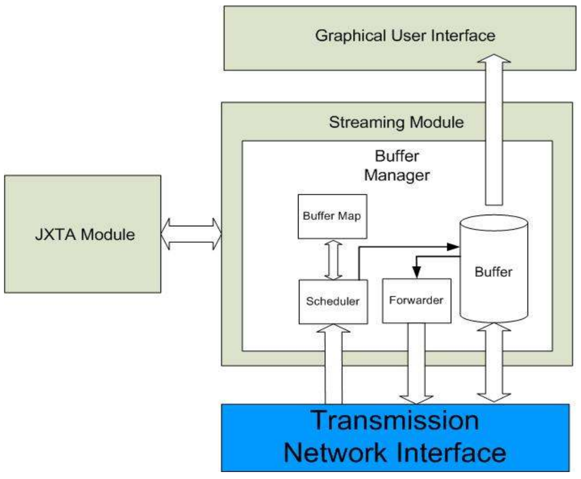


Fig. 4.5 Streaming Module

Buffer. This submodule storages in local memory the datagrams of the received media stream. The buffer is divided into uniform data blocks. It holds the data that will be sent to the player for local playback of the desired media stream and, at the same time, that data will also be made available to other peers into the system. It must be big enough in order to temporally accommodate a part of the media stream. There must be a trade-off between: required memory space, end-to-end delay requirements and peer failure detection and correction.

Buffer Map. This submodule manifests the state of the buffer.

- **Low bitrate**: indicates which blocks are available and the corresponding deadline of each block (timestamp).
- **High bitrate**: indicates the state of the buffer (how many blocks accommodates) and the current playback time.

Scheduler. This submodule decides which peer will be the supplier for each block (low bitrate) or media stream (high bitrate). Just selects the peer who will be asked for a part of its buffer.

Forwarder. This submodule is in charge of sending multimedia packets to the partners of a peer. When talking about low bitrate streams, it sends just the requested packets. Otherwise, in a high bitrate transmission, it sends to the partners (children) all the packets it receives, that is, just forwards the stream. These streaming mechanisms are further detailed in CHAPTER 5. The operations carried out by this module can be compared to the ones carried out by a RTP Proxy.

CHAPTER 5. P2P MEDIA STREAMING MECHANISMS

Two strategies are adopted when a peer wants to start receiving a media stream. Both strategies are focused on the kind of stream that is going to be asked for: low bitrate stream or high bitrate stream.

First, it must be clarified the conceptual separation between low bitrate streams and high bitrate streams. In this project it is referred as low bitrate stream those streams delivered at a bitrate lower than 2 Mbps, which is a common top bitrate used in popular P2PTV applications and offers a wide coverage for current broadband Internet access. So, when talking about high bitrate streams, it is assumed a bitrate above 2Mbps which allows delivering any other high bitrate format such as High Definition (HD), Standard Definition (SD), Digital Video (DV) and others.

These two strategies are adopted because the requirements of bandwidth and buffering of a peer are not the same when trying to receive a low or high bitrate stream. High bitrate streams require large bandwidth availability and strong buffering capabilities, uncommon in current Internet broadband domestic connections and devices.

Nevertheless, it must be mentioned that the logic applied by both streaming mechanisms can be updated when needed. The same occurs with the scheduling algorithms applied.

5.1. Low Bitrate Media Streaming

For low bitrate streams it is used buffer segmentation for subsequent sharing.

A video stream is divided into segments of uniform length by the source peer, and the availability of the blocks in the buffer of a node can be represented by a Buffer Map (BM).

Each peer periodically publishes its BM notifying the other peers which segments it has. Then, compares which segments lack in its BM and asks for them to the known best peers. An example of this situation can be seen in Fig. 5.1.

Fig. 5.1 manifests the situation where Peer A has as partners both Peer B and Peer C. Peer A does not have in its buffer the following blocks: 50, 69, 70, 101, 102 and 103, marked in red in the Buffer Map chart representation. After calculating which the best supplier for each block is, using a scheduling algorithm, it asks for the missing blocks. Consequently, Peer A retrieves 50, 69 and 70 from Peer B and 101, 102 and 103 from Peer C.

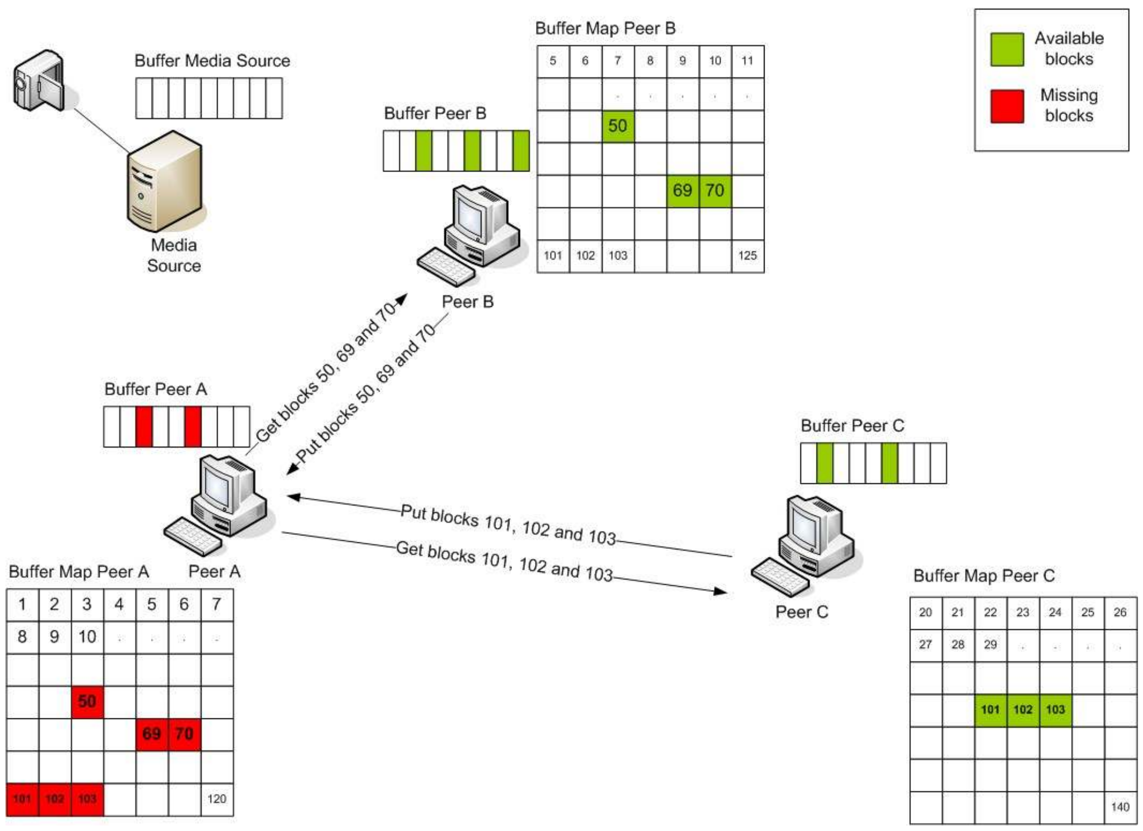


Fig. 5.1 Buffer representation and sharing

When a peer starts the media streaming process, it interacts with some known peers. But the more time it is up, the more new peers it can discover and can calculate the delay between them and how does it change in time by sending some probe packets. The delay can become a fine grained property, very important to determine the supplier of a media block. This is why the scheduler must be intelligent in order to optimize the task of media blocks requesting, so it is proposed to use the CoolStreaming/DONET scheduling algorithm. Given the BMs of a node and that of its partners, the schedule is to be generated for fetching the expected segments from the partners.

The scheduling algorithm strikes to meet two constraints.

- The playback deadline for each segment.
- The heterogeneous streaming bandwidth from the partners.

If the first constraint cannot be satisfied, then the number of segments missing should be kept as minimum as possible.

It is not easy to find an optimal solution, particularly considering that the algorithm must quickly adapt to the highly dynamic network conditions. Therefore, CoolStreaming/DONET resorts to a heuristic algorithm. The heuristic algorithm (Fig. 5.2) first calculates the number of potential suppliers for each media block by consulting the BMs of each partner. Notice that a segment with less potential suppliers is more difficult to meet deadline constraints, so the algorithm determines the supplier of each block starting from those with only one potential supplier, then those with two, and so forth. Among the multiple potential suppliers, the one with the highest bandwidth and enough available time is selected.

```
for block i Є set_of_segments_to_be_fetched do{
   n=0     // number of potential suppliers for segment i
   for j=1 to number_of_partners do{
      T[j,i] = deadline[i] – current_time //check deadline of segment i for partner j
      n=n+BM[j,i]  //Buffer Map node j, has segment i. Increment number of suppliers n
   } //end for j
  if n=1{ //segments with only one potential supplier
    supplier[i] = k //get the supplier for segment i

…
}//end for i

for n=2 to number_of_partners{
   for each i do{    //for each segment , i
      //get supplier[i] with better Bandwidth and enough available time
     …
   }
…
}

Output: supplier[i]: supplier for unavailable segment.
```

Fig. 5.2 Scheduling algorithm overview

The segments to be fetched from the same supplier are marked in a BM-like bit sequence which is sent to the supplier, and these segments are then delivered in order through the transmission network interface of each partner.

5.2. High Bitrate Media Streaming

High bitrate streaming needs high bandwidth and buffering requirements. This is why an ALM structure of paths is proposed (connection-oriented-like scheme) formed by the peers that receive and forward the stream.

5.2.1. Mesh Organization

This project proposes a mesh-first [7], [17] approach which builds up a mesh among the participating peers. The mesh is optimized towards the application requirements and is dynamically adjusted to accommodate the underlying network changes: peer arrival or departure. The distributed ALM algorithm can be run at each node.

In this scenario (Fig. 5.3), there is no need of complex processing of the buffered data, just play it on local player and forward it to children. Peers do not only receive the requested stream, but also contribute to the media stream delivery by forwarding it to other peers.

When a high bitrate channel is joined by a peer, the peer tries to find the best parent, which is the closest to the source and has enough number of available connections.

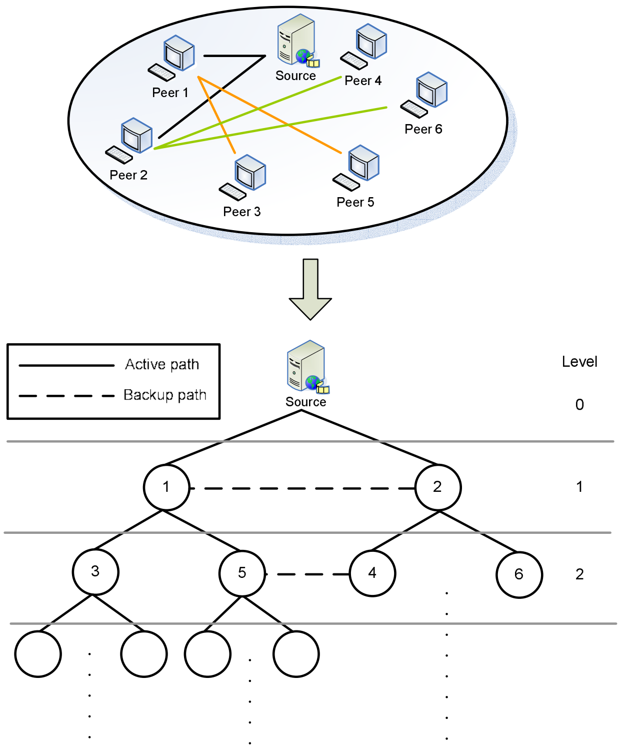


Fig. 5.3 Application Multicast Tree structure. It can be seen the active and backup paths that conform a mesh-first approach

Every peer supports a maximum number of connections (maxConnections) according to its available bandwidth. This maxConnections parameter determines how many children it can have.

Moreover, all the peers that join the high bitrate P2PTV channel have an active connection with a fixed parent and a backup parent peer in order to minimize the effects of network failure when a peer fails or leaves the channel.

In this connection-oriented scenario, the forwarding algorithm can be a simple Round Robin at delivery time. It must be said that, the proposed system allows updating the scheduling and forwarding algorithms in both cases (low and high bitrate schemes). It is not a closed issue. It can be easily reconfigured thanks to the modular design of the application.

Fig. 5.4 shows a generic process when a peer wants to start a media streaming playback once discovered the P2PTV channel in the JXTA network. The peer joins the discovered group, finds the peers in the group and discovers their features. With the gathered data it selects the potential suppliers and runs a parallel process to construct a backup list of partners. When the buffer is filled enough, the peer can start playing the stream.

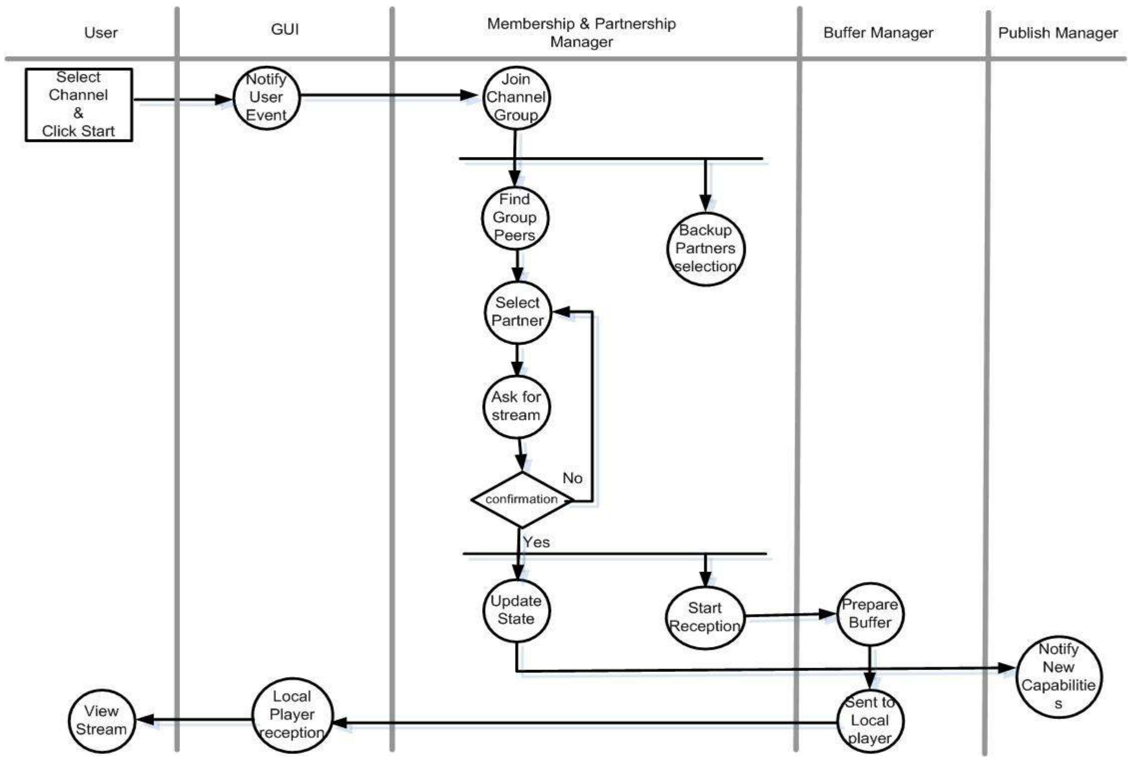


Fig. 5.4 Start a media streaming playback

The termination process of the streaming playback is shown in Fig. 5.5.

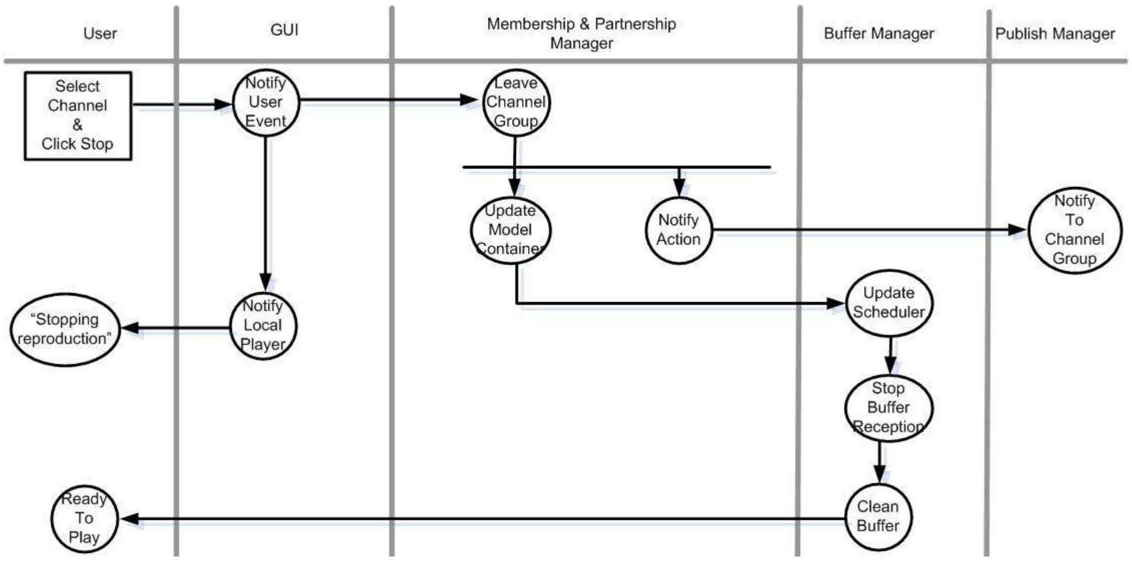


Fig. 5.5 Stop a media streaming playback

There are two ways of leaving a group: leave a group voluntarily or leave it involuntarily due to a peer failure. When a peer leaves voluntarily a group it must notify the event to the peer group through the JXTA Membership service. Otherwise, when a peer leaves involuntarily, it must be detected by its children or parent through the presence service.

In a high bitrate scenario, the mesh must be restructured. Nevertheless, other algorithms can be easily added as it has been said and will be further studied in future work.

5.2.2. Tree Organization

Otherwise, focusing on the realized implementation, it has been proposed an ALM tree structure rooted at source due to its simplicity in comparison with the mesh organization, especially focused on managing a peer departure situation which requires reorganizing the tree. Note that the reorganization of the tree is a critical process.

