1. Introducció

1.1 Motivació

Internet és un mitjà de comunicació molt nou i que, els últims anys, ha experimentat un creixement i un nivell d'implantació en les nostres vides que ningú s'esperava. Se'ns fa molt difícil avui en dia prescindir d'alguns dels serveis que la Xarxa ens ha proporcionat, com el correu electrònic i la comunicació instantània, que dia a dia li guanya terreny al correu tradicional.

Són molts els àmbits en els quals aquesta nova tecnologia ha suposat un canvi total de la manera de treballar, d'entendre un negoci o inclús de concebre l'oci. Des de les empreses que avui en dia s'han obert a mercats insospitats per ells anys enrere gracies a portals que els i permeten vendre els seus productes arreu del mon, fins a entitats bancàries que donen una sèrie de serveis als seus clients, que sense moure's de casa seva poden realitzar la majoria de les transaccions que abans tenien que anar a un banc o caixer expressament a fer.

Si alguna queixa es pot tenir d'aquestes noves tecnologies i aplicacions que s'estan integrant en el nostre dia a dia, és el fet de que no estan adaptades als requisits de tota la població. Així doncs, hi ha una massa de població que te unes necessitats especials, i que per tant, requereixen que tot aquest software s'adapti a les seves possibilitats i a la seva manera de treballar, per tal de que també en puguin fer ús.

Qui més qui menys en l'actualitat tothom ha fet servir algun cop o a diari el programa Microsoft Messenger, programa de missatgeria instantània que va prenen més força cada dia que passa. En aquest projecte s'ha volgut aprofundir en el coneixement d'aquest programa a nivell intern. Per tal de entendre el seu funcionament, desenvolupar-ne una versió més senzilla i amb menys funcionalitats, però a l'hora igual de robusta i a nivell bàsic amb un funcionament similar.

Així doncs el que s'ha volgut fer és assentar l'esquelet d'un programa de missatgeria instantània robust, que en una següent fase de desenvolupament, pugui

esdevenir una aplicació adaptada a aquest col·lectiu de gent. Que els i permeti comunicar-se entre ells amb aquest programa utilitzant les seves capacitats comunicatives especials. Això implica el fet que el programa funcioni en llenguatge SPC, enlloc de llenguatge escrit natural, un llenguatge de símbols que els hi permet la comunicació.

Per fer això ha sigut necessari integrar els coneixements tant del món de la informàtica com del de les telecomunicacions. Ha sigut imprescindible tenir la capacitat de programar utilitzant les eines que ofereixen les interfícies java.net i java.NIO, a l'hora que s'han tingut que fer servir tots els coneixements del món de les telecomunicacions en quant a connexions de xarxa mitjançant sockets. Es pot considerar que aquest projecte ha sigut la integració final de tots els coneixements adquirits en la realització de la doble titulació (Enginyeria Informàtica i Enginyeria Telemàtica) .

1.2 Introducció als sockets

Al igual que el protocol IP permet la comunicació entre ordenadors connectats a Internet, els ports permeten la comunicació entre les aplicacions que s'executen en els ordenadors (tota aplicació proporciona un o més serveis). En essència un port és la direcció de 16 bits associada, normalment, a una aplicació o servei. Una interfície d'ordenador de xarxa està dividida en 65536 ports (els primers 1024 solen estar reservats per serveis estàndards i processos del sistema operatiu). La divisió no és física, aquests ports no tenen res a veure amb els ports físics del hardware.

En una arquitectura client-servidor, el programa servidor es mante escoltant les peticions d tots els clients a través dels ports que te actius. La informació o els serveis sol·licitats son enviats pel servidor als clients, en concret als ports que aquests estiguin utilitzant per realitzar les peticions.

En una comunicació a través d'Internet, la direcció IP identifica de manera única l'ordinador al que es dirigeix el missatge i el número de port identifica a quin port o aplicació s'estan enviant les dades.

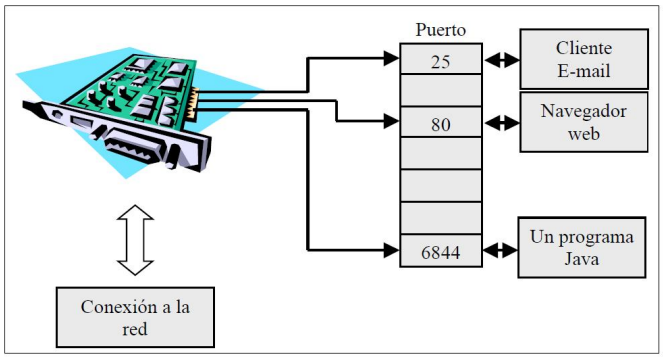

Figura 1. Esquema ports

En la família de protocols TCP/IP, existeixen dos protocols de transport (TCP i UDP) que s'encarreguen d'enviar dades d'un port a un altre per fer possible la comunicació entre aplicacions.

El protocol TCP (Transmission Control Protocol) és un servei de transport orientat a la connexió. Durant l'existencia de la comunicació hi ha un enllaç entre l'emissor i el receptor, enllaç que permet l'intercanvi bidireccional d'informació. El protocol inclou sistemes de detecció i correcció d'errors i garanteix que els paquets arriben en ordre al destí.

El protocol UDP (User Datagram Protocol) és un servei de transport sense connexió; el missatge de l'emissor es descompon en datagrames UDP i cada datagrama s'envia al receptor pel camí més oportú en aquell moment. Els datagrames poden arribar en qualsevol ordre o no arribar.

Tant TCP com UDP utilitzen sockets (literalment traduït com a endolls) per comunicar programes entre si en una arquitectura client-servidor. Un socket és un punt terminal o extrem en l'enllaç de comunicació entre dos aplicacions (que normalment s'executen en ordinadors diferents). Les aplicacions es comuniquen mitjançant l'enviament i recepció de missatges mitjançant els sockets. Un socket TCP es podria comparar a un telèfon, anàlogament un socket UDP seria la bústia de correus.

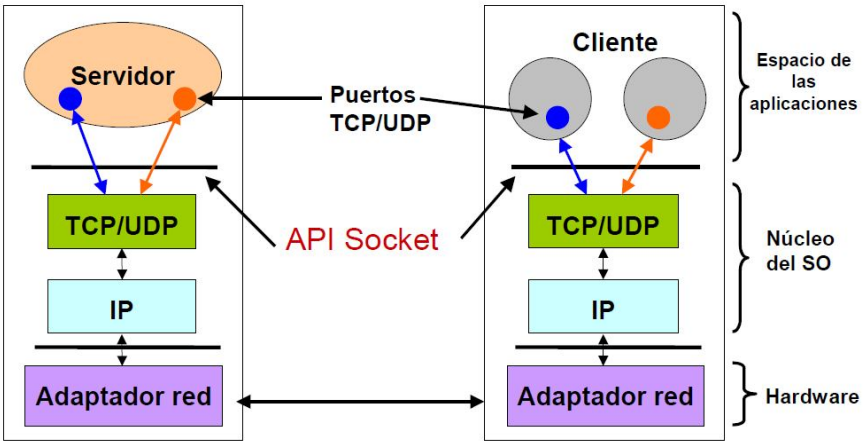

Figura 2. Comunicació client-servidor amb sockets

A part de classificar-se en sockets TCP i UDP, els sockets poden pertànyer a algun d'aquests dos grups:

- Sockets actius: Poden enviar i rebre dades a través d'una connexió oberta.
- Socksts passius: Esperen intents de connexió. Quan arriba una connexió entrant, li assignen un socket actiu. No serveixen per enviar o rebre dades.

Els socket son creats pel sistema operatiu i ofereixen una interfície de programació d'aplicacions (API) mitjançant la qual les aplicacions poden enviar missatges a altres programes, ja sigui en local o remotament. Les operacions dels sockets (enviar, rebre, etc.) s'implementen com a crides al sistema operatiu en tots els sistemes operatius actuals. Dit d'una altre manera; els sockets formen part del nucli del sistema operatiu. En els llenguatges orientats a objectes com el Java o el C#, les classes de sockets s'implementen sobre les funcions que dona l'API del sistema operatiu per a l'ús de sockets.

Per fer possible la comunicació client-servidor, el client ha d'establir sockets i connectar-los a sockets del servidor. Els passos que es segueixen en una comunicació típica amb sockets TCP són aquests:

- Es creen els sockets en el client i el servidor.
- El servidor estableix el port pel qual proporcionarà el servei.

- El servidor es mantén escoltant les possibles peticions dels clients pel port predefinit en el pas anterior.
- Un client connecta amb el servidor.
- El servidor accepta la connexió.
- Es realitza l'intercanvi de dades.
- El client o el servidor, o ambdós, tanquen la connexió.

Alguns comentaris al respecte d'aquest cinc primers passos. En primer lloc, el servidor ha d'estar en execució abans de que els clients intentin connectar-se a ell, i ha de mantenir actiu un socket passiu en tot moment que escolti qualsevol possible petició entrant.

En segon lloc, el client ha de crear un socket actiu en el qual s'especifiquin la direcció IP i el número de port de l'aplicació que proporciona el servei desitjat. És mitjançant aquest socket que el client comença la comunicació TCP amb el servidor.

En tercer lloc, l'intent de connexió TCP desencadena el fet que en el servidor es crea un socket actiu dedicat única i exclusivament a aquest client (durant el temps de vida d'aquest socket, no podrà ser assignat a cap altre client).

Finalment s'estableix la connexió TCP entre el socket del client i el socket actiu del servidor. A partir d'aleshores, un i altre poden enviar dades o rebre-les a través de la connexió establerta.

2. Objectiu

2.1 Objectiu general

Amb l'excusa de desenvolupar una aplicació de missatgeria instantània, aprofundir en els coneixements de les comunicacions de xarxa a nivell de sockets i buffers. Conèixer i aprendre a treballar amb les API que ens ofereix el Java dedicades a aquet propòsit, java.net i java.NIO. Conèixer de primera ma les avantatges i inconvenients d'aquestes interfícies.

2.2 Objectius específics

- Augmentar els coneixements sobre les comunicacions de xarxa al nivell més elemental, utilitzant els sockets.
- Conèixer la interfícies del Java dedicades a la programació de xarxa, java.net.
- Aprendre a fer servir la interfícies del Java java.NIO, dedicada a treballar amb buffers.
- Desenvolupar la capacitat d'aprendre noves metodologies de programació auto didàcticament.
- Experimentar en primera persona el desenvolupament d'una aplicació del tipus client-servidor.
- Desenvolupar una aplicació robusta i eficient.
- Aconseguir una aplicació escalable, que permeti ser ampliada en un futur.

3. Tecnologia i metodologia

En aquest apartar es parlarà de perquè s'ha escollit el Java com a eina per a desenvolupar l'aplicació i de les interfícies del Java que s'han utilitzat per a la realització d'aquest projecte, que com s'ha esmentat anteriorment, son la java.net i la java.NIO.

Per al programador de Java, un socket és la representació d'una connexió per a la transmissió d'informació entre dos aplicacions, ja s'executin en un mateix ordenador o en varis. Aquest abstracció d'alt nivell permet despreocupar-se dels detalls que hi ha a sota (corresponents als protocols subjacents). Per implementar els sockets TCP, el paquet java.net de Java proporciona dos classes: ServerSocket i Socket. La primera representa un socket passiu que espera connexions entrants dels clients i que s'ubica en el costat del servidor. La segona representa un socket actiu que pot enviar i rebre dades (pot trobar-se tant en el servidor com en el client).

En resum, un socket permet connectar-se a un equip a través d'un port, enviar o rebre dades i tancar la connexió establerta. Tots els detalls dels protocols de red, de la naturalesa física de les xarxes de telecomunicacions i dels sistemes operatius involucrats s'oculten als programadors. Fen així molt més senzilla la tasca de programar aplicacions de xarxa.

3.1 Perquè Java?

En aquest apartat es volen enumerar les avantatges que han portat a l'elecció del Java com a eina per al desenvolupament de l'aplicació de xarxa, i que fan que s'hagi considerat la millor opció per aquest projecte. Es farà alguna breu comparació amb altres llenguatges d'ús freqüent.

1- **Java és un llenguatge orientat a objectes.** Altres llenguatges com C++ també estan orientats a objectes, però són molt mes impurs, ja que permeten dades definides pel usuari que no són objectes, així com la barreja de codi

orientat a objectes amb codi modular o no modular. C# està en la mateixa línea que el Java en quant a orientació a objectes, però inclou (per eficàcia) estructures de dades que no són objectes.

2- **Independència de plataforma.** Els *bytecodes* produïts al compilar un programa Java poden executar-se en qualsevol plataforma amb una màquina virtual de Java. Degut a que les xarxes contenen ordenadors de molts fabricants, la independència respecte a la plataforma constitueix una característica molt valuosa. Els llenguatges inclosos en la plataforma .NET de Microsoft també es compilen a un codi entremig independent de la plataforma, però actualment .NET no s'ha propagat a tantes plataformes com el Java.

