JXTA Xat Segur

Jordi Serra i Puig
ETIS

Antoni Martínez i Ballesté

15/06/2006

Agraïments

A Brendan Wilson, autor d'un llibre de referència sobre JXTA que, a més, ha posat a la lliure disposició de tothom penjant-lo a la seva pàgina web; a Daniel Brookshier que, a més de coautor de textos sobre JXTA, té l'amabilitat de publicar articles sobre la mateixa plataforma que ens marquen línies de treball, i, en especial, a tots aquells que participen activament en fòrums i grups de debat a Internet, i que quan ens trobem davant un problema que ens sembla irresoluble, sempre ens acaben donant un cop de mà.

Resum

Actualment, les xarxes **P2P** –*Peer to Peer*– permeten que multitud d'equips, principalment ordinadors, es comuniquin emprant una xarxa de comunicacions que normalment és Internet. Tal com es comenta en una de les referències bibliogràfiques consultades[1], "una xarxa P2P és, bàsicament, una xarxa que no té clients ni servidors fixos, sinó tot un conjunt de nodes que es comporten alhora com a clients i com a servidors dels altres nodes de la xarxa". Les xarxes P2P ofereixen la infraestructura que permet als equips connectats fer ús d'aplicacions que fan possible l'intercanvi d'arxius, compartir recursos per a realitzar tasques de computació distribuïda o bé oferir serveis de xat i missatgeria instantània. Es pot considerar que el treball descrit en aquesta memòria s'encabeix en la darrera categoria. L'aplicació ***JXTA Xat Segur*** és una aplicació de xat, desenvolupada en Java, pensada per a ser executada en un entorn P2P. En ella, a l'inici de cada conversa, un cop la petició ha estat acceptada, cada parella de participants genera una clau de sessió emprant el mètode d'intercanvi de claus de **Diffie** i **Hellman**. A partir d'aquest moment, la clau generada s'empra per a xifrar els missatges que s'intercanvien els interlocutors, de forma que ningú fora d'ells pot conèixer el contingut de la conversa. L'aplicació, que permet mantenir vàries converses alhora, amb la possibilitat de varis participants en la mateixa conversa, ha estat desenvolupada a partir de la implementació Java del marc de treball ofert per la plataforma **JXTA** (l'origen del nom és la paraula anglesa *juxtapose*). Aquesta plataforma, basada en estàndards com XML i Java, és el resultat d'un projecte iniciat per *Sun Microsystems* l'any 2001 i proporciona un conjunt de protocols no propietaris que serveixen de marc de referència en les comunicacions a través de xarxes P2P.

[1] R.J. Millán Tejedor, *Domine las redes P2P*, pàg. 4.

Índex de Continguts

Índex de Figures

1.- Introducció.

1.1.- Descripció i justificació del TFC.

El projecte que es descriu en aquest document consisteix en el desenvolupament d'una aplicació de xat emprant la tecnologia **P2P**, en la qual els missatges que s'intercanvien els distints interlocutors es transmeten xifrats de forma que ningú fora d'ells pot conèixer el contingut de la conversa.

Per tal de complir amb aquest requeriment de seguretat, a l'inici de cada sessió, els participants generen una clau de sessió emprant el mètode d'intercanvi de claus de **Diffie-Hellman**. Aquest mètode permet usar un mitjà insegur i públic com Internet per generar una clau secreta que només coneixeran els interlocutors.

Actualment, moltes de les aplicacions P2P utilitzen protocols de comunicació propietaris i incompatibles. **JXTA** és un projecte iniciat per *Sun Microsystems* l'any 2001 amb l'objectiu d'especificar un conjunt de protocols no propietaris que serveixin de marc de referència en les comunicacions a través de xarxes P2P. Basat en estàndards com XML i Java, no requereix estrictament l'ús d'aquest darrer llenguatge ja que els protocols es poden implementar també en C, C++, Perl o d'altres llenguatges. Tanmateix, n'existeix una implementació en Java que s'ha emprat en el desenvolupament de l'aplicació.

1.2.- Objectius.

Tal com es comenta en l'apartat anterior, el programari desenvolupat és una aplicació de xat en la què els missatges viatgen xifrats. Com a aplicació de xat, el programa permet:

- Administrar llistes de contactes, donant-los d'alta, modificant les seves propietats o bé donant-los de baixa.
- Possibilitar a l'usuari la gestió del seu estat quan hagi iniciat l'aplicació.
- L'estat inicial –activació de la part servidora– i d'altres característiques de l'interlocutor es poden configurar.

Tenint en compte el mètode d'intercanvi de claus escollit –Diffie-Hellman–, quan es crea un nou contacte es generen els paràmetres públics previstos en el mètode: el nombre primer **p** i l'enter **g** generador del grup multiplicatiu $\mathbb{Z}_p$. Aquests dos paràmetres es guarden, juntament amb les altres dades del contacte, en un fitxer. Aquesta informació s'emmagatzema també xifrada amb una clau generada a partir de la contrasenya del propietari de l'agenda.

En iniciar-se la conversa, els dos interlocutors generen una clau de sessió emprant l'algorisme bàsic de Diffie-Hellman, clau que es manté en el transcurs de la conversa.

L'aplicació, desenvolupada en Java, compleix també els següents objectius:

- Es poden mantenir diverses converses al mateix temps, amb la possibilitat de varis participants en una conversa.
- La interacció de l'usuari amb l'aplicació es realitza emprant una interfície gràfica.

Finalment, tenint present el model de xarxa que es troba en la base de la proposta, un altre objectiu del treball ha estat el coneixement de les xarxes P2P i la recerca de marcs de referència a l'hora de desenvolupar una aplicació per aquest tipus de xarxa. En el transcurs d'aquest procés de recerca, s'ha descobert la plataforma JXTA, la qual, un cop analitzada, ha esdevingut la base sobre la qual s'han construït els processos de comunicació que tenen lloc entre els equips que usen l'aplicació. L'estudi i coneixement de la plataforma, així com de la implementació en Java, han esdevingut, des d'aleshores, altres objectius del projecte.

1.3.- Enfocament i mètode seguit.

Abans d'indicar quins han estat l'enfocament i el mètode seguits tant a l'hora de planificar el treball com en el moment de desenvolupar l'aplicació que es lliura, cal fer esment a la situació de partida de l'estudiant en l'instant d'iniciar el treball:

- En el transcurs dels seus estudis, havia cursat matèries –Xarxes, Estructura de Xarxes, Seguretat en Xarxes i Criptografia– que li aportaven la formació bàsica suficient tant en el què es refereix a temes de comunicacions com a qüestions de criptografia.
- Disposava d'un coneixement bàsic del llenguatge Java, tot i que desconeixia aspectes relacionats amb la programació de les interfícies gràfiques i, sobretot, la forma de programar el treball amb diferents fils d'execució.

- Tenia un coneixement superficial de les xarxes P2P i de les aplicacions de missatgeria instantània com ICQ i Messenger.

Tenint presents aquests antecedents, la primera tasca a realitzar fou la cerca d'informació relacionada amb aquells temes on es presentaven dèficits. Referent a les xarxes P2P, aquest procés de cerca va permetre arribar a les següents conclusions:

- Tal com es comenta en el resum, una xarxa P2P és, bàsicament, una xarxa entre iguals que no té clients ni servidors fixos, sinó un conjunt de nodes que es comporten alhora com a clients i servidors.
- Ara bé, en la gran xarxa que és Internet, molts d'aquests equips es troben situats darrera de tallafocs o encaminadors amb prestacions **NAT**. El problema que es planteja és el següent: com pot un ***peer*** descobrir-ne un altre si un d'ells –o ambdós– es troba situat darrera un dels elements esmentats i no es coneix –perquè no és pública– la seva identitat?
- Han existit –com era el cas de **Napster**– o existeixen –**SETI@home**– sistemes o aplicacions que treballen en entorns P2P, que solucionen aquest problema emprant un servidor central, al què s'adrecen els altres equips actuant com a clients per tal de descobrir-se mútuament.
- Altres sistemes com **Gnutella** utilitzen un model descentralitzat –anomenat P2P pur–, però el protocol emprat és exclusiu del sistema.
- Com ja s'ha citat en paràgrafs anteriors, la plataforma JXTA especifica un conjunt de protocols no propietaris que hom pot emprar en les comunicacions a través de xarxes P2P. Aquest conjunt de protocols –a més d'una sèrie d'equips, anomenats **relays** i **rendezvous**, que actuen com a intermediaris– fa possible la descoberta dels equips connectats encara que es trobin darrera tallafocs o encaminadors NAT.

Una vegada analitzada la informació obtinguda, tenint presents els objectius plantejats –sobretot que tot s'ha de concretar obtenint un producte de programari–, l'enfocament donat al treball es pot resumir en els següents punts:

- En l'aplicació, cal desenvolupar tant una part servidora com una part client. En aquest sentit, com a primera idea de disseny, es considera que en posar en marxa l'aplicació s'activarà –si no s'ha configurat el contrari– la part servidora, de forma que els clients que vulguin iniciar la conversa puguin connectar-s'hi.
- Emprar la plataforma JXTA com a base en els processos de comunicació ja que, d'acord amb les especificacions, facilita la comunicació entre els distints equips sigui quina sigui la seva situació.
- L'aplicació haurà de disposar d'un gestor de persistència per tal de gestionar les llistes de contactes.
- També s'hauran de crear les classes d'utilitat necessàries per a fer possible la generació i l'intercanvi de claus, així com el processos per a xifrar i desxifrar els missatges.

La metodologia seguida en el desenvolupament ha estat la clàssica: *recollir requeriments*, *analitzar*, *dissenyar*, *implementar* i *provar*, efectuant aquestes tasques de forma seqüencial. La línia de treball seguida es concreta en la planificació que es comenta tot seguit.

1.4.- Planificació Temporal.

Al final d'aquest apartat (figures 1 i 2) es mostra un diagrama de Gantt amb la planificació inicial del projecte. Aquesta va realitzar tenint presents, entre altres qüestions, les proves d'avaluació continuada –el Pla de Treball i les dues PAC– a lliurar en el transcurs del semestre. Coincidint amb les dates de lliurament es van marcar fites de control (veure Taula 2). Aquesta planificació s'ha complert de forma prou satisfactòria al llarg del semestre.

Gairebé totes les feines s'han desenvolupat una rere l'altra. Només ha existit un cert paral·lelisme en el què es refereix a l'elaboració del Pla de Treball i a la cerca d'informació, així com en l'elaboració de la documentació final. És lògic suposar que al final es presenti un cert encavalcament de les darreres tasques documentals amb les proves finals del programari.

La relació de les tasques principals desenvolupades és la següent:

- **Pla de Treball:**

  En aquesta fase s'ha elaborat el document en què s'ha establert l'abast del projecte, els requisits funcionals i tècnics, així com la temporalització. S'ha considerat com a data inicial el 6 de març, i com a data final la de lliurament del pla.

- **Cerca d'informació:**

  Es tractava de cercar i recollir informació sobre els distints conceptes i/o eines a utilitzar en el transcurs del treball:

  - Mètodes i algorismes emprats en criptografia per a crear claus, intercanviar-les, xifrar,...
  - Arquitectura i característiques de les xarxes P2P.
  - Eines de programació en Java per a treballar els temes de comunicació i de criptografia.
  - Possibles marcs de treball que puguin servir de base al projecte.
  - Aplicacions existents de xat.

  Part d'aquesta fase es realitzà en paral·lel amb l'anterior, ja que havia de servir per a escollir i començar a centrar el treball.

- **Estudi i Anàlisi de la informació:**

  Una vegada acabada la fase de cerca, calia analitzar la informació recollida per tal d'acabar de definir el treball a realitzar i les eines a emprar en el desenvolupament.

- **Anàlisi i Disseny**

  Aquesta fase es va pensar amb una durada de gairebé tres setmanes dividida en dues parts:

  - L'anàlisi i disseny del programa a nivell de processos.
  - El disseny de la interfície gràfica.

  El final de la primera coincidia amb el lliurament de la PAC1. Tot i intentar que aquest fet coincidís amb el final de tota la tasca, va semblar més realista mantenir la planificació que es presenta. Com s'ha esmentat anteriorment, es va marcar el lliurament de la prova com a fita de control.

- **Implementació:**

  Fase, amb una durada de 5 setmanes, que com l'anterior, es va dividir en dues parts:

  - La programació dels processos i les proves unitàries de cadascun d'ells.
  - La implementació de la interfície gràfica.

  El final de la primera part coincidí amb el lliurament de la PAC2.

- **Proves d'integració:**

  L'objectiu de la fase era acabar de depurar l'aplicació i preparar-la per al lliurament final. Havia d'estar acabada el 24 de maig per tal de dedicar les darreres setmanes a l'elaboració de la memòria i la presentació.

- **Elaboració de la memòria:**

  S'han dedicat unes tres setmanes a l'elaboració de la documentació final, tot i que en la primera setmana ha existit encavalcament amb les proves finals. La idea fou que quedessin tres setmanes dedicades plenament a la memòria i la presentació.

- **Elaboració de la presentació:**

  Aquesta tasca coincideix amb l'anterior. El dia 15 de juny s'ha de lliurar el producte final juntament amb la memòria i la presentació.

- **Debat:**

  Fase que s'inicia una vegada s'ha lliurat el projecte. Se li ha assignat una durada d'uns quinze dies. És la darrera fase del projecte, prèvia al seu tancament.

1.4.1.- Tasques i precedències.

La taula següent mostra el detall de les distintes tasques així com les dependències entre elles.

Codi Activitat	Nom	Precedències
01	**Inici del TFC**	
02	**Elaboració del Pla de Treball**	01
03	**Cerca d'informació**	
03.01	Intercanvi de claus de Diffie-Helman	01
03.02	Xifratge simètric	03.01
03.03	Xarxes P2P	03.02
03.04	Plataforma JXTA i Java	03.02
04	**Estudi i anàlisi de la informació**	03
05	**Anàlisi i disseny**	
05.01	Processos	04
05.02	Interfície gràfica	05.02
06	**Implementació**	
06.01	Programació dels processos i proves unitàries	05
06.02	Implementació de la interfície gràfica	06.01
07	Proves de integració	06
08	**Elaboració de la memòria**	
08.01	Objectiu del projecte, estat de l'art i recursos	07
08.02	Dissenys i implementació del programa	08.01
08.03	Manual d'usuari, proves i conclusions	08.02
09	**Elaboració de la presentació**	07
10	**Presentació del Projecte i Debat**	08, 09
11	**Fi del Projecte**	10

Taula 1 - Tasques i precedències

1.4.2.- Diagrama de Gantt.

La figura adjunta mostra la planificació que es va proposar per al projecte. Per claredat, no s'han inclòs en el diagrama les fites de control.

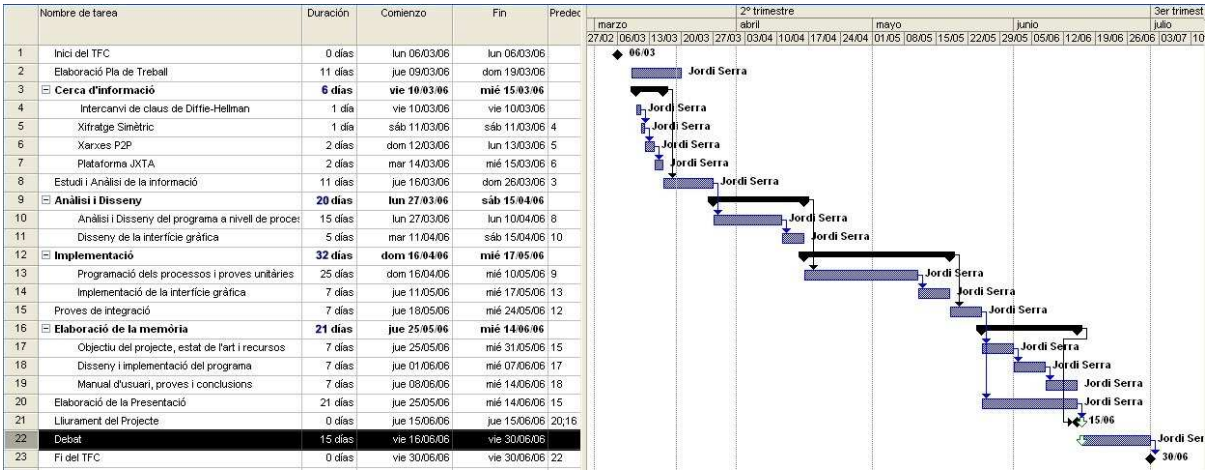

Fig. 1 - Diagrama de Gantt

En la següent figura es pot comprovar millor la planificació temporal i l'assignació de recursos.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
1	Inici del TFC	0 días	lun 06/03/06	lun 06/03/06		
2	Elaboració Pla de Treball	11 días	jue 09/03/06	dom 19/03/06		Jordi Serra
3	**Cerca d'informació**	**6 días**	**vie 10/03/06**	**mié 15/03/06**		
4	Intercanvi de claus de Diffie-Hellman	1 día	vie 10/03/06	vie 10/03/06		Jordi Serra
5	Xifratge Simètric	1 día	sáb 11/03/06	sáb 11/03/06	4	Jordi Serra
6	Xarxes P2P	2 días	dom 12/03/06	lun 13/03/06	5	Jordi Serra
7	Plataforma JXTA	2 días	mar 14/03/06	mié 15/03/06	6	Jordi Serra
8	Estudi i Anàlisi de la informació	11 días	jue 16/03/06	dom 26/03/06	3	Jordi Serra
9	**Anàlisi i Disseny**	**20 días**	**lun 27/03/06**	**sáb 15/04/06**		
10	Anàlisi i Disseny del programa a nivell de proces	15 días	lun 27/03/06	lun 10/04/06	8	Jordi Serra
11	Disseny de la interfície gràfica	5 días	mar 11/04/06	sáb 15/04/06	10	Jordi Serra
12	**Implementació**	**32 días**	**dom 16/04/06**	**mié 17/05/06**		
13	Programació dels processos i proves unitàries	25 días	dom 16/04/06	mié 10/05/06	9	Jordi Serra
14	Implementació de la interfície gràfica	7 días	jue 11/05/06	mié 17/05/06	13	Jordi Serra
15	Proves de integració	7 días	jue 18/05/06	mié 24/05/06	12	Jordi Serra
16	**Elaboració de la memòria**	**21 días**	**jue 25/05/06**	**mié 14/06/06**		
17	Objectiu del projecte, estat de l'art i recursos	7 días	jue 25/05/06	mié 31/05/06	15	Jordi Serra
18	Disseny i implementació del programa	7 días	jue 01/06/06	mié 07/06/06	17	Jordi Serra
19	Manual d'usuari, proves i conclusions	7 días	jue 08/06/06	mié 14/06/06	18	Jordi Serra
20	Elaboració de la Presentació	21 días	jue 25/05/06	mié 14/06/06	15	Jordi Serra
21	Lliurament del Projecte	0 días	jue 15/06/06	jue 15/06/06	20;16	
22	Debat	15 días	vie 16/06/06	vie 30/06/06		Jordi Serra
23	Fi del TFC	0 días	vie 30/06/06	vie 30/06/06	22	

Fig. 2 - Relació de Tasques

1.4.3.- Fites de Control.