So, when the ALM must be reorganized due to a peer departure, it is proposed that just a branch of the ALM tree will suffer the effects of the disconnection. Note that this approach implies that the source peer at level 0 must have global information of the connected peers, because it will determine which peer updates its relationships into the ALM tree or which one will switch to its backup path. The cost of having centralized global view on the source is not a restriction due to the fact that it consists of metadata, not the exchanged data, and it does not suppose a heavy task of maintenance. Moreover, when having a global view of the ALM structure, it can be guaranteed a logarithmic cost proportional to the number of users, *O* log (n), when it is required to search a peer. Another advantage of constructing an ALM tree managed at a single point is that the tree can be always balanced in order to always guarantee a logarithmic lookup cost.

Fig. 5.6 shows the departure of peer C in the ALM tree structure. Then, Fig. 5.7 shows the tree view rooted at source after the departure.

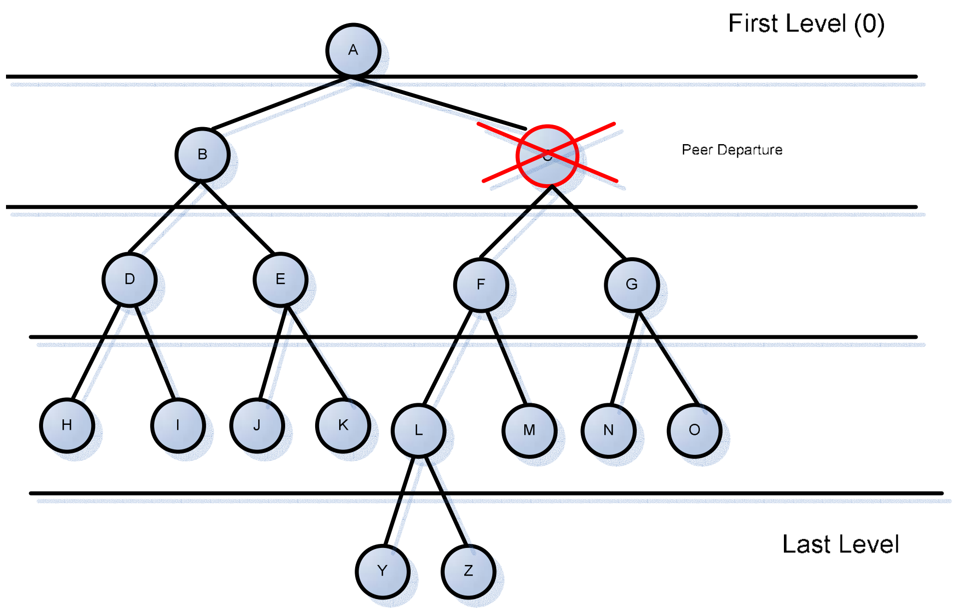


Fig. 5.6 Peer C gets disconnected

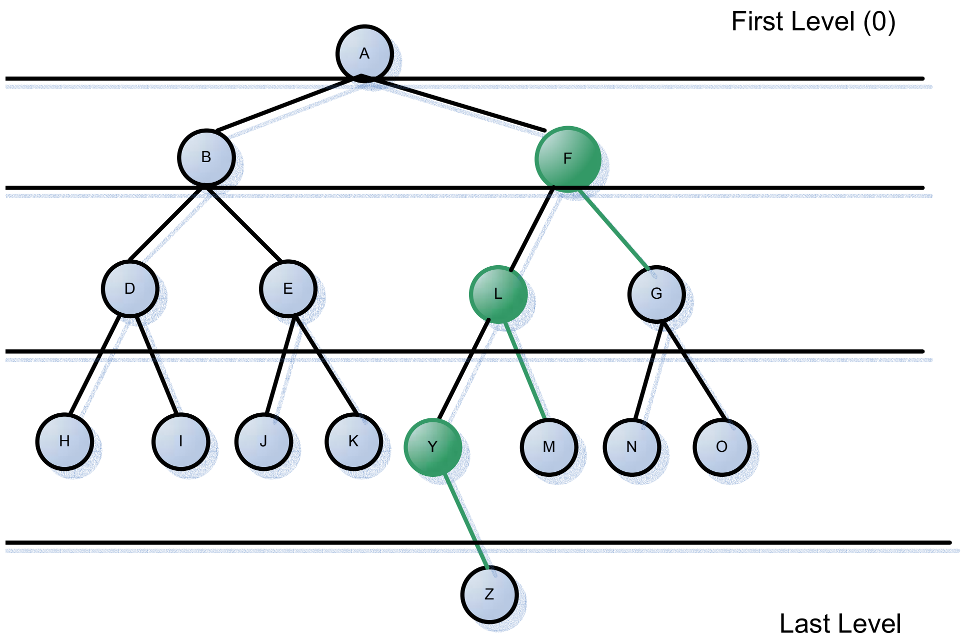


Fig. 5.7 Peer F and its children are reorganized in a tree source-rooted scenario

CHAPTER 6. IMPLEMENTATION AND TESTING

This chapter describes the main issues related to the realized implementation according to the design which has been carried out.

First, it is defined the test bed scenario where the implementation has been done and the technologies and tools that allowed creating a prototype of the CIMS-Live application.

6.1. Test Bed

The test bed manifests the hardware and software requirements for the project.

The whole development of the application has been done in the room SI2CAT. This space was enabled by MediaEntel, located in the Escola Politècnica Superior de Castelldefels (EPSC).

In order to test the communication among peers and enable multimedia streams delivery, it was needed a switch to connect the different involved PCs (Fig. 6.1). Some tests where also done using a Mobile PC connected to the scenario via a Wireless access point. When developing the software it was mainly used two PCs. However, it was also used up to four PCs to test the streaming platform.

There were used two switches during the testing periods: a Fast Ethernet switch and a Gigabit Ethernet Switch. The usage of one or other was determined according to the required bandwidth of the test. That is, when the total bandwidth exceeded 80 Mbps, it was used a Gigabit Ethernet switch.

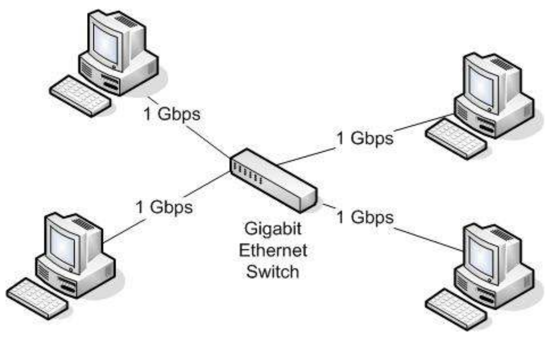


Fig. 6.1 Network diagram

The specification of the PCs used for development and test is shown in Table 6.1.

Table 6.1 PC specifications

Desktop PC	Intel ® Pentium ® 4 CPU HT at 3.2 GHz RAM Memory: 1024 MB 10/100/1000 Ethernet card Operating System: Windows XP Pro
Mobile PC	Intel ® Pentium ® M CPU at 1.6 GHz RAM Memory: 1024 MB 10/100/1000 Ethernet Card Operating System: Windows XP Pro

The only requirement that must be emphasized is the necessity of providing the PCs involved in development tasks with a Fast or Giga Ethernet card. Wireless card are accepted for low bitrate transmissions, but when the stream bitrate exceeded 8 Mbps it was required wired Ethernet network interfaces.

All the software development has been done under Windows XP Professional Operating System but the code can also run under Linux Distributions. In spite of having used open source and multiplatform technologies, the code can only work on both above mentioned operating systems due to the fact that jVLC is currently available for these two platforms.

VideoLan Client (VLC) is also a key software requirement. It is used as streaming server for the time being.

The programming tool used for the implementation of the Java-based application was Eclipse Project. This powerful tool offers the developer a user-friendly GUI and many debugging and compilation options in order to make easier the development task.

Java version 5 has been selected as the version of Java Virtual Machine because it is the most stable release and it is fully compatible with the rest of the used software components.

6.2. Technologies and Tools

This section enumerates and describes the different technologies and tools used by this project.

JDK 5.0: Java Virtual Machine and Development Kit version 5. The different libraries that it provides are used in order to develop the Java-based application.

Eclipse SDK 3.1: Open development and compilation platform for Java comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. A large and vibrant ecosystem of major technology vendors, innovative start-ups, universities, research institutions and individuals extend, complement and support the Eclipse platform. Its basic features can be extended through plugins. The whole application has been developed using this development platform.

JXTA (Juxtapose) 2.3.7: Opensource-based peer-to-peer infrastructure developed by Sun Microsystems. This is the P2P Java-based API used in order to develop the P2P functionalities. This platform is defined as a set of XML based protocols that allow any device connected to a network to exchange messages and collaborate in spite of the network topology. JXTA is the most mature P2P framework currently available and was designed to allow a wide range of devices such as PCs, cell phones and PDAs to communicate in a decentralized manner. The main functionalities offered by JXTA are:

- Dynamic discovery of JXTA resources such as peers, groups, pipes, sockets and so on even with the presence of firewalls.
- Creation of groups that offer services.
- Remote monitoring of peers.
- Secure communication with other peers.

JMF (Java Media Framework) 2.1.1e: Java Library that enables audio, video and other time-based media to be added to Java applications and applets. This optional package, which can capture, playback, stream, and transcode multiple media formats, extends the Java Platform, Standard Edition (Java SE) and allows development of cross-platform multimedia applications. This framework is used for enabling videoconference and multiconference services.

VideoLan Client (VLC): Free media player software and streaming server. It is a highly portable multimedia player, encoder and streamer that supports many audio and video codecs and file formats as well as DVDs, VCDs and various streaming protocols. It is able to stream over networks and to transcode multimedia files and save them into various different formats. This is the software used for streaming tasks.

jVLC [14]: Java VLC Bindings. It provides some libraries that allow embedding a VLC player in a Java application. For the moment, it is just available for Linux and Windows platforms. These libraries allowed embedding the local player of the application into the GUI.

Log4j [20]: Java-based logging utility. The main criterion for introducing a logging system to the application is that it helps with debugging tasks. Furthermore, it is especially useful for distributed application such as the one proposed in this project. With Log4j it is possible to enable logging at runtime without modifying the application binary. The Log4j package is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behaviour can be controlled by editing a configuration file, without touching the application binary.

SAX (Simple API for XML) [24]: Serial access parser API for XML. SAX provides a mechanism for reading data from an XML document. It is a popular alternative to the Document Object Model (DOM). This API allowed parsing the XML contents added to the XML-based advertisements of JXTA.

Apache Ant 1.6 [21]: Java-based build tool. It is like a Makefile when talking about a C environment. It allows a lot of tasks such as compile, execute, copy files, create directories, create JAR/WAR files, run unitary tests, and so on. It uses XML syntax and it is platform independent.

Apache Commons Collections – Buffer [22]: This API offers some implementations of the buffer interface such as priority buffer and circular buffer. It is used for allocate the data received in a streaming task.

CVS (Concurrent Version System) [23]: Implements a version control system. It keeps track of all work and all changes in a set of files, typically the implementation of a software project, and allows several (potentially widely separated) developers to collaborate. CVS has become popular in the free software and open-source worlds.

Excelsior JET [25]: Excelsior JET is a toolkit and complete runtime environment for optimizing, deploying and running applications written in the Java programming language. JET is a powerful solution that enables your desktop and server side Java applications to take full advantage of the underlying hardware. Moreover, JET effectively protects your intellectual property by making your programs as hard to reverse engineer as if they were written in C++. Excelsior JET was used in this project to translate Java code to Native Code in order to increase the performance of the developed prototype.

Advanced Installer [37]: Windows Installer tool. It offers a friendly and easy to use Graphical User Interface for creating and maintaining installation packages (EXE, MSI, etc.) based on the Windows Installer installation technology. It was used for creating an installer of CIMS-Live from the Java code (ANNEX E.1.2).

6.3. Forwarding Method

The final prototype of CIMS-Live is the consequence of developing some previous prototypes that allowed forwarding multimedia streams from a source to other peers using simple unicast UDP connections.

First, the peers forwarded the stream in a cascade scenario as can be seen in Fig. 6.2.

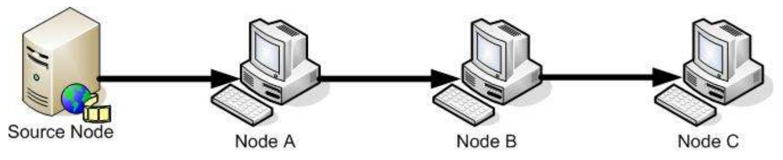


Fig. 6.2 Cascade forwarding

Then, this basic prototype was extended by adding a simple protocol of communication in order to allow mesh and tree configurations for forwarding the stream as is shown in Fig. 6.3. This forwarding method is the one used in the final prototype of CIMS-Live.

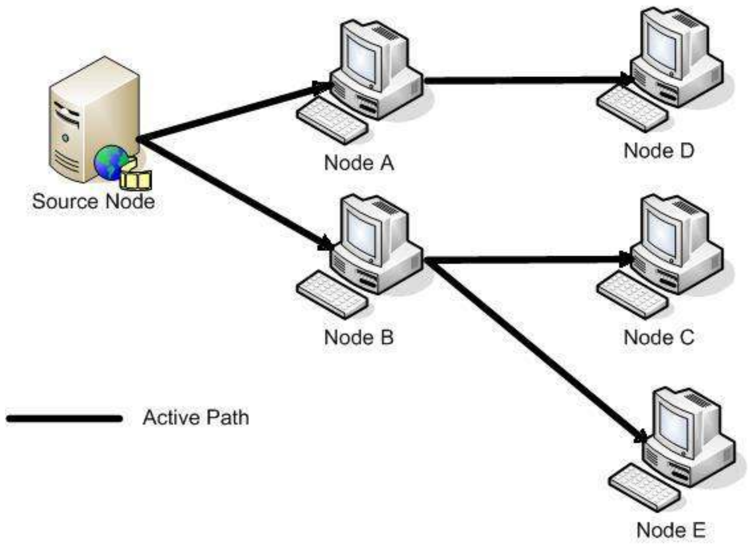


Fig. 6.3 Multiple node forwarding

Every peer receives UDP datagrams (stream) continuously from a single parent and forwards it to the local player and to its connected peers, called children. The forwarding algorithm in this scenario is a simple Round Robin.

6.3.1. Implementation Tests

It must be mentioned that the execution of the prototypes is single thread, that is, just a thread accesses to the buffer, gets and deletes information from the buffer and then forwards it to the children of a peer. It was also developed a multithread prototype where a dedicated thread was used for attending each connected child. Nevertheless, it was very difficult to manage and control the access to the buffered multimedia data by all the threads and there were lots of losses of multimedia data when forwarding. It could be noticed in the field study carried out of P2P streaming applications that many developers avoid multithread environments due to synchronisms problems.

In a first single thread approximation, no incoming buffer (reception buffer) was specifically implemented in order to allocate the stream coming from the parent peer. This lack of buffering capabilities produced high losses when receiving high bitrate streams (concretely when exceeding 6 Mbps). This is why it was introduced a buffer in order to allocate some packets. Subsequent tests and prototypes where carried out using the implemented buffer.

From the operative point of view, all the tests of the prototypes were done using Java Blocking Sockets (java.io). However, it was also ran a test using Java Non-Bloking Sockets (java.nio). The difference between blocking and non-blocking sockets is that when a packet arrives to a computer, java.io processes the packet if a thread is available, otherwise it drops the packet. Nevertheless, java.nio allocates internally a packet until a thread in the system can process it.

Initially, it was expected better results in a non-blocking sockets scenario in comparison with blocking ones, but in the tests that were carried out the performance of non-blocking sockets was poorer. The reason of these results points out to inefficient code implementation.

Another forwarding method based on JXTA multicast sockets was tested but there were lots of losses at playback time due to the heavy encapsulation of the payload into XML messages, as it was expected.

In addition, in order to improve the performance of the single thread scenario, the prototype was translated to Native code using Excelsior JET application. Java is an interpreted programming language which needs the Java Virtual Machine in order to translate Java code to a bytecode that the computer can compile. This translation process makes Java less inefficient than compiled code such as the one generated by C programming language. After translating the Java code to native code, the performance was improved but the playback of a HD stream was still unsatisfactory under Windows XP due to some playback losses.

Finally, in order to improve this transmission layer, it has been tested the UDP Packet Reflector / Forwarder [26] code developed by CESNET with no significant improvement in performance. This program is written in C and it was expected to obtain considerably better performance than in a single thread scenario.

Next, Table 6.2 synthesizes the obtained results running on Windows XP Professional Operating System.

Table 6.2 Test results on Windows XP Pro

Method	Stream (VBR)		Comments	CPU Usage	RAM Memory Usage	End User Evaluation	
	Low bitrate stream	High bitrate stream					
java.io single thread (no buffering capability)	Streams up to 2 Mbps	3.5 Mbps streams - 8 Mbps (SD) streams - 20 Mbps (HD) streams	Streams above 6Mbps present losses at playback time.	5 % - 6%	48 MB aprox.	The playback quality is fully acceptable up to 6Mbps.	✓
						Streams above 6Mbps offer unsatisfactory playback quality.	✗
java.io single thread + 5 MiB buffer			Streams above 10Mbps aprox. present less losses at playback time in comparison with no buffering capability.	5 % - 6%	48 MB aprox.	The playback quality is fully acceptable up to 10Mbps.	✓
						Streams above 10Mbps offer unsatisfactory playback quality.	✗

java.nio single thread	Streams up to 2 Mbps	3.5 Mbps streams - 8 Mbps (SD) streams - 20 Mbps (HD) streams	Streams above 8Mbps present losses.	6 % - 7%	55 MB aprox.	The playback quality is acceptable up to 8Mbps. ✓ Streams above 8Mbps offer unsatisfactory playback quality. ✗
java.io multithread			All kind of streams present losses.	Not measured	Not measured	Fully unsatisfactory ✗
java.nio multithread			All kind of streams present losses.	15% - 20%	80 MB aprox.	The playback quality is acceptable up to 8Mbps. ✓ Streams above 8Mbps offer unsatisfactory playback quality. ✗
JXTA multicast sockets			All kind of streams present a lot of losses.	100 %	Not measured	Fully unsatisfactory ✗
Packet Reflector			Streams above 15Mbps present lots of losses.	4% - 8%	2.5 MB aprox.	The playback quality is acceptable up to 15Mbps (present some notable losses). ✓ Streams above 15Mbps offer unsatisfactory playback quality. ✗
Native Excelsior JET			It can be appreciated that streams above 10Mbps present less losses at playback time in comparison with java.io + buffering capability.	3% – 4%	29 MB aprox.	The playback quality is fully acceptable up to 10Mbps. ✓ Streams above 10Mbps offer unsatisfactory playback quality. ✗

Due to unsatisfactory results obtained when receiving high bitrate steams above 10 Mbps, it was decided to run some tests under a Linux operating system. The Linux distribution that was used is Ubuntu 6.06 LTS (Debian-based). Table 6.3 synthesizes the obtained results running on Ubuntu Operating System.