3- **Descarga dinàmica de classes.** Java permet descarregar els *bytecodes* d'una classe mitjançant una connexió de xarxa, crear una instància de la classe i incorporar el nou objecte a l'aplicació en execució. En conseqüència, les aplicacions en xarxa poden distribuir-se de forma molt més senzilla (mitjançant servidors web on es guarden *bytecodes*, navegadors i *applets*) i son capaços d'incorporar objectes desconeguts en el temps de compilació. C++ no ofereix unes capacitats dinàmiques com les de Java.

4- **Inclusió estàndard de vàries API per a la programació de xarxa.** El paquet java.net ofereix una API d'alt nivell per treballar amb el protocol HTTP, amb les URL i els sockets. La plataforma .NET també en proporciona una, sospitosament similar a la del Java. Per exemple, qualsevol programador de Java que utilitzi .NET enseguida s'adonarà de la similitud entre les classes System.Net.Sockets.TCPClient i java.net.Socket. C++ també pot treballar amb sockets però exigeix un major esforç al programador, la lectura del codi es més complicada i les API tradicionals no estan tant orientades a objectes com les de Java.

5- **Java es manté al dia en quant a tendències en tecnologies distribuïdes.** Encara que Microsoft va apostar per les tecnologies de serveis web abans que Sun, avui en dia Java disposa d'una sèrie d'eines excel·lents per treballar amb aquestes noves tecnologies.

6- **Comunitat d'usuaris.** Java disposa d'una comunitat molt àmplia i internacional d'usuaris preocupats per difondre el llenguatge i per perfeccionar-lo.

7- **Java s'ensenya a moltes universitats.** És el cas de la meva, per tant el Java és el llenguatge amb el que més he treballat i amb el que mes còmode em sento desenvolupant.

8- **Disponibilitat d'eines gratuïtes i de codi obert.** Java disposa d'eines de desenvolupament gratuïtes d'excel·lent qualitat, com és el cas del NetBeans, entorn de desenvolupament integral que s'ha utilitzat per a l'elaboració d'aquest projecte. En el cas de .NET, l'eina de desenvolupament oficial és Visual Studio .Net; una molt bona solució i completament integrat amb Windows i SQL Server. Té en contra el seu preu, uns 2500 dollars actualment i que només es pot utilitzar en entorns de treball Windows.

3.2 El paquet java.net

Gran part de la popularitat del Java es deu a la seva orientació a Internet. Cal afegir que aquesta acceptació resulta merescuda, Java proporciona una interfície de sockets orientada a objectes que simplifica moltíssim el treball en xarxa.

Amb aquest llenguatge, comunicar-se amb altres aplicacions a través d'Internet és molt similar a obtenir l'entrada del usuari a través de la consola o de llegir arxius, en contrast amb el que succeeix amb llenguatges com el C. Cronològicament, Java va ser el primer llenguatge de programació on la manipulació de l'entrada i sortida de dades a través de la xarxa es realitzava igual que la E/S amb arxius.

El paquet java.net, que proporciona una interfície orientada a objectes per crear i manipular sockets, connexions HTTP, localitzadors URL, etc., esta formada per classes que es poden dividir en dos grans grups:

- Classes que corresponen a les API (interfícies de programació d'aplicacions) dels sockets: **Socket**, **ServerSocket**, **DatagramSocket**, etc.

- Classes corresponents a eines per treballar amb URL: **URL**, **URLConnection**, **HttpURLConnection**, **URLEncoder**, etc.

El paquet java.net permet treballar amb els protocols TCP i UDP. La classe **java.net.Socket** permet crear sockets TCP per al client; la classe **java.net.ServerSocket** fa el mateix per al servidor. Per les comunicacions UDP, Java ofereix la classe **java.net.DatagramSocket** per als dos costats de la comunicació UDP, i la classe **java.net.DatagramPacket** per crear datagrames UDP.

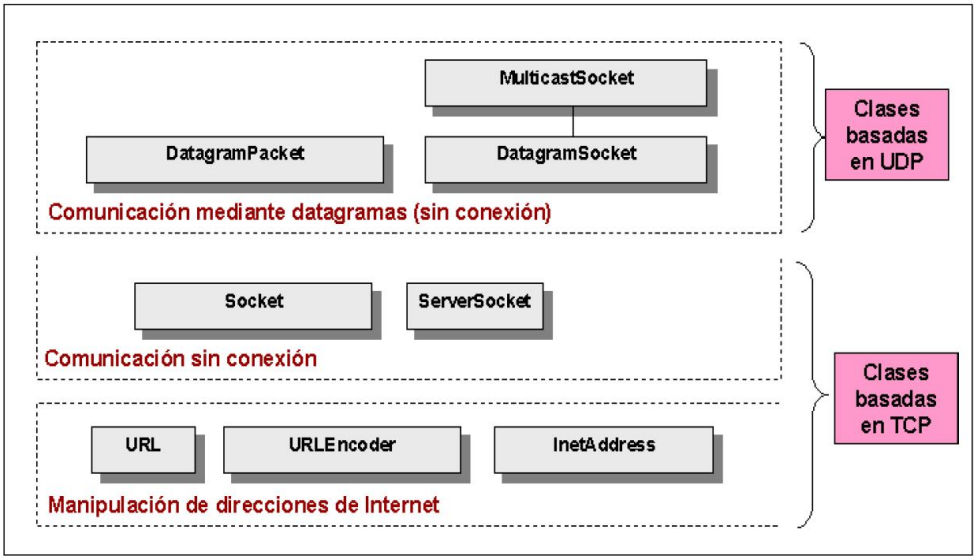

Figura 3. Esquema de les classes basades en TCP i UDP

3.2.1 La classe java.net.ServerSocket

Aquesta classe es la que s'utilitza pel servidor per rebre connexions dels clients mitjançant el protocol TCP. Tota aplicació que actuí com a servidor crearà una instància d'aquesta classe i cridarà al seu mètode *accept();* la crida farà que l'aplicació es

bloquegi (que estigui en espera) fins que arribi una connexió per part d'algun client. Quan això succeeixi, el mètode accept() crearà na instancia de la classe java.net.Socket que s'utilitzarà per comunicar-se amb el client. Res no impedeix que un servidor tingui un únic objecte ServerSocket i molts objectes Socket associats (cada un a un client en concret).

Resulta important ressaltar que la cua de connexions entrants és del tipus FIFO (First in, First Out: primer en entrar, primer en sortir) i que la seva longitud màxima ve determinada pel sistema operatiu. En el cas de que la cua estigui plena, l'objecte ServerSocket rebutjarà noves connexions fins que es buidin posicions de la cua.

A continuació es presenta una taula resum on es poden veure alguns dels mètodes més utilitzats de la classe ServerSocket:

Mètode	Descripció
accept()	Escolta connexions a aquest socket de servidor i les accepta.
getInetAddress()	Retorna la direcció IP local del socket de servidor.
getLocalPort()	Retorna el port pel qual escolta el socket servidor.
close()	Tanca el socket servidor.

Taula 1. Mètodes classe ServerSocket

En el cas de que un objecte ServerSocket intenti escoltar peticions dels clients per un port ocupat per un altre ServerSocket, es produirà una excepció **java.net.BindException**.

La implementació de qualsevol programa servidor en Java segueix la següent seqüència de passos:

1- Es crea un objecte ServerSocket per escoltar les peticions que arriben al port associat al servei.
2- Quan es crida al mètode *accept()*, el socket de servidor es queda a l'espera de peticions de clients pel port.
3- Al arribar una sol·licitud es produeixen els següents tres passos:

3.1- S'accepta la connexió, el qual genera un objecte Socket associat al client.

3.2- S'associen objectes de les classes contingudes en el paquet java.NIO als fluxes d'entrada i sortida (buffers) del socket.

3.3- Es llegeix i s'escriu en els buffer, es a dir es processant els missatges entrants i sortints.

4- Es tanquen els buffers.

5- Es tanca el socket vinculat al client.

6- el socket de servidor continua a l'espera de noves peticions.

3.2.2 La classe java.net.Socket

Aquesta classe implementa sockets TCP actius (capaços de rebre i transmetre dades). El constructor ***public Socket (String host, int port) throws UnknownHostException, IOException*** és un dels mes utilitzats. Aquest mètode crea un objecte Socket i el connecta amb el port amb el número de *port* del ordenador de nom *host*.

Els mètodes més utilitzats d'aquesta classe es resumeixen en la següent taula:

Mètode	Descripció
getInetAddress()	Retorna la dirrecció InetAddress a la qual està connectat el socket.
getPort()	Retorna el número de port al que està connectat el socket.
getLocalAddress()	Retorna la direcció local a la qual està connectat el socket.
getLocalPort()	Retorna el número de port local al qual està connectat el socket.
getInputStream()	Retorna u flux d'entrada per al socket.
getOutputStream()	Retorna un flux de sortida per al socket.
close()	Tanca el socket.

Taula 2. Mètodes classe Socket

La implementació de qualsevol programa client ha de seguir aquet passos:

1- Es crea un objecte Socket, que te associat un node destí i un port on s'executa el servei sol·licitat.

2- S'associen objectes de les classes contingudes en el paquet java.NIO als fluxes (buffers) d'entrada i de sortida del socket.

3- Es llegeix o s'escriu en els buffers.
4- Es tanquen els buffers.
5- Es tanquen els sockets.

El tancament dels sockets no s'ha de passar per alt, per la seva naturalesa bidireccional consumeixen molts recursos, tant de la màquina virtual Java com del sistema operatiu on s'estiguin executant. Els sockets es tanquen quan:

- Finalitza el programa que els ha creat.
- Es crida al seu mètode *close()*.
- Es tanca un dels buffers associats.
- S'elimina pel recol·lector d'escombraries de Java.

3.3 El paquet java.NIO

La versió 1.4 de Java va incorporar noves característiques al llenguatge, mantingudes en la posterior versió 5.0. Una de les més importants és el paquet java.nio. Aquest paquet ofereix una nova API d'entrada i sortida (E/S), coneguda com NIO (New Input/Output). La nova API pot dividir-se en tres grans paquets:

- **java.nio** defineix buffers, que s'utilitzen per emmagatzemar seqüències de bytes o d'altres valors primitius (int, float, char, double, etc.).

- **java.nio.channels** defineix canals, això és, abstraccions mitjançant les quals poden transferir-se dades entro els buffers i les fonts o els consumidors de dades (un socket, un fitxer, un dispositiu de hardware). A més, proporciona classes que permetin E/S sense bloqueig.

- **java.nio.charset** conté classes que converteixen de manera molt eficaç buffers de bytes en bytes de caràcters i permeten l'ús d'expressions regulars.

NIO proporciona grans avantatges sobre l'E/S estàndard de java (java.io). El problema de la ineficàcia de la E/S en Java ha estat present desde les primeres versions.

Resultava contradictori trobar-se davant d'un llenguatge amb moltes característiques enfocades a Internet i, al mateix temps, amb un rendiment tant pobre en quant a la transferència massiva de dades.

A continuació s'exposen les principals característiques de NIO, en contraposició amb la E/S estàndard de Java:

- Incorpora buffers (contenidors de dades), canals (encarregats de la transferència de dades entre els buffers i les peticions de E/S) i selectors (encarregats de proporcionar el estat dels canals: si estan preparats per llegir, per escriure, rebre connexions).

- Permet la transferència eficaç de grans quantitats de dades. **Java.io** utilitza fluxes de bytes; **java.nio**, en canvi, usa blocs de dades.

- Els sockets de NIO permeten treballar sense bloqueig (també permeten treballar amb bloqueig si és necessari). Sense bloqueig, un sol fil pot encarregar-se de controlar tants canals com sigui necessari. No es necessita destinar un fil a cada canal, lo qual augmenta el rendiment de la E/S. Així mateix al no necessitar-se un fil per a cada socket, la complexitat del codi es redueix notablement.

- Aprofita les prestacions del sistema operatiu on s'executa la MVJ.

3.3.1 La classe java.nio.Buffer

La classe java.nio.Buffer és una classe abstracta que te, de moment, set implementacions (una per a cada tipus de dada). Cada una d'aquestes classes és un contenidor de dades primitives (*byte, char, int*, etc.) amb una capacitat limitada. En cada una, les dades s'emmagatzemen de manera lineal i seqüencial. En comparació amb les col·leccions de Java, els buffers eviten la sobrecàrrega de la ubicació dels objectes a la pila i de la recol·lecció de brossa.

En un buffer, la capacitat és el número d'elements que pot contenir, el límit l'índex del primer element que no pot llegir-se o escriure's (perquè no hi ha res més en aquella posició ni en les següents). La posició i el límit són menors o iguals a la capacitat, i la posició no pot ser major que el límit.

3.3.2 La classe java.nio.channels.ServerSocketChannel

Un canal és una connexió entre un buffer i una font o un consumidor de dades (un socket, un arxiu, etc.). Les dades poden llegir-se dels canals mitjançant buffers, i les dades d'un buffer poden escriure's en els canals. Es a dir, els buffers poden escriure's en canals o llegir-se d'aquests.

Els canals poden operar amb bloqueig o sense. Una operació de E/S amb bloqueig no retorna fins que s'ha produït una d'aquestes situacions:

- Es completa l'operació.
- Es produeix una interrupció deguda al sistema operatiu.
- Es llença una excepció.