Finalment, les fites de control previstes –gairebé totes ja realitzades– es recullen en la taula següent.

Data	Fita de Control	Participants
06-03-2006	Inici del projecte	Consultor i alumne
20-03-2006	Lliurament Pla de Treball	Consultor i alumne
10-04-2006	PAC1	Consultor i alumne
10-05-2006	PAC2	Consultor i alumne
05-06-2006	Lliurament pre-memòria	Consultor i alumne
15-06-2006	Lliurament del projecte	Consultor i alumne
30-06-2006	Final del projecte	Consultor i alumne

Taula 2 - Fites de Control

1.5.- Productes obtinguts.

A més del present document i de la presentació que l'acompanya, el resultat del TFC és un producte de programari, l'aplicació ***JXTA Xat Segur***. Com ja s'ha indicat, és una aplicació de xat en què els missatges viatgen xifrats; l'usuari pot gestionar el seu estat de connexió; administrar els seus contactes, i mantenir diverses converses alhora, amb la possibilitat de varis participants en la mateixa conversa. A més, la interacció amb l'aplicació es realitza de forma gràfica en un típic entorn de finestres.

Juntament amb el programa en format JAR, contingut dins la carpeta **dist** en el fitxer **JxtaXatSegur.zip**, es lliuren les fonts i la documentació de les classes: carpetes **src** i **docs**. El codi font es lliura també en format PDF: fitxer **jserrapu_CodiPrograma**.

1.6.- A partir d'ara.

Els capítols que segueixen constitueixen la raó de ser d'aquest document. El segon capítol tracta dels conceptes que fonamenten el producte desenvolupat. Tot i que amb una extensió reduïda per tal de mantenir el document en els límits demanats, es tracten les qüestions criptogràfiques, què són les xarxes P2P i s'explica l'estructura bàsica de la plataforma JXTA.

El capítol 3 parla del disseny de l'aplicació: introdueix quins han estat els principis del disseny, esmenta com s'ha dissenyat el programa a nivell de processos i acaba comentant les bases del disseny de la interfície gràfica. El

capítol següent tracta dels aspectes més destacables de la implementació realitzada: com s'han programat els temes relacionats amb la criptografia, com s'ha implementat la persistència, el control d'usuaris i sessions i, finalment, la interfície d'usuari.

El capítol 5 està dedicat als manuals d'usuari. Explica com instal·lar i executar el programa, i incorpora un petit manual d'ús de l'aplicació. El capítol 6 mostra una sèrie de proves realitzades –a més de les que es poden observar en el manual d'usuari del capítol anterior. Finalment, abans d'un breu glossari i de la bibliografia emprada, el capítol 7 presenta les conclusions i els comentaris finals. El document acaba amb un annex que conté el codi implementat.

2.- Fonaments.

2.1.- Criptografia.

2.1.1.- Introducció.

Quan dos usuaris **A** i **B** utilitzen un canal de comunicació públic i insegur com és Internet per a conversar –com és el cas del problema que ens ocupa–, el seu diàleg està sotmès a tot un conjunt de problemes de seguretat, entre els quals podem citar:

- Altres usuaris malintencionats, com un tercer usuari **C**, poden:
  - **Interceptar** la conversa, sense alterar-la, amb l'ànim només d'escoltar-la.
  - **Interrompre** la comunicació.
  - Interceptar el diàleg amb l'intenció de **modificar** el contingut.
  - Prendre el paper d'algun dels usuaris i iniciar un diàleg amb l'altre fent-se passar pel primer, fet que es coneix amb el nom de **impostura**.
- Un dels usuaris, **A** per exemple, pot enviar un missatge a **B** i posteriorment negar-ho, produint-se el que anomenem **repudi**.
- Qualsevol dels usuaris pot rebre un missatge del seu interlocutor i negar-ho posteriorment, la qual cosa es coneix amb el nom de **negació de recepció**.

Per tal d'evitar aquests problemes, els sistemes de comunicacions han de proporcionar mecanismes de seguretat, de forma que es pugui assegurar:

- La **confidencialitat**, és a dir, que el missatge només el puguin conèixer els interessats.
- La **integritat**, cosa que garanteix que el missatge no ha estat alterat.
- L'**autenticitat** que permet evitar que es suplanti la identitat de qualsevol dels interlocutors.
- El **no rebuig**, per a garantir que no es pugui negar ni l'enviament ni la recepció del missatge.

La criptografia ajuda als sistemes de comunicacions en la implementació de mecanismes de seguretat. Per tal de garantir la confidencialitat de les dades, aquestes es poden transmetre codificades després de ser sotmeses a un procés de **xifratge**, mitjançant el qual se'ls hi aplica una sèrie de transformacions –basades en un algorisme conegut– amb l'ajut de l'anomenada **clau**. Els mètodes criptogràfics garanteixen –fins a cert punt– que encara que es conegui l'algorisme de xifra, no es pot obtenir el missatge en clar a partir del xifrat si no es coneix la clau emprada.

Existeixen actualment dos sistemes criptogràfics: l'anomenada **criptografia de clau simètrica** i la que rep el nom de **criptografia de clau pública o asimètrica**. Ambdós sistemes presenten les següents característiques:

- Tal com es pot observar en la figura adjunta, en la **criptografia de clau simètrica**, es fa servir la mateixa clau –**k**– per a xifrar i desxifrar les dades (també es pot usar per a desxifrar una clau obtinguda a partir de l'utilitzada per a xifrar). L'algorisme de xifratge especifica les funcions a emprar en el processos per a xifrar –**E**– i desxifrar –**D**–.

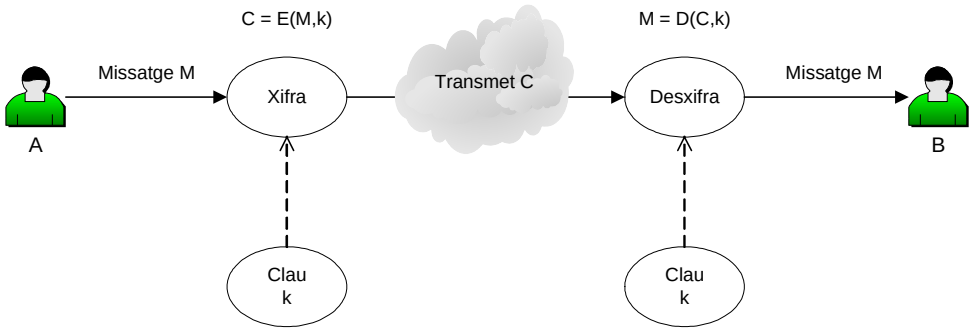

Fig. 3 - Criptografia de clau simètrica. Confidencialitat.

La criptografia de clau simètrica permet garantir la confidencialitat en la comunicació, utilitza algorismes que normalment són senzills i ràpids, però presenta un desavantatge: els interlocutors s'han de posar d'acord amb la clau a utilitzar, la qual cosa és un veritable problema quan s'han de comunicar a través d'un mitjà no del tot segur.

També es pot emprar la criptografia de clau simètrica per a proporcionar serveis d'autenticitat i integritat. En aquests casos, si no es desitja confidencialitat en la comunicació, s'utilitzen algorismes basats en **funcions hash segures** –anomenades també **funcions de resum de missatge**. Aquestes funcions proporcionen un resum **H** calculat a partir del missatge a transmetre **M**, **H=h(M)** , i compleixen dues condicions:

- Per una banda, són unidireccionals, és a dir, és gairebé impossible –es diu que és computacionalment molt difícil– obtenir **M** a partir de **H**.
- Per altra banda, és també computacionalment molt difícil obtenir un missatge **M'≠M** tal que **h(M')=h(M)**. Aquest fet s'expressa dient que la funció és resistent a les col·lisions.

Per tal d'assegurar l'autenticitat i/o la integritat d'un missatge es pot emprar la següent tècnica[2]: calcular el seu resum, xifrar-lo amb la clau compartida i transmetre el missatge conjuntament amb el resum xifrat.

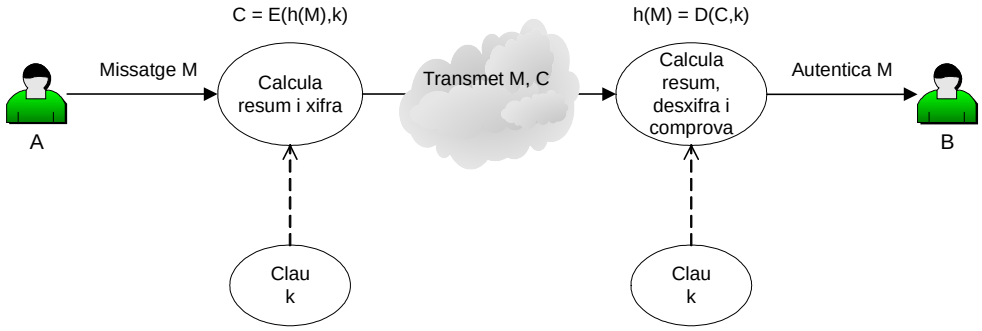

Fig. 4 - Criptografia de clau simètrica. Autenticació i integritat.

L'altra part desxifrarà amb l'ajut de la clau coneguda el resum i comprovarà, després de calcular també el resum del missatge, que aquest no ha estat alterat i que prové de la font autèntica.

o En la **criptografia asimètrica** es treballa amb dues claus, una emprada per a xifrar i l'altra per a desxifrar. Cada usuari disposa d'una **clau privada** que només coneix ell i d'una **clau pública**, calculada a partir de l'anterior, que com indica el seu nom pot donar a conèixer públicament. El sistema proporciona seguretat sempre i quan sigui impossible –és a dir, computacionalment molt difícil– conèixer la primera a partir de la segona.

Aquest sistema garanteix la confidencialitat si s'utilitza tal com mostra la figura adjunta: es xifra amb la clau pública de l'usuari al què s'adreça el missatge, la qual cosa assegura que només ell el podrà desxifrar amb la seva clau privada.

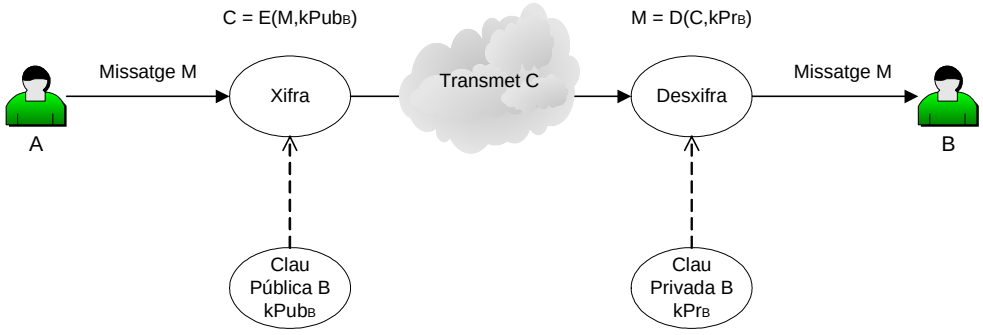

Fig. 5 - Criptografia de clau pública. Confidencialitat.

L'avantatge d'aquest tipus de criptografia és que no requereix l'intercanvi de claus secretes, ja que els usuaris fan públic un dels components de la seva clau. No obstant, presenta un inconvenient: els algorismes són més complexos i, per tant, més lents. Un altra qüestió a resoldre en aquest sistema és com saber si una clau que es presenta pertany efectivament a un determinat usuari. Respondre amb propietat a aquesta darrera qüestió depassa l'àmbit d'aquest

[2] Es pot trobar informació respecte altres tècniques a l'obra –referenciada en la bibliografia– J.Herrera, J.García i X.Perramon, *Seguretat en Xarxes de Computadors*, Mòdul 3, pàg. 19.

treball. Indicarem, només, que la resposta la proporciona la **infraestructura de clau pública** a través dels anomenats **certificats de clau pública**[3].

A causa de la lentitud esmentada, la criptografia de clau pública sol emprar-se, en la pràctica, per a realitzar processos d'intercanvi de claus, autenticació i no repudi. Per a efectuar un procés d'autenticació amb aquest tipus de criptografia, es pot actuar en la forma següent: l'usuari que vol autenticar un missatge, el xifra –el missatge o un resum– amb la seva clau privada, la qual cosa s'expressa dient que signa el missatge; posteriorment transmet el missatge i la signatura, que s'anomena **signatura digital**.

El receptor, que disposa de la clau pública del signant, pot autenticar l'origen del missatge ja que:

- Pot desxifrar la part rebuda que venia xifrada.
- Pot comprovar que el què obté coincideix amb el missatge o el seu resum.
- Sap que només qui disposa de la clau privada pot haver xifrat el contingut

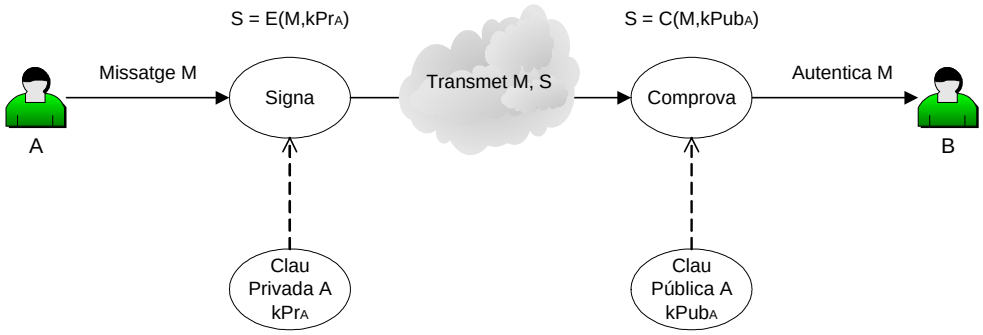

Fig. 6 - Criptografia de clau asimètrica. Autenticació.

La figura adjunta il·lustra aquesta qüestió. Naturalment, els dos processos comentats es poden produir alhora, amb la qual cosa el sistema ofereix tant servei de confidencialitat com d'autenticació i integritat.

2.1.2.- Intercanvi de claus de Diffie-Hellman.

S'ha esmentat en el subapartat anterior que un dels problemes de la criptografia de clau simètrica era el de la distribució de la clau compartida. Per tal de fer-ho, o bé els interlocutors s'han de trobar personalment, o bé han d'emprar un canal segur, amb la qual cosa en algun moment hom es pot preguntar que perquè cal compartir una clau per a poder codificar un missatge secret, si emprant el canal segur es podria transmetre el secret compartit.