Table 6.3 Test results on Ubuntu 6.06 LTS

Method	Stream (VBR)		Comments	CPU Usage	RAM Memory Usage	End User Evaluation
	Low bitrate stream	High bitrate stream				
java.io single thread + 5 MiB buffer	Streams up to 2 Mbps	3.5 Mbps stream - 8 Mbps (SD) streams - 20 Mbps (HD) streams	No losses noticed when delivering any kind of media stream [1.5 Mbps to 20 Mbps]	5 % - 6%	Not measured	Fully satisfactory playback ✓
java.nio single thread			Not tested due to satisfactory results with java.io single thread			
java.nio multithread						
Packet Reflector			No losses noticed when delivering any kind of media stream [1.5 Mbps to 20 Mbps]	5 % - 6%	2.5 MB	Fully satisfactory playback ✓

When comparing both tables (Table 6.2 vs. Table 6.3) we can realise that the developed implementations work much better running on a Linux platform. It is supposed that this occurs because the implementation of the IP stack on this distribution of Linux is optimized for networking purposes. No considerable losses were observed when running the mentioned forwarding prototypes on Linux when using streams up to 20 Mbps. Moreover, it could not be tested the top bitrate tolerated by the implemented methods because there were not available videos at bitrates above 20 Mbps at that moment. Nevertheless, it is supposed that the maximum bitrate tolerated by a forwarding method based in C, such as Packet Reflector, would be higher than the one that can be achieved by a Java-based prototype.

Finally, the single thread forwarding prototype based on Java Blocking Sockets (Fig. 6.3) with a 5 MiB buffer was used for implementing the transmission layer (Transmission Network Interface) of the CIMS-Live prototype. The final prototype is the consequence of integrating the transmission prototype with a graphical user interface and a JXTA core application.

6.4. Functionalities

CIMS-Live, finally is a P2P application that offers the following functionalities to the peers which join the JXTA network created by the application.

- **1:1 Chat**

Each peer has a ContactSocket and a UserSocket. The ContactSocket is implemented using a JXTA server socket that waits for connections coming from other peers through a UserSocket. The UserSocket is a JXTA socket that allows starting one-to-one communications. The peers exchange JXTA messages which contain the text messages of the chat (Fig. 6.4).

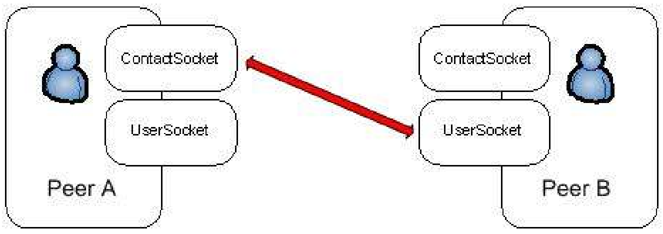


Fig. 6.4 Chat 1:1

The application recognises that the exchanged messages are chat messages because it is used a simple text protocol that adds the following key tag.

```
TXTMSG:[UserName]:[data]
```

- **1:N Chat**

Each peer creates a JXTA multicast socket in each group that it joins (Fig. 6.5). This multicast socket allows receiving and sending text messages from/to all the known peers in a group. It uses the same protocol as 1:1 chat in order to send messages.

It must be commented that the group created by the application which is joined by all the peers at start-up, uses a JXTA multicast socket in order to update the state of the peers. This is called presence service and uses the following message format.

```
PRESENCE:[UserName]:[state]
```

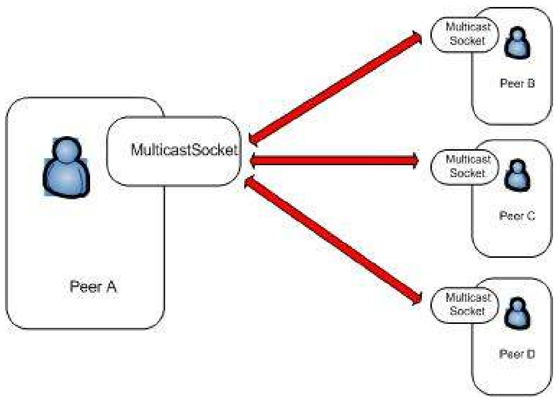


Fig. 6.5 Chat 1:N

- **1:1 File sharing**

It is used the same JXTA socket connection defined in chat 1:1, but it defines a new type of message which allows defining the start / stop of file transmission.

```
FILESTART:[fileName]:[data]
```

- **Videoconference**

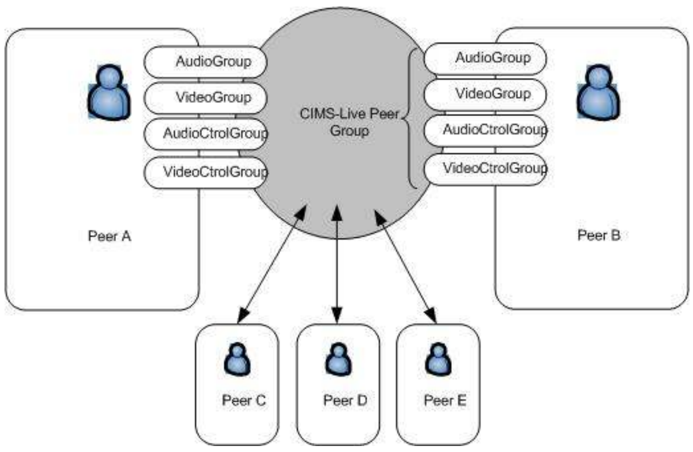


Fig. 6.6 Videoconference

Each peer that wants to start a videoconference must publish and discover four peergroup advertisements first (Fig. 6.6).

- AudioGroup: peergroup for transmitting the audio stream (RTP).
- VideoGroup: peergroup for transmitting the video stream (RTP).
- AudioCtrolGroup: peergroup for transmitting the control packets (RTCP) for the audio flow and to generate statistics of QoS.
- VideoCtrolGroup: peergroup for transmitting the control packets (RTCP) for the video flow and to generate statistics of QoS.

The data is gathered from a webcam thanks to Java Multimedia Framework (JMF) [34]. The media streams and control streams are sent through JXTA multicast sockets on the created peer groups.

The videoconference module is reused from a module developed in a subject named “Design of Telematic Networks and Applications” (DXAT) cursed in EPSC (2006).