Una operació de E/S sense bloqueig retorna al instant, retornant algun valor de retorn que indica si l'operació s'ha realitzat correctament o no. Un programa o un fil que executi una operació sense bloqueig no es quedarà "dormit" esperant dades, una interrupció o una excepció; el seu curs d'execució continuarà.

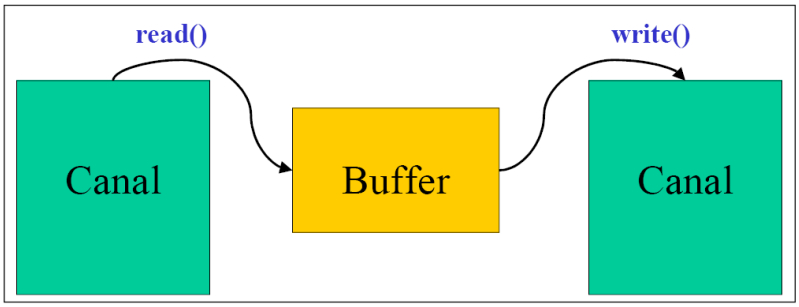

Figura 4. Esquema de canals i buffer

3.3.3 La classe java.nio.channels.SocketChannel

La classe java.nio.channels.SocketChannel és un canal seleccionable per a sockets TCP actius. Un objecte SocketChannel ve a ser un embolcall per a un objecte Socket, que permet associar-li un canal. Les operacions d'enllaçat amb una direcció IP i un port, de tancament etc., es realitzen mitjançant el socket associat.

Igual que passa amb els ServerSocketChannels, els SocketChannels també poden configurar-se sense bloqueig, de forma que les operacions de E/S no bloquegin el programa en execució.

En el cas de les crides al mètode *connect()* sempre existeix un curt bloqueig encara que haguem configurat amb no bloqueig. Aquest és degut a la naturalesa del protocol TCP; es necessita un cert temps perquè el client connecti amb el servidor i perquè aquest enviï una resposta que indiqui la seva disponibilitat per acceptar peticions.

3.3.4 La java.nio.channels.Selector i java.nio.channels.SelectionKey

La classe **java.nio.channel.Selector** és una de les principals de la API NIO. Un Objecte Selector controla una sèrie de canals i llença un avis quan un d'ells llença un succés de E/S. La classe *Selector* informa a l'aplicació de les operacions de E/S que succeeixen en els canals que aquesta mantén actius.

La informació sobre les operacions de E/S es registra en un conjunt de claus, que són instancies de la classe **java.nio.channel.SelectionKey**. Cada clau emmagatzema informació sobre el canal que desencadena l'operació i el tipus d'operació de la que es tracta (lectura, escriptura, connexió entrant, connexió acceptada).

Per saber quins successos de E/S dels que estem interessats es produeixen en un determinat canal és necessari registrar el canal amb el selector i especificar els tipus de successos en els que estem interessats. Els successos de E/S per els quals es pot registrar un canal mitjançant un selector s'especifiquen amb les següents constants enteres:

- **SelectionKey.OP_READ.** Registra el canal per successos de lectura.
- **SelecionKey.OP_WRITE**. El registra per successos d'escriptura.
- **SelectionKey.OP_ACCEPT**. El registra per les peticions entrants dels clients (utilitzat en els servidors).
- **SelectionKey.OP_CONNECT**. El registra per les connexions acceptades pels servidors (es fa servir al costat del client).

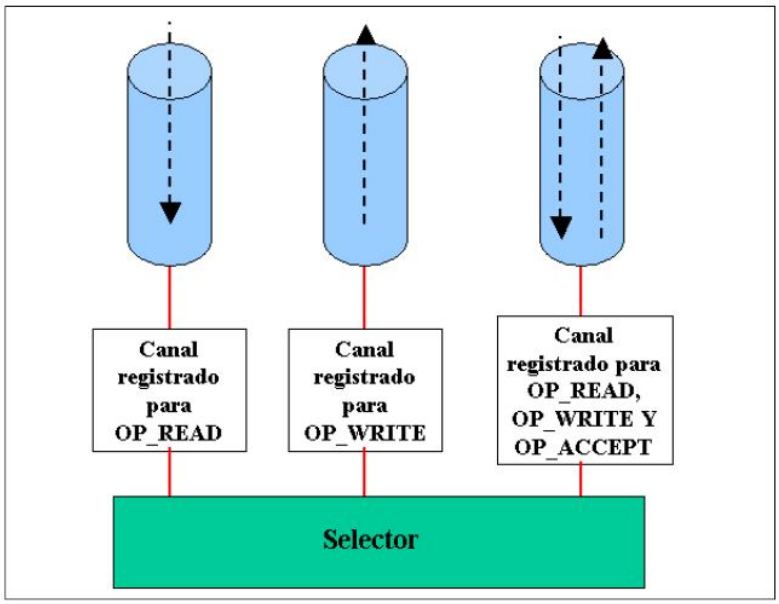

Figura 5. Esquema de canals i successos

4. Pressupost

4.1 Planificació

Tasques	Hores
Aprenentatge de java.net i java.nio	60 h
Especificació dels requeriments	5 h
Diagrama de casos d'ús	2 h
Especificació de casos d'ús	3 h
Anàlisi de les tecnologies i eines necessàries	3 h
Anàlisi de l'estructura de l'aplicació	20 h
Disseny pantallas inicials	3 h
Programació del servidor v1(Threads)	50 h
Programació del servidor v2 (java.NIO)	50 h
Programació del client	15 h
Comunicació client-servidor	20 h
Gestió usuaris online	15 h
Comunicació client-client	40 h
Interfície gràfica final	10 h
Testeig de l'aplicació	5 h
Documentació	35 h
TOTAL	**371 hores**

Taula 3. Planificació de tasques

4.2 Cost

Personal

Hores	Preu/hora
371 h	12 €/h
TOTAL	**4452 €**

Taula 4. Cost del personal

Software

Producte	Preu
Microsoft Windows XP Professional	139 €
Microsoft Office 2003	154 €
NetBeans	Gratuït
TOTAL	**293 €**

Taula 5. Cost del software

Hardware

Equip	Preu	Amortització (4 mesos)
Acer Aspire 9300	800 €	267 €

Taula 6. Cost del hardware

Total

Concepte	SubTotal
Cost del personal	4452 €
Cost del software	293 €
Cost del hardware	267 €
TOTAL	**5012 €**

Taula 7. Cost Total

4.3 Gràfic de costos

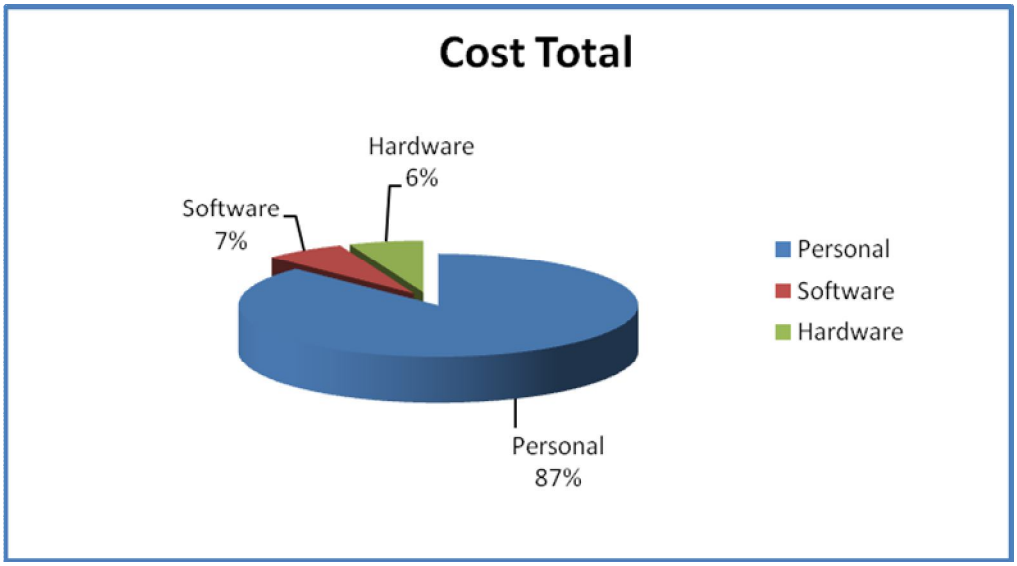

Figura 6. Gràfic de costos

5. Anàlisi

5.1 Estudi de mercat

Avui en dia son varis els programes que s'utilitzen com el que es coneix com a programes de missatgeria instantània. El propòsit d'aquests programes és, com el seu nom indica, permetre a dos usuaris que es trobin connectats a Internet i amb el programa encès comunicar-se via missatges de text i en temps real.

En un principi eren varis els programes que van sortir que permetien fer aquesta tasca, un dels primers va ser el ICQ, programa que sortia de l'evolució dels xats que tan de moda estaven en aquella època, i que venia a ser un xat en el que nomes podies parlar amb aquells contactes que prèviament havies afegit a la teva llista, de manera que atorgava al usuari un cert nivell de privacitat que no existia en els xats on qualsevol persona podia entrar.

Amb el temps alguns programes s'han anat consolidant i d'altres han desaparegut. Actualment podem parlar de tres grans referents en el món de la missatgeria instantània. Aquest són el Microsoft Messenger, el Skype i el Google Talk. La característica comú de tots ells és el fet de que son gratuïts i qualsevol persona s'els pot instal·lar i utilitzar fàcilment en el seu ordinador.

En quan al Microsoft Messenger, podríem dir que és el més utilitzat de tots, sobretot per una gran massa d'usuaris joves o inclús adolescents. Alguna de les característiques més importants que té, és el fet que incorpora un gestor de correu electrònic i cada cop dona més aplicacions. Actualment ens permet la videoconferència, les trucades de veu i la compartició de carpetes amb tot tipus d'arxius entre els usuaris. Així doncs també es caracteritza per un entorn molt visual i molt personalitzable.

El Skype es tracta d'un programa que inclou missatgeria instantània, però que està molt més enfocat a la trucada de veu a través d'Internet. És un dels programes que més telèfons IP porten incorporats. Les avantatges son clares, la trucada per Internet és

gratuïta front la que es pot fer amb qualsevol telèfon fix o mòbil. Això si, requereix un micròfon i que d'interlocutor estigui en el seu ordinador amb el programa engegat.

En quant al Google Talk destaca per la lleugeresa de l'aplicació. La seva gran aposta és el fet de disposar d'un client que s'executa directament desde la web sense requerir de la instal·lació de cap programa. Una interfície molt minimalista i lleugera que l'únic que pretén es fer la seva tasca de missatgeria instantània accessible desde qualsevol lloc sense requerir cap instal·lació.

Per acabar, comentar que en aquest projecte no s'ha volgut imitar ni superar cap d'aquestes aplicacions, doncs són aplicacions amb potents empreses al darrere amb un fort equip que les mantén constantment al dia i en prefecte funcionament. El que s'ha fet ha sigut agafar la idea de com treballen aquests aplicatius i intentar desenvolupar un programa amb menys funcionalitats però que fes be la seva feina, per en un futur permetre utilitzar aquest treball fet per adaptar l'aplicació a les necessitats específiques d'un grup d'usuaris.

5.2 Requeriments funcionals

- Un usuari ha de poder mantenir converses amb altres usuaris connectats.
- Un usuari ha de poder enviar símbols SPC a qualsevol altre usuari connectat.
- L'usuari ha de accedir al programa amb un alies o "nick".
- Un usuari ha de poder mantenir tantes converses simultànies amb diferents usuaris com desitgi.
- Un usuari ha de veure en tot moment quins usuaris estan connectats al programa.
- S'han de desenvolupar dos programes, un servidor encarregat de gestionar totes les connexions centralitzadament i un client que permeti la missatgeria instantània.
- El servidor ha de gestionar els diversos usuaris que hi ha connectats al programa.

5.3 Requeriments no funcionals

- L'aplicació ha d'estar desenvolupada en Java.
- El sistema ha de ser robust, escalable i estable.
- El programa ha de ser independent de la plataforma, suportat tant per Windows com per Linux.
- S'ha de seguir l'arquitectura client-servidor.
- La interfície gràfica ha de ser usable a l'hora que senzilla.
- El client s'ha de programar seguin el patró factoria / tres capes.