Deixant de banda aquesta darrera qüestió, el problema de l'intercanvi de claus a través d'un canal insegur el van resoldre Whitfield Diffie i Martin E. Hellman l'any 1976 amb la publicació de l'article "New directions in cryptography", obrint alhora la porta a la criptografia de clau pública. L'algorisme descrit en aquest article permet a dos usuaris intercanviar una clau secreta, sense necessitat de trobar-se, emprant un canal insegur. Es basa en la seqüència d'esdeveniments que es comenta tot seguit:

- S'escull públicament –per part d'ambdós interlocutors o a proposta d'un d'ells– un nombre primer molt gran **p** i un enter **g**, més petit que el número anterior, generador del grup multiplicatiu $\mathbb{Z}_p$, és a dir, que compleix:

  $$\forall\ 1 \le n \le p-1,\ \exists\ k\ /\ n = g^k \bmod p$$

  En el cas de l'aplicació de xat segur, quan un *peer* –actuant com a client– contacti amb un altre *peer* que ha posat en marxa la seva part servidora, li transmetrà aquest dos números –**p** i **g**– en el primer dels missatges que li adreçarà.

- Tot seguit, cada interlocutor genera un valor aleatori privat **x**, calcula dins el grup $g^x \bmod p$ i transmet públicament aquest resultat a l'altra part.

  Aplicant aquest fet al cas que ens ocupa, si el client genera el nombre **a**, transmet al servidor el valor $g^a \bmod p$, la qual cosa farà incloent també aquesta dada en el missatge inicial.

  En rebre aquest missatge, el servidor coneixerà **p** i **g**, generarà aleatòriament un número **b** i retornarà al client un missatge de resposta que contindrà $g^b \bmod p$.

[3] Es pot trobar informació a l'obra –referenciada en la bibliografia– J.Domingo, J.Herrera i H.Rifà, *Criptografia*, Mòdul 7.

- Quan els dos interlocutors han rebut la resposta de l'altra part calculen, privadament, **$val\_public^{val\_privat}$ mod p**.

  El resultat d'aquest càlcul proporciona la clau secreta que comparteixen.

  Efectivament, si apliquem aquest fet al nostre model client-servidor, el client calcularà:

  $$(g^b)^a \bmod p = g^{ba} \bmod p$$

  Mentre que el servidor farà:

  $$(g^a)^b \bmod p = g^{ab} \bmod p$$

Observem que ambdues parts han obtingut el mateix resultat: $g^{ab} \bmod p$. Tot i que el valors que permeten arribar al resultat són públics –**p**, **g**, $g^a \bmod p$, $g^b \bmod p$–, el resultat final només el poden conèixer els dos interlocutors ja que són els únics que saben els valors privats **a** i **b**. Aquest valor constitueix la clau secreta compartida. La figura il·lustra l'intercanvi.

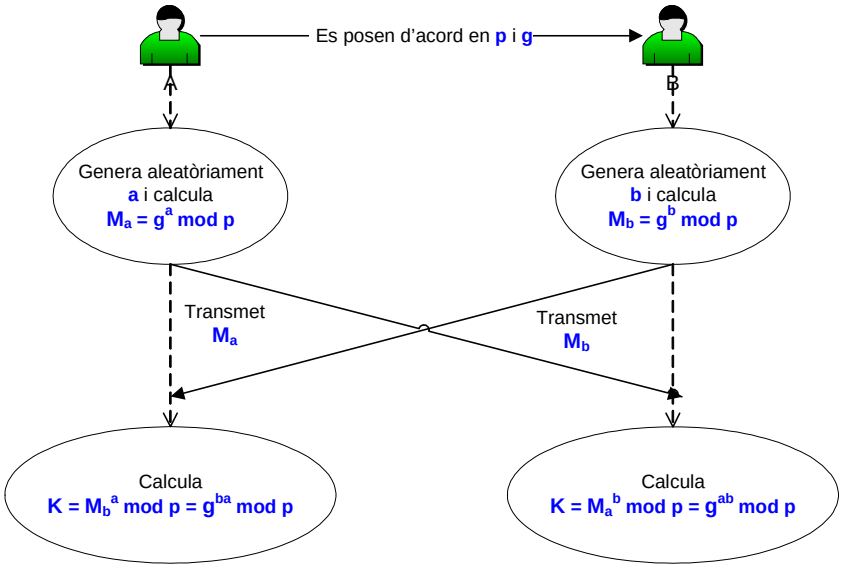

Fig. 7 - Intercanvi de claus de Diffie-Hellman

La seguretat del mètode rau en la dificultat –computacionalment molt difícil– de poder conèixer quan **p** és un número primer de moltes xifres el número g^{ab} (sempre en aritmètica mòdul **p**), tot i saber –en ser públics– els valors $\mathbb{Z}_p$, **p**, **g**, g^a i g^b. Aquest problema, anomenat problema de Diffie-Hellman generalitzat, és molt semblant al problema del logaritme discret[4] generalitzat, que es pot formular de la forma següent:

Es desitja saber el valor de **a** coneixent $\mathbb{Z}_p$, **p**, **g**, i g^a.

No s'ha demostrat l'equivalència entre ambdós problemes, però no s'ha trobat la forma de resoldre el problema de Diffie-Hellman sense resoldre el del logaritme discret. Si es pogués resoldre aquest, es podria resoldre el de Diffie-Hellman, ja que si a partir de g^a podem saber **a**, es podria –de forma anàloga– trobar també **b** i, per tant, g^{ab}. En ser el problema del logaritme discret un problema NP-difícil, es considera que també ho és el de Diffie-Hellman.

L'intercanvi de claus de Diffie-Hellman és vulnerable a l'atac anomenat **d'home al mig** *–man-in-the-middle atack–*. La figura adjunta el representa.

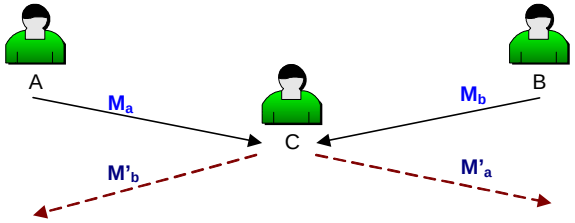

Fig. 8 - Atac a l'intercanvi de claus de Diffie-Hellman

[4] Hi ha informació sobre el problema del logaritme discret a J.Domingo, J.Herrera i H.Rifà, *Criptografia*, Mòdul 5, pàg. 17.

En aquest atac, un altre usuari **C** intercepta el component públic de la clau de **A** (**M_a**) i el substitueix per un component propi (**M'_a**). El mateix fa amb l'altre usuari **B**. Així aconsegueix tenir una clau compartida amb cadascun dels usuaris, amb la qual cosa pot interceptar –i manipular si ho desitja– el missatges que **A** i **B** s'adrecen a través d'un canal que creuen, ignorants de la situació, protegit.

Aquesta vulnerabilitat del protocol d'intercanvi és deguda a que els participants no s'autentiquen. Per tant, una possible prevenció davant l'atac és la utilització de signatures digitals per tal d'autenticar els interlocutors[5].

2.1.3.- Algorismes de xifratge simètric: DES, Triple DES i AES.

Encara que a vegades es creu així, Diffie-Hellman no és un algorisme de xifratge sinó un sistema que fa possible –gràcies a l'intercanvi– generar la clau que posteriorment s'emprarà en els processos de xifratge i desxifratge. Tot i que la clau disponible al final de l'intercanvi –comuna per ambdós interlocutors– podria emprarse per a xifrar, la forma en què matemàticament s'ha obtingut fa que a vegades se la qualifiqui de clau asimètrica, la qual cosa comporta lentitud en els processos de xifratge i desxifratge.

Això fa que sovint es prefereixi generar al final del procés d'intercanvi una clau simètrica que és la que s'acabarà utilitzant en la comunicació. Aquest és el cas de l'aplicació ***JXTA Xat Segur***. Aquesta clau la genera cada participant a partir de la seva clau privada –el valor aleatori **x** que ell ha elegit–, del component públic del seu interlocutor –**M_x**– i ho fa per a l' algorisme de xifratge simètric que es pensa emprar en la comunicació: **DES**, **Triple DES**, **IDEA**, **AES**,...

No és l'intenció de l'autor del treball tractar abastament de les xifres de clau compartida. Hi ha informació amb escreix en les referències bibliogràfiques sobre criptografia que es detallen al final del treball[6]. En els següents paràgrafs es comenten els aspectes que es creuen més interessants des d'un punt de vista pràctic.

2.1.3.1.- DES.

El criptosistema DES –*Data Encryption Standard*– és un sistema de xifratge simètric de bloc que xifra dades especejant-les en blocs de 64 bit i utilitzant una clau de 56 bit. En el desenvolupament de l'algorisme hi va participar activament IBM –de fet l'algorisme és una millora de l'algorisme Lucifer desenvolupat per IBM a principis dels 70–, i en les etapes finals del procés la *National Security Agency* (NSA) i el que actualment és el *National Institute of Standards and Technology* (NIST).

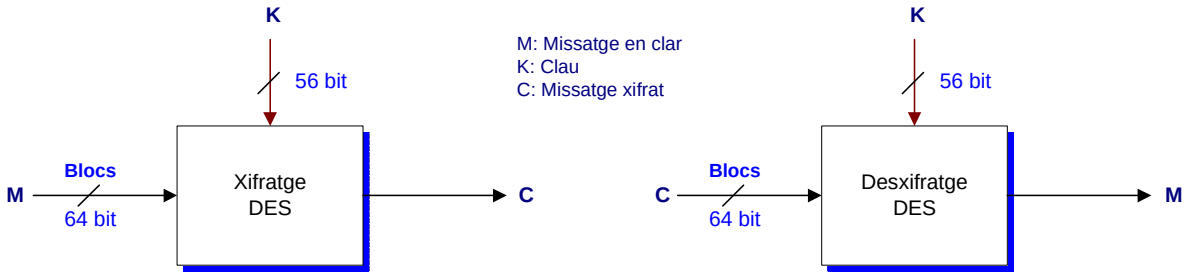

Fig. 9 - Xifra DES

Dissenyat originalment per ser implementat en maquinari, DES pot treballar amb tots els modes de xifratge de blocs –**CBC**, **CFB** i **OFB**[7]–. Es considera que DES presenta els següents inconvenients:

- La clau té poca allargada, la qual cosa el fa vulnerable als **atacs per força bruta**. Fa uns quants anys –finals del 70, principis dels 80–, Diffie i Hellman havien previst que una màquina amb una arquitectura de processadors paral·lels pogués arribar a trencar el DES a través d'un atac d'aquest tipus.

  Darrerament, el supercomputador DES Cracker, amb l'ajuda de 100.000 ordinadors personals, aconseguí trencar DES en 22 hores i pocs minuts.

- En el transcurs de l'execució de l'algorisme associat a DES, es fan operacions de substitució en què intervenen les anomenades caixes S –a cada caixa se li entra un bloc de 6 bit i en retorna un de 4–. Hi ha qui creu que la forma en què esdevé la substitució és arbitrària i podria estar relacionada amb l'existència d'una clau mestre que permetés desxifrar qualsevol missatge.

---

[5] Es pot trobar més informació sobre l'intercanvi de Diffie-Hellman a A.Menezes, P.Van Oorschot i S.Vanstone, *Handbook of applied cryptography*, pàg. 515-516, 519-520.

[6,7] Entre d'altres referències, es pot consultar sobre les xifres de bloc a J.Domingo, J.Herrera i H.Rifà, *Criptografia*, Mòdul 4, i a A.Menezes, P.Van Oorschot i S.Vanstone, *Handbook of applied cryptography*, Capítol 7.

Per tot això, es considera que DES no és prou segur (el govern dels Estats Units no permet el seu ús) i es recomana, mentre no arribi el nou estàndard previst –AES– emprar la variant de DES que utilitza el xifratge triple: Triple-DES.

2.1.3.2.- Triple-DES.

Amb l'objectiu de dificultar l'atac per força bruta, es pot augmentar l'espai de la clau efectuant diferents xifratges de bloc en cadena, de forma que es faci servir una clau distinta en cada bloc. Aquesta tècnica –anomenada xifratge múltiple– necessita que el criptosistema que s'utilitza no constitueixi un grup, és a dir, que donades dues claus qualsevol, K_1 i K_2, no n'existeixi una tercera K que compleixi:

$$E(M,K) = E(E(M, K_1), K_2)$$

Si existís aquesta clau, xifrar aplicant el xifratge múltiple donaria el mateix resultat que fer-ho de forma simple. DES compleix aquest requeriment, per la qual cosa es pot emprar la tècnica del xifratge múltiple per tal d'incrementar la seguretat del criptosistema.

En la pràctica, es fa servir el xifratge triple per evitar l'atac anomenat del punt intermedi[8]. La figura adjunta mostra l'esquema de xifratge que s'utilitza per al criptosistema DES que rep, en aquest cas, el nom de Triple-DES. Sovint també se l'anomena DES-EDE ja que, com es pot comprovar, primer es xifra (E), després es desxifra (D) i, finalment, es torna a xifrar (E).

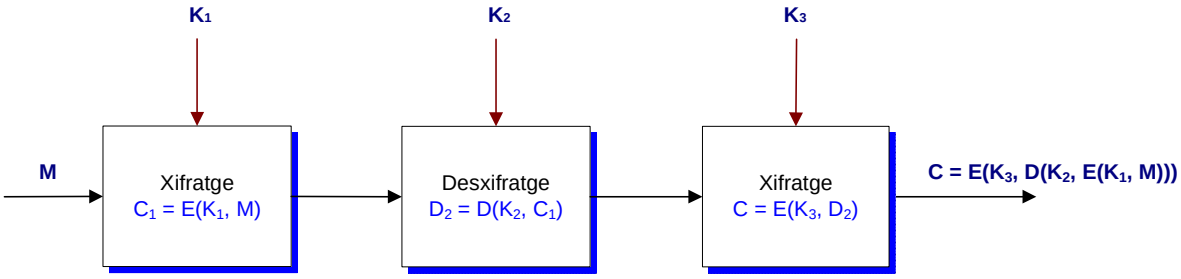

Fig. 10 - Xifratge triple

Tot i que en la figura apareixen claus distintes en cada bloc –cosa que representa treballar amb una clau de 168 bit (56x3)–, l'estàndard definit en ANSI X9.52 contempla també la possibilitat que K_1 i K_3 siguin iguals, resultant llavors una clau de 112 bit (56x2).

2.1.3.2.- AES.

Tenint en compte els problemes de seguretat del criptosistema DES i, d'altra banda, la relativa ineficiència de la implementació en programari del Triple-DES, l'any 1997 el NIST va convidar a la comunitat criptogràfica a fer propostes per a un nou estàndard anomenat *Advanced Encryption Standard* –AES–.

Es va especificar que el nou algorisme de xifratge de bloc hauria de permetre treballar amb blocs de 128 bit, amb els modes estàndard (CBC, CFB i OFB), i que la clau hauria de poder ser de longitud variable: 128, 192 o 256 bit. A més, calia que es pogués emprar en distints àmbits –governamentals, institucionals, empresarials i comercials– i que es millorés l'eficiència del Triple-DES.

L'octubre del 2001 es va seleccionar el criptosistema Rindjael[9] –inventat pels criptògrafs belgues Joan Daemen i Vincent Rijmen– com a AES, i el nou estàndard es va publicar el novembre del 2002.

2.2.- Les xarxes P2P.

2.2.1.- Introducció.

Segurament, la traducció més correcta de l'expressió anglesa *peer-to-peer* (P2P) és d'igual a igual. Mitjançant aquest terme, en el camp de la informàtica i les comunicacions, solem referir-nos a un model de xarxes descentralitzat –i distribuït– en el qual les aplicacions que s'executen en els equips connectats a la xarxa es poden comunicar directament entre si sense la intervenció d'un servidor central –qüestió aquesta última que no sempre és del tot certa.

El terme vol deixar clar que tots els nodes de la xarxa són iguals –fet que és cert amb matisos–, tal com queda reflectit en la definició que proporciona del terme l'organisme encarregat de la terminologia en la llengua

[8] En principi, el fet d'emprar dues claus independents portaria a pensar que l'espai de claus és igual a 2^{2n}. S'ha demostrat, però, que es pot trencar un criptosistema DES de dues etapes realitzant només 2^{n+1} iteracions. Aquest fet es deu a l'aplicació d'una tècnica de criptoanàlisi anomenada atac del punt intermedi. Veure J.Domingo, J.Herrera i H.Rifà, *Criptografia*, Mòdul 4, pàg. 37.

[9] Es pot trobar el detall de l'algorisme a J.Domingo, J.Herrera i H.Rifà, *Criptografia*, Mòdul 4, pàg. 25.

catalana, el TERMCAT: "Dit de la tecnologia d'intercanvi de fitxers entre processadors d'una xarxa que permet que es puguin comunicar directament entre si sense haver de passar per un servidor central que redistribueixi les dades".

Tal com s'ha esmentat en la introducció, una xarxa P2P és, bàsicament, una xarxa entre iguals que no té clients ni servidors fixos, sinó un conjunt de nodes que es comporten alhora com a clients i servidors dels altres elements de la xarxa. Aquest model, que sembla contrastar amb el model tradicionalment emprat en les comunicacions via Internet –client-servidor–, ha esdevingut molt popular en els darrers anys degut, entre altres raons, a l'increment del nombre d'ordinadors connectats a la xarxa, a la millora en les seves prestacions, al major ample de banda disponible, i a la proliferació de continguts i la disponibilitat de molts usuaris a compartir-los.