- **Live video streaming**

When a peer wants to become a streaming source, that is, create a media streaming channel, it must publish a JXTA PeerGroup advertisement, showed in Fig. 6.7.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jxta:PGA>
<jxta:PGA xmlns:jxta="http://jxta.org">
      <GID>
            urn:jxta:uuid-
FEB568DACA1643368C5B8590C90136C2547A93762DF54C69834EB93D3BD3383A02
      </GID>
      <MSID>
            urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010306
      </MSID>
      <Name>
            P2PTV-Mediacat
      </Name>
      <Desc>
            <channelDesc>
                  <Source>169.254.153.237</Source>
                  <Type>Films</Type>
                  <Desccription>Mediacat</Description>
            </channelDesc>
      </Desc>
</jxta:PGA>
```

Fig. 6.7 Media streaming channel JXTA advertisement

- *Source* field specifies the IP address of the source peer.
- *Type* field describes the theme of the channel.
- *Description* field is a brief description that the streaming peer wants to include.

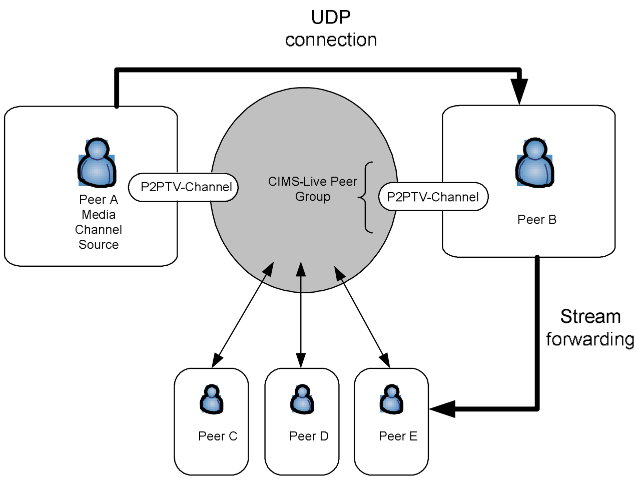


Fig. 6.8 Video streaming

6.5. Application Layer Multicast (ALM) Structure

In this project, it has been implemented a simple ALM source-rooted tree structure for creating the source view that will allow delivering high bitrate streams. In order to construct the ALM structure it has been used an AVL tree implementation.

An AVL tree is a self-balancing binary search tree. In an AVL tree the heights of the two child subtrees of any node differ by at most one, therefore it is also called height-balanced. Lookup, insertion, and deletion all take *O*(log n) time in both the average and worst cases. Additions and deletions may require the tree to be rebalanced by one or more tree rotations. The balance factor of a node is the height of its right subtree minus the height of its left subtree. A node with balance factor 1, 0, or -1 is considered balanced. A node with any other balance factor is considered unbalanced and requires rebalancing the tree. The balance factor is either stored directly at each node or computed from the heights of the subtrees. Fig. 6.9 shows a Non-AVL tree and the same tree height balanced (AVL tree).

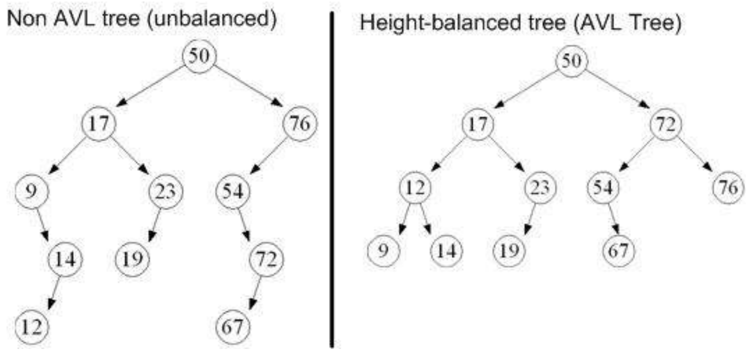


Fig. 6.9 Non AVL tree and AVL tree

When a peer in a channel group wants to start receiving a media stream, it asks the source to get a parent peer, which will be its media supplier.

In order to determine this parent supplier, the source inserts into its AVL the new peer according to the number of hops and delay to the source provided by the new peer. The source has a Hashtable structured according to the number of hops (integer) to the source. Inside each entry of this Hashtable, it contains an AVL tree where peers are self-introduced and balanced according to the delay to the source (Fig. 6.10). AVL tree structures guarantee that the searching cost is proportional to the logarithm of the number of entries.

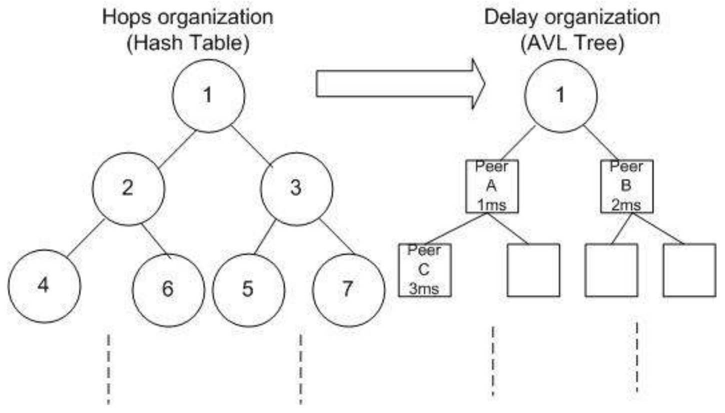


Fig. 6.10 Peer organization by hops and delay

According to this information, the source tries to find the best parent for this new peer, which is the one that has similar cost to the source. Once, the new peer obtains its parent he asks the parent to start receiving the desired media stream. This connection process can be seen in Fig. 6.11.

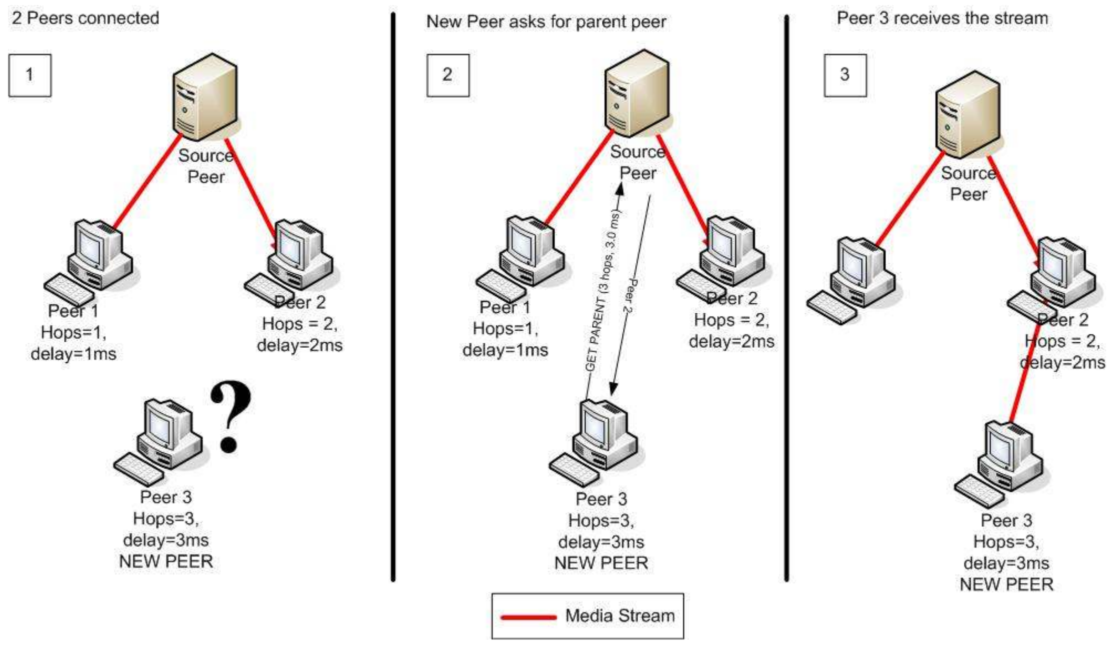

2 Peers connected
1
Source Peer
Peer 1 Hops=1, delay=1ms
Peer 2 Hops = 2, delay=2ms
?
Peer 3 Hops=3, delay=3ms NEW PEER
New Peer asks for parent peer
2
Source Peer
GET PARENT (3 hops, 3.0 ms)
Peer 2
Peer 1 Hops=1, delay=1ms
Peer 2 Hops = 2, delay=2ms
Peer 3 Hops=3, delay=3ms NEW PEER
Peer 3 receives the stream
3
Source Peer
Peer 2 Hops = 2, delay=2ms
Peer 3 Hops=3, delay=3ms NEW PEER
Media Stream

Fig. 6.11 Connection process

However, it must be noticed that this returned parent may be not the most suitable for the new peer because nobody can guarantee the minimal separation between the new peer and its parent. This is why it is proposed that peers can get restructured themselves the more time they are up and the more peers they discover through the most abstract layer which is the logical layer built by CIMS-Live on top of the architecture. Thanks to this high level layer, peers can have a global view of the groups they join, and can discover peers with specific capabilities and can also test performance of the connection by sending probe packets in order to find out which the delay between them is.

6.6. Graphical User Interface

This section shows the developed GUI in detail. The main views that must be described are the publication menu and the media streaming channels view. The whole GUI has been implemented using Java Swing Framework.

- **Publication menu**

  This menu (Fig. 6.12) allows any peer to publish a media streaming channel into the P2P application.

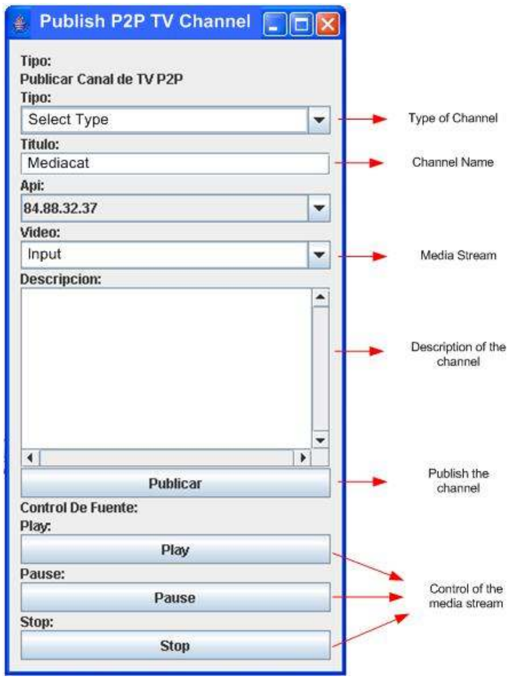
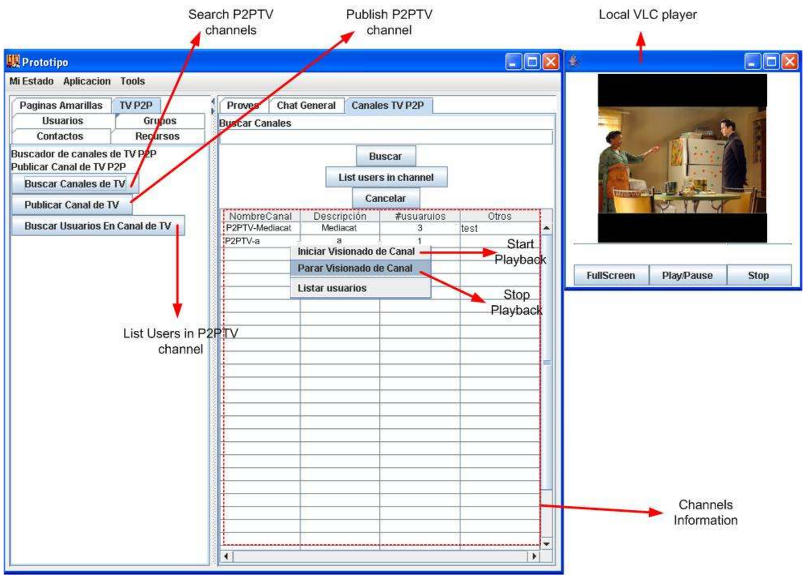


Fig. 6.12 Publication menu

- **Media streaming channels view**

The application allows the user to find media streaming channels and list the connected peers in order to interact with them (Fig. 6.13).

When a media streaming channel is discovered, a peer can ask for its playback. Then, the application launches a local VLC player embedded into the GUI, implemented with jVLC, and starts the playback. The user can also stop the playback of the stream whenever he wants.



NombreCanal	Descripción	#usuaruios	Otros
P2PTV-Mediacat	Mediacat	3	test
P2PTV-a	a	1	

Fig. 6.13 Channels manager view

6.7. CIMS-Live Prototype Test

CIMS-Live uses JXTA for discovery and publication tasks. Blocking UDP unicast sockets have been used in order to receive and forward the high bitrate and low bitrate media streams. In Table 6.4 can be seen the specification of the different streams that were delivered by the prototype at testing time.

Table 6.4 Delivered Media

	High Definition (HD)	Standard Definition (SD)	Low bitrate stream
Video Codec	MPEG 2	MPEG 2	MPEG 1
Resolution	1280 x 720	720 x 576	352 x 288
Duration	52 min	19 min	4 min
Stream bitrate	18 Mbps	8 Mbps	1.5 Mbps
Stream encapsulation	MPEG_TS	MPEG_TS	MPEG_TS
Source codification	VBR	VBR	VBR

One requirement needed for media streaming delivery is that the streams must be encapsulated using a Transport Stream (TS) protocol. Transport stream (TS, TP, or MPEG-TS) is a communications protocol for audio, video, and data which is specified in MPEG-2 Part 1, Systems (ISO/IEC standard 13818-1). Its design goal is to allow multiplexing of digital video and audio and to synchronize the output. Transport stream offers features for error correction for transportation over unreliable media, and is used in broadcast applications such as DVB and ATSC. If the video stream is not multiplexed with audio into a single stream, two streams would be needed in order to deliver video with audio. Note that this situation supposes a greater effort for stream synchronization.

In all the tests, a source peer creates and publishes a P2PTV channel into the application group and acts as Rendezvous. This peer loads video files encoded with the features shown in Table 6.4. It can also load the media stream provided by a DV Camera for live streaming if the corresponding control layer is developed. It was run a test in order to check the correct delivery of DV streams using VideoLan as streaming software. It was verified that VideoLan allows sending DV transcoded streams, but it is not possible to send raw DV stream.

Table 6.5 DV stream specification

Source	Digital Video (DV) Camcoder
Video Codec	MPEG 2
Resolution	1024 x 576
Duration	Live
Stream bitrate	3.5 Mbps
Stream encapsulation	MPEG_TS
Source codification	VBR

Then, the source waits for some requesting peers in order to start the forwarding process. The deployed scenario can be seen in Fig. 6.14.

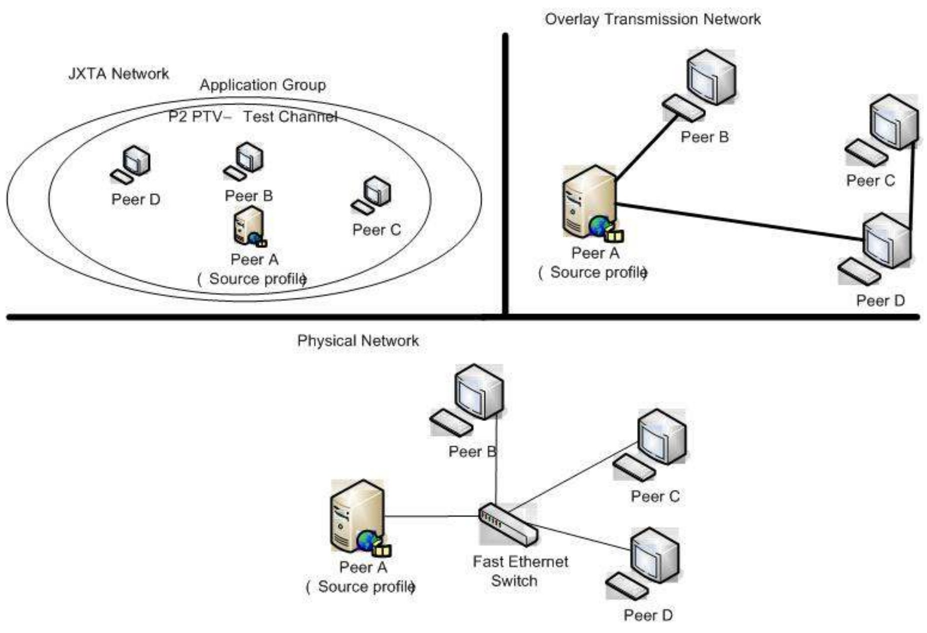


Fig. 6.14 Development scenario

Every peer that acts as source configures the transmission network interface to receive the desired stream from a streaming server such as VideoLan (VLC) [12]. This is achieved by adding a control layer to VLC [30].

The peers connected to the source receive the media stream from the most suitable peer (parent), which is the one returned by the source. Then, they can start forwarding media to other connected peers.

When a peer receives a media stream, it is temporally accommodated into a 5 MiB buffer. Taking into account that each media packet sent from the source has a length of 1316 Bytes, the buffer can store up to 3983 packets. The time equivalence of this amount of packets depends on the bitrate of the stream.

The buffer has been built using the Priority Buffer implementation of Apache Commons Collection [13]. The main criterion for choosing one type of buffer is the process of the contained data that is going to be done. In our case it is proposed to order the received packets according to a sequence-like number. This will allow getting a defined BM.

The received media stream is sent to the local player provided with the prototype application and to the children of the peer. This player is implemented using the jVLC Project [14], which enables to embed a VideoLan Player into the GUI developed with the Java Swing Framework.

The prototype development has been done following some well-known design patterns. The main one is Model-View-Controller (MVC), which allows separating the data plane (model) and the user interface (GUI) by introducing an intermediate component known as controller. The benefits of applying this pattern is that the application can grow easily when it is required a modification or addition in its components.

CHAPTER 7. PLANNING AND COST ESTIMATION

This chapter describes the planning and cost estimation of the realized project. In the planning section are described the fulfilled tasks and also specifies how long did it take to finalize them. Finally, it is provided a cost estimation of the project.

7.1. Planning

The planning is focused on the following tasks:

- **Previous research**

  It has been done a deep research of the current state-of-the art in order to determine the mechanisms applied in P2P streaming for live multimedia spreading and the technologies that can be used to implement them. This research allows identifying which problems must be overcame.

- **Design**

  Once defined the problems to fulfill, it must be done a proposal specifying how they are going to be overcame. This task requires a definition of the functionalities, architecture and design of the application that is going to be implemented.

- **Implementation**

  This task consists of developing the designed application using the technologies and tools selected, which are supposed to best fit our proposal. First, it is done a simple prototype and then it can be grown over it by adding new functionalities. Unitary tests must be carried out in order to test the correctness of every new addition to the application.

- **Testing**

  Testing allows verifying the correctness of the implemented application on a testbed. Several tests must be done in order to determine the correctness of the operations carried out by the application. The testing process allows fixing existing problems or finding new ones if they are present. Unitary tests have been done in order to test little pieces of code before integrating them into the final code.

- **Documentation**

  Elaboration of a written document including all the tasks carried out during the development of the project. This document is necessary to close the project; it consists of the final report of the project.

- **Project Tracking**

  This task consists of periodically revisions of the state of the project carried out by the tutor of the project.

Table 7.1 shows the time estimation of the main tasks that have been realized in this study. The total amount of dedicated hours is approximated to 660 hours. A Gantt diagram can be seen in ANNEX D.

Table 7.1 Tasks

Phase	Task	Description	Hours
Previous research	First contact	Search of P2P streaming applications on the Internet (PPLive, SopCast, PPStream, AnySee).	20
Previous research	First tests	Test some P2P streaming applications.	25
Previous research	P2P mechanism	Study of P2P streaming mechanisms used by commercial and academic applications. Many papers have been searched and read. This task was done in parallel with other tasks too.	200
Design	Features required	Defining all functionalities that our project needs to implement.	25
Design	Architecture	Definition of the proposed collaborative and interactive architecture based on JXTA..	20
Implementation / Testing	First prototype (Cascade)	Java.io./ Java.nio Cascade forwarding implementation and testing. Delivery of different types of multimedia streams.	35
Implementation / Testing	Second prototype (Mesh + Buffer)	Java.io Mesh/tree forwarding implementation and testing. Delivery of types of multimedia streams.	70
Implementation / Testing	Multithread prototype	Implementation of the multithread prototype and testing.	15
Implementation / Testing	JXTA multicast prototype	Implementation and testing of a JXTA multicast prototype.	15
Implementation / Testing	Native translation	Native translation of the second prototype with Excelsior JET.	10
Implementation / Testing	Packet Reflector	Test the Packet Reflector code provided by CESNET.	10
Implementation / Testing	CIMS-Live	Integrate the second prototype with a graphical user interface and the JXTA core application.	100
Testing	Linux test	Test the second prototype and the packet reflector on Linux.	30
------	Paper	Paper submitted to the 3rd EURO-NGI conference. Many parts of the paper have been reused in the final report document.	8
Documentation	Final report	Write the final report of the developed study.	80

7.2. Cost Estimation

In order to make a cost estimation of the project, some aspects must be considered.

- **Delivery Time**

  The planning showed in section 7.1 details the delivery time. The project must be realized in five months approximately, which supposes a little time for implementing the whole specified and designed application. A previous study was realized in order to determine the state-of-the-art in P2P streaming and the technologies and strategies that guided the design and implementation.

- **Achieved objectives**

  The cost estimation can depend on the achieved objectives and final results obtained at the deadline of the project. Depending on the number and type of the achieved objectives (milestones) the project can produce a positive balance or a negative balance. The worst situation would be when any objective is achieved at the deadline of the project, so the final balance of the project would be considered negative (loss of time, money and resources).

- **Risks**

  Some risks related to the project are: results of the research (field study), selection of technologies and tools and unpredicted problems.

  Initial complexity of the project is based on the ignorance of the great amount of paradigms that P2P streaming proposes and has to overcome.

  This is a vast project which requires the study of high performance mechanisms and strategies for an efficient stream delivery. The application has to overcome critical issues such as latency and losses decrease in order to achieve a pleasant user experience and consequently achieve successfully results.

- **Staff**

  A single student of Telecommunications Engineering realizing the final thesis of the M.S. degree in Telematic Engineering. The student does not receive any salary during the project duration.

  The average time estimation is 6 hours per day.

- **Tools and equipment**

  The development of the project has been realized on Windows XP Professional. This operating system requires the payment of a license.

However, the whole development could be done on Linux distributions, which is free software.

The development tool, Eclipse Project, is also free software. Other technologies and tools (6.2) are also free, but not Excelsior JET (it has been used a trial version).

Two PCs were used in order to make easy the development of the P2P application.

It has been used a Fast Ethernet switch and a Gigabit Ethernet switch in order to test the developed prototypes with four PCs.

The hardware, software and Internet connection costs are assumed by the University, so a real cost estimation is difficult to obtain. Hardware and software can be reused for other projects.

- **Economical Result**

Once considered all the issues listed in this section, it can be noticed that precise cost estimation can not be obtained. The final cost can be expressed as a function where the involved variables are: time, staff, achieved results and material costs.

Nevertheless, it can be mentioned that the global cost of the project is bearable. So, an approximated cost can be estimated by calculating the salary that the student should have received.

In Table 7.2 is estimated the cost of the project. The price of a working hour fixed in the table is obtained from a reference scale proposed by the university. The university determines the minimum cost of the hour that a person can receive according to a specific level of education and training.

Table 7.2 Cost estimation of the project

Hours per day (h/day)	6
Total amount of working days (day)	110
Cost per hour (€/h)	7
TOTAL	4620 €

CHAPTER 8. CONCLUSIONS

8.1. Achieved Objectives

In this project it was pointed out some topics of P2P media streaming and it was proposed CIMS-Live as the platform that offers solutions to them, specially focused on improving the collaboration among peers and enabling an interactive experience. This platform is a novel hybrid solution based on JXTA, DONET and ALM which allows delivering any kind of media stream. Besides, it provides a flexible and extensible platform according to new requirements and future value-added services.

It was implemented a Java-based prototype that enabled the provisioning of the interactive services described in section 6.4 and allowed P2P multimedia streaming using an ALM tree mechanism. These functionalities were checked on a test bed.

The main conclusion inferred from this project is that thanks to JXTA it is possible to construct logical overlay networks in an easy manner. This technology enabled the interactivity among users thanks to powerful discovery, publication and binding mechanisms. Nevertheless, it defines a heavy weight messaging system based on XML (Fig. 3.4), this is why JXTA was basically used as a signalling layer that enables the configuration of the transmission layer according to the information gathered from the JXTA layer, which is the most abstract one. Thanks to the global view of the JXTA layer and the exchange of the capabilities among the present peers, the transmission layer can be configured in an optimum way when a user multimedia wants to start receiving a media stream, that is, collaborate selecting the best possible peers.

The results showed that the prototype allowed delivering any kind of multimedia stream, including High Definition streams using ALM structures.

Another interesting conclusion is that Windows XP Professional is not an optimum operating system for High Definition streams delivered at bitrates above 20 Mbps. On the contrary, the Linux distribution called Ubuntu supported these bitrates without problems.

Finally, it can be mentioned that the proposed objectives were achieved.

It must be also mentioned that the development of this project allowed submitting a paper to the 3rd *Euro-NGI Conference on Next Generation Internet Networks - Design and Engineering for Heterogeneity* which presented the described platform and the prototype developed called CIMS-Live. The paper titled “CIMS-Live: Collaborative and Interactive Media Streaming Platform in P2P Environment'” is attached in ANNEX C.

8.2. Future Work

The P2P platform proposed in this project is pure and unstructured, so it is under consideration to develop a structured organization of peers in order to offer a better and more reliable service. This is the reason why an overlay network of Rendezvous/Relay super peers [16] can be built on top of the architecture (Fig. 8.1). This new layer can maintain and guarantee the existence of a unique logical network and can also provide a powerful discovery mechanism based on Distributed Hash Algorithms (DHT). The use of these techniques makes possible to improve the performance of the searching mechanisms by reducing the delay of a discovery query in a time proportional to the logarithm of the number of nodes that join the P2P network, *O*log(n). Moreover, the overhead weight can be reduced, and the availability of a found resource can be guaranteed. However, it supposes higher storage and update costs.

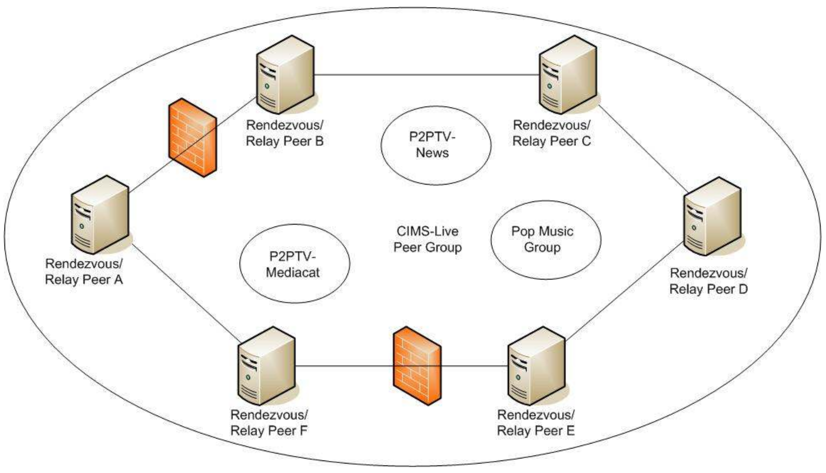


Fig. 8.1 Super peer architecture