5.4 Diagrama de casos d'ús

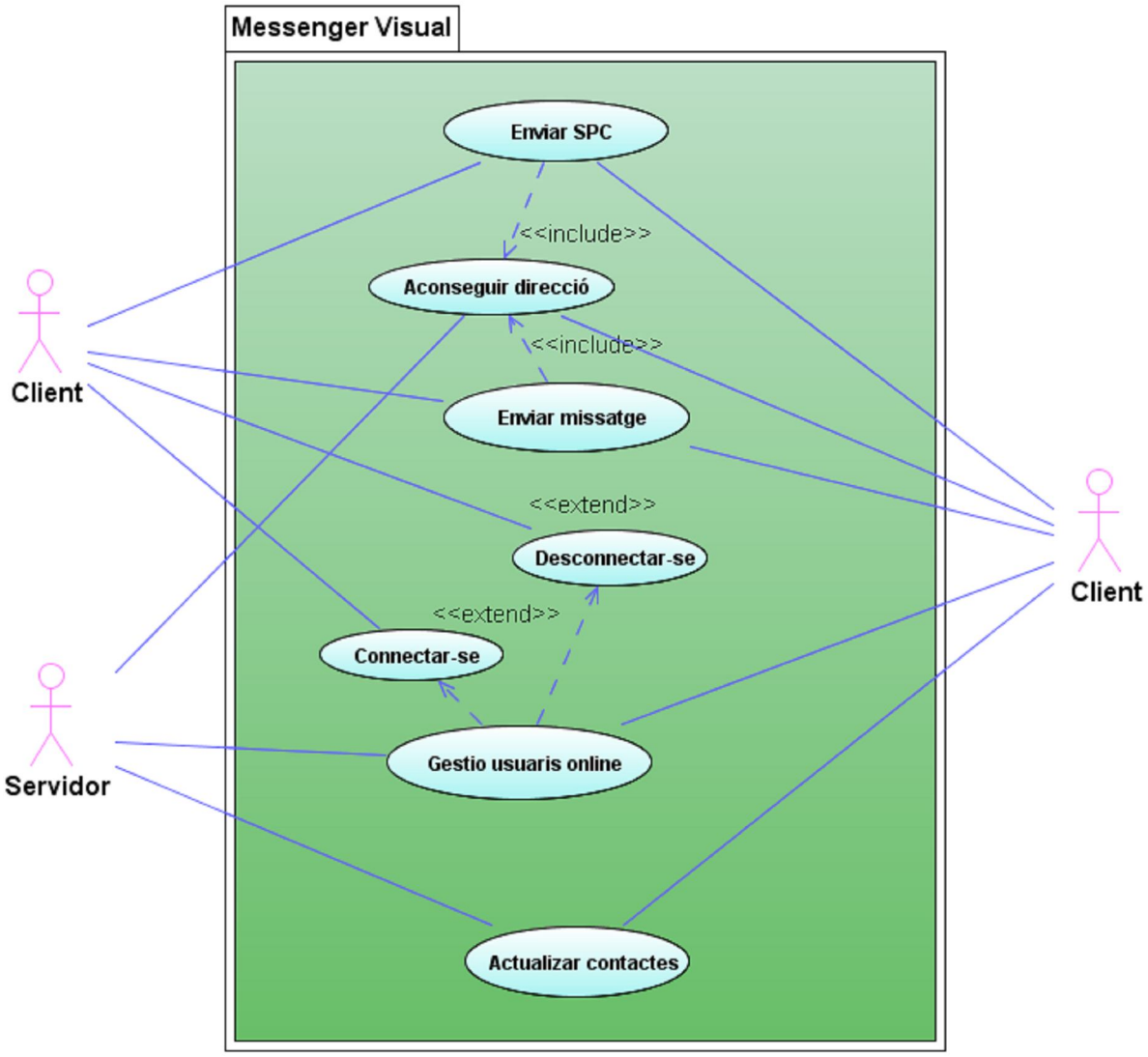

Figura 7. Diagrama de casos d'ús

- Enviar SPC: Un usuari connectat ha de poder enviar un símbol SPC a qualsevol altre usuari que estigui connectat al messenger visual.

- Enviar missatge: Els usuaris que estiguin utilitzant el messenger visual poden enviar missatges de text a altres usuaris que també es trobin connectats simultàniament.

- Aconseguir direcció: El servidor ha de poder subministrar la direcció de xarxa i el port amb el que un usuari vol mantenir una conversació, ja sigui via missatge de text o enviant un símbol SPC.

- Connectar-se: Els usuaris del programa quan l'obren es registren amb un nick i comencen a utilitzar-lo, de manera que queden registrats en el servidor com a usuaris actius.

- Desconnectar-se: Els usuaris un cop hagin acabat de fer servir l'aplicació la poden tancar, aquest esdeveniment farà que el servidor els doni de baixa i ja no apareguin com a usuaris actius.

- Gestió d'usuaris online: El servidor ha de mantenir en tot moment una llista actualitzada dels usuaris que estan connectats per tal de poder facilitar la comunicació entre ells en cas de que ho sol·licitin.

- Actualitzar contactes: El servidor s'ha de preocupar de tenir la llista de contactes actualitzada en tot moment i de que els clients en rebin les actualitzacions periòdicament.

5.5 Especificació de casos d'us

5.5.1 Connectar-se

Cas d'ús "Connectar-se"

Actor/s: Usuari, Servidor.

Pre-condicions

1. L'usuari està connectat a Internet.

Flux normal

1. L'usuari inicia el cas d'us connectar-se.
2. El sistema demana el nick amb el que es vol connectar.
3. L'usuari introdueix el nick i prem el botó acceptar.
4. El servidor verifica l'usuari, li dona d'alta la sessió i li envia la llista d'usuaris online.
5. El sistema obra la pantalla de contactes.
6. Finalitza el cas d'us connectar-se.

Flux alternatiu

3.a.1 L'usuari no introdueix cap nick i prem el botó acceptar.

3.a.2 El sistema retorna detecta l'error i retorna al punt 2.

4.a.1 El servidor detecta que el nick està en ús i retorna un error.

4.a.2 El sistema informa al usuari del error i retorna al punt 2.

Post-condicions

1. S'inicia la sessió del usuari registrat.

5.5.2 Enviar missatge

Cas d'ús "Enviar missatge"

Actor/s: Usuari.

Pre-condicions

1. L'usuari està connectat a Internet.
2. L'usuari està connectat al messenger visual.

Flux normal

7. L'usuari inicia el cas d'us enviar missatge fen click sobre el contacte al que vol enviar el text.
8. El sistema obre la pantalla de xat.
9. El sistema connecta amb el servidor i executa el cas d'us aconseguir direcció.
10. El sistema estableix la connexió amb el usuari amb el que vol parlar i envia el missatge de text.
11. El client amb el que es vol parlar rep el missatge i s'inicia la conversa.
12. Finalitza el cas d'us enviar missatge.

Flux alternatiu

9.a.1 El servidor no respon.

9.a.2 El sistema informa al usuari de que el servidor no esta operatiu.

10.a.1 El client amb el que es vol iniciar la conversa no respon.

10.a.2 El sistema informa al usuari del error i retorna a la pantalla de contactes.

5.5.3 Enviar SPC

Cas d'ús "Enviar SPC"

Actor/s: Usuari.

Pre-condicions

1. L'usuari està connectat a Internet.
2. L'usuari està connectat al messenger visual.

Flux normal

13. L'usuari inicia el cas d'us enviar SPC fen click sobre el contacte al que vol enviar el text.

14. El sistema obre la pantalla de xat.

15. El sistema connecta amb el servidor i executa el cas d'us aconseguir direcció.

16. El sistema estableix la connexió amb el usuari amb el que vol parlar i envia el missatge de text.

17. El client amb el que es vol parlar rep la imatge i s'inicia la conversa.

18. Finalitza el cas d'us enviar missatge.

Flux alternatiu

15.a.1 El servidor no respon.

15.a.2 El sistema informa al usuari de que el servidor no esta operatiu.

16.a.1 El client amb el que es vol iniciar la conversa no respon.

16.a.2 El sistema informa al usuari del error i retorna a la pantalla de contactes.

5.5.4 Aconseguir direcció

Cas d'ús "Aconseguir direcció"

Actor/s: Usuari, Servidor.

Pre-condicions

1. L'usuari està connectat a Internet.

2. L'usuari està connectat al messenger visual.

3. S'han iniciat els casos d'ús Enviar missatge o Enviar SPC.

Flux normal

19. El servidor rep una petició del usuari de direcció i port i inicia el cas d'ús aconseguir direcció.

20. El servidor actualitza la llista de contactes connectats.

21. El servidor busca l'usuari a la llista de contactes online i guarda la seva IP i port.

22. El servidor envia les dades sol·licitades al client que les ha sol·licitat.

23. Finalitza el cas d'us enviar missatge.

Flux alternatiu

21.a.1 El servidor no troba l'usuari que sol·licita el client.

21.a.2 El servidor informa al client de que ja no existeix el contacte que sol·licita.

21.a.3 El servidor executa el cas d'ús actualitzar contactes.

5.5.5 Desconnectar-se

Cas d'ús "Desconnectar-se"

Actor/s: Usuari, Servidor.

Pre-condicions

1. L'usuari està connectat a Internet.
2. L'usuari esta connectat al messenger visual.

Flux normal

24. L'usuari inicia el cas d'us desconnectar-se tancant l'aplicació.

25. El sistema envia un missatge de desconnexió al servidor informant que l'usuari s'ha desconnectat.

26. El servidor actualitza la llista d'usuaris connectats i inicia el cas d'ús actualitzar contactes.

Post-condicions

1. L'usuari ha quedat donat de baixa de la llista de usuaris connectats.

5.5.6 Gestió usuaris online

Cas d'ús "Gestió usuaris online"

Actor/s: Servidor.

Pre-condicions

1. El servidor esta en funcionament i hi ha usuaris online.

Flux normal

27. El servidor rep una petició del usuari resta a l'espera de qualsevol connexió o desconnexió.

28. El servidor rep una petició.

29. El servidor comprova si la petició es d'un usuari que es connecta o d'un usuari que es desconnecta.

30. El servidor actualitza la llista de contactes connectats.

31. El servidor inicia el cas d'ús actualitzar contactes.

32. Finalitza el cas d'ús gestió usuaris online.

Post-condicions

1. La llista d'usuaris queda actualitzada i tots els clients en son informats.

5.5.7 Actualitzar contactes

Cas d'ús "Actualitzar contactes"

Actor/s: Servidor, Usuaris.

Pre-condicions

1. El servidor esta en funcionament i hi ha usuaris online.

2. Algun procés ha disparat el cas d'ús.

Flux normal

33. El servidor rep una petició i inicia el cas d'ús actualitzar contactes.

34. El servidor actualitza tots els contactes que es troben connectats.

35. El servidor envia la llista de contactes connectats a tots els clients que estàn amb el programa client en execució.

36. El servidor resta a l'espera de qualsevol event.

37. Finalitza el cas d'ús actualitzar contactes.

6. Disseny

6.1 Client-Servidor

La majoria de les aplicacions que s'executen a Internet (WWW, FTP, Telnet, correu electrònic, etc.) corresponen a una arquitectura denominada client-servidor. Es diu que una aplicació informàtica – definida com a un conjunt de programes i dades- te una estructura client-servidor quan els programes i dades que la constitueixen es troben repartides entre dos o més ordinadors.

En una arquitectura client-servidor, un ordenador (denominat client) conté la part de l'aplicació que s'encarrega de la gestió de la interacció immediata amb l'usuari i, possiblement, de part de la lògica i de les dades de l'aplicació. Les operacions o les dades que no resideixen en el client però que aquest necessita, les sol·licita a la altre part de l'aplicació mitjançant el protocol de l'aplicació corresponent. Aquesta part es troba en un ordenador compartit (el servidor), normalment més potent i de major capacitat que el client, i esta formada per la resta de dades i funcions de l'aplicació informàtica. Aquest equip s'encarrega de servir (és a dir, atendre i contestar) les peticions realitzades pels clients. Els termes client i servidor s'utilitzen tant per referir-se als ordenadors com a les parts de l'aplicació.

Atenent a la ubicació de la lògica i de les dades de l'aplicació, es poden establir diferents classificacions dins de l'arquitectura (lògica distribuïda, presentació descentralitzada, lògica descentralitzada). En el nostre cas ens trobem davant del cas de la lògica distribuïda, ja que el client i el servidor es reparteixen la feina, alleugerin així les tasques. El client s'encarrega de la interacció amb l'usuari, de mostrar els resultats de les seves peticions i de controlar la part bàsica de la lògica de l'aplicació (comprovació de camps, introducció de valors en els camps obligatoris, etc.). El servidor és el que accedirà a la base de dades i l'encarregat de la part important de la lògica de l'aplicació (assignació a cada client de la llista d'usuaris connectats, per exemple).

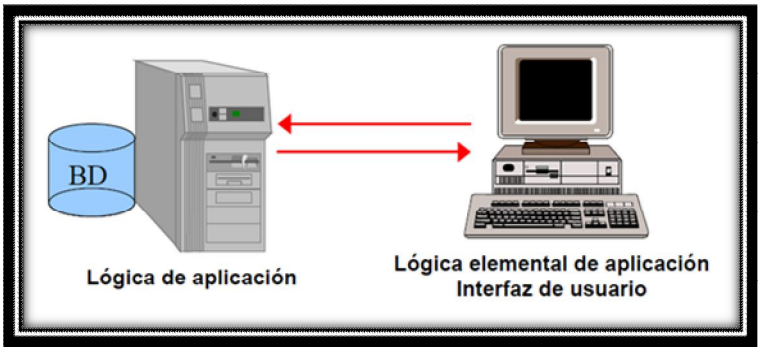

Figura 8. Esquema arquitectura client-servidor

Un exemple típic d'arquitectura client-servidor el constitueix un servidor web i un navegador web. L'ordenador client és el de l'usuari que desitja accedir a una pàgina web i te en execució un programa client (un navegador web, en aquest cas). Quan un usuari introdueix una direcció web en el navegador, aquest sol·licita, através del protocol HTTP, la pàgina web a l'aplicació de servidor web que s'executa en el ordenador servidor on resideix la pagina. L'aplicació de servidor web envia la pàgina per Internet a l'aplicació client (el navegador) que la ha sol·licitat. Un cop rebuda el navegador la mostra al usuari.