Tot i que hi ha qui considera que des dels principis d'Internet han existit aplicacions P2P –els serveis de notícies, el servei DNS, els protocols d'encaminament,...–, segurament un fet clau en el desenvolupament de les xarxes P2P fou l'aparició, l'any 1999, de Napster com a servei de distribució d'arxius de música en format MP3. Napster, no obstant, disposava d'ordinadors centrals per tal d'emmagatzemar les dades dels equips i dels recursos existents en cadascun d'ells, encara que posteriorment les transferències esdevenien entre els equips que es connectaven directament. Com bé es coneix, el Napster primitiu –que no es pot considerar del tot un sistema P2P pur– fou tancat per ordre judicial el setembre del 2001, i el seu successor opera en l'actualitat com a servei de pagament.

2.2.2.- Elements de les xarxes P2P.

L'element principal de les xarxes P2P és el *peer*[10]. Tot i que molt sovint es considera que un *peer* és una aplicació que s'executa en un ordinador connectat a Internet, la realitat, tal com mostra la figura adjunta[11], és que un *peer* pot ser qualsevol dispositiu amb capacitat de procés i de comunicació.

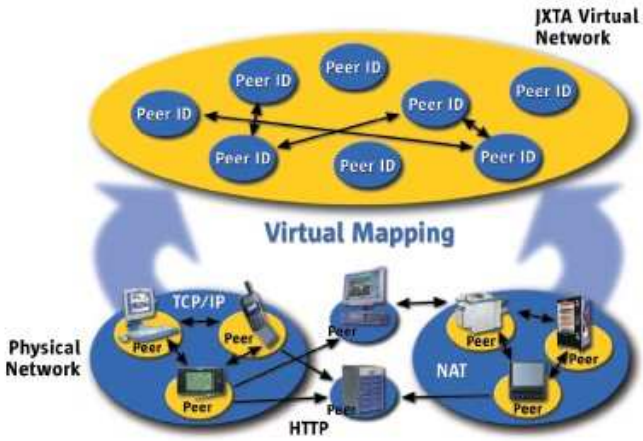

Fig. 11 - Dispositius en xarxes P2P

En una de les referències bibliogràfiques consultades[12] es defineix un *peer* –de forma prou encertada– en els següents termes: "Un *peer* és una entitat capaç de desenvolupar alguna tasca útil i de comunicar els resultats de l'execució a una altra entitat de la xarxa, bé sigui de forma directa o indirectament".

La mateixa font documental considera dos tipus de *peer* que, en el fons, coincideixen amb els tipus que hom pot trobar en la plataforma JXTA:

- El ***peer* senzill**, destinat a ser emprat pels usuaris finals, mitjançant el qual aquests ofereixen els seus serveis o fan ús dels serveis que ofereixen d'altres *peers*.

  En aquest tipus, es troben distintes classes de dispositius –ordinadors, mòbils, PDA,...– amb connexió intermitent a xarxa. A més, quan s'hi connecten, ho fan amb adreçament canviant i molt sovint es troben darrera tallafocs i/o encaminadors NAT.

- El ***superpeer***, la missió del qual és ajudar a què els altres peers es descobreixin, actuant com a ***rendezvous peer***, o bé fent possible la comunicació entre ells, prenent el paper de ***relay peer*** o ***router peer***.

[10] Encara que com ja s'ha esmentat, el terme català *d'igual a igual* defineix la tecnologia *P2P*, no s'ha trobat cap terme en llengua catalana que s'ajusti totalment al sentit del terme anglès *peer*, per la qual cosa es mantindrà l'original.

[11] Extreta de Sun Microsystems, *Software Datasheet / Project JXTA Technology*.

[12] R.J. Millán Tejedor, *Domine las redes P2P*, pàg. 234.

En el primer cas, el *rendezvous peer* manté una memòria cau amb les dades associades a les darreres demandes satisfetes, de forma que pot resoldre ràpidament les noves peticions que se li adrecen. Si no disposa de la resposta a una determinada demanda, la redirigeix a un altre *rendezvous*.

Els *relay peers* actuen com a encaminadors que permeten que els *peers* més senzills puguin comunicar-se encara que es trobin protegits per tallafocs o situats darrera dispositius NAT.

Moltes de les aplicacions P2P actualment vigents són tancades i els protocols de comunicació que utilitzen propis de l'aplicació. Des d'aquest punt de vista, es pot considerar que els *peers* que usen una d'aquestes aplicacions formen un grup tancat, que no es pot comunicar amb els *peers* que fan ús d'altres aplicacions. El concepte de grup de *peers*, però, té una altra accepció després de l'aparició d'estàndards com JXTA. En casos com aquest, els grups de *peers* –anomenats ***peer group***– es formen, bàsicament, per dos motius:

- l'agrupament permet compartimentar la xarxa
- el grup és l'àmbit en el què s'ofereixen serveis, és a dir, un *peer* pot disposar d'un servei associat a un grup –per exemple, un servei de xat– si pertany al grup.

La figura mostra els conceptes esmentats en els darrers paràgrafs aplicats a la plataforma JXTA.

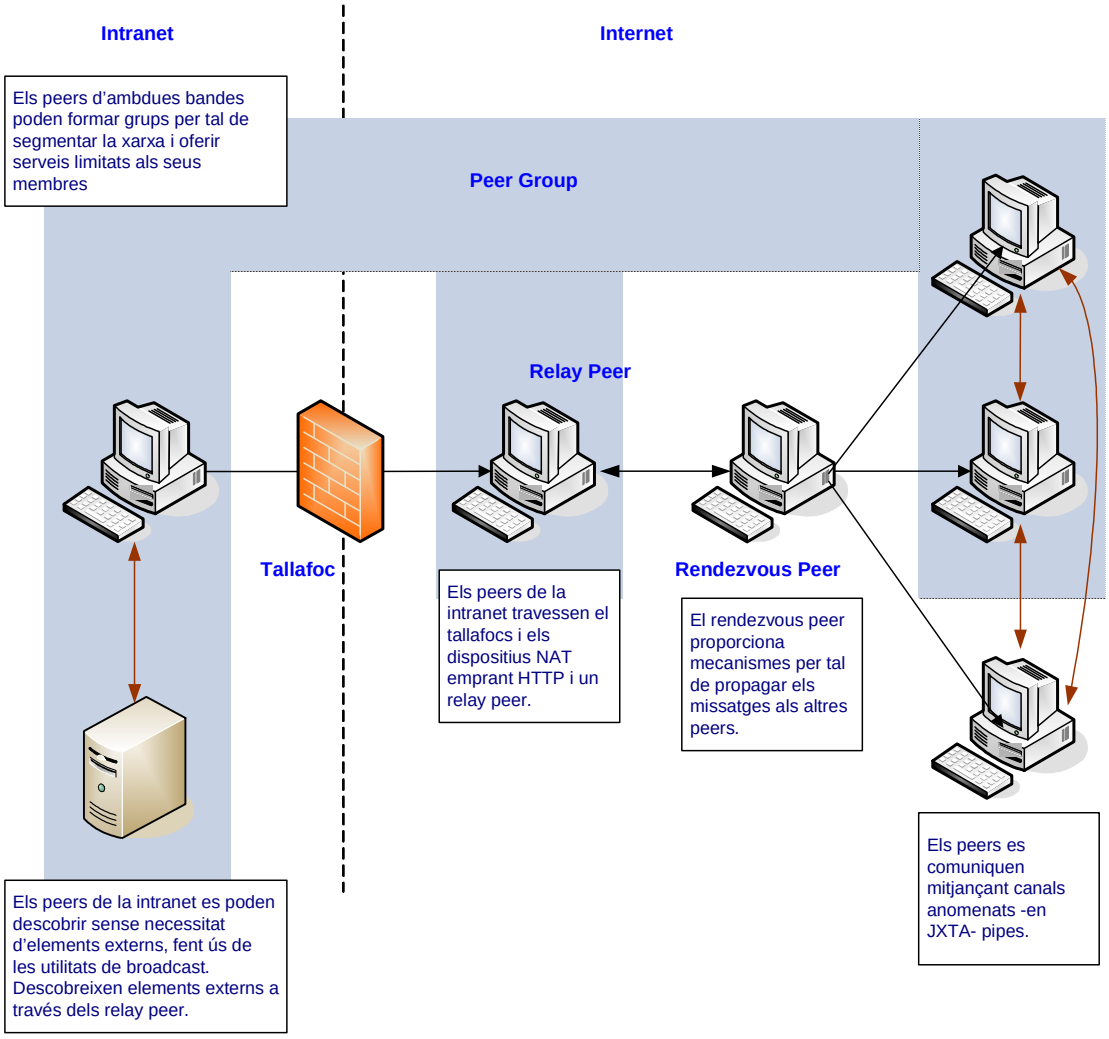

Fig. 12 - Elements d'una xarxa P2P.

2.2.3.- Arquitectura de les xarxes P2P.

Des d'un punt de vista jeràrquic, actualment es poden distingir les següents arquitectures en les xarxes P2P:

- El **model P2P pur**, en el qual no existeix cap servidor central. Aquest model, en el que tots els nodes són iguals, és l'utilitzat per aplicacions com Gnutella o Freenet.

És un sistema econòmic i tolerant a fallades, però pot esmerçar –comparat amb els altres– bastant de temps i recursos de xarxa a l'hora de cercar informació en el sistema, ja que qualsevol requeriment s'adreça a múltiples usuaris.

- El **model P2P híbrid** o **centralitzat**. En aquest cas, un *peer*, abans de contactar amb un altre, ho fa amb el servidor per tal de cercar dades respecte la identitat de l'altre *peer*, els serveis que ofereix o bé per qüestions de seguretat. Quan el *peer* té les dades del seu interlocutor, contacta amb ell directament.

  Aquest és el model seguit, entre d'altres, per Napster i Groove. Presenta un rendiment elevat en la localització de recursos, però és un sistema de cost elevat.

- El **model mixt** o basat en ***superpeers***. En aquesta arquitectura, existeixen equips –esmentats en l'apartat anterior– que contenen informació que els altres *peers* no tenen i de la qual aquests altres en poden disposar quan la necessiten. Aquest és el model utilitzat per Kazzaa i FastTrack, i el que segueix la plataforma JXTA.

  En aquest sistema, els *peers* normals mantenen connexions amb els seus *superpeers* –a més de les ja establertes amb els interlocutors descoberts. A l'hora de respondre als requeriments dels clients, el model presenta un bon rendiment i té, a més, dos avantatges: fa la xarxa escalable i és tolerant a fallades.

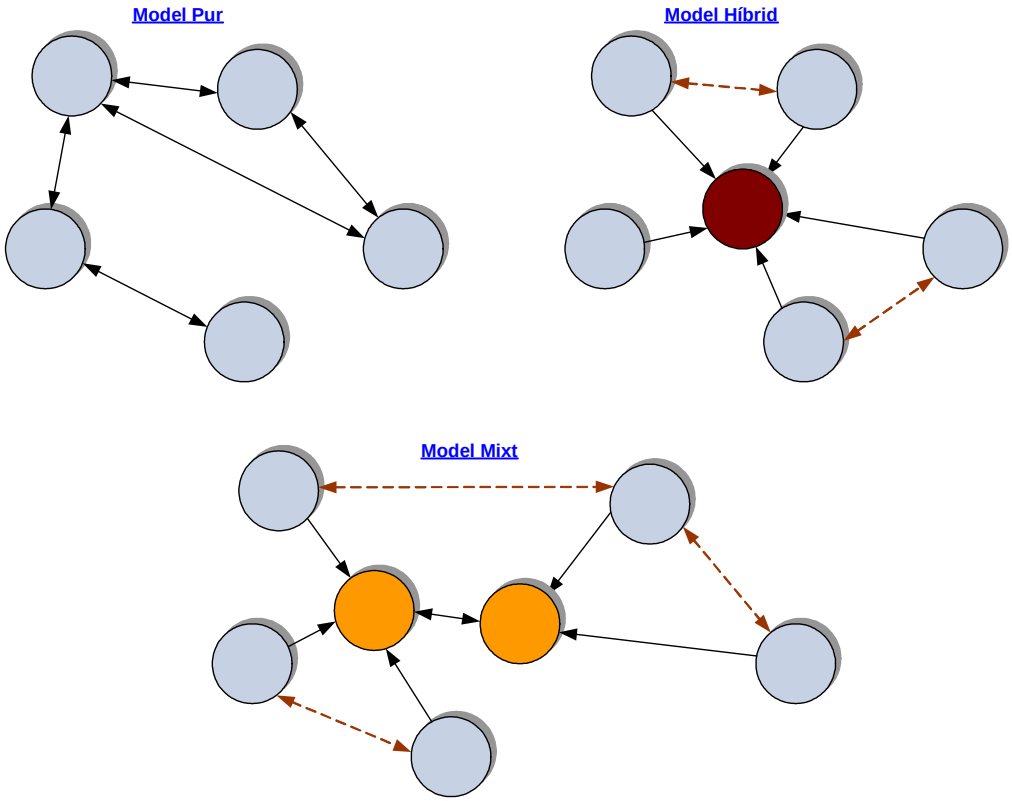

Fig. 13 - Models P2P

2.2.3.- Aplicacions P2P.

Per acabar aquest apartat dedicat als principis de les xarxes P2P, s'esmentaran els tipus d'aplicacions que usen el model P2P. Tot i que són molt diverses –i fan servir models distints–, totes tenen en comú un fet diferencial respecte el model client-servidor, en el que pocs equips –els servidors– ofereixen serveis a molts altres –els clients–: cada node és alhora client i servidor; proporciona serveis als altres, i gaudeix, també, dels serveis que els altres *peers* ofereixen.

Podem considerar la següent tipologia d'aplicacions P2P:

- Les anomenades **aplicacions col·laboratives** permeten que les persones es relacionin per motius socials o de feina. Exemples d'aquest tipus d'aplicacions serien els xats, la missatgeria instantània, la videoconferència, els jocs en xarxa,...

  Entre els productes d'aquesta mena, es poden citar Groove, myJXTA, Skype,...

- Les **aplicacions de compartició de fitxers i/o continguts**, que han estat les més famoses i, també, les que han aixecat més polèmica en les xarxes P2P. Permeten posar a l'abast d'altres usuaris el contingut d'arxius de distints formats –música, imatge, vídeo,...– que es troben en la pròpia màquina, la qual cosa genera problemes de propietat intel·lectual.

  Entre elles, podem esmentar aplicacions com Gnutella, FreeNet, Kazzaa, eDonkey, BitTorrent,...

- Les **aplicacions de computació distribuïda**. Aquest nom serveix per a designar aquelles aplicacions que permeten solucionar problemes complexes aprofitant –en temps d'inactivitat si el propietari ho desitja– la capacitat de càlcul dels ordinadors connectats a la xarxa, fent que treballin en paral·lel coordinats per un servidor central que s'encarrega del repartiment de les tasques a realitzar.

  Exemples d'aquest tipus d'aplicacions serien: la ja esmentada SETI@home –experiment científic a la recerca d'intel·ligència extraterrestre– i distributed.net –dedicada precisament a temes de criptografia.

2.3.- La plataforma JXTA.

2.3.1.- Introducció.

Com ja s'ha esmentat, moltes aplicacions P2P són tancades i utilitzen protocols de comunicació propis. La plataforma JXTA és la concreció d'un projecte iniciat per *Sun Microsystems* l'any 2001 amb la idea de dissenyar una solució vàlida per a gairebé tots els tipus d'aplicacions P2P.

JXTA –acrònim de la paraula anglesa JuXTApose[13]– especifica un conjunt de protocols no propietaris que serveixen de marc de referència en les comunicacions a través de xarxes P2P. Aquests protocols poden ser implementats en qualsevol llenguatge i per a qualsevol sistema operatiu, en estar basats en un estàndard com XML. A més, JXTA pot ser executat per distintes classes de dispositius: telèfons cel·lulars, PDA, ordinadors, servidors,... Actualment, n'existeixen implementacions en Java –tant per J2SE com per J2ME– i C/C++ que es poden descarregar des de la web de la plataforma: http://www.jxta.org.

2.3.2.- L'arquitectura de protocols JXTA.