Furthermore, it will be considered to separate and distribute the streaming tasks (Streaming Module) into a different computer by using a Service-Oriented Architecture (SOA) scenario (Fig. 8.2). This will enable to separate heavy buffering and stream processing tasks into a powerful computer which will offer these services in order to construct peers with low requirements. Then, the streaming module will interact with the rest of the peer entity by using suitable SOA interfaces, such as Web Services.

Another key feature, maybe the most important and value-added, of decoupling the streaming tasks using an interface based on Web Services is that the system will enable to configure any kind of Media Server based on any streaming software such as Digital Video Transmitting System (DVTS [27]), UltraGrid [28] or others. In this case, it is only necessary a Streaming Control Layer which allows configuring the streaming software. Basically, this layer consists of a software component that translates streaming commands such as *play* or *stop* to the format understood by the streaming software.

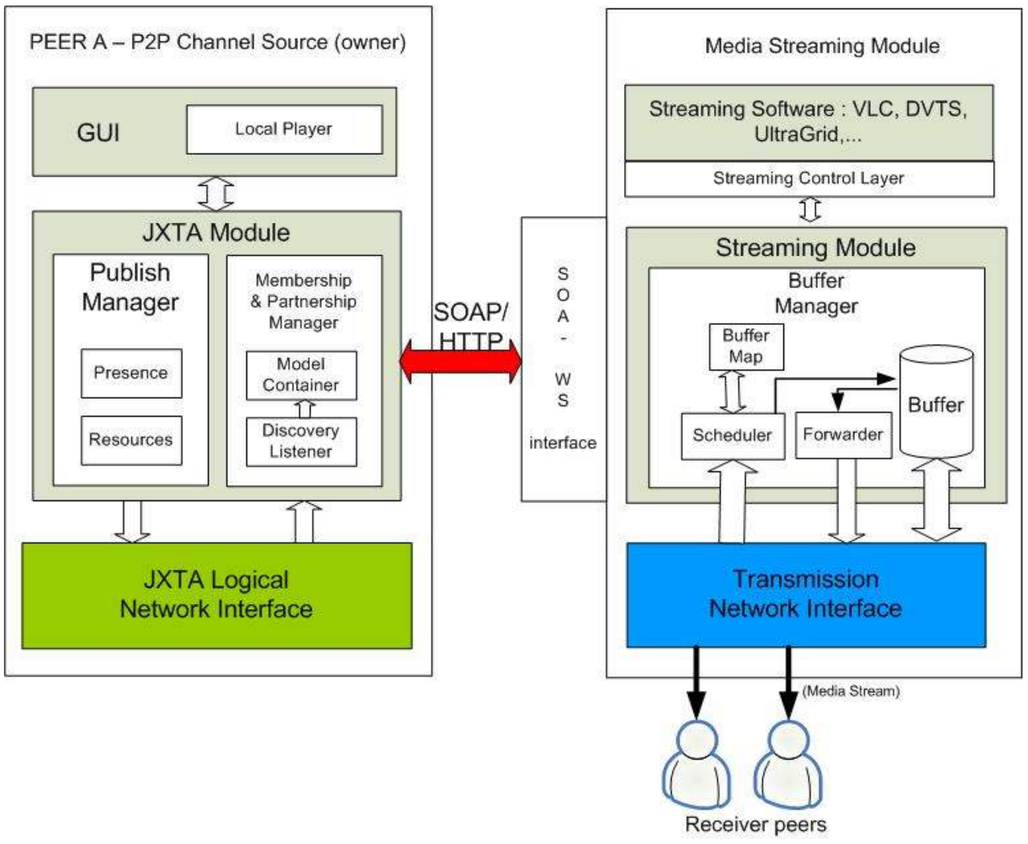


Fig. 8.2 SOA proposed architecture

Other future tasks that must be considered are listed below.

- Further study of ALM structures in order to improve the performance of high bitrate scenario and improve the involved management tasks, especially in those ones that refer to peer departures detection and reorganization.
- Implementation of the low bitrate scenario.
- Test high bitrate transmissions above 20 Mbps in order to determine the top transmission bitrate that can be achieved using the implemented methods (java.io with buffer and Packet Reflector).
- Secure communications between peers through JXTA sockets. JXTA allows adding security to the communications by using cryptographic protocols such as Transport Layer Security (TLS) [29].
- Optimize JXTA messaging by compressing the exchanged messages using algorithms such as GZIP.
- Consider the Session Initiation Protocol (SIP) [35] as P2P signalling protocol. It can be followed the SIP-CMI [36] guidelines. SIP-CMI platform is an open, flexible, scalable testbed to support a wide and extensible set of next-generation continuous media services. This platform follows the principle that any continuous media service can be accessed by using the SIP protocol, regardless of the nature of the service; for example videoconference or streaming.

8.3. Environmental Impact

Nowadays, the environmental impact of a project is a very important issue to take into account and on many occasions it determines the feasibility of a project and, therefore, its approval or rejection.

However, when talking about software projects the estimation of the environmental impact can be difficult to obtain or maybe it can not be noticed at first sight. Next are listed some positive and negative aspects consequence of the development of network applications, which is the environment where P2P applications are applied.

Negative aspects

- **Increase of the number of infrastructures**. Networks are composed by wires, data centres, high voltage power lines, aerials, computers, interconnection devices and so on. This supposes big infrastructures that the society and the environment must assume in order to guarantee the provisioning of network services. These infrastructures normally provoke visual impact.
- **Hardware devices get obsolete very fast**. Every year the requirements needed by some applications grow. Consequently, hardware must be updated in order to run the applications properly. This situation causes that many devices are thrown away and are replaced by new ones, increasing the generation of residues. Most of the residues derived from hardware devices are not biodegradable.
- **Health impact**. The presence of electromagnetic radiations in the environment is continuously growing mainly due to the intensive deployment of wireless technologies and infrastructures. Although there are no conclusive studies about the effects of electromagnetic radiations on human health, many people reject the deployment of these technologies because it is believed that these kinds of radiations are harmful.
- **Social impact**. Unfortunately, P2P is currently used mainly for the illegal spreading of copyrighted files. We believe that P2P is the future of content distribution and will mature from the current 'wild west' into a respectable business solution.
- **Network impact**. P2P is not in decline; in fact it is growing at a sharp rate [33]. The "vast majority" of P2P traffic (60 % of the total of the traffic in Internet in 2004) is of files more than 100MB. While most of this is video, there are other things such as CD images for open source software. This great volume of traffic can collapse current available bandwidth.

Positive aspects

- **Social networks**. People can communicate with their contacts thanks to text, audio and video applications everywhere and whenever they want. People can also meet new contacts according to specific interests.

- **Telematic communications**. Decrease of the necessity to use urban transport avoiding face-to-face communications or meetings. This implies a reduction of fuel consumption and other types of energies, decreasing environmental pollution.
- **Multimedia content digitalization**. Digital support supposes a decrease on the space required for contents storage. In addition, streaming applications allow the distribution of multimedia content without the necessity of material support and enables the reuse of these contents.
- **P2P computing**. P2P applications allow optimising digital transmissions creating a virtual network where all the peers can cooperate for achieving a common interest. When talking about applications that require a great usage of network resources such as bandwidth, P2P computing becomes a suitable scheme.

Once defined all the positive and negative aspects it is time for defining a valuation of the environmental impact of a project. A specific weight can be given for each point under consideration, and a final valuation can be obtained according to a symbolic mark.

This project consists of the creation of a virtual network that enables interactivity among its users and the addition of new value-added services. The proposed platform can be deployed under current network infrastructures, so the harmful environmental impact of the project can be valued as very low. However, the social impact can be high but in a useful or beneficial way.

8.4. Personal Conclusions

This project has been very interesting and motivating for me. P2P streaming supposes a hot topic in current network applications as manifests the increasing popularity of P2P streaming applications [32] that can be found on the Internet. Moreover, current dominant network operators are launching IPTV solutions to deliver TV and Radio on streaming over their multicast networks, these solutions represent a cornerstone of their commercial catalogue. P2P applications can be considered nowadays as killer applications, so it is a privilege for me to have the opportunity of contributing to this field.

It must be emphasized that it is being considered a big project. It supposes a great effort to develop an efficient P2P streaming application, specially focused on high quality streams of audio and video.

A part from learning new useful concepts, technologies and tools, the most important thing that I learned was to adopt an attitude when trying to solve the problems I found, which I needed to overcome. An enterprising spirit, self learning and workgroup capabilities are key features in the world of software development. Current technologies offer solutions to current necessities but as new problems appear or current necessities change, these technologies and tools get obsolete and new ones must appear. This is why an adaptable and innovative attitude must be adopted.

CHAPTER 9. REFERENCES

[1] D. Meddour, M. Mushtaq and T. Ahmed, “Open Issues in P2P Multimedia Streaming”, MULTICOMM’06, June 2006, Turkey.
[2] C. Yeh and L. S. Pui, “On the Frame Forwarding in Peer-to-Peer Multimedia streaming”, P2PMMS’05, November 2005, Singapore.
[3] PPLive software, Website: http://www.pplive.com/en/index.html
[4] SopCast software, Website: http://www.sopcast.org/
[5] Sun Microsystems, “JXTA v2.3.x Java Programmer’s Guide”, April 2005. http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf
[6] X. Zhang, J. Liuy, B. Liz, and T. P. Yum, “CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming”. In Proc. IEEE INFOCOM, March 2005.
[7] T. Su-Wei and G. Waters, “Building Low Delay Application Layer Multicast Trees”. In Proc. EPSRC, Liverpool, June 2003.
[8] D. A. Tran, K. A. Hua and T. Do, “ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming”, In Proc. IEEE INFOCOM 2003.
[9] T. Tsuchiya, H. Yoshinaga, K. Koyanagi, A. Honda and A. Minami, “STARCast: Streaming Collaboration Platform using the Overlay Technology”, In Proc. SAINTW’06, IEEE Computer Society, January 2006.
[10] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, “AnySee: Peer-to-Peer Live Streaming”, In Proc. IEEE INFOCOM, April 2006.
[11] B. Eckel, “Java Thinking in Patterns”, May 2003. http://www.mindview.net/Seminars/ThinkingInPatterns.
[12] VideoLan Home Site, WebSite: http://www.videolan.org/
[13] Apache Commons Collections of Buffers, Website: http://jakarta.apache.org/commons/collections/apidocs/org/apache/commons/collections/buffer/package-summary.html
[14] jVlc Project, Website: https://trac.videolan.org/jvlc
[15] Java Swing Framework. Website: http://www.newt.com/java/swing.html
[16] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J-C. Hugly, E. Pouyoul and B. Yeager, “Project JXTA 2.0 Super-Peer Virtual Network”, May 2003. http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf.
[17] N. Magharei and R. Rejaie, “Understanding Meshbased Peer-to-Peer Streaming”, in Proc. International Workshop on Network and Operating Systems Support for Digital Audio and Video, Newport, Rhode Island, May 2006.
[18] L. Gong, “Project JXTA: A Technology Overview”, Website: http://www.jxta.org/project/www/docs/jxtaview_01nov02.pdf
[19] MVC overview. Website: http://java.sun.com/blueprints/patterns/MVC-detailed.html
[20] Homesite of Log4J Project, Website: http://logging.apache.org/log4j/docs/index.html
[21] Homesite of Apache Ant Project, Website: http://ant.apache.org/manual/index.html

[22] Homesite of Apache Commons Collections, Webiste: http://jakarta.apache.org/commons/collections/apidocs/index.html
[23] CVS definition by Wikipedia, Website: http://en.wikipedia.org/wiki/Concurrent_Versions_System
[24] Homesite of SAX Project, Webiste: http://www.saxproject.org/
[25] Excelsior JET software homesite, Website: http://www.excelsior-usa.com/jet.html
[26] Packet Reflector Project developed by CESNET. Website: http://www.cesnet.cz/doc/techzpravy/2003/rtpreflector/
[27] DVTS information, Website: http://www.sfc.wide.ad.jp/DVTS/
[28] UltraGrid information, Website: http://ultragrid.east.isi.edu/
[29] T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.1”, Request for comments RFC number 4346, Network Working Group, April 2006.
[30] P. Lorente, “Diseño y desarrollo de un Media Procesor basado en servidores de streaming utilizados en Internet 2”, EPSC-UPC thesis, February 2005.
[31] A. J. González, “Sistema de Localización Multimedia Distribuido”, EPSC-UPC thesis, January 2005, http://bibliotecnica.upc.es/PFC/arxius/migrats/35777-1.pdf
[32] Complete list containing Opensource/freeware p2p streaming systems available on the internet today (February 2007), Website: http://orblive.com/modules/newbb/viewtopic.php?topic_id=13&forum=4
[33] Study of the impact of P2P in 2005, Website: http://www.cachelogic.com/home/pages/studies/2005_06.php
[34] Java Multimedia Framework, Website: http://java.sun.com/products/java-media/jmf/
[35] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," IETF RFC 3261, June 2002.
[36] M. Hurtado, A. Oller, and J. Alcober, "The SIP-CMI Platform- An Open Testbed for Advanced Integrated Continuous Media Services," TridentCom 2006.
[37] Advanced Installer by Caphyon, Webiste: http://www.advancedinstaller.com/java.html

CHAPTER 10. ACRONYMS

ADSL	Asymmetric Digital Subscription Line
ALM	Application Layer Multicast
API	Application Programming Interface
ATSC	Advanced Television Systems Committee
AVL	balanced binary search tree
BM	Buffer Map
CBR	Constant BitRate
CIMS-Live	Collaborative and Interactive Media Streaming Platform
CPU	Central Processing Unit
CVS	Concurrent Version System
DHT	Distributed Hash Table
DONET	Data driven Overlay Network
DV	Digital Video
DVB	Digital Video Broadcasting
DVD	Digital Versatile/Video Disc
DVTS	Digital Video Transmitting System
GUI	Graphical User Interface
HD	High Definition
IP	Internet Protocol
IPTV	Television over IP
JDK	Java Development Kit
JMF	Java Multimedia Framework
jVLC	Java VideoLan Client
JXTA	Juxtapose
LTM	Location aware Topology Matching
MPEG-TS	Moving Pictures Expert Group Transport Stream
MVC	Model View Controller
P2P	Peer to Peer
P2PRadio	Radio streaming over P2P
P2PTV	TV streaming over P2P
PC	Personal Computer
PDA	Personal Digital Assistant
QoS	Quality of Service
RAM	Random Access Memory
RTCP	Real Time Control Protocol
RTP	Real Time Protocol
SAX	Simple API for XML
SD	Standard Definition
SMS	Short Message Service
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VBR	Variable BitRate
VCD	Compact Disc Digital Video
VLC	VideoLan Client
XML	eXtensible Markup Language

ANNEX A. TECHNICAL SPECIFICATION

This annex contains an extension of technical specifications not described in the main document.

A.1. Technical specification

The application interacts with different modules in order to offer the functionalities described.

Next it is shown some use case diagrams that allow observing clearly the functionalities that the system offers to the users. Proceedings are described using cards.

A.1.1. Front-end Module (GUI)

The GUI contains several flaps as the following ones:

- List of contacts
- Groups. Shows a chart of the found groups. Making double click over a precise group, it is possible to see expand the tree in order to see which peers are contained.
- Channels. The same as the groups but with media channels information
- File sharing. Offers the user an interface for searching contents published on the P2P application.

A.1.2. Start-up Module

This module is in charge of (Fig. A.1):

- Authentication of the user in the application
- Configuration of the user preferences and profile

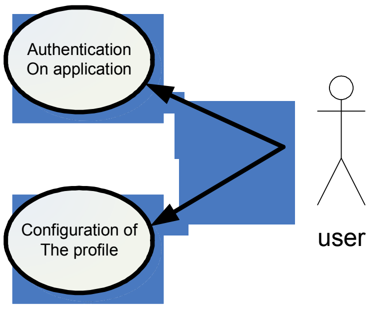


Fig. A.1 Start-up

The application accesses will have to interact with the authentication service. This service can be implemented in a distributed way across the P2P net (it could be considered a supernode net in charge to keep the net authentication service). A more centralized vision alternative it would be to create an authentication service based in web services.

Nevertheless, first the authentication will be local. That is, the access data is going to be read from a local file.

Once the user starts the application, it is well authenticated, it can set-up its user preferences. These preferences allow setting up the following issues.

- GUI Skin.
- Type of bandwidth, connection.
- Searching filters (e.g. parental control for adult contents).
- Force searching criterion (e.g. codecs, file types, and so on).

Preferences data is locally stored into an XML file.

Name	Start
Description	Authentication of the user
Actors	User without being authenticated
Preconditions	User has started the application
Normal flow	1. System shows the register panel to insert the user name and the password. 2. User inserts data. 3. System checks the validity of the data (in a local or distributed way, or using WebServices) and loads the application with the user information.
Alternative flow	3. If the introduced data is not correct, the user is able to try to authenticate once again.
Output	Known user.

Preferences

Name	Preferences
Description	Modify/Add user preferences
Actors	Authenticated user
Preconditions	User correctly authenticated
Normal flow	1. System shows for the first time the preferences configuration panel if the user accedes to the service for the first time. 2. The user inserts his/her preferences (skin, type and connection velocity, filters, credentials changes, multimedia module's options). 3. Once the data is inserted it is stored and each time the user authenticates itself in the application it is loaded.
Alternative flow	1. The authenticated user is not the first time that accedes to the service. 2. Loaded preferences. 3. It is possible to manual modify them through the application tool menu.
Output	Authenticated user and with stored preferences.

A.1.3. User/group management module

Fig. A.2 shows the actions that an application user can do with other users.

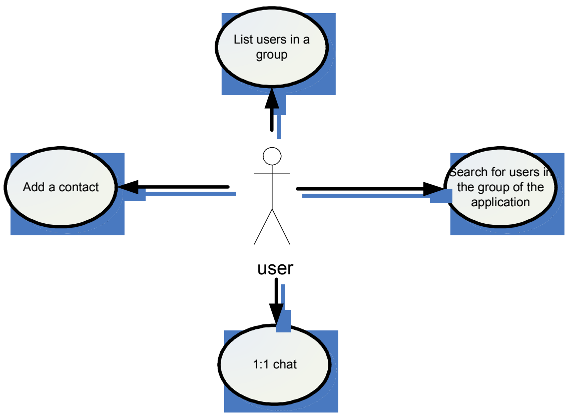


Fig. A.2 Actions with users

Fig. A.3 shows the actions that a user can do with the groups present in the application.

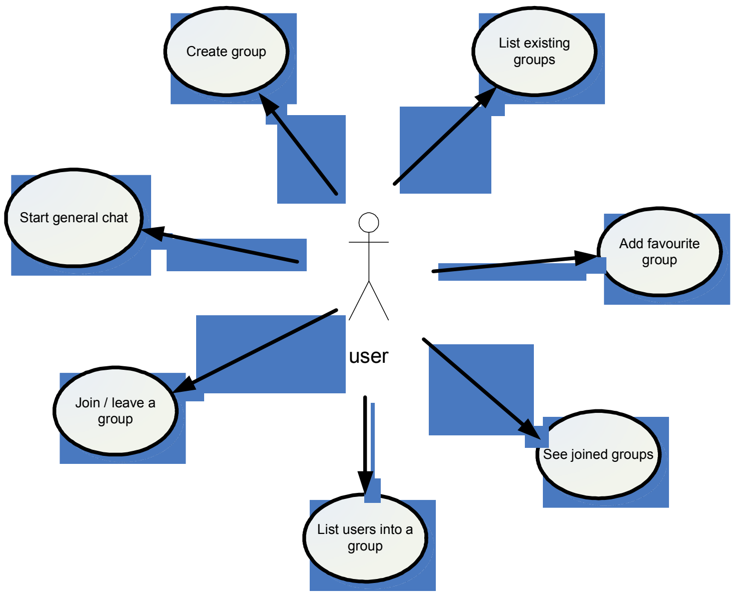


Fig. A.3 Actions with groups

Addition /suppression of a group or contact

Name	To Add/eliminate a group
Description	To add or to eliminate a group or a contact from a user's contact list
Actors	Authenticated User
Pre-conditions	The User clicks the button of adding/eliminating group/contact
Normal Flow	1. The application shows a panel to insert the data and to realize the requested action. 2. The user inserts the data 3. The application verifies the validity of the same ones and if it is possible it realizes the needed action.
Alternative Flow	3. If the data is not correct or it is incomplete the application allows the user to reinsert the data.
Output	Modification of the contact list.

Change of the state of the user

Name	Change of the state of a user
Description	Chance of changing the current user’s state.
Actors	Authenticated User
Pre-conditions	The User clicks the changing state button
Normal Flow	1. The application list the possible states that a user can set 2. The user chooses the conditions in which he/she wants to remain 3. The application spreads among the other peers its current state
Post-conditions	General modification of the user state

List groups/users

Name	To list groups and/or users
Description	To list concrete groups or users
Actors	Authenticated User
Pre-conditions	The user starts the application
Normal Flow	1. The user starts the application and can observe the state and in what group his/her contacts are present
Alternative Flow	1. The user wants to list the users of a concrete group and clicks the button for listing groups/peers 2. The application verifies the data and displays data
Post-conditions	The user knows the existence of other peers and groups

Favourite Groups

Name	Favourite groups
Description	To add a group into favourite groups
Actors	Authenticated User
Pre-conditions	The User clicks the favourite groups button
Normal Flow	1. The application shows a list with the groups which the user belongs to. 2. The user selects what groups he/she wants to add to his favourite groups. 3. The application verifies the information, realizes the action and adds the group or groups to a new one, which is more accessible for the user.
Post-conditions	The application has a new section where appear the groups that belong to favourite groups.

Chat

Name	Chat
Description	To initiate a text conversation between the user and one o more contacts
Actors	Authenticated User
Pre-conditions	The user clicks Chat's button
Normal Flow	1. The application shows a panel with the possible contacts for starting chat 2. The user selects the concrete contacts 3. The application loads a panel where the user can realize some actions as sending texts, files, synchronization of agenda’s information 4. The user realizes any action 5. The application takes charge of sending the action to one or more contacts.
Alternative Flow	If some type of problem exists, the application will notify to the user the inability to realize the action
Post-conditions	Two contacts are communicated

ANNEX B. INTERACTION FLOW

Fig. B.1 shows the interaction among different modules in order to provide a generic service.

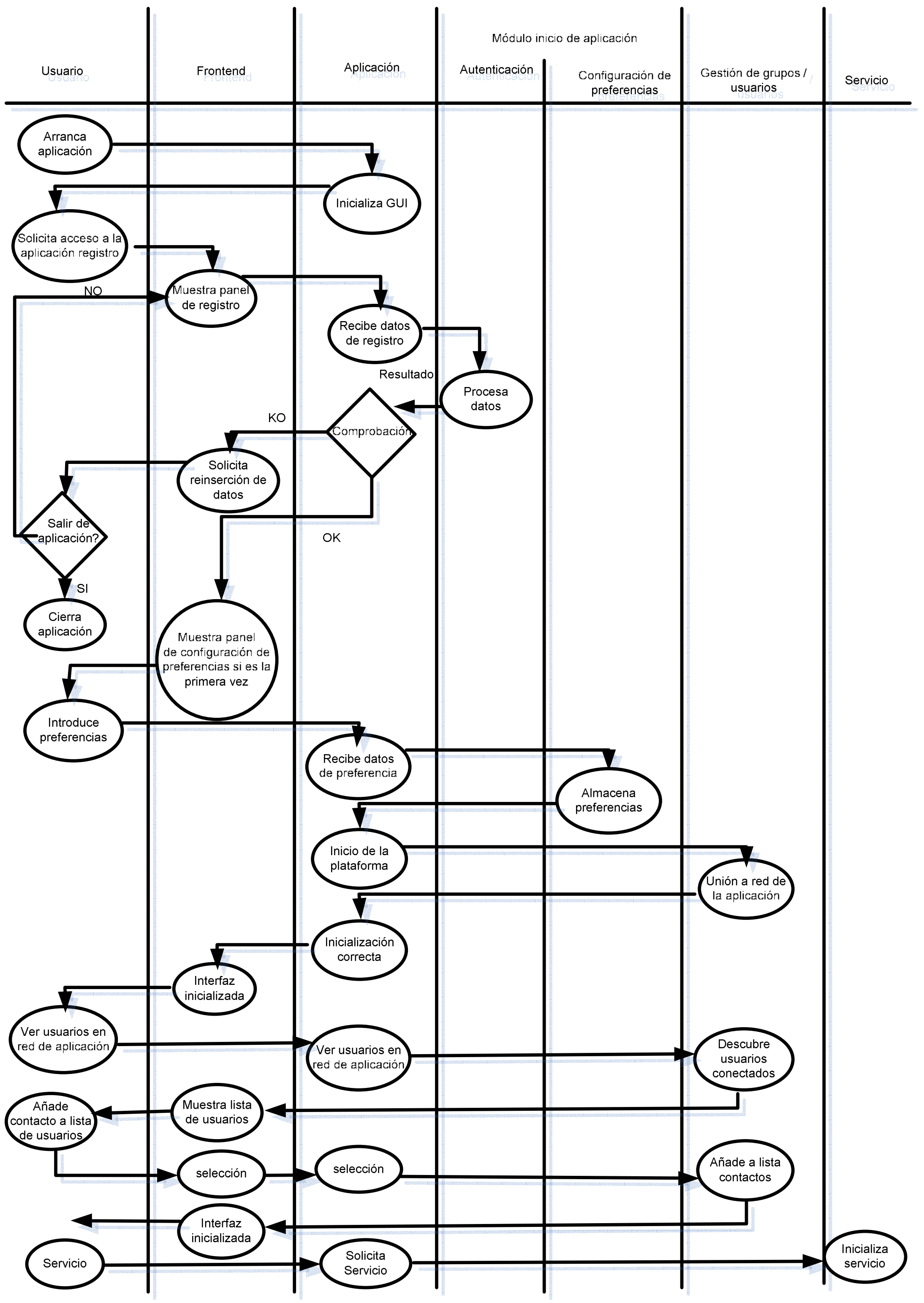

Módulo inicio de aplicación
Usuario
Frontend
Aplicación
Autenticación
Configuración de preferencias
Gestión de grupos / usuarios
Servicio
Arranca aplicación
Inicializa GUI
Solicita acceso a la aplicación registro
NO
Muestra panel de registro
Recibe datos de registro
Resultado
Procesa datos
KO
Comprobación
Solicita reinserción de datos
Salir de aplicación?
OK
SI
Cierra aplicación
Muestra panel de configuración de preferencias si es la primera vez
Introduce preferencias
Recibe datos de preferencia
Almacena preferencias
Inicio de la plataforma
Unión a red de la aplicación
Inicialización correcta
Interfaz inicializada
Ver usuarios en red de aplicación
Ver usuarios en red de aplicación
Descubre usuarios conectados
Añade contacto a lista de usuarios
Muestra lista de usuarios
selección
selección
Añade a lista contactos
Interfaz inicializada
Servicio
Solicita Servicio
Inicializa servicio