En el cas del projecte que s'ha desenvolupat clarament es tracta de l'estructura client-servidor de la que s'ha parlat. El servidor és el programa encarregat de fer totes les tasques de gestió d'usuaris, i en futurs desenvolupaments serà el que contindrà i gestionarà la base de dades del programa. En canvi el client, es un programa més lleuger que depèn del servidor per al seu funcionament, i que desenvolupa les tasques més bàsiques que permeten oferir la interfície d'usuari per tal de que aquest pugui treballar amb l'aplicació.

6.2 Patró arquitectònic

Quan es desenvolupa un software mai s'ha de perdre de vista el futur. Aquest és el principal argument pel qual s'ha escollit com a patró arquitectònic el Model-Vista-Controlador. I és que al cap i a la fi el que s'està desenvolupant és un software que es pretén que en un futur pugui ser ampliat. La lògica d'aquest software és complexa i per tant és interessant salvaguardar-la i aïllar-la de la resta de l'aplicació, sobretot de la presentació ja que és la que més tendeix a canviar.

Aplicant el patró MVC no només s'aconsegueix això, sinó que a més pots controlar a quines funcionalitats te accés cada interfície fent que apunti a un controlador principal diferent.

Vista/Presentació: La capa de presentació equival a la vista del patró MVC. Es tracta d'un component totalment aïllat de la resta de l'aplicació que tan sols té constància dels controladors principals que es defineixin. En aquest projecte s'ha implementat una interfície gràfica estàndard de Java am els components de tipus Java.swing.

Controlador/Aplicació: La capa d'aplicació equival al controlador del patró MVC. La seva responsabilitat és única i exclusivament delegar. És un error tant comú com greu carregar al controlador de feina que no li concerneix. El flux que segueixen les dades és el següent: captura les dades de la vista; crea els objectes del domini pertinents i n'executa els mètodes corresponents.

Model (Domini i Persistència): La capa de domini i la capa de persistència equivalen al model del patró MVC. És la part més acoblada de tota l'aplicació. En el domini resideix tota la lògica de l'aplicació mentre que a la persistència s'hi troben tots els mètodes que interaccionen amb la base de dades. Un objecte del domini mai accedirà a la base de dades directament sinó que sempre ho farà mitjançant una interfície.

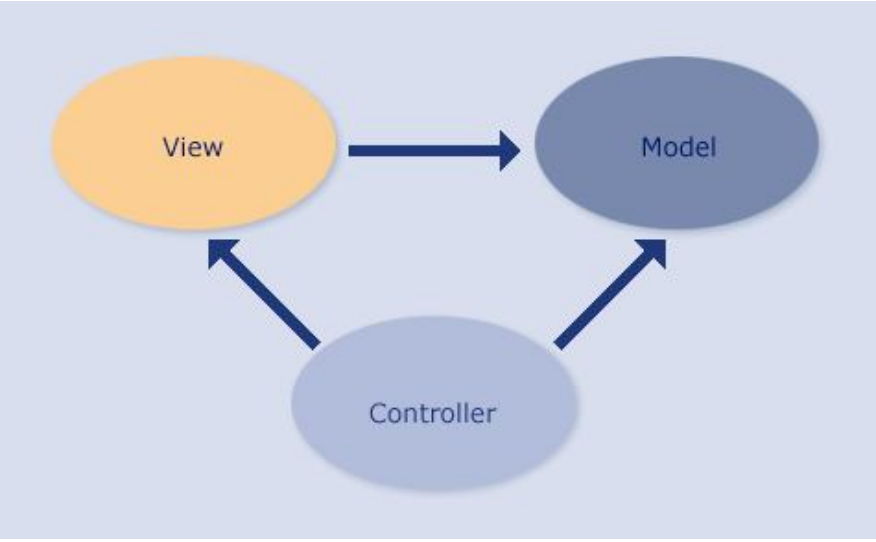

Figura 9. Esquema Model Vista Controlador

6.3 Patrons de disseny

Singleton: Es tracta d'un patró molt simple que ajuda a l'aplicació a no sobrecarregar-la amb objectes instanciats varies vegades innecessariament. S'utilitza quan en temps d'execució tan sols es necessita una instància d'un objecte determinat. Aquest patró s'ha aplicat, per exemple, a la classe que crea la connexió amb la base de dades.

6.4 Model de comunicacions

En aquest apartat s'explicarà el model de comunicacions que s'ha seguit per desenvolupar l'aplicació. Això vol dir, com parlen les dos aplicacions entre elles, com son els missatges que s'envien i de quina manera es fan les peticions. Es diferenciaran dos casos, la comunicació entre client i servidor i la comunicació entre dos clients.

6.4.1 Comunicació Client – Servidor

Primera connexió del client:

En aquest punt es parlarà del diàleg que es produeix quan s'arranca un client i estableix la primera comunicació amb el servidor que entre d'altres coses li subministrarà la llista de contactes connectats.

En primera instància cal remarcar que perquè es produeixi la comunicació client-servidor és condició indispensable el fet de que la màquina que conté el servidor tingui el programa en funcionament i hagi arrancat el servidor pròpiament dit. Al arrancar el servidor, el que estem fent és posar el programa en un bucle infinit que el que fa és escoltar permanentment la xarxa (pel port per el qual s'ha assignat prèviament que es vol que el servidor atengui les peticions).

Un cop el servidor està pendent de les peticions entrants, ja podem arrancar qualsevol dels clients. Els clients saben (perquè ho tenen en la seva configuració), la direcció IP on es troba la màquina que conté el servidor, i per quin port està pendent de les peticions entrants. Un cop el client ja sap el nick del usuari que s'ha connectat al programa, el que fa és enviar una petició de connexió al servidor a la IP i el port destí.

El servidor el que veu és una petició de connexió entrant i el que fa és acceptar la connexió. Al acceptar la petició el que passa és que es crea la connexió entre les dos aplicacions, això es tradueix en l'establiment d'un canal vinculat a les dos aplicacions i la inicialització d'un buffer contenidor de bytes que permet l'intercanvi d'informació entre les aplicacions.

Arribat aquest punt es guarda en un contenidor les dades del client. Això vol dir que el servidor ja disposa de el nick del client, la seva direcció IP i el port per el qual el mini-servidor del client escolta peticions. Un cop el servidor disposa de tota la informació que necessita del client, el que fa és aprofitar el canal obert per enviar-li la informació de tots els usuaris que estan connectats. En aquest punt el client ja pot arrancar la seva llista de contactes i es tanca la connexió entre el client i el servidor fins a futures peticions.

En la següent figura es mostra de manera esquemàtica el que s'acaba d'explicar sobre la comunicació client servidor el primer cop que el client inicia el programa.

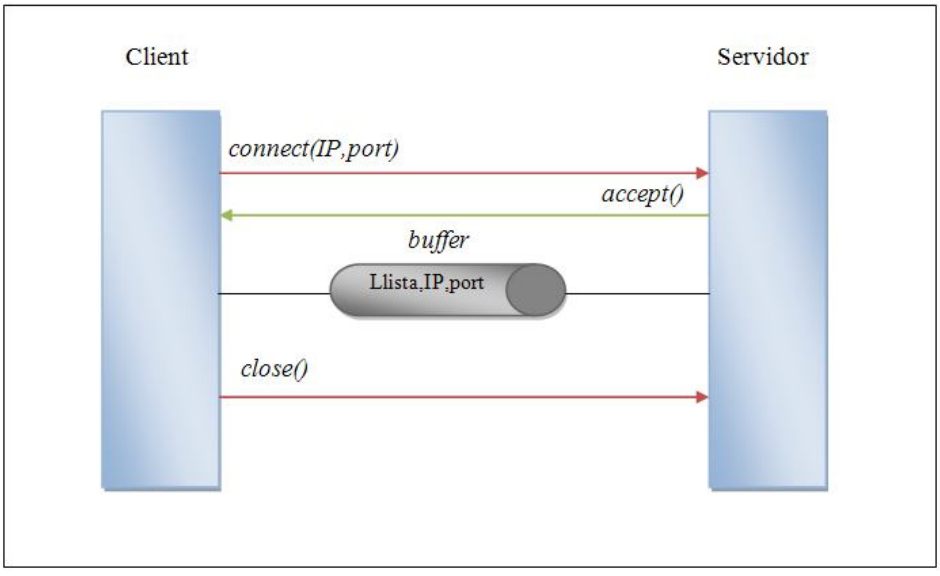

Figura 10. Esquema comunicació client-servidor

Actualització llista contactes:

Periòdicament els clients demanen una actualització al servidor, per tal de mantenir en tot moment la llista de contactes online actualitzada. Això es fa de la següent manera, el client envia una petició de connexió al servidor. Quan aquest accepta el client envia pel buffer la petició amb la qual demana la llista de contactes online i el servidor li subministra.

Un cop el client disposa de la llista actualitzada de contactes online tanca la connexió amb el servidor. Cal remarcar el fet que el servidor en tot moment serveix les peticions seguin una política de tipus FIFO (First In First Out), això significa que la primera petició que li arriba es la primera a la que atent, posa en espera les demés. Un cop a atesa la primera, atent la següent de la cua, i així va fent.

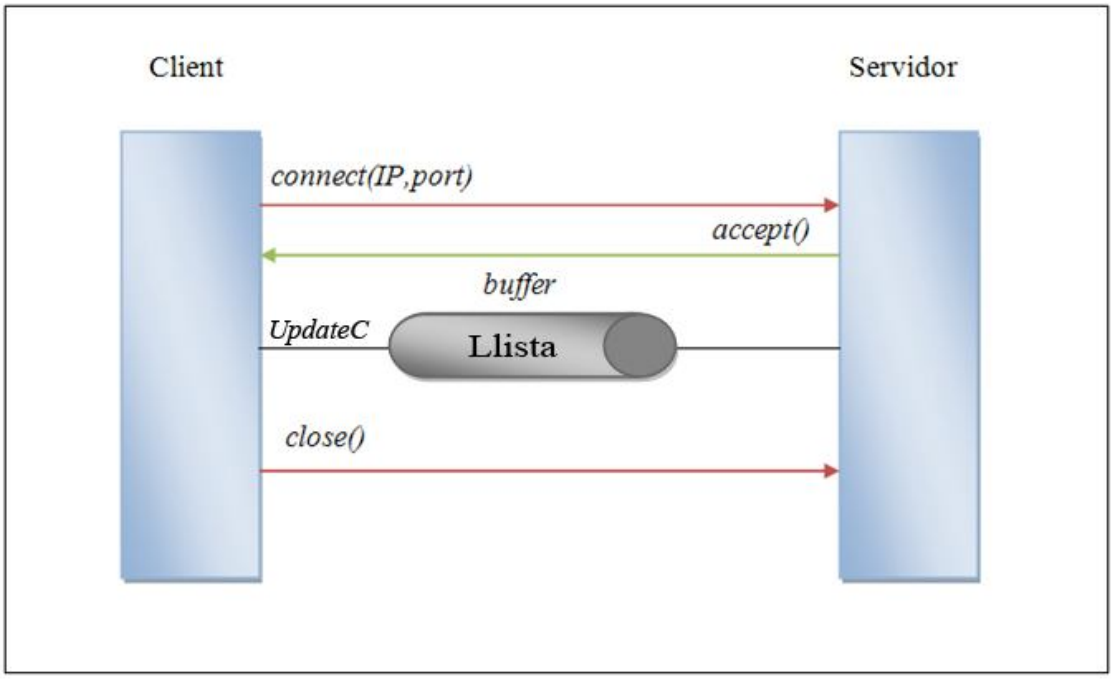

Figura 11. Esquema comunicació actualitzar contactes

Petició informació contacte:

Quan un client vol iniciar una conversació amb un altre client el que fa és comunicar-ho al servidor i aquest li subministra la IP i el port per el qual esta escoltant

aquest client, permeten així que el miniservidor del client pugui establir la connexió amb l'altre client.

Per fer-ho es seguiria la metodologia del cas anterior, l'únic que canviarà és la petició que faria el client, que aquest cop enlloc de demanar una actualització de contactes (mitjançant el updateC), demanaria les dades del contacte amb el que vol parlar. El servidor respondria subministrant la informació demanada, i posteriorment un cop el client revés la informació, tancaria la connexió. I el client amb aquesta informació procedirà a iniciar la conversa, però aquest punt es tractarà una mica més endavant.