Els protocols JXTA especifiquen la forma a través de la qual els *peers* es descobreixen, s'organitzen en grups, anuncien i descobreixen els recursos disponibles en la xarxa, es comuniquen i es supervisen. El conjunt de protocols especificats en la plataforma són els següents:

- **Peer Resolver Protocol** (PRP): emprat per a enviar requeriments a un conjunt de peers –o a equips específics– i rebre'n les respostes.
- **Peer Discovery Protocol** (PDP): és el mecanisme utilitzat per un *peer* per anunciar els seus recursos i per a descobrir els disponibles en d'altres *peers*. Cada recurs es descriu –i es publica– mitjançant un anunci que en la terminologia JXTA s'anomena ***advertisement***, representat a través d'un document XML.
- **Peer Information Protocol** (PIP): utilitzat per a obtenir informació referent a l'estat d'altres *peers*.
- **Pipe Binding Protocol** (PBP): mecanisme emprat per a que un *peer* estableixi canals virtuals de comunicació –anomenats ***pipes***– amb altres *peers*. Les *pipes* són en JXTA la base del mecanisme de comunicació entre *peers*.
- **Endpoint Routing Protocol** (ERP): usat per a cercar una ruta que permeti fer arribar un missatge a un altre *peer*. Si un *peer* no coneix la ruta que l'uneix a un altre, necessita utilitzar d'altres *peers* que li facin d'intermediaris. El protocol ERP s'utilitza per a descobrir el camí.
- **Rendezvous Protocol** (RVP): és el mecanisme a través del qual els *peers* es poden subscriure a un servei de propagació de missatges en la xarxa –o esdevenir-ne agents. Dins un grup, un *peer* pot ser un *rendezvous-peer* o bé dependre dels *rendezvous-peers*. El protocol RVP fa possible que un *peer* enviï missatges a tots els *peers* que implementen el servei, i és el protocol emprat per altres protocols –PRP i PBP– per propagar els seus missatges.

Cadascun d'aquests protocols és gairebé independent dels altres. No és necessari que un *peer* els implementi tots: només cal que implementi els que necessita. Per exemple, si un *peer* no necessita obtenir ni proporcionar informació d'estat als altres, no cal que implementi el Peer Information Protocol. Ara bé, sí cal que implementi dos protocols per tal de poder ser reconegut com a *peer*: el Peer Resolver Protocol i l'Endpoint Routing Protocol.

[13] Segons Bill Joy, que conjuntament amb Mike Clary dirigí inicialment el projecte, "[juxtapose] vol dir situar coses unes al costat de les altres, i el *peer-to-peer* tracta precisament d'això".

Aquests dos protocols i els seus anuncis –*advertisements*–, serveis i definicions són el nucli de l'especificació de la plataforma, el **JXTA Core Specification**, i constitueixen la infraestructura bàsica que usen els altres serveis i aplicacions. Els altres protocols, serveis i anuncis són opcionals i es coneixen amb el nom de **JXTA Standard Services**. La figura 14 mostra les relacions entre els protocols de la plataforma.

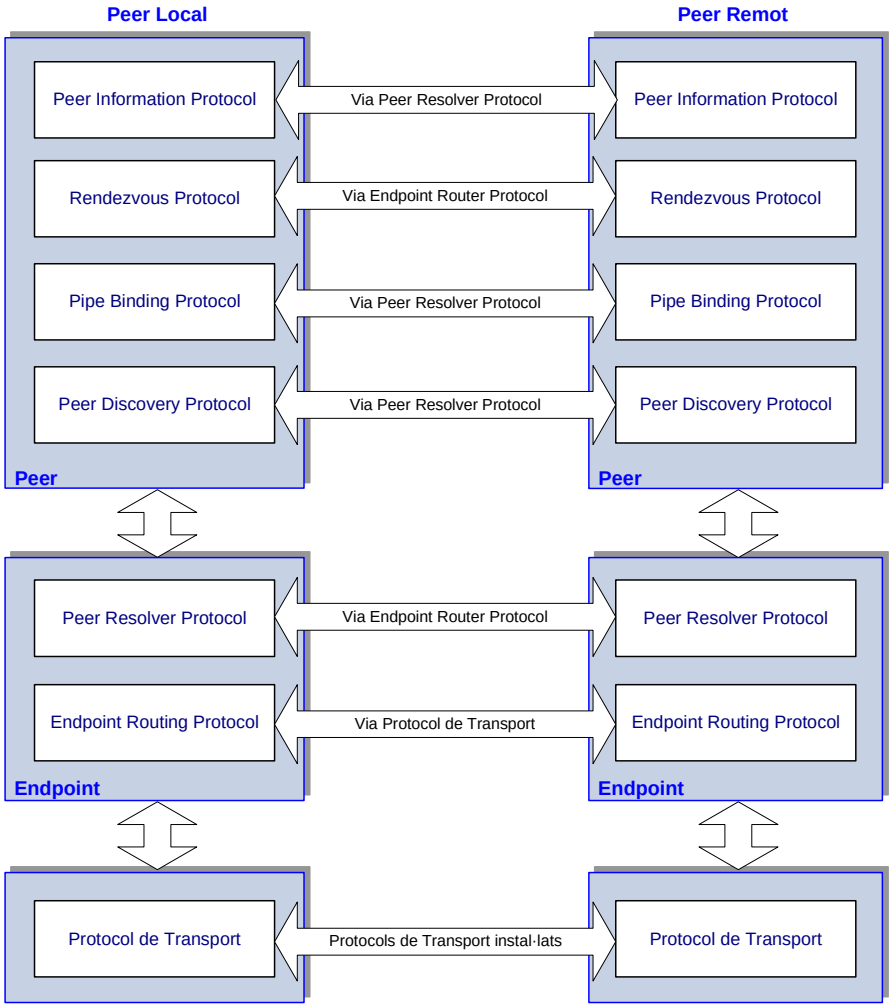

Fig. 14 - Protocols JXTA

2.3.3.- L'arquitectura de capes de JXTA.

Des del punt de vista de lògica de l'aplicació, la plataforma JXTA es pot considerar dividida en tres capes, cadascuna de les quals es construeix sobre les característiques de la capa inferior, afegint nova funcionalitat a la plataforma. La figura 15[14] mostra gràficament l'arquitectura. Es pot observar:

- La **capa central** de la plataforma –**JXTA Core**–, que encapsula els elements mínims i essencials que són comuns a qualsevol xarxa P2P:
  - *Peers*
  - Grups de *peers*
  - Anuncis –*advertisements*–
  - Eines de comunicació: *pipes*, *endpoints*,...
  - Identificadors de qualsevol tipus d'entitat
  - Protocols de descobriment, comunicació, supervisió,...
  - Elements per a realitzar polítiques de seguretat

[14] Extreta de J.C.Soto *JXTA-SAMS.pdf*. Disponible a http://www.jxta.org/community/WebServicesEdge-SanJose2002/JXTA-SAMS.PDF.

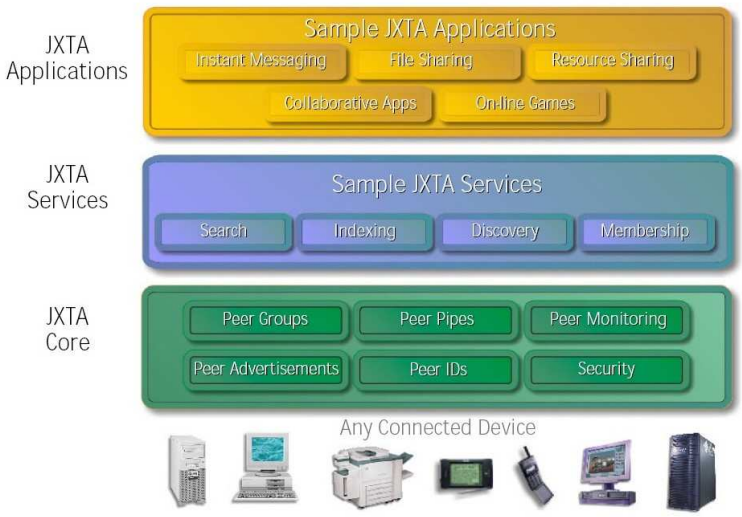

Fig. 15 - Arquitectura de JXTA

Aquesta capa es troba implementada en tots els dispositius P2P de forma que es puguin interconnectar. Inclou els protocols de la plataforma, tot i que aquests s'implementen com a serveis.

- La **capa de serveis** –**JXTA Services**–, on es troben els serveis que no són absolutament necessaris en una xarxa P2P, però que són desitjables:
  - per a cercar recursos, incorporant característiques d'indexació;
  - per a compartir recursos;
  - per a que dos sistemes que usin protocols diferents es puguin entendre,
  - o per a realitzar tasques d'associació a grups i d'autenticació.
- La **capa d'aplicacions** –**JXTA Applications**. Com és obvi, aquesta capa inclou les aplicacions típiques del món P2P esmentades en anteriors apartats. La frontera, però, entre servei i aplicació no és rígida: el que un usuari veu com una aplicació, un altre ho pren com un servei. Normalment, es considera que hi ha aplicació quan existeix alguna mena d'interfície d'usuari. En cas contrari, es suposa que ens trobem davant d'un servei.

2.3.4.- La xarxa virtual JXTA: conceptes bàsics.

Els protocols disponibles en JXTA permeten crear una xarxa virtual que es sobreposa a la infraestructura física existent. Aquesta estructura virtual possibilita a qualsevol *peer* la comunicació amb els altres, independentment de la seva situació, es trobi o no darrera tallafocs i/o encaminadors NAT, utilitzi o no xarxes IP. Els missatges són encaminats de forma transparent emprant diferents protocols –TCP/IP, HTTP– per tal d'aconseguir arribar al seu destí. La figura adjunta –ja reproduïda i del mateix origen que l'anterior– mostra aquest fet.

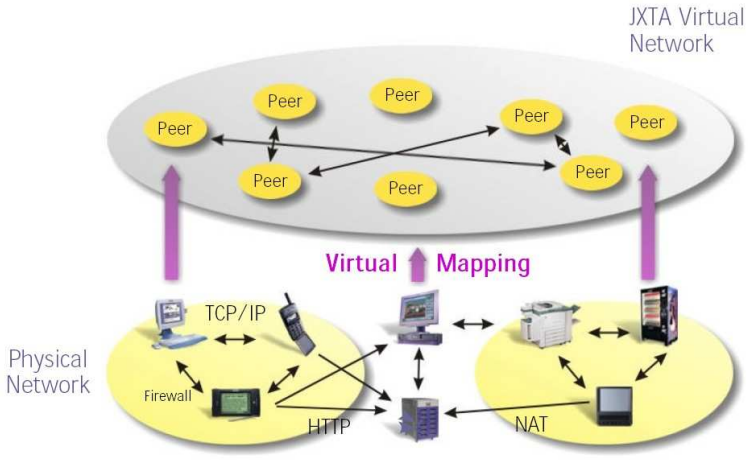

Fig. 16 - Xarxa virtual JXTA

Mitjançant la xarxa virtual, els *peers* gaudeixen de les següents possibilitats:

- Es poden comunicar sense necessitat de conèixer o gestionar l'estructura física de la xarxa, la qual cosa els hi permet moure's de forma transparent d'un lloc a un altre.
- Disposen d'una forma estàndard per a descobrir-se –ells i els recursos–, organitzar-se en grups i comunicar-se.

Per tal de poder complir amb aquests requeriments, la plataforma incorpora a aquesta abstracció –la xarxa virtual– els següents conceptes:

- Una forma lògica d'adreçament, els anomenats **JXTA ID**, que constitueixen la manera uniforme de fer referència a diferents recursos –*peer*, *pipe*, *peergroup*, *advertisement*,...–. Els JXTA ID són objectes que permeten a d'altres identificadors –com adreces IP o MAC– coexistir dins la mateixa xarxa virtual JXTA. La implementació Java de referència de la plataforma permet que cada *peer* els generi de forma aleatòria.

  Així, per exemple, un *peer* en JXTA és identificat pel seu *peer-ID*, la qual cosa fa que tothom es pugui adreçar a ell independentment de la seva localització, és a dir, de la seva adreça física. A cada *peer-ID* se li associa un *peer-endpoint* que encapsula totes les adreces físiques de xarxa vàlides per al *peer*. Així, quan un altre *peer* rep un anunci relacionat amb el *peer-endpoint* del primer, pot elegir la forma de comunicació més adient, escollint, si és el cas, entre una comunicació directa emprant TCP/IP o bé utilitzant HTTP sobre TCP si cal travessar un tallafoc.

- Els grups de *peers* –***peer group*** en la terminologia original. Són entitats que representen conjunts de *peers* que col·laboren oferint determinats serveis. L'especificació JXTA no indica quan, on o perquè crear un *peer group*, ni posa límit al nombre de grups als que un *peer* pot pertànyer. Sí que defineix, en canvi, com es poden crear, publicar i descobrir els grups emprant el Peer Discovery Protocol.

  Cal indicar, també, que els protocols especificats en la plataforma s'implementen mitjançant serveis i que aquests s'ofereixen en el marc dels grups. A més, cada peer pertany, com a mínim, a dos grups: el **WorldPeerGroup** i el **NetPeerGroup**.

- Els anuncis, originalment ***advertisements***. Es poden definir com una representació estructurada d'una entitat, servei, o recurs que un *peer* o grup de *peers* posa a disposició d'altres en una xarxa P2P. JXTA utilitza abastament aquests anuncis, emprant l'XML a l'hora d'estructurar tot el què publica: *peer*, *peer group*, *pipe*, servei, ruta, contingut, *rendezvous*, *peer-endpoint* o tipus de transport emprat. A més dels definits explícitament en l'especificació de la plataforma, cada servei i aplicació que es crea pot definir-ne de nous.

  La figura adjunta en mostra un que s'utilitza en l'aplicació de xat per tal d'anunciar l'estat presencial d'un usuari. A més de l'anunci, es pot observar la forma que pren l'identificador del *peer*.

```
public void announcePresence(char presenceStatus, String emailAddress, String name) {
    if (discovery != null)
    {
        // Creem l'advertisement
        PresenceAdvertisement pres
                (PresenceAdver
        // I el configurem.
        presenceInfo.setPresenceSt
        presenceInfo.setEmailAddre
        presenceInfo.setName(name)
        presenceInfo.setPeerID(loc
        try {
            discovery.publish(pres
        } catch (IOException ex) {
            System.out.println("Er
        }
        discovery.remotePublish(presenceInfo, ConstantsXatSegur.TEMPS_EXPIRA);
    }
```

```
presenceInfo = (xatsegur.presence.PresenceAdv) <?xml version="1.0"?>
<!DOCTYPE PresenceAdvertisement>
<PresenceAdvertisement>
        <PeerID>
                urn:jxta:uuid-596162616461626 14A78746150325033 4DF58B7F6D8B4A48A8F8DBA9C453890A03
        </PeerID>
        <EmailAddress>
                aniolserra@hotmail.com
        </EmailAddress>
        <Name>
                aniol
        </Name>
        <PresenceStatus>
                O
        </PresenceStatus>
</PresenceAdvertisement>
```

Fig. 17 - Anuncis i Identificadors

- L'anomenat ***resolver***. A l'hora de resoldre –en el sentit de trobar o reconvertir– un *peer* en una adreça IP, lligar un *socket* a un port, localitzar una entitat a través d'un servei de directori o cercar un contingut, JXTA utilitza un sistema universal que rep el nom de *resolver* especificat en el Peer Resolver Protocol. Aquest servei nuclear proporciona la infraestructura que permet enviar i propagar requeriments, i rebre'n les respostes. Si convé, autentica i verifica credencials, i esborra els missatges que no considera vàlids.

Per tal de realitzar les tasques que té encomanades, aquest servei es recolza –en determinades situacions– en *peers* que presenten característiques especials: els *rendezvous peers* i els *router peers*. El següent apartat explica amb una mica més de detall com es porten a la pràctica aquestes qüestions.

- Els canals virtuals de comunicació o ***pipes***. S'usen a l'hora de rebre i enviar missatges entre serveis i aplicacions, i proporcionen una abstracció –que es situa per sobre dels punts finals d'origen i de destí– que permet creure en l'existència de bústies virtuals d'entrada, *input pipe*, i de sortida, *output pipe*, no lligades a una localització física concreta.

  Com altres elements dins JXTA, s'identifiquen per un *pipe-ID*, i es publiquen i descobreixen mitjançant un *advertisement*. En temps d'execució, el Pipe Binding Protocol, emprant els serveis del *resolver*, les lliga al punt final del *peer* que s'usa en la comunicació. Les *pipes* ofereixen dues modalitats a l'hora d'establir comunicacions:

  - El mode punt a punt connecta dues pipes a través d'un canal unidireccional i asíncron –un punt final d'entrada que rep dades enviades des d'un punt final de sortida.
  - El mode propagació, que connecta una *pipe* de sortida a múltiples d'entrada, de forma que el missatge enviat té molts destinataris.

  A més dels serveis nuclears relacionats amb les *pipes* que ofereix la plataforma, aquesta també proporciona –construïts a partir dels anteriors– serveis de *pipes* bidireccionals, de confiança i segures. Finalment, cal indicar que la implementació 2.0 de la plataforma incorpora al Pipe Service el concepte de ***socket***, la qual cosa facilita la programació.