Fig. B.1 Start-up and service interaction

ANNEX C. PAPER SUBMITTED TO EURO-NGI

The paper submitted to the *3rd EURO-NGI Conference on Next Generation Internet Networks - Design and Engineering for Heterogeneity* titled "*CIMS-Live: Collaborative and Interactive Media Streaming Platform in P2P Environment*" which presents the proposed P2P platform is attached next. This paper is currently under review and pending for approval.

CIMS-Live: Collaborative and Interactive Media Streaming Platform in P2P Environment

André Ríos, Alberto J. González, Antoni Oller and Jesús Alcober

Department of Telematics Engineering, Technical University of Catalonia / I2cat Foundation.
Barcelona, Spain.
Email: {andre.rios\antoni.oller\ jesus.alcober}@upc.edu, alberto.jose.gonzalez@i2cat.net

Abstract--. **Media streaming has become a hot topic in Internet research, especially when it is related to live streaming audio and video applications and peer-to-peer (P2P) networks. In general, they have to cope with challenging problems in their design and implementation, due to the dynamic and heterogeneous nature of P2P networks. In this paper, some new features that other P2P media streaming applications do not consider or barely specify are introduced. The connection and hardware requirements are different if low bitrate or high bitrate streams are transmitted, but all peers are usually considered identical, which is not a good approach considering an heterogeneous environment. Furthermore, in order to achieve distinction among peers, it is necessary to provide a mechanism that enables the exchange of the specific capabilities (including transcoding) of each peer. Moreover, these applications have difficulties to extend its features by adding new services or when trying to modify any preloaded data such as the channel list provided to the user, making impossible to guarantee the source availability all the time. Finally, interactive services haven't been deeply discussed. This paper proposes the design and implementation of a collaborative and interactive media streaming platform that allows coping with the issues before mentioned. The platform integrates different mechanisms that permit real-time distribution of multimedia contents with different qualities including high bitrate media streaming (e.g. HD). This platform is a novel hybrid solution based on JXTA, DONET and ALM (application layer multicast), providing a flexible and extensible platform according to new requirements and future value-added services. In addition, this platform allows creating an architecture with two overlay layers: a JXTA based logical layer, in charge of signalling and metadata exchange, and a UDP socket based layer, in charge of the multimedia transmission of unicast traffic. Consequently, the mismatch between transmission and physical layers can be reduced by means of the data obtained from the JXTA layer, which is updated through time.**

Index terms—**Peer-to-Peer, DONET, JXTA, Media Streaming, HD.**

I. Introduction

Recently, P2P networks have gained popularity thanks to the deployment of file-sharing applications. However, nowadays real-time media streaming applications arouse a great interest both to commercial level and academic research. However, live streaming introduces new challenging problems [1], [2] different to ordinary file-sharing. In general, media streaming solutions have different features that determine the operation of the applications. For example: the large volume of media data along with stringent timing constraints, the dynamic and heterogeneous nature of P2P networks and the unpredictable behaviour of peers.

It is possible to emphasize some important issues closely related to P2P media streaming which are mainly associated to two aspects: collaboration between peers and interactivity. Related to collaboration between peers, the following issues have to be taken into account: dynamic peer discovery, peer relationship maintenance and exchange of capabilities (transcoding information). Related to interactivity, the issues are the membership service and the messaging exchange.

From the operative point of view, when a peer starts a streaming P2P application, it has no information about other peers, which are also present in the virtual network. This implies that an efficient discovery mechanism must be initiated in order to find other peers and start receiving the desired media stream from the best ones.

Another key point is the maintenance of the relationship among peers and their update. Due to the heterogeneous environment of P2P networks and the unpredictable behaviour of peers (e.g. constant appearance and disappearance and stable and dynamic membership), presence mechanisms must be incorporated to know which peers are up, especially to ensure the state of the provider peers (partners) of a specific peer. Therefore, a peer must update its relationships to replace those partners that are down and, consequently, to not decrease the received stream quality.

The process of discovery is especially important when referring to live streaming applications, because depending on the number and specific features of the found peers, the desired stream will be received under some conditions. In the worst case, it will not be received or it will be received with losses or delay, causing the rejection of the final user due to unacceptable visual quality.

P2P networks are composed by thousands of different peers, each of them with specific features, such as bandwidth, decoding capabilities or storage capacity. It is a main issue to select the providers that better fulfil the capabilities and requirements of a peer that wants to start receiving a media stream; therefore, a dynamic mechanism to advertise the concrete capabilities of each peer must exist. Provided a stream is delivered at a specific bitrate and codec, not all of the peers in the network can receive this stream because they have not got enough downlink bandwidth. This problem is solved if the streaming platform enables to advertise peers with adaptation functions or transcoding. For example, if a source is sending a high definition stream of 20 Mbps (MPEG2-TS, 1280x720), maybe most of the peers with current Internet access (e.g. ADSL or Cable Modem) can not receive this stream delivered at that bitrate, but if a peer has transcoding capability, possibly this stream can be adapted to a smaller bitrate.

Related to interactivity among peers, existing P2P applications do not promote value-added interactive services among connected peers, currently known as social networks. When talking about terrestrial TV and radio broadcasting the interaction with the viewers or listeners is a hot topic. A way to interact with the audience is to make available an infrastructure to receive SMS, which can be expensive and complex. It is also usual to install devices dedicated to calculate the share gathered from a TV programme and determine the success of the programme according to the obtained data. P2P applications offer the possibility to create interactive experiences to the users by providing cheap and simple mechanisms to interact with the connected user. For instance, peers can chat at playing time. Using P2PTV, the broadcasters can get the feedback of online users at real time of the same way that TV broadcasters do it in current TV channels with SMS, but with the difference that IP infrastructure is much cost effective. Statistical information such as number of current/average viewers, time of playback, user profile can be directly obtained thanks to an efficient membership service.

It must be also mentioned that popular streaming P2P applications focused in delivering TV streams on the Internet, such as PPLive [3], SopCast [4], have a default static list of P2PTV channels, which identify the well known media streaming sources. In spite of the fact that these lists can be updated, this process is inflexible because when a change in the contents is required, retrieval or addition of channels, the whole application must be online updated or, even, it is mandatory to download a new release of the whole application. This feature causes that these applications can not be easily extended.

A problem derived from the provided static channel list is related to the source availability. The fact that the list of channels is prefixed, it does not guarantee that they are fully available. In this sense, it is when a user wants to view a channel that the system checks if the source is available. In the case of being unavailable, the user has wasted its time trying to view this channel.

Taking into account this environment and with the idea to solve the issues before mentioned, this paper proposes the design and implementation of a media streaming platform called: CIMS-Live (**C**ollaborative and **I**nteractive **M**edia **S**treaming Platform). This platform is a hybrid solution based on the P2P Java API called JXTA [5], DONET (Data-driven Overlay Network) [6] and ALM (Application Layer Multicast) structure [7], [8]. Thanks to this platform, the publication and discovery of media streaming channels and peers can be fully decentralized and automated. This infrastructure makes possible to create an extensible application which allows incorporating new services. Furthermore, the flexible publication mechanisms provided by JXTA allows announcing the specific capabilities of the different peers present in the network and, consequently, finding the best peers for critical services such as media streaming.

This paper will first introduce in the section II the related works about P2P streaming applications. In section III, the system overview is presented and also the work environment and the P2P architecture used is described. In section IV, the system design is presented and the different elements that constitute the application are detailed. Next, in section V, two P2P media streaming mechanisms used in the solution are described, which allow the distribution in real time of multimedia content with different qualities. Finally, in section VI, a developed prototype and its development environment are described in order to check the different programmed functionalities and to validate the concepts on which the application is based. In the section VII and VIII concludes this paper with the future woks and conclusions.

This work was supported by MCyT (Spanish Ministry of Science and Technology) under the project TIC2003-08129-C02, which is partially funded by FEDER.

II. Related Works

In recent years, there have been significant researches into a variety of issues related to P2P media streaming. However, it is necessary to mention that there are some popular media streaming applications whose internal operation is barely known; therefore, it is difficult to analyze the algorithms and mechanisms used. On the other hand, there are academic developments associated to universities or developer communities that have been well specified and published. One example of this last type is STARCast [9], which is a JXTA based platform that offers streaming services.

STARCast uses a ALM structure for streaming tasks. But, opposite to CIMS-Live, it treats all the streams in the same manner, that is, it does not distinguish between low and high bitrate streams.

Moreover, there are different strategies related to the selection of the partners and to the construction of an efficient overlay network for multimedia streaming. Next, some techniques applied or associated to this study will be mentioned.

Borrowing ideas from IP multicast technology, tree-based protocols can be considered simple, efficient and scalable. Specifically, the mail goal of single tree protocols is to build a scalable multicast tree with high efficiency. A representative example of this is ZIGZAG [8], which is a P2P technique that allows the media server distributing content to many clients, by organizing them into an appropriate tree rooted in the server. Basically, it is ALM tree that has height logarithmic with the number of clients and a node degree bounded by a constant.

Another well known example is DONET, which is a P2P technique that does not need any kind of complex tree structure for data transmission. DONET includes a Gossip based partnership management algorithm and an intelligent scheduling algorithm in order to provide a continuous distribution of streaming contents. DONET is currently implemented in a commercial application called CoolStreaming [6].

AnySee [10] is a P2P live streaming system based on an inter-overlay optimization scheme and where the resources can join multiple overlays. This system creates an overlay network of the peers in the application according to location-aware topology matching (LTM), but it supposes a prominent management effort.

III. System Overview

In this section it is going to be explained which are the mechanisms that allow constructing the collaborative and interactive P2P platform and how the peers are organized into different overlay networks.

The proposed platform is based on the P2P Java API provided by the JXTA project, which offers automatic mechanisms for publishing and discovering peers, groups of peers, and every desired resource. Thanks to this API it is easy to develop a full featured P2P application where peers are self organized into logical groups. The self-grouped peers can discover each others in an easy way, and it is simple to create membership and presence mechanism.

In addition, a goal feature offered by the JXTA platform is that the more time the application is running, the more global knowledge of the logical network or group can be achieved thanks to the discovery requests and events generated or caught by the peers. According to the information gathered from the discovery, membership and presence processes, a peer can optimize its transmission layer by updating its relationships with other discovered peers.

The powerful mechanism for publishing advertisements of JXTA, also makes easy to advertise the specific capabilities that each peer has, even it can be periodically done if they change in time. Peers can work attending to three profiles: source, transcoder and simple peer. Tin the next section, the profiles will further be described.

As it has been said, the platform makes possible to deliver media streams, which supposes a key point. In this paper it is proposed to adopt two different strategies depending on the kind of stream that is going to be delivered: low (< 2Mbps) or high (> 2Mbps) bitrate steams, because the connection and hardware requirements are not the same in both environments. For further detail, see section V.

On one hand, it is used a similar mechanism to the one suggested by DONET in order to deliver low bitrate streams, which is based on buffer segmentation and multiple-peer retrieving of data blocks. Nevertheless, it is used the information exchanged in the JXTA network to configure and update the scheduling algorithm and also exchange the required information such as the Buffer Map of each peer. Thanks to the JXTA layer, each peer can dynamically discover other peers and optimize its transmission layer with the obtained information.

Moreover, a peer can obtain information of proximity to other peers by exchanging some messages at application level in order to discover what the delay between two peers is. Once obtained the delay and its variation (jitter), a peer can establish relationships with the best found peers.

Another new feature presented by this platform is that it enables everyone to publish a stream, that is, to be a streaming source and make it available to everybody. It is also allowed publishing into a specific streaming group (channel) those peers with specific capability of transcoding in order to adapt the media stream to limited devices.

Finally, new interactive services such as chat, file sharing, opinion poll or survey can be added to this platform.

A. P2P Architecture

This section shows the proposed pure P2P architecture based on the Java API named JXTA.

The architecture can be mapped to different layers. All the peers are nodes physically separated by a certain number of hops and a variable delay, which supposes that a data packet sent from one node to another can go through many different links and cross many nodes until it arrives to destiny. The physical topology is a layer joined by lots of heterogeneous peers.

This physical topology is mapped to a logical overlay network, where all the peers that join the application are self organized. Thanks to the JXTA API, these nodes can be advertised, discovered and self organized in a logical group of peers. See Figure 2.

When the application starts (Figure 1), the first thing it must be done is to find the peer group created by the application, which contains all the peers connected to the application. Then, if the peer finds the group it will join it and then look for a Rendezvous peer in the group. If the group was not discovered, it will create the group. This also occurs with the rendezvous peer, that is, if the peer does not find the rendezvous, and it has enough capability, this peer will become rendezvous. Once connected to the rendezvous, the peer will start a thread for managing discovery events.

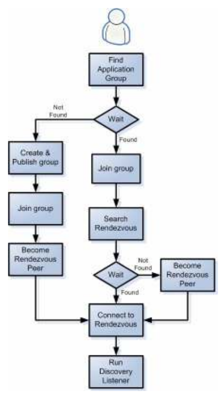


Figure 1: Start of the application.

In spite of the fact that JXTA provides mechanisms to send and receive data (Pipes and JXTA Sockets) they are not optimum for high data volume transferences due to the heavy weighted encapsulation of this data into SOAP/XML messages. This is the reason why it is used a unicast data transmission layer, specially aimed at media stream delivery. The JXTA layer is the responsible of signalling and other light services provided by the platform such as presence, chat, and others.

Thanks to the dynamic information exchanged among different peers in the JXTA network, a global view of the group can be achieved and the transmission layer can be set-up to achieve the best possible performance. The peers are always listening to the changes produced in the network, that is, the state of the peers and their features. It is proposed a feedback from the top layer to the transmission layer.

This architecture enables to optimize the relationships among peers as more time they are up by discovering other new peers.

Another key issue is that when a peer generates a discovery request it waits for some answers and, implicitly, just the closest peers will answer. With this approximation, the mismatch between physical and logical layers can be decreased. But this is not the only parameter to be taken into account when constructing logical links between two peers; it is also considered the delay between them.

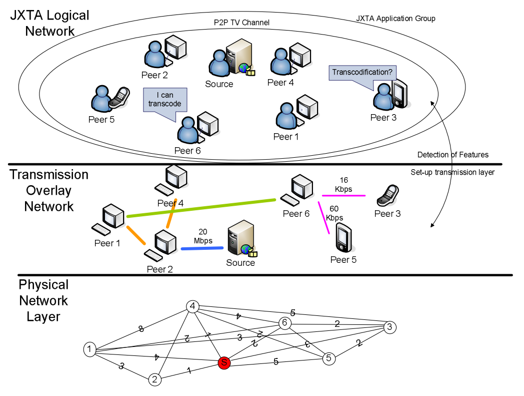


Figure 2: Layered architecture. 1) Physical Network Layer: it represents the physical topology of the different peers separated by physical links, the number placed over the link represents a symbolic cost estimated by *f*(hops, delay) between two end points; 2) Transmission Overlay Network: this layer consists of a group of unicast links between peers, this layer is responsible of transmitting the media packets; 3)JXTA Logical Network: this is the most abstract level which contains all the peers self organized. This layer is in charge of signalling and discovery tasks.

B. Peer Profiles

It must be specified the different roles carried out by each actor of the P2P system. It must be noticed that all peers in the application belong to a common group. The application creates a logical group that contains all the peers in the system and all the groups created by the peers of the application.

Different types of peers can be found according to its features. Each user connected to the application will become a peer of the logical group of the application and will be capable to discover and join other peers and groups present in the logical system. Each peer publishes its features: bandwidth, maximum number of connections accepted, delay to the source, hops to the source and role into the peer group: source, transcoder or simple peer.

Source: the owner peer of a specific channel group of media streaming. A P2PTV or P2PRadio channel can be mapped to a logical group in JXTA environment. This group is joined by all the peers that want to play a concrete stream, that is, all the peers have a common interest. When a peer wants to become a stream deliverer it must create a JXTA peer group advertisement describing the characteristics of the P2PTV channel: name, theme, description, bit rate, codec and any other required parameter. When peers discover the advertisements, they get ready to try to receive the media stream.

Transcoder: a peer that can receive a media stream and transcode it to a specific media format. Some peers can collaborate in the peer group by adapting the streams for other limited peers present in the group, for instance: PDA and mobile phones.