Desconnexió d'un client:

Quan un client es desconnecta segueix l'esquema que s'ha vist fins ara. Primer estableix la connexió amb el servidor com s'ha vist, a continuació envia al servidor el missatge informant que el client es desconnecta.

En aquest punt el servidor el que fa és actualitzar la seva llista de contactes connectats treien d'ella el client que s'ha donat de baixa, i acte seguit fa un broadcast a tota la seva llista de contactes enviant la llista actualitzada de contactes connectats. Tal i com es mostra en el següent esquema.

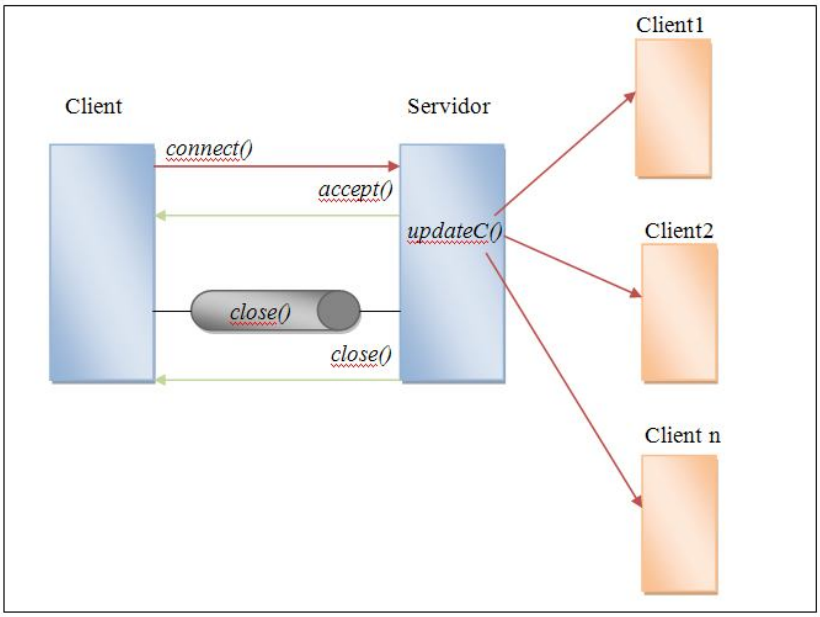

Figura 12. Esquema comunicació desconnexió

6.4.2 Comunicació Client – Client

Tal i com s'ha comentat de passada en algún apartat anterior, els clients disposen d'un petit servidor, de funcions limitades, que te menys tasques a controlar que el servidor gran, però que permet que les converses entre clients es facin independentment del servidor. Això s'ha decidit fer d'aquesta manera perquè està demostrat que carregar el servidor en excés és molt contraproduent.

Si totes les converses que mantenen tots els clients connectats entre ells tinguessin que passar per el servidor, el més probable és que quan el número de converses fos una mica elevat, el servidor es començaria a saturar, i començaria a anar lent i a tardar molt en complir les seves tasques més importants com són, les de gestió de clients connectats o de peticions de conversa o desconnexió. En un cas extrem on el nombre de converses simultànies fos molt gran, podria arribar a desbordar la pila de la maquina virtual de Java o inclús la del propi sistema operatiu.

Per tot això s'ha decidit seguir el model que fa servir el Microsoft Messenger, on es descentralitza la conversa entre clients del servidor, i es situa en el client. D'aquesta manera ens estalviem la sobrecàrrega del servidor i podem garantir que per moltes converses simultànies que pugui mantenir un client, mai arribarà a saturar el seu mini servidor.

Així doncs, els clients disposen de un petit servidor que està escoltant la xarxa per un port en concret (cada client per un port diferent per evitar errors) a l'espera de qualsevol connexió entrant que pertanyi a algun contacte que vol iniciar una conversa. Quan es rep una conversa entrant, el que es fa és preguntar al servidor que ens subministri la informació d'aquell contacte (IP i port destí) per tal de poder-li respondre. Un cop tenim aquesta informació la conservem fins al final de la conversa, d'aquesta manera no hem d'anar fent peticions al servidor.

La conversa es produeix seguint l'esquema que es mostra a continuació. Cada cop que un dels dos clients envia un missatge s'estableix la connexió, es transfereix el missatge i es tanca la connexió.

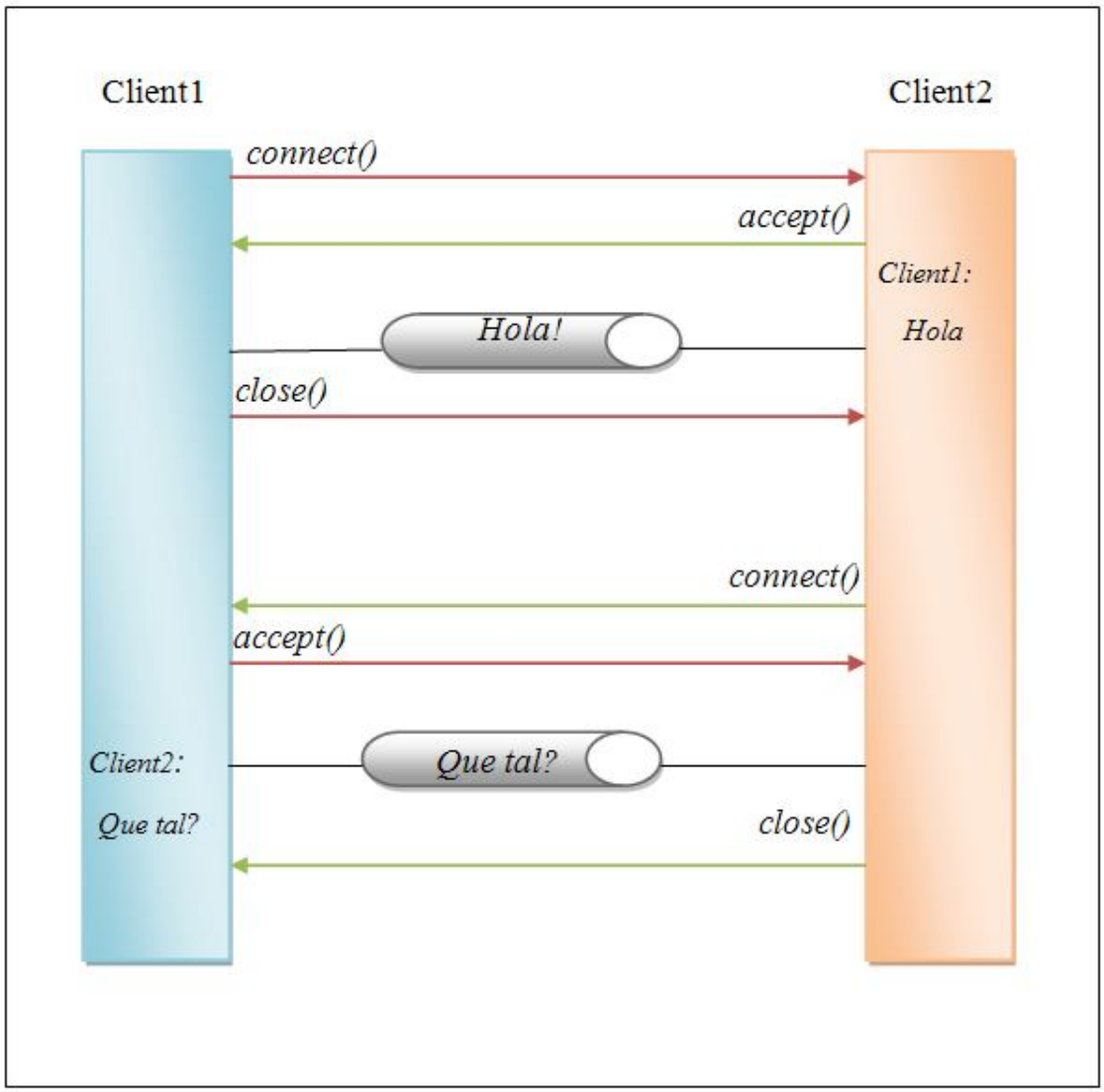

Figura 13. Esquema comunicació client-client

Això es fa així perquè com sabem, els sockets requereixen de molts recursos tant de la MVJ com del sistema operatiu i seria molt poc eficient mantenir totes les connexions obertes tota l'estona, per tant es prefereix anar obrint i tancant-les.

D'aquesta manera, el mini servidor estarà constantment escoltant la xarxa i reben missatges identificats pel nom de l'usuari que els envia, així sabrà en quina finestra de xat li pertoca col·locar-los. Sobre aquest mini servidor, cal comentar que segueix la mateixa política per servir les peticions que el servidor gran (FIFO). També s'ha de dir que aquest mini servidor és totalment transparent al usuari, s'arranca sol i s'executa de manera paral·lela a l'execució del client en un *thread* específic que el client ni tan sols coneix.

7. Implementació

En aquest apartat s'exposaran alguns dels mètodes i funcions més rellevants dels dos programes desenvolupats (servidor i client), o que per algun motiu mereixen algun tipus de menció especial.

7.1 Mètode decoder

Un dels problemes amb els que em vaig trobar, era el fet de que els buffers amb els que treballa el Java, i en concret el paquet java.nio, son buffers de bytes (ByteBuffer), això el que fa és que es necessitin d'alguns mètodes per transformar les cadenes de caràcters (String), amb les que habitualment treballem els programadors i que finalment seran el que se'ns mostrarà per pantalla.

El Java ens ofereix una operació cridada directament desde un String que permet transformar un String a una cadena de bytes directament, per tal de poder introduir la cadena en qualsevol buffer. El problema resideix quan el que es necessita és transformar una cadena de bytes a String. El Java no ens ofereix cap mètode per fer aquesta conversió i es va haver de desenvolupar el mètode anomenat *decoder* que es mostra a continuació per poder realitzar aquesta tasca, que a l'hora és una feina que es repeteix per tot el programa (tant el client com el servidor), i que ha anat bé tenir centralitzada en un mètode.

```
protected String decoder(ByteBuffer b){
      Charset charset= Charset.forName("ISO-8859-1");//Conversor del ByteBuffer a
      String
      CharsetDecoder decoder=charset.newDecoder();
      CharBuffer charBuffer;
   try {
      charBuffer = decoder.decode(b);
   }
   catch (CharacterCodingException ex) {
      return null;
   }
   return charBuffer.toString();}
```

7.2 Thread MiniServer

Com s'ha comentat en apartats anteriors, els clients disposen de un petit servidor que és l'encarregat de gestionar les converses que mantenen els clients entre ells. Per desenvolupar aquest servidor, ha fet falta que estigués executant-se en segon pla a l'hora que s'executava el client, sense intervenir en aquest, ja que requeria d'una execució constant per tal de poder estar escoltant la xarxa permanentment.

Si aquest mini servidor s'hagués executat junt amb el programa principal client, hi haurien hagut problemes de concurrència i bloqueig que impedien el correcte funcionament del client. El que s'ha fet ha sigut recórrer als Threads o fils del Java.

Això ha permès tenir en el client dos fils d'execució simultanis que no interfereixen l'un en l'altre per res. Per un cantó tenim el fil que porta el programa client i tota la transferència de dades i connexions, i per l'altre el servidor que constantment esta escoltant la xarxa i reben missatges. L'únic punt d'intersecció entre aquest dos fils, és el moment en el que el mini servidor envia els missatges rebuts a les pantalles de l'execució del fil principal.

En el següent fragment de codi es mostra el tros del client on es defineix el segon fil d'execució. Aquest fil és el que tindrà el mini servidor en execució constant.

```
private class ThreadServidor extends Thread{

      public synchronized void run (){
         arrancarMiniServidor();
         procesarConversa();

      }//fi public void run()
}
```

7.3 UpdateContactes

S'ha volgut presentar aquest mètode en aquest apartar perquè es considera que és un mètode molt complet en el qual es poden veure vàries coses interessants. Com per exemple, com s'estableix una connexió amb el servidor mitjançant els sockets i els mètodes que ens proporciona la classe java.net.

També podem observar com es tracta amb els buffers mitjançant les opcions que ens ofereix la classe java.nio i el ús que fem del, anteriorment comentat, mètode decoder per poder transformar les cadenes de Bytes en Strings. Finalment comentar, que aquest és el mètode que periòdicament es connecta al servidor i actualitza la llista de contactes online,

```
public void updateContactes (){

    ByteBuffer buffer = ByteBuffer.allocate(TAMANY_BUFFER);
    SocketChannel sc;
    model.clear();
    nickPort.clear();
    buffer.clear();

        try {

            sc = SocketChannel.open();
            sc.connect(new InetSocketAddress(MAQUINA, PORT1));
            sc.read(buffer);//Llegim cadena nicks del server

            buffer.flip();
            sc.close();

            nicks = decoder(buffer);
            jList1.setModel(vectorDeNicks(nicks));//Omplim el Jlist
            buffer.clear();

        }
        catch(IOException ex){System.out.println("Error en vectorDeNicks");}
}
```

7.4 Desconnexió d'usuari

El mètode que es presenta a continuació és un mètode que respon a un event produït a la interfície gràfica d'usuari. És tracta del mètode que s'activa quan l'usuari tanca el programa client. El que passa és que davant d'aquest event, el programa just abans de tancar-se fa una ultima connexió amb el servidor per informar-lo.