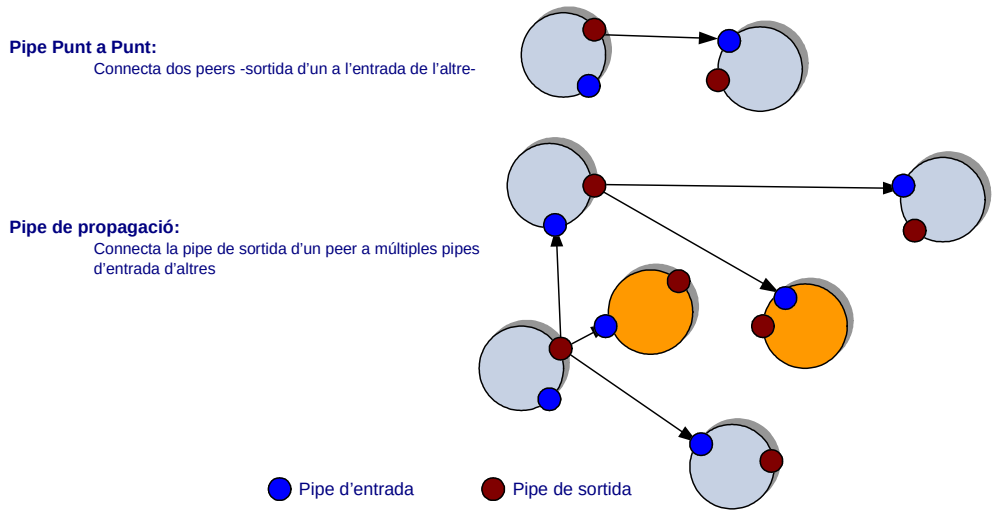

Fig. 18 - Tipus de pipes

2.3.5.- Descoberta d'advertisements.

En una xarxa P2P cal resoldre dos problemes importants: com descobrir *peers* i serveis, i com un dispositiu situat en una intranet pot participar com un més en la xarxa. Com s'ha esmentat en l'apartat anterior, en JXTA qualsevol entitat s'anuncia a través d'un *advertisement*, per la qual cosa els problemes anteriors es concreten en la forma de descobrir aquests anuncis en la xarxa. Els paràgrafs següents es dediquen a aquesta qüestió. Naturalment, no es pretén realitzar una explicació exhaustiva, que es pot trobar en les referències bibliogràfiques que es citen al final del treball.

Podem distingir tres formes de descobriment:

- El **no descobriment**. Aquest sistema parteix del fet d'haver realitzat prèviament una descoberta d'algun altre tipus. El *peer* confia en què disposa en la seva cau[15] de dades relacionades amb *advertisements* anteriors i les utilitza per contactar amb altres *peers*.

  Aquest sistema és ràpid, redueix el tràfic en la xarxa, però pot ser que s'obtinguin –com a punt de partida– dades que no estan actualitzades si s'han publicat amb temps de vida llargs.

[15] En JXTA, cada *peer* disposa d'una cau situada en un directori anomenat **cm** que es troba sota el directori en què s'està executant la plataforma. En la implementació Java, aquesta cau és controlada per la classe **Cm** –*Cache Manager*–.

Normalment, les implementacions controlen aquest fet i inicien noves descobertes, emprant els altres mètodes, quan observen una entrada obsoleta o bé no troben el recurs cercat.

- La **descoberta directa**, realitzada pels *peers* que es troben en la mateixa LAN, en ser capaços de descobrir-se sense necessitat d'emprar intermediaris.

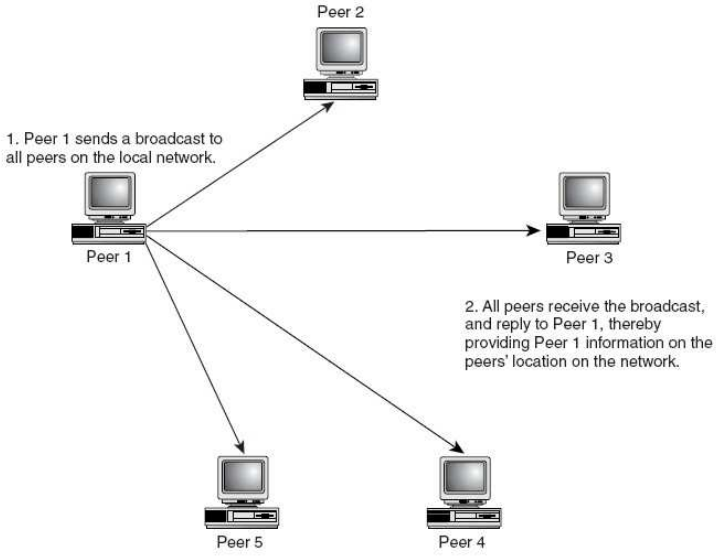

Fig. 19 - Descoberta directa

La figura, extreta del text sobre JXTA de Brendan Wilson referenciat en la bibliografia, mostra com té lloc la descoberta:

- Un *peer* anuncia la seva presència i/o els seus serveis als altres emprant tècniques de broadcast.
- Una vegada ha estat descobert, els altres s'hi poden adreçar directament, bé per comunicar-s'hi o per a descobrir nous serveis.

- La **descoberta indirecta** que requereix l'ús de *rendezvous-peers* que mantenen una base de dades amb anuncis procedents d'altres *peers* i que, alhora, poden propagar la cerca en nom del *peer* que realitza el descobriment. Aquest tipus de descobriment el poden realitzar equips dins una LAN, però, sobretot, es fa servir per a què els equips situats dins una intranet puguin relacionar-se amb els què estan fora del perímetre.

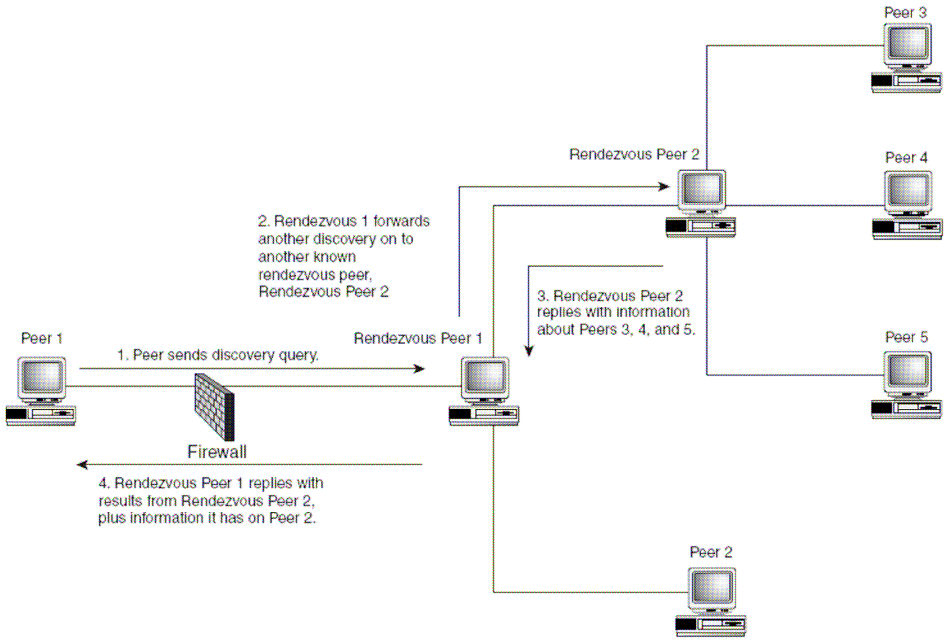

Fig. 20 - Descoberta indirecta

La figura anterior, procedent de la referència esmentada anteriorment, mostra aquest tipus de descoberta. Els *rendezvous-peers* treballen de dues maneres a l'hora de resoldre el requeriment:

- Cercant en la seva cau.
- Propagant la cerca a d'altres *peers* i/o *rendezvous-peers* que coneixen.

Per tal d'evitar que es produeixi congestió en la xarxa, cada missatge incorpora un camp *Time To Live* (TTL). Com sol ser usual en aquests casos, cada *peer* que rep el missatge decrementa el valor del camp i quan aquest arriba a zero, el missatge no es propaga.

Finalment, introduirem, per acabar l'apartat, com treballa un *peer* amb els seus *rendezvous* i *relays peers*. Aquest tema és clau per aquells *peers* que es troben situats en xarxes privades protegides per tallafocs i/o dispositius NAT, ja que sense l'ajut d'aquests altres *peers* no podrien participar en la xarxa. Normalment, el què es fa és configurar el *peer* amb una llista de *rendezvous* i *router peers* què li serveixin de punt de contacte amb la xarxa P2P, de forma que a partir d'ells pugui descobrir la resta de *peers* i serveis.

A més, a l'hora de pensar en la comunicació entre el *peer* i aquests *super-peers*, cal tenir en compte que els tallafocs restringeixen –per motius de seguretat– el tràfic entre la intranet i la xarxa pública. Per tant, un *peer* situat a l'interior pot trobar-se amb restriccions en el tràfic adreçat a l'exterior, la qual cosa limita la comunicació i obliga a emprar protocols –com HTTP– que corresponen a trànsit normalment permès.

Els dispositius NAT es solen emprar per evitar que cada màquina dins la intranet necessiti una IP pública si s'ha de comunicar amb l'exterior. L'encaminador NAT té assignat un conjunt reduït d'adreces IP públiques –que representen la xarxa privada– i s'encarrega de mapar les adreces privades a les públiques –mitjançant taules que contenen equivalències @IP_privada-port⇔@IP_pública-port–, possibilitant l'accés des de la intranet a l'exterior. Ara bé, molt sovint els administradors de la xarxa privada fan servir també els dispositius NAT[16] per a protegir la intranet, permetent només connexions d'entrada a determinades màquines o bé aquelles connexions que són la resposta a d'altres què s'han iniciat en la intranet. Per aquest motiu, un *peer* situat fora del perímetre de la xarxa interna no pot contactar amb un situat a l'interior si aquest no inicia la sessió. En resum, la combinació tallafoc/NAT dificulta la comunicació entre *peers*.

El problema es pot solucionar tal com mostra la figura adjunta extreta, com les darreres, del text esmentat anteriorment de Brendan Wilson.

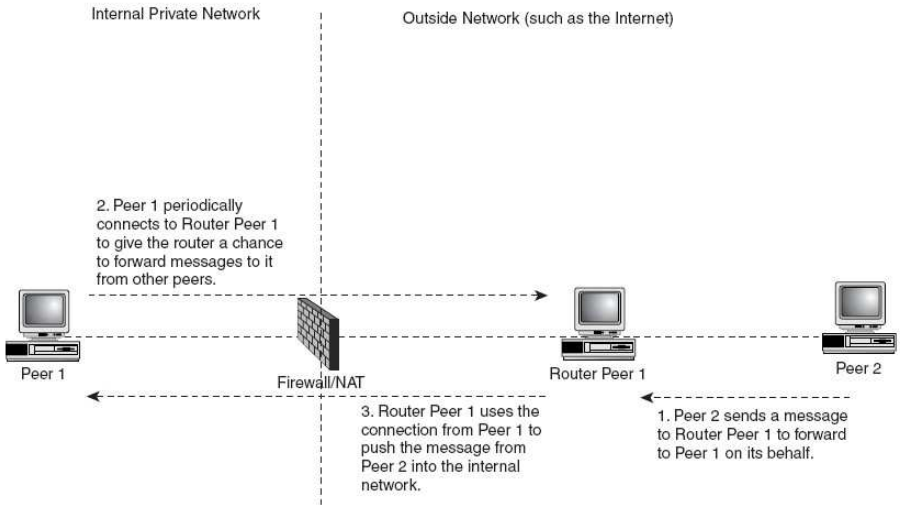

Fig. 21 - Travessant un tallafocs/NAT

Tenint present que el *peer* situat en la intranet pot iniciar connexions amb l'exterior a través de ports permesos, aquestes es poden emprar per a crear túnels de connexió amb l'exterior. A més, aquests túnels poden ser utilitzats per a què una màquina fora del perímetre enviï dades a l'interior[17].

Com molt sovint és permeten les connexions HTTP sortints, aquest és el protocol utilitzat. Ara bé, tenint present que HTTP és un protocol que no contempla per se el concepte de sessió persistent, en ser un protocol del tipus demanda/resposta, i que els *peers* situats en l'exterior no poden adreçar-se directament als *peers* interns a causa del tallafocs, es soluciona el problema amb l'ajut d'un *router peer*, actuant de la forma següent:

[16] És clar que en aquests casos es pot parlar perfectament de tallafocs amb prestacions NAT.

[17] Cal tenir present, però, que si el tallafocs no permet les connexions a l'exterior, la comunicació amb el *peer* intern és impossible.

- Quan un *peer* situat fora del perímetre vol contactar amb un situat a l'interior, connecta amb el *router peer*.
- Mentrestant, el *peer* de la intranet connecta sovint amb el *router-peer* emprant un protocol com HTTP, amb la qual cosa l'encaminador li pot transmetre els missatges de l'altre *peer* amb la resposta a la comanda HTTP.

Si és el *peer* situat en la intranet qui vol contactar amb un que es troba fora, la mecànica de treball és la que mostra aquesta altra figura i es comentat tot seguit:

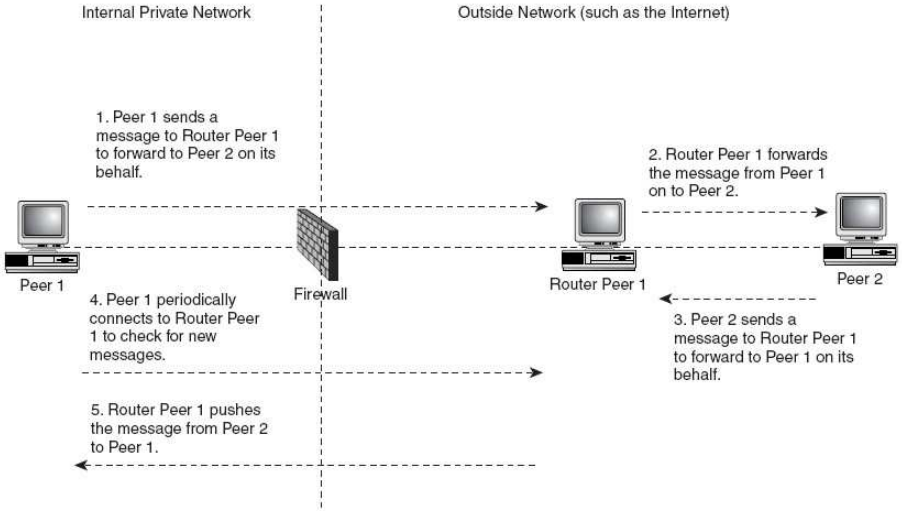

Fig. 22 - Travessant un tallafocs. Missatge des de l'interior.

- El *peer* intern contacta amb el seu *router-peer*, a través d'un protocol com HTTP capaç de travessar el tallafocs, i li demana que adreci un missatge en nom seu a l'altre *peer*.
- L'encaminador connecta amb l'altre *peer* i li adreça –mitjançant qualsevol protocol del nivell de transport comú– el missatge que ha rebut del *peer* intern, actuant com a element intermediari.
- A partir d'aquest instant es reprodueix la situació analitzada anteriorment: el *peer* extern dirigeix la resposta a l'encaminador amb l'encàrrec de fer-la arribar al *peer* intern i aquest, que connecta periòdicament amb el *router-peer* emprant el túnel, acaba rebent la resposta.

Naturalment, si ambdós interlocutors es troben dins intranets protegides, s'actua reproduint per partida doble la situació del *peer* 1 de la figura anterior. La figura següent mostra aquest fet.

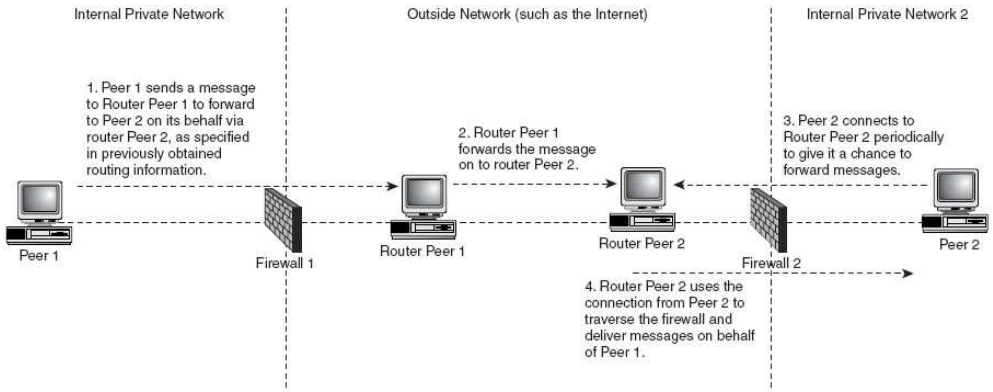

Fig. 23 - Travessa de dos tallafocs

2.3.6.- Els sockets en JXTA.

Com és prou conegut, en Informàtica mitjançant el terme anglès *socket* –connector o sòcol en català– es defineix un punt d'accés a un servei de comunicació en l'àmbit dels protocols del nivell de transport.

Tot i que els *sockets* permeten comunicar processos que s'executen en una mateixa màquina, s'utilitzen sobretot per a comunicar processos que poden córrer en màquines distintes connectades en xarxa. En aquest sentit, els més emprats són els *sockets* TCP que usen l'espai de noms Internet. En aquest cas, l'adreça que identifica el

socket –necessària per a poder-s'hi comunicar– està formada, a més del protocol emprat en la comunicació (TCP o UDP), per l'adreça IP de la tarja de xarxa emprada i pel número de port associat què distingeix l'aplicació què es comunica mitjançant el *socket*.