Simple Peer: this is the default profile of a user in the application group. Each peer can discover and join the different groups in the application JXTA world and discover the services it offers: media streaming, file sharing, and so on. Also, if the peer has enough capacity, it can become a source peer if it wants to deliver a media stream.

IV. System Design

The design of the peer application can be seen in Figure 3. The proposed design is done following the Model-View-Controller design pattern [11].

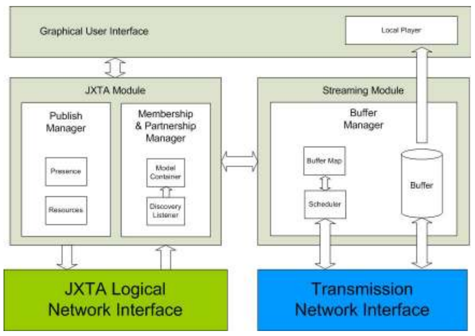


Figure 3: Generic system diagram of a peer.

Figure 3. depicts the system diagram of a peer: (1) Graphical User Interface (GUI); (2) JXTA Module, which maintains global view of the logical group, establishes and maintains the partnership with other peers, publishes its features and discovers resources on the peer group; (3) Streaming Module, which schedules the transmission of a media stream to other peers in the system according to the information gathered from the JXTA Module and also forwards the stream to the local player for user playback.

The system has two key interfaces: a) JXTA Logical Network Interface, which enables all the signalling; b) Transmission Network Interface, which allows the user receiving and sending streaming packets.

A. Graphical User Interface (GUI)

The application view (Figure 4) is constructed using the Java Swing framework [15]. The user can view all the gathered data from the JXTA network referenced to groups: media channels, chat rooms, connected peers, advertising, shared resources and any other new service added to the platform thanks to a generic services directory (yellow pages). The GUI listens to the actions of the user and generates events for its later processing.

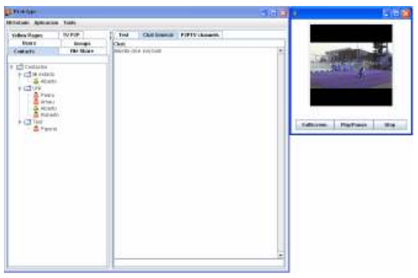


Figure 4: Graphical User Interface.

B. JXTA Module

This module is the responsible of interacting with the JXTA network interface. The main task it must carry out is the signalling of the system which will be managed by the partnership and membership submodules. All the signalling is based on JXTA socket messages and publish/discovery events.

All the information gathered from peers, groups, resources and services is stored in a model container submodule which represents the model layer of the application.

Membership Manager submodule: this submodule is entrusted of managing all the discovered data from the JXTA overlay network thanks to a dedicated thread (Figure 5) that listens to the generated events. These events are linked to new advertisements of peers, groups and other JXTA resources discovered by the application. When any advertisement or event of this kind is found, the GUI is automatically updated.

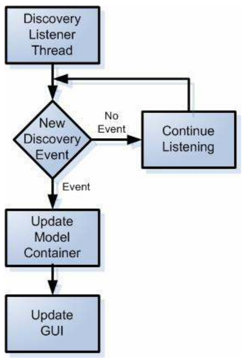


Figure 5: Discovery Listener Thread.

The main functions of the membership manager are:

- Manage connections and departures of peers
- Manage groups: connection and disconnection through JXTA Membership service.
- Update of the model container

This submodule can discover any resource published in the JXTA network such as peers, groups, resources, publishing, state of peers and Buffer Maps.

Partnership Manager submodule: is focused in maintaining a selection of the best found peers with which a user will interact in a streaming task. The selection of these peers is based on different parameters (Table 1) according to the kind of desired stream and to the specific features of the requesting peer.

TABLE 1: PARAMETERS UNDER CONSIDERATION

High bitrate stream	maximum number of connections accepted, time of playback, proximity to the source, and end-to-end delay
Low bitrate stream	available bandwidth, end-to-end delay, all the Buffer Maps of the known peers and the number of peers with the desired data available

Referring to low bitrate streaming, this module contains a list of parent peers and their Buffer Maps. It will interact with the Publish Manager sub module in order to publish the Buffer Map periodically through JXTA Sockets. When talking about high bitrate streaming it will contain a list of the main parent, its children and a backup parent.

Publish Manager submodule: this module is dedicated to any process that requires a publish operation in the JXTA network. It can publish among others:

- Peer Advertisement: contains the basic description of a peer (peer ID).
- Communication point: JXTA socket (Pipe)
- PeerGroup Advertisement: it is the specification of a group of peers. It is used a specific peer group advertisement in order to advertise media streaming channels: basically P2PTV and P2P Radio.
- Digital resources: media files, digital documents, and so on.
- Publishing

This module is also the responsible of periodically propagating the state of a peer through the presence service, its features and the Buffer Map when being part of a low bitrate streaming task. This module is implemented with JXTA multicast and unicast sockets in a specific peer group.

The JXTA network interface is based on JXTA Sockets, JXTA multicast Sockets and the JXTA discovery and publishing services.

C. Streaming Module

This module is the responsible of managing the streaming operations based on the updated information contained in the best peers list provided by the partnership manager. It is basically composed by different submodules: scheduler, buffer map and the buffer.

Buffer: storages the media stream datagrams received in local memory. The buffer is divided in uniform data blocks. It holds the data that will be sent to local player for the user local playback of the desired media stream and the data that will be made available to other peers into the system. It must be enough big in order to temporally accommodate a part of the media stream. There must be a trade-off between: required memory space, end-to-end delay requirements and peer failure detection and correction.

Buffer Map: manifests the state of the buffer.

- Low bitrate: indicates which blocks are available and the corresponding deadline of each block (timestamp).
- High bitrate: indicates the state of the buffer (how many blocks) and the current playback time.

Scheduler: this submodule decides which will be the supplier peer for each block (low bitrate) or media stream (high bitrate). Just selects the peer who will be asked for a part of its buffer.

V. P2P Media Streaming Mechanisms

Two strategies are adopted when a peer wants to start media streaming reception. Both strategies are focused on the kind of stream that is going to be asked for: low bitrate stream or high bitrate stream.

First, it must be clarified the conceptual separation between low bitrate streams and high bitrate streams. In this paper it is referred as low bitrate stream those streams delivered with a bitrate smaller than 2 Mbps, which is a common top bitrate used in popular P2PTV applications and offers a wide coverage for current broadband Internet access. So, when talking about high bitrate streams, it is assumed a bitrate higher than 2Mbps which allows delivering any other high bitrate format such as High Definition (HD), Standard Definition (SD), Digital Video (DV) and other high bitrate formats.

These two strategies are adopted because the requirements of bandwidth and buffering of a peer are not the same when trying to receive a low or high bitrate stream. High bitrate streams require a large bandwidth and buffer space available, uncommon in current Internet broadband domestic connections and devices.

A. Low bitrate media streaming

For low bitrate streams it is used buffer segmentation for subsequent sharing. Each peer periodically publishes its Buffer Map (BM) notifying which segments it has. Then, compares which segments lack in its buffer map and asks for them to the known best peers. When a peer starts, it interacts with some known peers. But the more time it is up, the more new peers it can discover and can calculate which the delay is and how does it changes in time among them by sending some probe packets. The delay can become a fine grained property, very important to determine which will be the supplier of a media block. The scheduler must be intelligent in order to optimize the task of media blocks requesting, so it is proposed to use the CoolStreaming/DONET scheduling algorithm.

B. High bitrate media streaming

High bitrate streaming needs high bandwidth and buffering requirements. This is why an ALM structure of paths is proposed (connection-oriented) formed by the peers that receive and forward the stream. In this paper is proposed a mesh-first [7][17] approach which builds up a mesh among the participating peers. The mesh is optimized towards the application requirements and is dynamically adjusted to accommodate the underlying network changes: peer arrival or departure. The distributed ALM algorithm can be run at each node.

In this scenario (Figure 6), there is no need of complex processing of the buffered data, just play it on local player and forward it to children. Peers do not only receive the requested stream, but also contribute to the media stream delivery by forwarding the stream to other peers.

When a high bitrate channel is joined the peer tries to find the best parent, which is the closest to the source and has enough number of available connections.

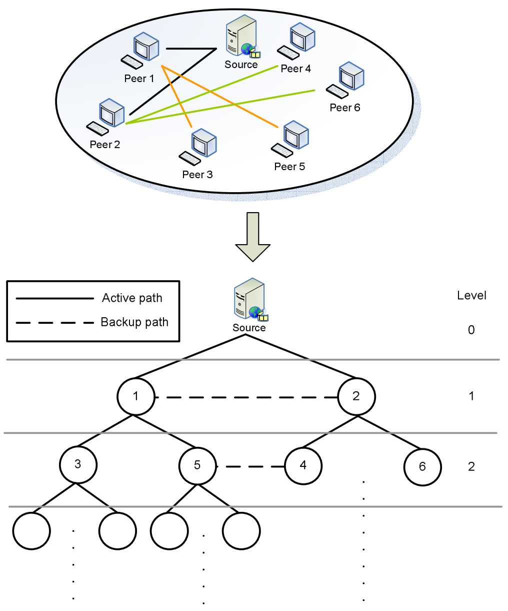


Figure 6: Application Multicast Tree structure. It can be seen the active and backup paths that conform a mesh-first approach.

Every peer supports a maximum number of connections (maxConnections) according to its available bandwidth. This maxConnections parameter determines how many children it can have.

Moreover, all the peers that join the high bitrate P2PTV channel have an active connection with a fixed parent and a backup parent peer in order to minimize network failure when a peer fails or leaves the channel.

In this connection-oriented scenario, the scheduling algorithm can be a simple round robin at delivery time. It must be said that, the proposed system allows updating the scheduling algorithms in both cases. It is not an open issue. It can be easily reconfigured thanks to the modular design of the application

Figure 7 shows a generic process when a peer wants to start a media streaming playback once discovered the P2PTV channel in the JXTA network.

The peer joins the discovered group, finds the peers in the group and discovers their features. With the gathered data it selects the best found potential suppliers and runs a parallel process for constructing a backup list of partners. When the buffer is enough filled, the peer can start playing the stream.

The termination process of the streaming playback is shown in Figure 8.

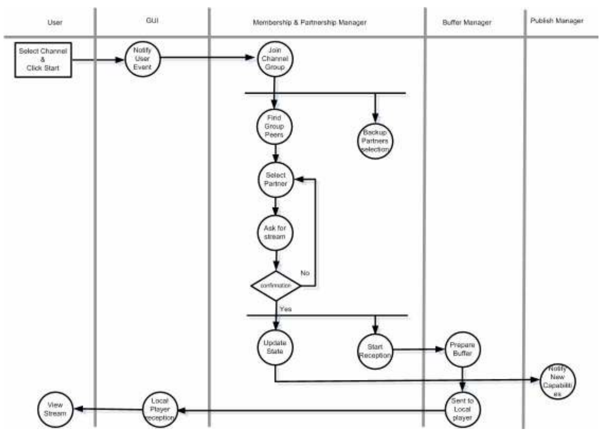


Figure 7: Start a media streaming playback

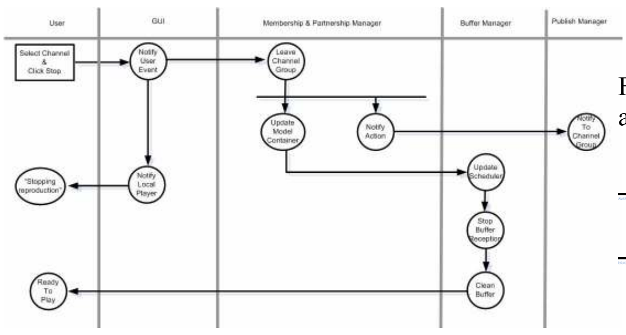


Figure 8: Stop a media streaming playback

There are two ways of leaving a group: leave a group voluntary or leave it involuntary due to a peer failure. When a peer voluntary leaves a group it must notify the event to the peer group through the JXTA Membership service. When a peer involuntary leaves, it must be detected through presence service.

In a high bitrate scenario, the mesh must be restructured. This paper proposes an easy process to redistribute the connections of the affected peers. Nevertheless, other algorithms can be easily added as it has been said and will be further studied in future work.

Otherwise, focusing on the realized implementation, it has been created a ALM tree structure rooted at source. So, when the ALM must be reorganized due to a peer departure, it is proposed that just a branch of the ALM tree will suffer the effects of the disconnection. Note that this approach implies that the source peer at level 0 must have global information of the connected peers, because it will determine which peer updates its relationships into the ALM tree or which will switch to its backup path. The cost of having centralized global view on the source is not a restriction due to the fact that it consists of metadata, not the data exchanged, and it does not suppose a heavy task of maintenance. Moreover, when having a global view of the ALM structure, it can be guaranteed a logarithmic cost proportional to the number of users, O log (n), when it is required to search a peer.

.

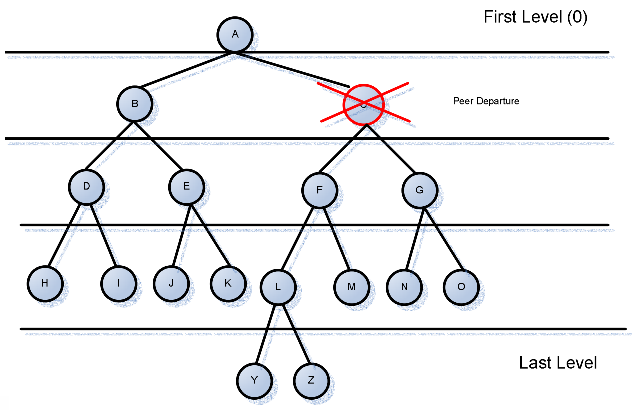


Figure 9: Peer C gets disconnected.

Figure 10 shows the tree view rooted at source after a peer departure.

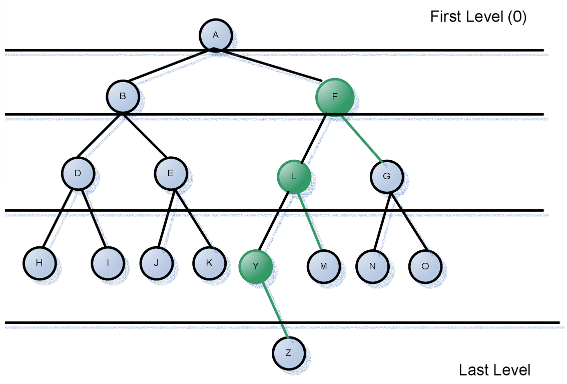


Figure 10: Peer F and its children are reorganized in a tree source-rooted scenario.

VI. Implementation and Testbed

It has been developed a simple prototype application based on JXTA. This application offers the following functionalities: 1:1 chat, 1:N chat, file sharing, videoconference and live video streaming. However, it will focus on live video streaming.

The application uses JXTA for discovery and publication tasks. Blocking UDP unicast sockets have been used in order to receive and forward the high bitrate media streams.

TABLE 2: DELIVERED MEDIA

	High Definition (HD)	Standard Definition (SD)	Digital Video (DV)
Video Codec	MPEG 2	MPEG 2	MPEG 2
Resolution	1280 x 720	720 x 576	1024 x 576
Duration	52 min	19 min	Live
Stream bitrate	18 Mbps	8 Mbps	3.5 Mbps

Stream encapsulation	MPEG_TS	MPEG_TS	MPEG_TS

Table 2 shows the characteristics of the media streams delivered when testing the prototype.

In all the tests, a source peer publishes a P2PTV channel into the application group and acts as rendezvous. This peer loads High Definition (HD), Standard Definition (SD) video files encoded with the features shown in Table 2. It can also load the media stream provided by a DV Camera for live streaming. Then, the source waits for some requesting peers in order to start the forwarding process. The deployed scenario can be seen in Figure 11.

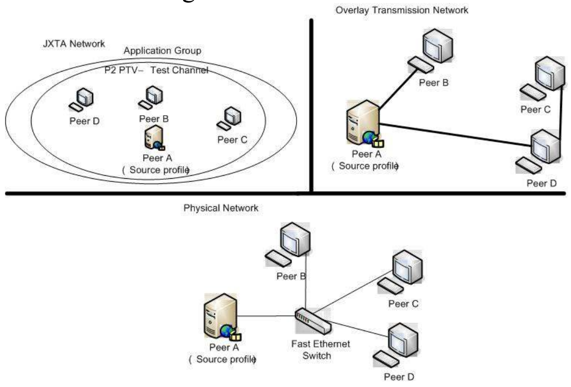


Figure 11. Development scenario

Every peer that acts as source configures the transmission network interface to receive the desired stream from a streaming server such as VideoLan (VLC) [12]. The connected peers receive the media stream from the most suitable peer. In this testbed it is just a peer with enough connections available, and then can start forwarding media to other connected peers.

When a peer receives a media stream, it is temporally accommodated into a 10 MiB buffer. The buffer has been built using the Priority Buffer implementation of Apache Commons Collection [13]. The main criterion for choosing one type of buffer is the process of the data that is going to be done. The received media stream is sent to the local player provided with the prototype application and to the children of the peer. This player is implemented using the jVlc Project [14], which enables to embed a VideoLan Player into the GUI developed with the Java Swing Framework.

The prototype development has been done following some well-known design patterns; the main one is Model-View-Controller (MVC), which allows separating the data plane (model) and the user interface (GUI) by introducing an intermediate component known as controller. The benefits of applying this pattern is that the application can easily grow when it is required a modification or addition in its components.

VII. Future Work

The P2P network proposed in this paper is unstructured, so it is under consideration to develop a structured organization of peers in order to offer a better and more reliable service. This is the reason why an overlay network of Rendezvous/Relay super peers [16] can be built on top of the architecture. This new layer can maintain the existence of the application logical network and provide a powerful discovery mechanism based on Distributed Hash Algorithms (DHT). The use of these techniques makes possible to improve the performance of the searching mechanisms by reducing the delay of a discovery query in a time proportional to the logarithm of the number of nodes, Olog(n). Moreover, the overhead weight can be reduced, and the availability of a found resource can be guaranteed. However, it supposes higher storage and update costs.

Moreover, it will be considered to separate the streaming tasks (Streaming Module) into a different computer by using a Service-Oriented Architecture (SOA) based scenario. This will enable to separate the heavy buffering and stream processing tasks into a powerful computer which will offer these services in order to construct peers with low requirements. Then, the streaming module will interactuate with the rest of the peer entity by using suitable SOA interfaces, such as Web Services.

VIII. Conclusions

In this paper, it was pointed out some topics of P2P media streaming and it was proposed CIMS-Live as the platform that offers solutions to them, specially focused on improving the collaboration among peers and enabling an interactive experience. This platform is a novel hybrid solution based on JXTA, DONET and ALM which allows delivering any kind of media stream. Besides, it provides a flexible and extensible platform according to new requirements and future value-added services.
It was implemented a Java based prototype and its functionalities were checked on a testbed.

REFERENCES

[1] D. Meddour, M. Mushtaq and T. Ahmed, “Open Issues in P2P Multimedia Streaming”, MULTICOMM’06, June 2006, Turkey.

[2] C. Yeh and L. S. Pui, “On the Frame Forwarding in Peer-to-Peer Multimedia streaming”, P2PMMS’05, November 2005, Singapore.

[3] PPLive software, Website: http://www.pplive.com/en/index.html

[4] SopCast software, Website: http://www.sopcast.org/

[5] Sun Microsystems, “JXTA v2.3.x Java Programmer’s Guide”, April 2005. http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

[6] X. Zhang, J. Liuy, B. Liz, and T. P. Yum, “CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming”. In Proc. IEEE INFOCOM, March 2005.

[7] T. Su-Wei and G. Waters, “Building Low Delay Application Layer Multicast Trees”. In Proc. EPSRC, Liverpool, June 2003.

[8] D. A. Tran, K. A. Hua and T. Do, “ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming”, In Proc. IEEE INFOCOM 2003.

[9] T. Tsuchiya, H. Yoshinaga, K. Koyanagi, A. Honda and A. Minami, “STARCast: Streaming Collaboration Platform using the Overlay Technology”, In Proc. SAINTW’06, IEEE Computer Society, January 2006.

[10] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, “AnySee: Peer-to-Peer Live Streaming”, In Proc. IEEE INFOCOM, April 2006.

[11] B. Eckel, “Java Thinking in Patterns”, May 2003. http://www.mindview.net/Seminars/ThinkingInPatterns.

[12] VideoLan Home Site, WebSite: http://www.videolan.org/

[13] Apache Commons Collections of Buffers, Website: http://jakarta.apache.org/commons/collections/apidocs/org/apache/commons/collections/buffer/package-summary.html

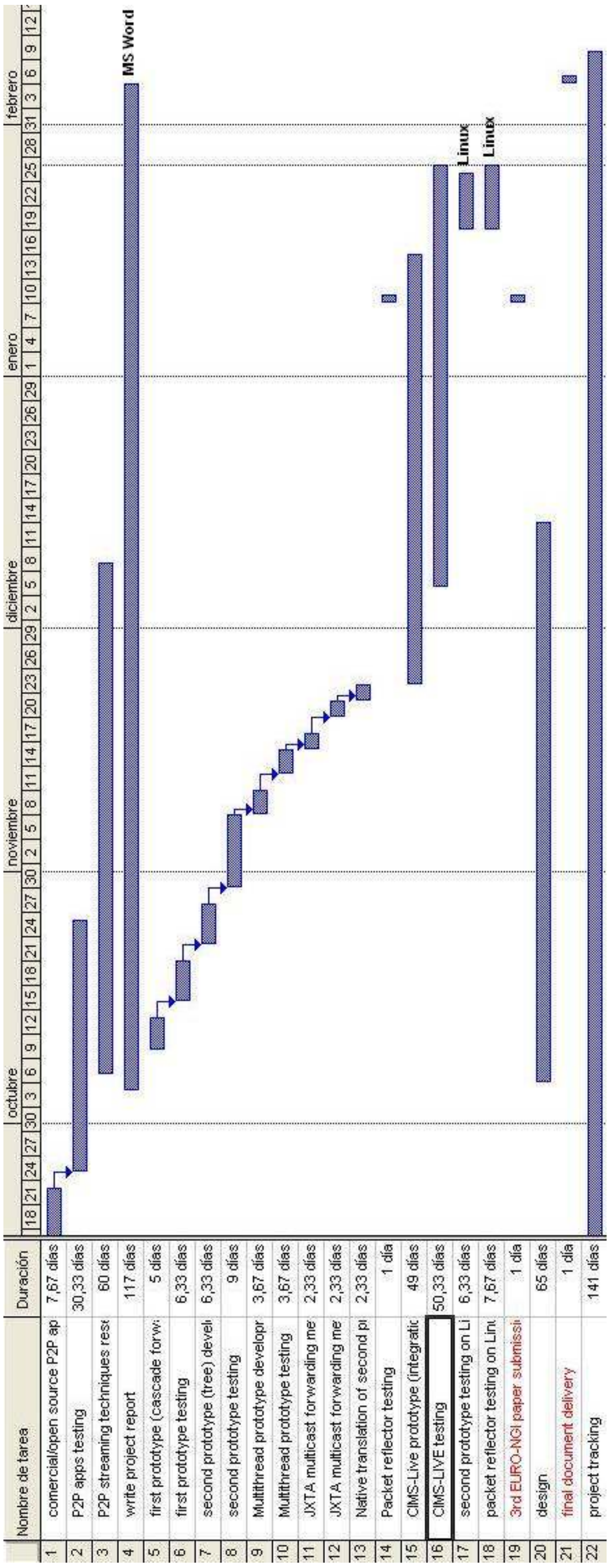
[14] jVlc Project, Website: https://trac.videolan.org/jvlc

[15] Java Swing Framework. Website: http://www.newt.com/java/swing.html

[16] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J-C. Hugly, E. Pouyoul and B. Yeager, “Project JXTA 2.0 Super-Peer Virtual Network”, May 2003. http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf.

[17] N. Magharei and R. Rejaie, “Understanding Meshbased Peer-to-Peer Streaming”, in Proc. International Workshop on Network and Operating Systems Support for Digital Audio and Video, Newport, Rhode Island, May 2006.

ANNEX D. GANTT DIAGRAM



	Nombre de tarea	Duración
1	comercial/open source P2P ap	7,67 días
2	P2P apps testing	30,33 días
3	P2P streaming techniques rese	60 días
4	write project report	117 días
5	first prototype (cascade forw	5 días
6	first prototype testing	6,33 días
7	second prototype (tree) devel	6,33 días
8	second prototype testing	9 días
9	Multithread prototype developr	3,67 días
10	Multithread prototype testing	3,67 días
11	JXTA multicast forwarding me	2,33 días
12	JXTA multicast forwarding me	2,33 días
13	Native translation of second pi	2,33 días
14	Packet reflector testing	1 día
15	CIMS-Live prototype (integratic	49 días
16	CIMS-LIVE testing	50,33 días
17	second prototype testing on Li	6,33 días
18	packet reflector testing on Linu	7,67 días
19	3rd EURO-NGI paper submissi	1 día
20	design	65 días
21	final document delivery	1 día
22	project tracking	141 días

Fig. D.1 Gantt Diagram

ANNEX E. CIMS-Live QUICK GUIDE

This quick guide is a little assistance for CIMS-Live installation and use.

E.1. Installation

There are two ways of installing CIMS-Live.

1. From the provided ANT file.
2. Installing the application using the provided Installer (Windows only).

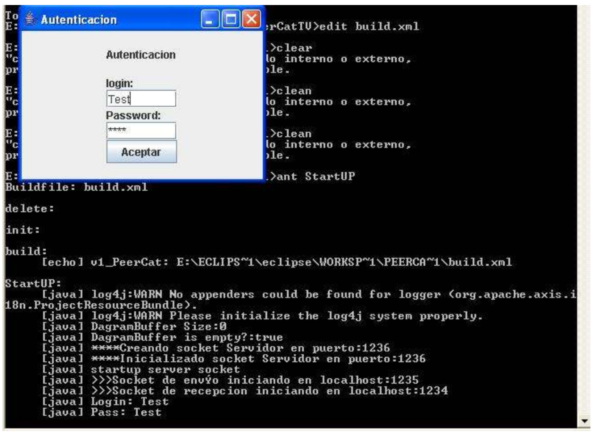
E.1.1. ANT File

It is provided a *build.xml* ANT file which allows the following tasks.

- init: creates the binaries output folder
- clean: deletes the content of the binaries output folder
- build: compiles the CIMS-Live source code
- delete: deletes the JXTA cache folder
- StartUP: runs CIMS-Live. This task also makes delete and build

In order to install and run CIMS-Live just type the following command in CIMS-Live home folder that contains the build.xml file (Fig. E.1).