En el codi d'aquest mètode podem veure una de les crides que fa el client al servidor quan li passa el paràmetre "close/...", d'aquesta manera es com es comuniquen el client i el servidor. Les diferents peticions que pot fer el client, es diferencien per comandes com aquesta, que son interpretades d'una manera o altre en el canto del servidor.

Aquesta comanda el servidor la interpreta com que l'usuari que li ha establert la connexió tanca el programa i per tant passa a estar en estat de no connectat. Acte seguit el servidor treu aquest usuari de la llista de contactes connectats i fa un bradcast per informar a tothom del nou estat de la llista de contactes.

```
private void formWindowClosing(java.awt.event.WindowEvent evt) {

   try{
         SocketChannel sc = SocketChannel.open();
         sc.connect(new InetSocketAddress(MAQUINA, PORT1));

         // Se crea un ByterBuffer y se limpia.
         ByteBuffer buffer = ByteBuffer.allocate(TAMANY_BUFFER);
         String aux = "close/" +selfNickPort;
         buffer.clear();
         buffer.put(aux.getBytes());
         buffer.flip();// listo para leer

         sc.write(buffer);
      }
      catch(Exception ex){
         System.out.println("Error al tancar la sessió");
      }
      System.exit(0);

}
```

7.5 Acceptar connexions

Finalment s'ha volgut incloure en aquest apartat un tros de codi del servidor. En ell podem veure el fragment que s'encarrega de acceptar les connexions entrants i convertir el ServerSocket que escolta la xarxa en un socket dedicat a cada client i per tant establir la connexió per tal de poder realitzar qualsevol de les tasques possibles.

Es pot observar que el mètode que ens permet fer això és el *accept()* del paquet java.net. El que passa quan cridem aquest mètode és que creem (a partir del ServerSocket existent) un nou socket que al guardar-lo en una instancia del socket entrant queden vinculats i d'aquesta manera s'estableix la connexió.

Cap al final del codi també es pot observar com es registren els diferents events per els quals es vol preparar el canal, events de lectura o d'escriptura. També podem veure com guardem en el contenidor de clients online tota la informació de la connexió del client (IP i port).

```
private void aceptarConexion(SelectionKey clave) {

    canalSocketServidor = (ServerSocketChannel) clave.channel();
    SocketChannel sc = null;

    ByteBuffer bb = ByteBuffer.allocate(TAMANY_BUFFER);

    // Tratamiento de la conexión y envío mensaje bienvenida.
    try {
       // Se acepta la petición del cliente y se obtiene un canal asociado al socket del
       cliente
       sc = canalSocketServidor.accept();

       if (sc == null) {
          return;
       }

       // Se configura sin bloqueo el socket del cliente y se registra
       // en selector para los sucesos de lectura i escritura
       sc.configureBlocking(false);
       sc.register(selector, SelectionKey.OP_READ | SelectionKey.OP_WRITE);

        // Se añade a la lista de clientes activos.       clientesActivos.add(sc.socket());
```

8. Guia d'usuari

8.1 Interfície Servidor

Figura 14. Interfície Servidor

S'observa que la interfície del programa servidor és força senzilla, consta de dos botons, un per poder arrancar el servidor i l'altre per parar-lo, tal i com indiquen els rètols. Al ser un programa que s'està executant en segon pla no s'ha volgut recarregar massa la seva GUI i s'ha considerat que aquesta era la manera més optima.

8.2 Pantalla Login

A continuació es presenta la pantalla de login del programa client. La finalitat d'aquesta interfície és que l'usuari que entra a l'aplicació de messenger esculli el seu "alias" o nom que voldrà que vegi la gent en la seva llista de contactes.

És en aquesta pantalla on es comprova també que aquest nom que s'ha escullit sigui únic i no estigui sent usat per algun altre usuari del programa. En el cas de que el nom ja existeixi o no s'introdueixi cap nom el sistema informarà de la incidència i no permetrà continuar endavant.

Figura 15. Interfície Login

8.3 Interfície Contactes

La pantalla que es presenta ara és la pantalla on els usuaris podran veure tots els contactes que estan connectats al programa en cada moment. Aquesta pantalla se'ls i obrirà automàticament just després de logejar-se anb un nick correcte.

És desde aquí que l'usuari podrà iniciar les converses amb els usuaris que desitgi. Per fer-ho només tindrà que fer click amb el seu ratolí en el nom del contacte amb el que desitgi parlar. Només una pantalla de contactes podrà existir per cada usuari.

Figura 16. Pantalla contactes

8.4 Pantalla Chat

Un cop s'inicia la conversa amb qualsevol usuari, se'ns obra la pantalla de xat. Podrem tenir oberta una pantalla de xat per cada conversa que tinguem. En la pantalla es poden observar clarament dos sectors, el dedicat a text i el dedicat a símbols.

El primer sector, el de text permet mantenir independentment converses via missatges de text. El segon permet l'enviament de símbols SPC i també és totalment independent de la pantalla de enviament de text.

Un cop s'ha escrit el missatge que es vol enviar s'envia mitjançant el botó Enviar. El mateix passa si el que volem enviar és un símbol SPC.

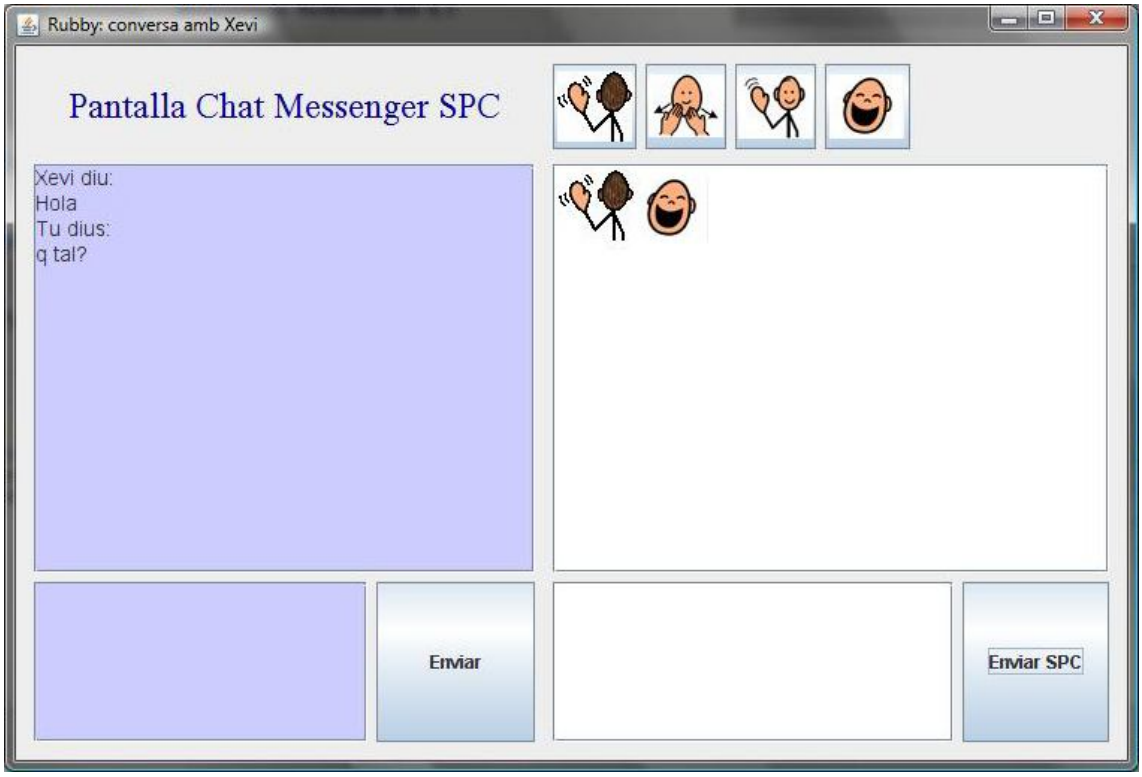

Figura 17. Pantalla xat

9. Propostes de futur

En un principi aquest projecte es va plantejar com un projecte amb dos etapes molt diferenciades. La primera és una etapa en la qual el que s'ha fet és desenvolupar tot l'esquelet de l'aplicació. La part on han sigut necessaris unificar els coneixements de informàtica i telecomunicacions per tal de crear l'estructura bàsica de l'aplicació que permetés fer la missatgeria instantània a través d'una xarxa. Com a resultat d'aquesta primera iteració s'han obtingut els dos programes que s'han presentat, el client i el servidor.

La segona etapa del projecte que no s'ha pogut fer per manca de temps, esdevindria la part en la qual s'adaptaria tota la interfície gràfica a les necessitats especials de la gent a la que va dirigit aquest projecte. A continuació proposaré alguns dels punts que s'haurien de completar per tal de donar per tancat i acabat l'aplicatiu, i que a la meva manera de veure, constituïren feina suficient com per desenvolupar tot un nou projecte final de carrera (unes 250 hores).

Pantalla de Login:

Actualment quan un usuari arranca l'aplicació client el que fa el programa és demanar-li un nick per tal de que els altres usuaris puguin identificar-lo en la seva pantalla de contactes. Doncs bé, per adaptar el software al que el client vol, s'hauria de substituir aquesta pantalla per una pantalla on hi hagués un llistat de fotografies dels usuaris del programa. Així doncs, l'usuari que es connecta al client l'únic que hauria de fer abans d'iniciar la seva sessió seria escollir de la llista la seva foto.

Es desitja que aquestes fotografies puguin ser gestionades per el usuari administrador, que serà aquell que tindrà accés a la base de dades. Per tant, en el servidor s'haurà de muntar una base de dades que contingui totes les fotos, i un gestor que permeti introduir, borra i editar aquestes imatges. Cada cop que un client es connecti al servidor, aquest l'haurà d'informar de si hi ha hagut alguna actualització de la base de dades i si és així enviar-li al client les imatges pertinents.

Pantalla Contactes:

La millora que caldria aplicar en aquesta pantalla, és que els usuaris connectats enlloc de mostrar-se en la llista de contactes amb el seu nick, s'haurien de mostrar amb les fotografies dels usuaris que prèviament hem rebut del servidor i mantenim en local a l'aplicació.

Diccionari SPC:

Una altre de les coses que el client sol·licita, és que l'usuari administrador que té accés al programa servidor pugui actualitzar el diccionari de símbols SPC. Per tant caldrà afegir a la base de dades del servidor tot el conjunt de símbols SPC i una interfície que permeti afegir o borra símbols.

Es vol que quan un client es connecta al servidor aquest l'informi de quina es la ultima versió del diccionari SPC i si no coincideixen caldrà iniciar la transferència del nou diccionari per tal de que el client l'actualitzi i utilitzi sempre la última versió del diccionari de símbols SPC.

Pantalla Xat:

La pantalla del xat seguirà en línies generals l'esquema que hi ha plantejat actualment, però s'haurà de retocar la distribució dels símbols SPC al gust dels pedagogs especialistes en el tema, que decidiran quina és la manera més eficient i ergonòmica per a les seves necessitats.

Una altre de les funcionalitats que es desitja desenvolupar, és que el client pugui escriure tant en el canto del xat de text com en el canto de símbols, i que si per exemple posem el símbol SPC equivalent a la paraula "hola" al costat dels símbols, automàticament ens aparegui al costat (en el xat de text) la paraula "hola". El mateix desde la pantalla de text, es desitja que quan s'escrigui amb text "hola" automàticament surti el símbol SPC equivalent a aquesta paraula en el xat de símbols..

10. Conclusions

Finalment i després de molt esforç es presenta una aplicació com a projecte final de carrera formada per dos programes. El primer programa és un servidor que gestiona tots els usuaris que es connecten al programa, i el segon és un client que instal·lat en tants ordinadors com es desitgi permetrà xatejar a tots els usuaris entre ells.

Cal remarcar el fet que s'ha fet una forta inversió en temps per poder aprendre aquelles eines necessàries que el Java ens ofereix per tal de poder desenvolupar una aplicació amb un fort caràcter telemàtic degut al nivell de comunicacions dins de la xarxa amb el que treballa. Ha sigut totalment imprescindible unificar els coneixements adquirits en la doble titulació que he cursat (informàtica i telecomunicacions) i es pot afirmar amb orgull que aquest projecte ha sigut una síntesi del que s'ha après al llarg de tot aquest temps en l'escola universitària.

El fet de programar a nivell de sockets, canals i buffers m'ha donat una visió molt més acurada i un nivell de comprensió molt mes elevat de les comunicacions entre aplicacions a través de la xarxa. Per tal de poder començar a programar, ha sigut totalment necessari plantejar quin model de comunicació es volia seguir en cada fase del projecte, i un cop decidit procedir a programar-lo, d'aquesta manera he entès com funcionen les comunicacions entre clients i servidors.

Un altre dels coneixements en els quals he aprofundit molt gràcies al desenvolupament d'aquesta aplicació ha sigut en el de l'estructura client-servidor. E pogut observar de primera ma quines son les avantatges i quins son els inconvenients d'aquest sistema.