Quan es comuniquen dues aplicacions a través dels *sockets*, molt sovint se segueix el model client-servidor, en el qual una de les aplicacions, el servidor, està esperant que li arribin peticions procedents de l'altra part, el client. Normalment, aquest és l'encarregat d'iniciar la comunicació, mentre el servidor ha de mantenir obert el port associat al servei, per la qual cosa ha de tenir una adreça pública ben determinada per a què es pugui establir la connexió. Tenint presents aquestes qüestions, se sol parlar de *sockets* servidor i sockets client.

També, a l'hora de comunicar-se amb un altre equip cal indicar quin serà l'estil de la comunicació:

- **Orientat a connexió**, anomenat també seqüència de bytes, en el qual les dades es transmeten com un flux –*stream*– de bytes entre els interlocutors. Aquest estil correspon, en l'espai de noms Internet, al protocol TCP.
- **No orientat a connexió** o estil datagrama. En aquest model, es transmeten paquets individuals de dades, anomenats datagrames, que en arribar al destí hauran de ser ordenats per part de l'aplicació destinatària. En Internet, és el model que segueix el protocol UDP. Com és conegut, aquest model no proporciona la fiabilitat que ofereix un model orientat a connexió,

Quan es desitja una comunicació segura i fiable en la què les dades arribin ordenades, sense pèrdues ni duplicitats, cal emprar un estil orientat a la connexió. La figura mostra en aquest cas quina ha de ser la seqüència d'operacions –en la part client i en la part servidora– a l'hora de portar a la pràctica la comunicació.

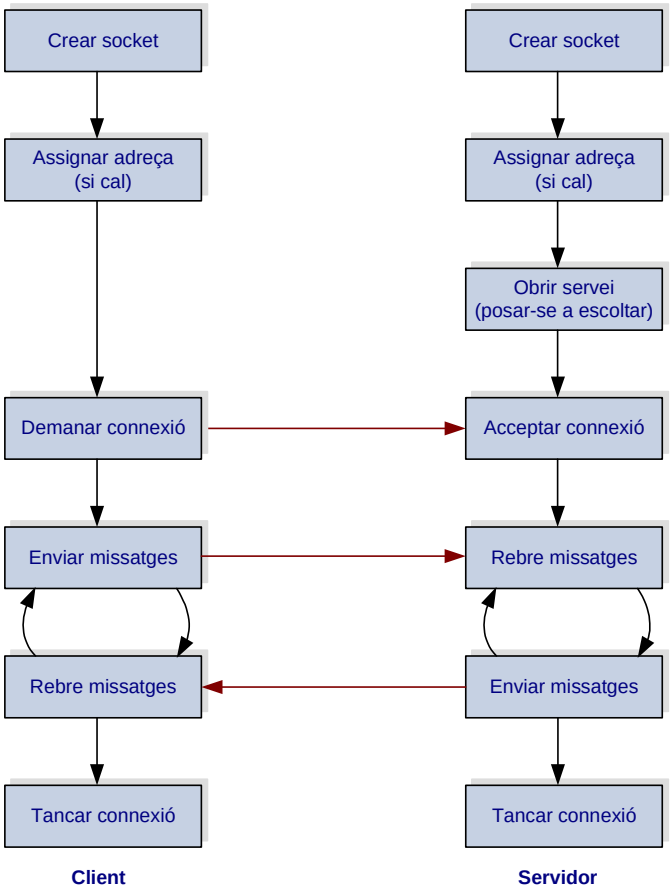

Fig. 24 - Operacions sockets client i servidor

Com s'ha esmentat anteriorment, en JXTA els protocols especificats en la plataforma s'implementen mitjançant els serveis que s'ofereixen dins els grups. Un dels serveis disponibles en la plataforma és el **Pipe Service**. Aquest servei, associat al Pipe Binding Protocol, és el responsable de la creació dels objectes *pipe* i de lligar-los als elements finals encarregats de la comunicació: els ***endpoints***, abstracció de les interfícies de xarxa emprades tant en enviar com en rebre dades.

Si l'aplicació de xat segur s'hagués desenvolupat a partir de les primeres versions de JXTA, la comunicació entre els participants es basaria en la utilització de *pipes* bidireccionals obtingudes a partir d'un servei ja obsolet anomenat BidirectionalPipe Service. Ara bé, a partir de la versió 2.0, l'API de JXTA incorpora classes hereves de

les classes que encapsulen *sockets* en l'API de Java. En concret, si pensem en el què s'ha esmentat en paràgrafs anteriors, són interessants les següents:

- **JxtaServerSocket**: classe que encapsula un *socket* servidor JXTA que espera connexions des de clients.
- **JxtaSocket**: classe que permet crear fluxos de E/S tant per clients com per a servidors.

La diferència respecte als *sockets* estàndard es troba en la forma d'adreçament emprada. Com és fàcil suposar, JXTA ho fa mitjançant un anunci o *advertisement* relacionat amb la *pipe* sobre la què es construeix el *socket*. En l'apartat de la memòria dedicat a la implementació s'indicarà com es crea aquest identificador.

A més de les classes esmentades, la implementació Java de la plataforma proporciona classes que permeten treballar amb fluxos –i a través d'ells enviar i transmetre les dades. Són les següents:

- **JxtaSocketInputStream**: classe que hereta de la classe abstracte **InputStream**, a partir de la qual podem obtenir un flux d'entrada.
- **JxtaSocketOutputStream**: classe que hereta de la classe abstracte **OutputStream**, a partir de la qual podem obtenir un flux de sortida de dades.

3.- Disseny.

3.1.- Introducció.

L'aplicació desenvolupada es pot encabir en el tipus d'aplicacions P2P anomenat aplicacions col·laboratives i, en concret, en les de xat o missatgeria instantània. Es pretén que qualsevol usuari de l'aplicació pugui iniciar converses amb altres usuaris, mantenir-ne vàries alhora i convidar a d'altres a participar en una conversa ja iniciada. Com la comunicació fa servir la infraestructura de xarxa que proporciona un mitjà públic i insegur com Internet, per tal de mantenir la privacitat, els missatges viatgen xifrats.

En aquest sentit, cada parella de participants genera una clau de sessió. Per a generar-la es posa en marxa un procés d'intercanvi que segueix el model de Diffie-Hellman. Un cop obtinguda la clau, tots els missatges s'envien xifrats. Cal remarcar que la clau de sessió s'estableix, sempre, entre parelles de participants. Així, si en un instant determinat intervenen tres usuaris en la conversa, la generació de claus ha passat per les següents etapes:

- Si la conversa l'inicia **A** –actuant com a client– i es posa en contacte amb **B** –que ha activat el seu servidor–, ambdós generen la seva clau de sessió K_{AB} i la utilitzen per xifrar i desxifrar els missatges que s'adrecen.
- Posteriorment **A**, per exemple, convida a **C** a la conversa. Aquest fet ocasiona:
  - **A** –client– inicia un procés d'intercanvi amb **C** –servidor– que acaba generant K_{AC}, emprada en la comunicació entre ells dos.
  - **B** rep un missatge de **A** indicant-li que convidi a **C** a la mateixa conversa. Per tant, **B** inicia un procés d'intercanvi amb **C** que conclou amb la generació de K_{BC}.

Acabem, per tant, disposant de tres canals segurs i independents pels que circulen els missatges de la conversa comuna xifrats amb tres claus diferents. La figura il·lustra aquest fet.

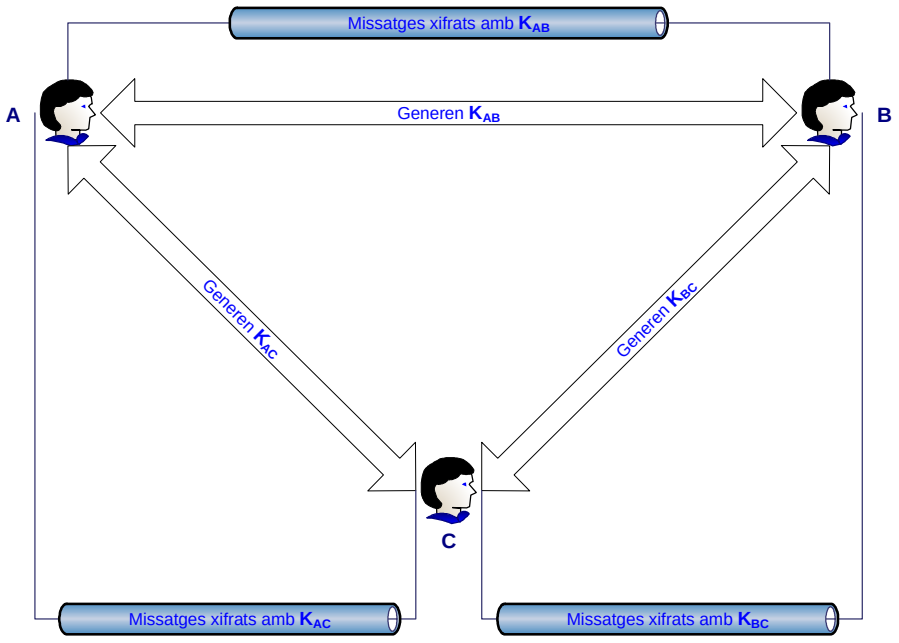

Fig. 25 - Claus i participants

Tenint present el què s'acaba d'esmentar, els objectius i l'enfocament indicats en l'apartat d'introducció d'aquest document, el disseny de l'aplicació s'ha realitzat d'acord amb les següents pautes:

- En ser una xarxa P2P una xarxa entre iguals, en què tots els nodes es comporten alhora com a clients i com a servidors, cal desenvolupar en cada *peer* una part servidora i una part client.
- S'utilitza la plataforma JXTA com a base en els processos de comunicació, en facilitar la comunicació entre els distints *peers* sigui quina sigui la seva situació en la xarxa.
- En iniciar-se la conversa, els dos interlocutors generen una clau de sessió emprant l'algorisme bàsic de Diffie-Hellman, clau que es manté en el transcurs de la conversa.
- Tenint en compte el mètode d'intercanvi de claus escollit –Diffie-Hellman–, en crear un nou contacte es generen els paràmetres públics previstos en el mètode i es guarden, juntament

amb les altres dades del contacte, en un fitxer. Aquesta informació s'emmagatzema xifrada amb una clau generada a partir de la contrasenya del propietari de l'agenda.

- Per tant, l'aplicació ha de disposar d'un gestor de persistència que permeti gestionar les llistes de contactes.
- L'aplicació ha de permetre mantenir vàries converses al mateix temps, amb la possibilitat de varis participants en la mateixa conversa.
- La interacció de l'usuari amb l'aplicació s'ha de realitzar emprant una interfície gràfica. Aquesta interfície ha de permetre:
    - Iniciar sessió –prèvia validació de l'usuari– i tancar-la.
    - Crear un nou usuari, gestionar el seu estat inicial –activació de la part servidora– i editar les altres característiques –nom, adreça de correu,... –.
    - Administrar la llista de contactes, donant-los d'alta, modificant les seves propietats o bé donant-los de baixa.
    - Iniciar converses, convidar-hi a d'altres usuaris i tancar-les.
    - Permetre a l'usuari gestionar el seu estat de connexió –en línia, desconnectat, ocupat,...– quan hagi iniciat l'aplicació.
- S'han de crear les classes d'utilitat necessàries per a fer possible la generació i l'intercanvi de claus, així com els processos per a xifrar i desxifrar els missatges.

Arquitectònicament, l'aplicació s'ha desenvolupat d'acord amb el model MVC, conegut també com a Model 2. En aquest, els objectes que formen part de l'aplicació es divideixen en tres tipus –**Model**, **Vista** i **Control**– i s'associen respectivament a tres capes, cadascuna de les quals presenta un comportament perfectament definit.

La capa de presentació engloba les vistes, que s'encarreguen de mostrar a l'usuari la part del domini que requereix en un instant determinat. El model ens permet representar els objectes que pertanyen al domini del problema. En l'aplicació què ens ocupa, són objectes com l'usuari, els seus contactes, les sessions de xat, els participants en la sessió,... Aquests objectes formen part de la capa de dades i alguns d'ells són persistents –en l'aplicació desenvolupada, l'usuari i l'agenda de contactes.

Finalment, l'arquitectura MVC contempla un tercer tipus d'objecte: els objectes de control. Aquests estan relacionats amb una capa, anomenada d'aplicació, i implementen la lògica d'aquesta, proporcionant-li la funcionalitat desitjada.

En el sistema desenvolupat, les peticions que genera l'usuari mitjançant la seva interacció amb les finestres de l'aplicació –la vista–, son recollides per una classe controladora –**XatSegurControl**– que podrà accedir, directament o a través de classes d'utilitat, als objectes del domini –el model– o crear-ne de nous. Aquesta informació generada pel sistema, la posarem de nou a disposició de l'usuari final mitjançant un canvi en la vista.

De cara al desenvolupament i al manteniment de l'aplicació, aquesta divisió de responsabilitats permet separar la lògica de l'aplicació –el control– de la lògica de la presentació –la vista–, la qual cosa permet reutilitzar la lògica de l'aplicació i evita que els canvis en la presentació influeixin en la lògica de l'aplicació.

En els paràgrafs següents, s'explica com s'han concretat aquestes pautes de disseny amb l'ajut, en algun cas, de diagrames UML.

3.2.- Disseny del programa.

La figura 26 mostra el diagrama de casos d'ús que –d'acord amb les pautes especificades en l'apartat anterior– ha servit de punt de partida en el disseny de l'aplicació.

Observant el diagrama, es pot concloure que es poden classificar els casos d'ús tenint en compte quina és la funcionalitat principal que realitzen. En aquest sentit, establirem la següent classificació que ens servirà de base en els apartats que segueixen:

- Processos relacionats amb el manteniment d'usuaris i grups.
- Processos d'intercanvi, generació de claus i xifratge.
- Processos relacionats específicament amb les sessions de xat.

Aquesta classificació no és estanca i s'utilitza només amb criteris d'explicació.

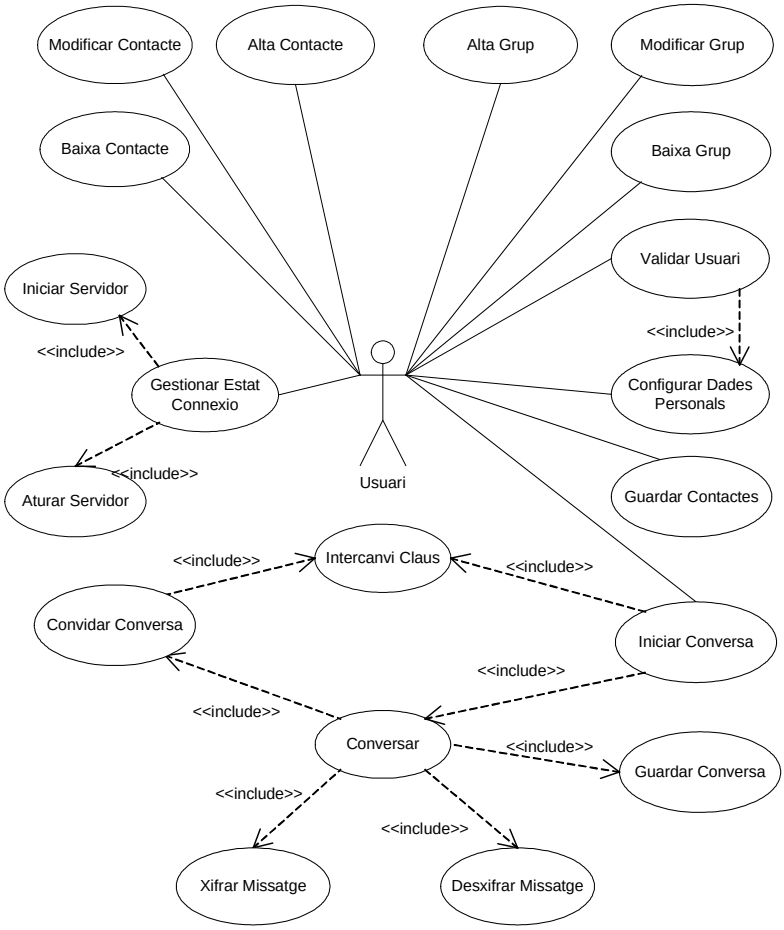

Fig. 26 - Casos d'ús

3.2.1.- Processos de manteniment d'usuaris, contactes i grups.

Per tal d'explicar quina ha estat la base del disseny d'aquests processos, se seguirà el fil d'un possible procés d'execució de l'aplicació. La seqüència podria ser la següent:

- En iniciar-se l'aplicació, es posa en marxa la plataforma JXTA –que es suposarà de moment adequadament configurada–, i tot seguit es demana a l'usuari que s'identifiqui a través del seu nom i contrasenya. Si la identificació és correcta, el sistema presenta a l'usuari l'entorn de treball.
- Si l'usuari no existeix, se'l convida a donar-se d'alta, cosa que fa introduint les dades següents:
  - Nom.
  - Adreça de correu electrònic.
  - Contrasenya –per duplicat, per tal de poder-la confirmar.
  - Estat inicial que desitja –a triar entre: *connectat*, *sense connexió*, *no disponible* i *temporalment ocupat*.