```
# ant StartUP
```



```
To
E:                         rCatTV>edit build.xml

E:                         .>clear
"c                         lo interno o externo,
pr                         ble.

E:                         .>clean
"c                         lo interno o externo,
pr                         ble.

E:                         .>clean
"c                         lo interno o externo,
pr                         ble.

E:                         .>ant StartUP
Buildfile: build.xml

delete:

init:

build:
     [echo] v1_PeerCat: E:\ECLIPS~1\eclipse\WORKSP~1\PEERCA~1\build.xml

StartUP:
     [java] log4j:WARN No appenders could be found for logger (org.apache.axis.i
18n.ProjectResourceBundle).
     [java] log4j:WARN Please initialize the log4j system properly.
     [java] DagramBuffer Size:0
     [java] DagramBuffer is empty?:true
     [java] ****Creando socket Servidor en puerto:1236
     [java] ****Inicializado socket Servidor en puerto:1236
     [java] startup server socket
     [java] >>>Socket de envÿo iniciando en localhost:1235
     [java] >>>Socket de recepcion iniciando en localhost:1234
     [java] Login: Test
     [java] Pass: Test
```

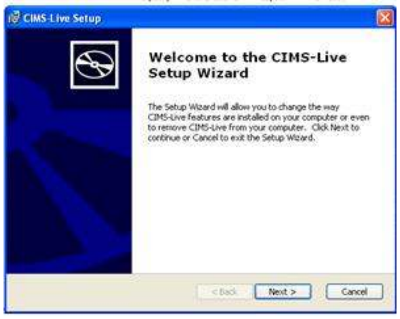
Fig. E.1 Actions with users

E.1.2. Windows Installer

It is provided a Windows installer [37] called *Cimslive.msi* (Fig. E.2). It consists of an executable file that creates a directory tree in "\Archivos de programa\ArGoN\CIMS-Live".

Fig. E.2 Installer icon

In order to install CIMS-Live, just make double-click on the installer icon and an Installation Wizard will appear in order to guide the setup process (Fig. E.3). Follow the steps indicated by the wizard to complete the installation.



Welcome to the CIMS-Live Setup Wizard

The Setup Wizard will allow you to change the way CIMS-Live features are installed on your computer or even to remove CIMS-Live from your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Fig. E.3 Installation Wizard

The installer creates a shortcut in the Start menu (Fig. E.4) and another one in the Desktop.

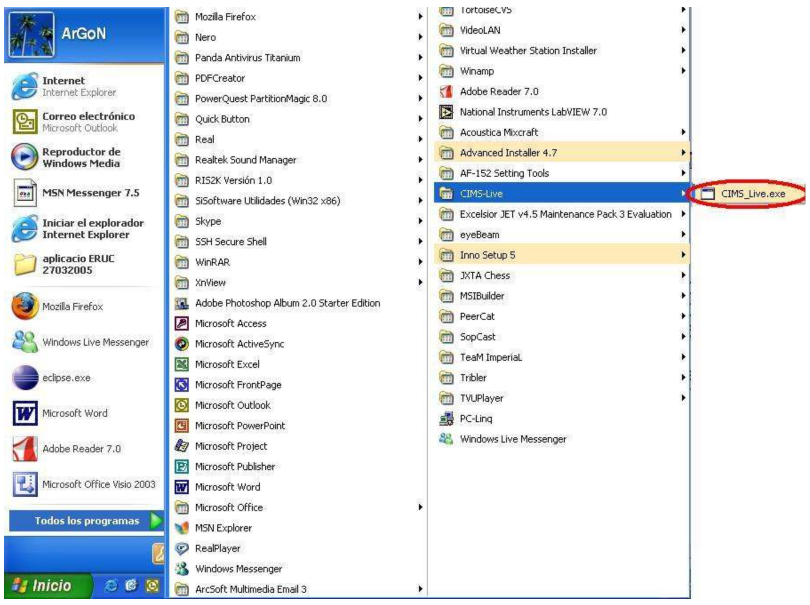
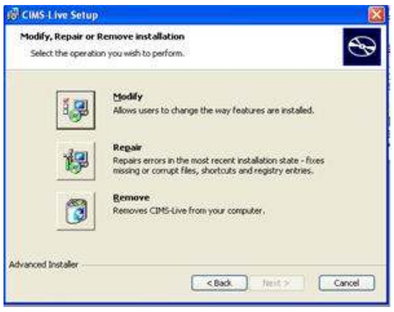


Fig. E.4 Start menu shortcut

This installer also edits the registry of Windows. If the user wants to remove CIMS-Live from the system (uninstall), it must be done through Control Panel. CIMS-Live can be also removed from the installer (Fig. E.5).


CIMS-Live Setup

Modify, Repair or Remove installation

Select the operation you wish to perform.

Modify
Allows users to change the way features are installed.

Repair
Repairs errors in the most recent installation state - fixes missing or corrupt files, shortcuts and registry entries.

Remove
Removes CIMS-Live from your computer.

Advanced Installer

Fig. E.5 Remove

E.2. Requirements

For the correct operation of the application it is necessary to install the following software:

- Java JDK 5.0
- Apache ANT 1.6 or upper
- VLC (VideoLAN Client)
- JMF (Java Media Framework)

It is also necessary to own a WebCam, correctly configured according to JM Studio in order to enable the videoconference functionality.

- Supported Operating Systems: Windows and Linux.

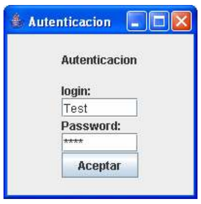
E.3. Use guide

E.3.1. Start-up

In order to run the application it can be used the shortcut created in the start menu. Otherwise, in the root of the directory it is also included a “build.xml” file for its use with ANT. Thus it is possible to run the application in any platform.

When the application starts running, it shows a window of authentication (Fig. E.6). In order to access to the application it is required to introduce the login and password of the current user. Next it is listed the default access data.

- *login: Alberto*
- *Password: Gonzalez*


Autenticacion

Autenticacion

login: Test

Password: ****

Fig. E.6 Authentication window

Note that both fields are case sensitive, be careful with capital letters.

E.3.2. Stop the application

To correctly stop the application it is necessary to use the menu: “Application” → “Exit” (Fig. E.7).

When stopping the application this way, the statistics generated by the application at run time can be saved in *“\Archivos de programa\ArGoN\CIMS-Live\var\monitor”.* This is a CSV file that can be opened with an editor of spreadsheets (like MS Excel).

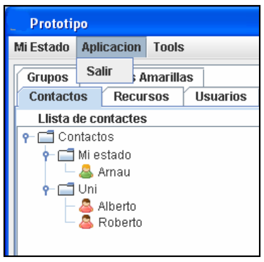


Fig. E.7 Exit

E.3.3. User registration

To start the CIMS-Live application it is necessary to be authenticated.

In the current prototype this authentication is made local. In the root of the directory tree, it can be found a file called “*users.xml*”. This file stores the name and password of the user.

To add a new user it has to be added a new entry in the xml file with the following format.

```
<user>
<login>[UserName]</login>
<pass>[Password]/pass>
</user>
```