En quan a avantatges, la majoria son a nivell d'usuari final i d'eficiència, ja que aquesta estructura allibera molt el client de tasques innecessàries i el fa molt més lleuger. Permet tenir centralitzades les dades en una sola màquina i tots els processos complexos junts, facilitant així el fet que l'única màquina potent que es necessiti sigui la del servidor.

Com a inconvenient, el principal amb el que jo m'he trobat, és el fet que per el programador és bastant més complex desenvolupar una aplicació d'aquest tipus que no pas una clàssica aplicació d'escriptori. Això és així perquè en l'estructura client-servidor el programador ha d'estar constantment pensant en els fluxos de dades i les converses que mantenen els dos programes, per evitar errors de concurrència o sobrecàrrega.

Per acabar m'agradaria concloure que no som conscients de que les noves tecnologies no arriben a tothom per igual, i que hi ha gent que té una sèrie de necessitats especials que mereixen, igual que tothom, el poder accedir-hi. S'ha d'intentar aportar en la mida del possible, per tal de poder ajudar a que això es vagi solucionant mica en mica i que cada cop més, les noves tecnologies i els nous softwares arribin a tothom per igual.

Xavier Costa Carmona
Escola Universitària Politècnica de Mataró
Enginyeria Tècnica Telecomunicacions

coscarxa@eupmt.upc.edu

Messenger Visual

Resum: Amb el desenvolupament d'Internet han aparegut noves aplicacions que cada dia augmenten el seu nombre d'usuaris. Un exemple son els programes de missatgeria instantània, que permeten al usuari parlar amb els seus contactes via missatges de text i en temps real. El problema que ens trobem és que aquests programes no estan adaptats a les necessitats de tothom. Aquest projecte pretén donar el primer pas per posar remei a aquest inconvenient.

1. Introducció

En una arquitectura client-servidor, el programa servidor es mante escoltant les peticions d tots els clients a través dels ports que te actius. La informació o els serveis sol·licitats son enviats pel servidor als clients, en concret als ports que aquests estiguin utilitzant per realitzar les peticions.

En la família de protocols TCP/IP, existeixen dos protocols de transport (TCP i UDP) que s'encarreguen d'enviar dades d'un port a un altre per fer possible la comunicació entre aplicacions.

Tant TCP com UDP utilitzen sockets (literalment traduït com a endolls) per comunicar programes entre si en una arquitectura client-servidor. Un socket és un punt terminal o extrem en l'enllaç de comunicació entre dos aplicacions (que normalment s'executen en ordinadors diferents). Les aplicacions es comuniquen mitjançant l'enviament i recepció de missatges mitjançant els sockets. Un socket TCP es podria comparar a un telèfon, anàlogament un socket UDP seria la bústia de correus.

A part de classificar-se en sockets TCP i UDP, els sockets poden pertànyer a algun d'aquests dos grups:

- Sockets actius: Poden enviar i rebre dades a través d'una connexió oberta.
- Sockets passius: Esperen intents de connexió. Quan arriba una connexió entrant, li assignen un socket actiu. No serveixen per enviar o rebre dades.

Els socket son creats pel sistema operatiu i ofereixen una interfície de programació d'aplicacions (API) mitjançant la qual les aplicacions poden enviar missatges a altres programes, ja sigui en local o remotament. Les operacions dels sockets (enviar, rebre, etc.) s'implementen com a crides al sistema operatiu en tots els sistemes operatius actuals. Dit d'una altre manera; els sockets formen part del nucli del sistema operatiu. En els llenguatges orientats a objectes com el Java o el C#, les classes de sockets s'implementen sobre les funcions que dona l'API del sistema operatiu per a l'ús de sockets.

2. El paquet java.net

Gran part de la popularitat del Java es deu a la seva orientació a Internet. Cal afegir que aquesta acceptació resulta merescuda, Java proporciona una interfície de sockets orientada a objectes que simplifica

moltíssim el treball en xarxa.

Amb aquest llenguatge, comunicar-se amb altres aplicacions a través d'Internet és molt similar a obtenir l'entrada del usuari a través de la consola o de llegir arxius, en contrast amb el que succeeix amb llenguatges com el C. Cronològicament, Java va ser el primer llenguatge de programació on la manipulació de l'entrada i sortida de dades a través de la xarxa es realitzava igual que la E/S amb arxius.

El paquet java.net, que proporciona una interfície orientada a objectes per crear i manipular sockets, connexions HTTP, localitzadors URL, etc., esta formada per classes que es poden dividir en dos grans grups:

- Classes que corresponen a les API (interfícies de programació d'aplicacions) dels sockets: **Socket**, **ServerSocket**, **DatagramSocket**, etc.

- Classes corresponents a eines per treballar amb URL: **URL**, **URL-Connection**, **HttpURLConnection**, **URL-Encoder**, etc.

El paquet java.net permet treballar amb els protocols TCP i UDP. La classe **java.net.Socket** permet crear sockets TCP per al client; la classe **java.net.Server-Socket** fa el mateix per al servidor. Per les comunicacions UDP, Java ofereix la classe **java.net.Datagram-Socket** per als dos costats de la comunicació UDP, i la classe **java.net.DatagramPacket** per crear datagrames UDP.

3. El paquet java.nio

La versió 1.4 de Java va incorporar noves característiques al llenguatge, mantingudes en la posterior versió 5.0. Una de les més importants és el paquet java.nio. Aquest paquet ofereix una nova API d'entrada i sortida (E/S), coneguda com NIO (New Input/Output). La nova API pot dividir-se en tres grans paquets:

- **java.nio** defineix buffers, que s'utilitzen per emmagatzemar seqüències de bytes o d'altres valors primitius (int, float, char, double, etc.).
- **java.nio.channels** defineix canals, això és, abstraccions mitjançant les quals poden transferir-se dades entro els buffers i les fonts o els consumidors de dades (un socket, un fitxer, un dispositiu de hardware). A més, proporciona classes que permetin E/S sense bloqueig.
- **java.nio.charset** conté classes que converteixen de manera molt eficaç buffers de bytes en bytes de caràcters i permeten l'ús d'expressions regulars.

NIO proporciona grans avantatges sobre l'E/S estàndard de java (java.io). El problema de la ineficàcia de la E/S en Java ha estat present desde les primeres versions. Resultava contradictori trobar-se davant d'un llenguatge amb moltes característiques enfocades a Internet i, al mateix temps, amb un rendiment tant pobre en quant a la transferència massiva de dades.

A continuació s'exposen les principals característiques de NIO, en contraposició amb la E/S estàndard de Java:

- Incorpora buffers, canals i selectors.

- Permet la transferència eficaç de grans quantitats de dades.

- Els sockets de NIO permeten treballar sense bloqueig.

- Aprofita les prestacions del sistema operatiu on s'executa la MVJ.

4. Client-Servidor

En una arquitectura client-servidor, un ordenador (denominat client) conté la part de l'aplicació que s'encarrega de la gestió de la interacció immediata amb l'usuari i, possiblement, de part de la lògica i de les dades de l'aplicació. Les operacions o les dades que no resideixen en el client però que aquest necessita, les sol·licita a la altre part de l'aplicació mitjançant el protocol de l'aplicació corresponent. Aquesta part es troba en un ordenador compartit (el servidor), normalment més potent i de major capacitat que el client, i esta formada per la resta de dades i funcions de l'aplicació informàtica. Aquest equip s'encarrega de servir (és a dir, atendre i contestar) les peticions realitzades pels clients. Els termes client i servidor s'utilitzen tant per referir-se als ordenadors com a les parts de l'aplicació.

Atenent a la ubicació de la lògica i de les dades de l'aplicació, es poden establir diferents classificacions dins de l'arquitectura (lògica distribuïda, presentació descentralitzada, lògica descentralitzada). En el nostre cas ens trobem davant del cas de la lògica distribuïda, ja que el client i el servidor es reparteixen la feina, alleugerin així les tasques. El client s'encarrega de la interacció amb l'usuari, de mostrar els resultats de les seves peticions i de controlar la part bàsica de la lògica de l'aplicació (comprovació de camps, introducció de valors en els camps obligatoris, etc.). El servidor és el que accedirà a la base de dades i l'encarregat de la part important de la lògica de l'aplicació (assignació a cada client de la llista d'usuaris connectats, per exemple).

Un exemple típic d'arquitectura client-servidor el constitueix un servidor web i un navegador web.

5. Comunicació client-servidor

Al arrancar el servidor, el que estem fent és posar el programa en un bucle infinit que el que fa és escoltar permanentment la xarxa

Un cop el servidor està pendent de les peticions entrants, ja podem arrancar qualsevol dels clients. Els clients saben la direcció IP on es troba la màquina que conté el servidor, i per quin port està pendent de les peticions entrants. Un cop el client ja sap el nick del usuari que s'ha connectat al programa, el que fa és enviar una petició de connexió al servidor a la IP i el port destí.

El servidor el que veu és una petició de connexió entrant i el que fa és acceptar la connexió. Al acceptar la petició el que passa és que es crea la connexió entre les dos aplicacions, això es tradueix en l'establiment d'un canal vinculat a les dos aplicacions i la inicialització d'un buffer contenidor de bytes que permet l'intercanvi d'informació entre les aplicacions.

6. Comunicació client-client

els clients disposen de un petit servidor que està escoltant la xarxa per un port en concret (cada client per un port diferent per evitar errors) a l'espera de qualsevol connexió entrant que pertanyi a algun contacte que vol iniciar una conversa. Quan es rep una conversa entrant, el que es fa és preguntar al servidor que ens subministri la informació d'aquell contacte (IP i port destí) per tal de poder-li respondre. Un cop tenim aquesta informació la conservem fins al final de la conversa, d'aquesta manera no hem d'anar fent peticions al servidor. Cada cop que un dels dos clients envia un missatge s'estableix la connexió, es transfereix el missatge i es tanca la connexió.

7. Conclusions

El fet de programar a nivell de sockets, canals i buffers m'ha donat una visió molt més acurada i un nivell de comprensió molt mes elevat de les comunicacions entre aplicacions a través de la xarxa. Per tal de poder començar a programar, ha sigut totalment necessari plantejar quin model de comunicació es volia seguir en cada fase del projecte, i un cop decidit procedir a programar-lo, d'aquesta manera he entès com funcionen les comunicacions entre clients i servidors.

Un altre dels coneixements en els quals he aprofundit molt gràcies al desenvolupament d'aquesta aplicació ha sigut en el de l'estructura client-servidor. E pogut observar de primera ma quines son les avantatges i quins son els inconvenients d'aquest sistema.

En quan a avantatges, la majoria son a nivell d'usuari final i d'eficiència, ja que aquesta estructura allibera molt el client de tasques innecessàries i el fa molt més lleuger. Permet tenir centralitzades les dades en una sola màquina i tots els processos complexos junts, facilitant així el fet que l'única màquina potent que es necessiti sigui la del servidor.

Com a inconvenient, el principal amb el que jo m'he trobat, és el fet que per el programador és bastant més complex desenvolupar una aplicació d'aquest tipus que no pas una clàssica aplicació d'escriptori. Això és així perquè en l'estructura client-servidor el programador ha d'estar constantment pensant en els fluxos de dades i les converses que mantenen els dos programes, per evitar errors de concurrència o sobrecàrrega.

8. Bibliografía

I. Ron Hitchens, *Java.NIO*, Ed. O'Reilly 2002.

II. Elliote Rusty Harold, *Java Network Programming, Second Edition*, Ed.O'Reilly 2000.

III. Craig Larman, *UML y Patrones*, 2a Edición. Pearson, Prentice Hall.

IV.J.API, http://java.sun.com/j2se/1.5.0/docs/api/ 2004

V. Elliote Rusty Harold, *Java I/O, Second Edition*, Ed.O'Reilly 2006.

VI. Steven Holzner, *La Biblia de Java2*, Ed.Anaya Multimedia, 2005

Annex II - Contingut del CD

- **Aplicació**
  - Codi font de l'aplicació Client.
  - Codi font de l'aplicació Servidor.

- **Documentació**
  - Memòria del projecte (Microsoft Office Word i Adobe Reader).
  - Portada de la memòria.
  - Article del projecte (Adobe Reader).

- **Fitxers d'instal·lació**
  - Fitxer d'instal·lació de NetBeans.
  - Fitxer d'instal·lació de MySQL.
  - Fitxer d'instal·lació de SQLyog.

Bibliografia

I. Ron Hitchens, *Java.NIO,* Ed. O'Reilly 2002.

II. Elliote Rusty Harold, *Java Network Programming, Second Edition,* Ed.O'Reilly 2000.

III. Craig Larman, *UML y Patrones*, 2a Edición. Pearson, Prentice Hall.

IV. Java API, http://java.sun.com/j2se/1.5.0/docs/api/ 2004

V. Elliote Rusty Harold, *Java I/O, Second Edition*, Ed.O'Reilly 2006.

VI. Steven Holzner, *La Biblia de Java2*, Ed.Anaya Multimedia, 2005