El què succeeix indica que l'aplicació pot tenir en una mateixa màquina distints usuaris, cadascun dels quals s'identifica a través de l'email –igual que ho faran els contactes. Per tant, l'adreça de correu electrònic identifica a tots els usuaris de l'aplicació. Com es veurà posteriorment, en l'aplicació ***JXTA Xat Segur***, l'identificador de la *pipe* sobre la qual es creen els *sockets* que s'utilitzen en la comunicació –tant el servidor com el client– es crea a partir de l'identificador del *peergroup*, del tipus de *pipe* i de l'*email*.

Un cop s'han validat les dades, el gestor de persistència les guarda en el disc protegides amb una clau que es genera a partir de la contrasenya de l'usuari –en un fitxer de nom: *nom_usuari.jxs*. Els detalls de la implementació es donaran en el següent capítol de la memòria.

Dins el seu entorn de treball, l'usuari gaudeix de les següents possibilitats respecte les qüestions tractades en aquest punt:

- Editar les seves dades personals.
- Crear, editar i esborrar grups de contactes. El treball amb els grups és el més senzill, ja que aquests s'identifiquen només pel nom.
- Referent als contactes, pot donar-ne d'alta, editar-los i esborrar-los. Les dades de cada contacte són les següents:
  - Nom.
  - Adreça de correu electrònic.
  - Grups –si cal, a escollir entre els disponibles.

  Quan es dóna d'alta un contacte, el sistema realitza també una tasca relacionada amb l'intercanvi de claus que esdevé en iniciar una conversa: genera els paràmetres –nombre primer, generador, i longitud de l'exponent– corresponents a l'algorisme de Diffie-Hellman. Aquests paràmetres s'empraran posteriorment en la comunicació sempre que l'usuari actuï com a client[18], i es guarden en l'agenda de contactes de l'usuari –fitxer *nom_usuari.cxs*.

Ara bé, les dades dels contactes i dels grups –fitxer *nom_usuari.gxs*– no es fan persistents fins el moment de tancar l'aplicació, a menys que l'usuari executi expressament la salvaguarda a través del cas d'ús **Guardar Contactes**. Com en el cas de les dades pròpies, es xifren amb una clau que es genera a partir del *password* de l'usuari. Tots els fitxers es troben en una carpeta amb el nom de l'usuari i els pot transportar, si ho desitja, a una altra màquina. Com ja s'ha esmentat, en parlar de la implementació es completarà aquest tema.

3.2.2.- Intercanvi i generació de claus. Xifratge i desxifratge.

El procés d'intercanvi es pot considerar que s'inicia quan es dóna d'alta un contacte i es completa quan es genera la clau de sessió. La primera part ja ha estat tractada en els paràgrafs anteriors i la segona passa per les següents fases –es suposa que el servidor està escoltant i el client inicia la sessió:

- En començar la conversa, qui actua com a client crea la seva parella de claus a partir dels paràmetres generats anteriorment que té emmagatzemats en la seva agenda en l'entrada corresponent al servidor.
- Tot seguit, envia un missatge en què expressa que vol iniciar el procés d'intercanvi. Aquest missatge conté, a més de les dades que identifiquen l'usuari –el correu sobretot–, els paràmetres què ha emprat per a crear la seva parella de claus, la seva clau pública i un identificador de sessió generat aleatòriament què s'emprarà posteriorment si es desitja convidar a algun altre contacte a la conversa.
- El servidor, per la seva banda, envia només un missatge inicial de salutació.
- En rebre el missatge, si el servidor accepta la conversa, cosa que fa si té el client donat d'alta en la seva llista de contactes[19], completa l'intercanvi enviant com a resposta a l'anterior un altre missatge d'intercanvi què conté la clau pública de la part servidora. A partir d'aquest moment, el servidor ja pot calcular la clau de sessió ja que coneix tant el component públic de la clau del client com el component privat propi.
- El mateix podrà fer el client en rebre la instància del missatge d'intercanvi adreçada pel servidor.
- Quan cadascun d'ells disposa de la clau de sessió, s'adrecen mútuament un missatge d'acceptació, instant a partir del qual els missatges s'envien xifrats.
- En cas de produir-se algun problema en el procés d'intercanvi, qui el detecta adreça un missatge a l'altra part comunicant l'errada, es tanca la sessió tot just començada i s'adreça un missatge a l'usuari explicant el fet.
- Com s'ha esmentat, un cop creada la clau de sessió els missatges s'envien xifrats. L'algorisme previst, tant per a xifrar com per a desxifrar, és el Triple-DES.

⇒ Una vegada la conversa s'ha iniciat, qualsevol dels participants pot convidar-ne un altre. Per a fer-ho, un d'ells posa en marxa el procés de convidar, el qual genera:

[18] Tot i que és de suposar que l'altre peer també disposa de dades semblants, el client és qui inicia l'intercanvi i proposa els valors dels paràmetres. Quan la conversa l'inicia la part que actua ara de servidora, és aquesta qui –actuant llavors com a client– proposa els valors de partida.

[19] Si no existeix l'entrada en la llista de contactes, s'adreça un missatge oferint la possibilitat d'acceptar la proposta de xat. En cas de resposta afirmativa, el procés de generació de la clau de sessió és idèntic a l'esmentat, ja que la proposta porta incorporats els paràmetres de l'intercanvi.

- Respecte al convidat, un procés idèntic al descrit anteriorment quan la part client d'un usuari contacta amb la part servidora d'un altre –el paper de servidor l'adopta en aquest cas el convidat.
- Respecte als altres participants en la conversa, se'ls hi envia un missatge especial que conté les dades –*email* i nom– del convidat. En rebre aquest missatge, la resta de participants inicien el contacte amb el convidat –que manté per a tots el paper de servidor–. Com el convidat detecta que tots els missatges porten el mateix identificador de sessió, integra a tots els participants en la mateixa conversa.

⇒ Quan un usuari vol finalitzar la seva participació en el xat, envia a la resta un missatge de fi de conversa.

Totes les tasques associades als processos esmentats en aquest subapartat s'implementen a través dels mètodes d'una classe d'utilitat: la classe **DiffieHellman**. Tot i que en l'apartat dedicat a la implementació es comentarà amb més detall, els mètodes d'aquesta classe usen classes incloses en l'API de Java.

DiffieHellman
`+generaParametres(longitud : int): DHParameterSpec`
`+generaParellaClaus(p : BigInteger, g : BigInteger, long : int): KeyPair`
`+getClauPublicaBytes(kp : KeyPair): byte[]`
`+setClauPublicaBytes(clPublicaB : byte[]): PublicKey`
`+generaClauSecreta(PrivateKey clPr, PublicKey clPub): SecretKey`
`+xifra(txtClar : byte[], clau : SecretKey, algorisme : String): byte[]`
`+desxifra(txtXifrat : byte[], clau : SecretKey, algorisme : String): byte[]`

Fig. 27 - Classe DiffieHellman.java

3.2.3.- Interaccions client/servidor. Sessions de xat.

Tal com s'ha esmentat vàries vegades, una xarxa P2P és una xarxa entre iguals en la què tots els nodes disposen d'una part servidora i d'una part client. També s'ha indicat en aquest document que el disseny de l'aplicació aprofita la infraestructura de la plataforma JXTA. En ella, els protocols especificats s'implementen mitjançant serveis que s'ofereixen en el marc dels grups de *peers*. Com és conegut, qualsevol *peer* pertany a dos grups: el **WorldPeerGroup** i el **NetPeerGroup**. Com la plataforma permet la creació de nous grups, s'utilitza aquesta possibilitat per a crear un nou grup, el **JXTA Xat Segur**, on s'encabeix el servei de xat.

Com s'ha indicat en l'apartat 2.3.6, un dels serveis disponibles en JXTA és el **Pipe Service**, responsable de la creació d'objectes *pipe* i de lligar-los als elements finals de comunicació: els *endpoints*. En emprar una versió de la plataforma superior a la 2.0, tant el servidor com el client es poden implementar a partir de *sockets* JXTA. Tenint present aquest fet, un cop s'ha posat en marxa la plataforma en iniciar l'aplicació, la qual cosa es fa trucant al mètode `iniciJxta()` de la classe de control **XatSegurControl**, quan un *peer* està disposat a rebre peticions de connexió, actua d'acord amb el què s'indica tot seguit:

- Crea un identificador per a la *pipe* que els altres *peers* aprofitaran per connectar-se a la part servidora del *peer*.

  En el cas de l'aplicació ***JXTA Xat Segur***, aquest identificador –i d'aquí la importància del correu electrònic– es crea a partir de l'identificador del *peergroup*, del tipus de *pipe* i de l'adreça de correu de l'usuari, emprant –seguint criteris recomanats per experts en la plataforma– una funció de resum. A continuació podem observar-ne un possible resultat:

  clLyJyaylklDeDQtyJKQ+i97fkoD2khVSVTud49+QRAee.uuid.4559EDE6C1D74F5288D1CB345E34598102

- Després de l'identificador, es crea un *socket* JXTA servidor associat a la *pipe* publicada i el sistema es disposa a rebre peticions.

  Per tal de fer-ho, el mètode `iniciaServidor()` de **XatSegurControl** inicia un nou fil d'execució –implementat mitjançant la classe **XatSegurServidor**– on s'escolten les peticions emprant el mètode `accept()` de **JxtaServerSocket**.

- Abans, però, cal fer pública la *pipe* mitjançant un *advertisement*, cosa que es fa a través del Discovery Service associat al grup.

- Publicada la *pipe*, amb el servidor activat per acceptar connexions, qualsevol altre *peer* s'hi podrà connectar si en coneix l'identificador, ja que aquest s'inclou amb l'URN que es fa servir en la plataforma JXTA per a les connexions:

  URN:JXTA:BINARYID-CILYJYAYLKIDEDQTYJKQ+I97FKOD2KHVSVTUD49+QRAEE.UUID.4559EDE6C1D74F5288D1CB345E34598102

  Com ja s'ha esmentat, aquest URN actua en JXTA de forma semblant a la parella @IP-port en l'arquitectura TCP/IP.

- Quan arriba una petició, el mètode `accept()` retorna un objecte *socket* JXTA. Aquest objecte es passa a un nou fil d'execució –implementat a través de la classe **ThreadParticipant**. A través dels dos *streams* lligats al *socket* esdevé l'intercanvi de claus i s'inicia la conversa després d'instanciar un objecte que representa la sessió.

  Si l'intercanvi de claus no té èxit –o bé, en cas de no figurar el sol·licitant en la llista de contactes del servidor, es rebutja el xat–, el fil es tanca. També es tanca la sessió, donant-la de baixa en la llista de sessions obertes, si no es detecta la presència de cap participant actiu.

- En confirmar-se la sessió de xat, se li associa una finestra des de la què l'usuari conversa. Posteriorment, des d'ella, l'usuari pot convidar a d'altres contactes al xat iniciat, engegant per a cadascun d'ells un nou fil d'execució. En fer-ho, genera un missatge on indica a la resta de participants la identitat del convidat, amb l'objectiu que aquests hi estableixin contacte.

Els darrers paràgrafs han tractat sobretot de la part servidora. Quan un *peer*, actuant com a client, vol contactar amb un altre ho fa d'acord amb el següent:

- Inicia un procés de petició adreçat al servidor a través del mètode `connectaAServidor()` de la classe controladora **XatSegurControl**. Aquesta petició porta incorporades les dades del servidor, a partir de les quals es crea l'URN que permet contactar-hi.

  Aquest fet ocasiona la creació d'un nou fil d'execució que com en el cas del servidor s'implementa a través de la classe **ThreadParticipant**.

- Aquesta classe, crea un *socket* JXTA a partir de l'*advertisement* associat a la *pipe* publicada pel servidor i, a través de l'*OutputStream* relacionat amb el *socket*, permet adreçar al servidor un missatge d'inici de connexió.

- En funció de la resposta del servidor, s'iniciarà o no el xat, actuant en aquest sentit de forma anàloga a com s'ha manifestat anteriorment en el cas del servidor.

Finalment, tot i que no es pot parlar en aquest cas de clients ni de servidors, comentarem les bases del disseny de l'estat presencial dels usuaris. La gestió d'aquest estat es fonamenta en un altre dels serveis oferts per la plataforma: l'anomenat **Discovery Service**, relacionat amb el Peer Discovery Protocol. Mitjançant aquest servei, emprant *advertisements*, els *peers* que pertanyen a un grup es poden descobrir uns als altres. En l'aplicació desenvolupada s'aprofita aquesta possibilitat per a gestionar la informació de l'estat presencial –*connectat*, *sense connexió*, *no disponible* i *temporalment ocupat*– dels usuaris. Per a fer-ho, s'actua en la forma següent:

- Quan es posa en marxa l'aplicació s'activa un fil de presència –que s'executa a intervals regulars de temps.

- Durant l'estat d'activitat del fil, cada *peer* publica els seu estat mitjançant un *advertisement*, emprant en darrer terme mètodes associats al Discovery Service.

- En el mateix fil, el *peer* utilitza també mètodes associats a aquest servei per a llegir els anuncis publicats que corresponen als altres *peers*.

Com és natural, aquest servei de presència s'associa al grup **JXTA Xat Segur**. El disseny i la implementació de les classes associades a aquest servei de presència, s'ha pres del llibre sobre JXTA publicat per Brendan Wilson. Pel que afecta a la implementació, l'autor del TFC s'ha limitat, en general, a canviar lleugerament el codi per tal d'ajustar-lo a l'aplicació desenvolupada, fora d'algun cas concret en què ha calgut fer alguna correcció per assegurar el funcionament. En el capítol dedicat a la implementació es tornarà a aquesta qüestió.

3.3.- Disseny de la interfície gràfica.

Un dels requeriments de l'aplicació és que l'usuari hi ha d'interaccionar mitjançant una interfície gràfica. Aquesta s'ha dissenyat com s'indica tot seguit.

3.3.1.- Finestra de validació de l'usuari.

Una vegada iniciada l'aplicació, es presenta a l'usuari la pantalla següent on se li demana el seu nom i el *password*:

Fig. 28 - Disseny GUI. Validació d'usuari.

3.3.2.- Dades de l'usuari.

Si l'usuari no existeix, se'l pot donar d'alta a través de la següent finestra, que també s'usa per a modificar les seves dades.

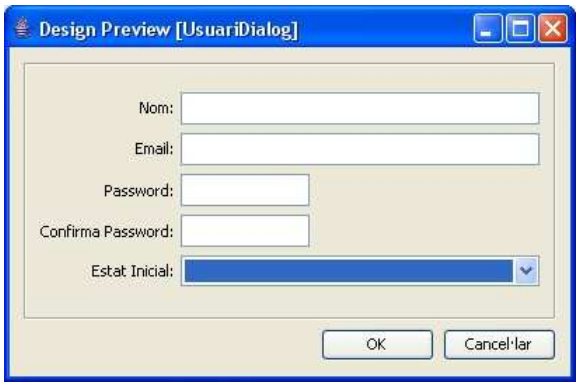

Fig. 29 - Disseny GUI. Dades Usuari.

3.3.3.- Finestra principal.

Un cop l'usuari s'ha validat, cal que es visualitzi la finestra principal de l'aplicació amb les distintes opcions de menú per poder gestionar els contactes, iniciar converses, tancar l'aplicació,...

Fig. 30 - Disseny GUI. Finestra Principal.

La part central de la finestra està ocupada per un objecte **JTree** del paquet **Swing** de Java que contindrà, en temps d'execució, la llista de contactes. Aquest objecte té associat un menú contextual amb accions associades als contactes: edició i inici de conversa.

3.3.4.- Manteniment de grups i contactes.

Les finestres a través de les quals es gestionaran els grups i els contactes apareixen en les figures següents.

Fig. 31 - Disseny GUI. Grups.

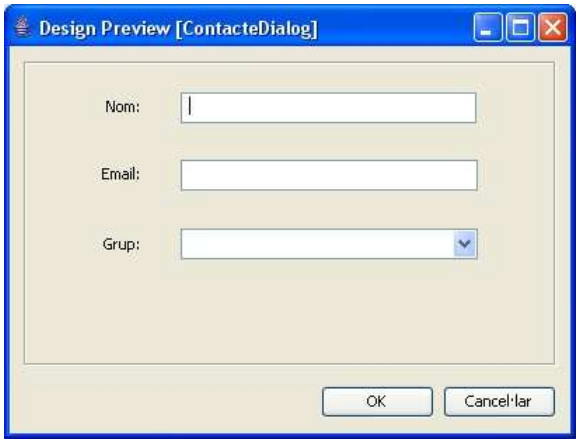

Fig. 32 - Disseny GUI. Contactes.

3.3.5.- Finestra de xat.

En iniciar una sessió de xat, si l'intercanvi de claus té èxit, l'usuari disposa d'una finestra associada a la sessió amb les distintes opcions de menú per poder iniciar noves converses, convidar a d'altres contactes a l'actual, guardar l'estat de la conversa en disc i tancar la sessió.

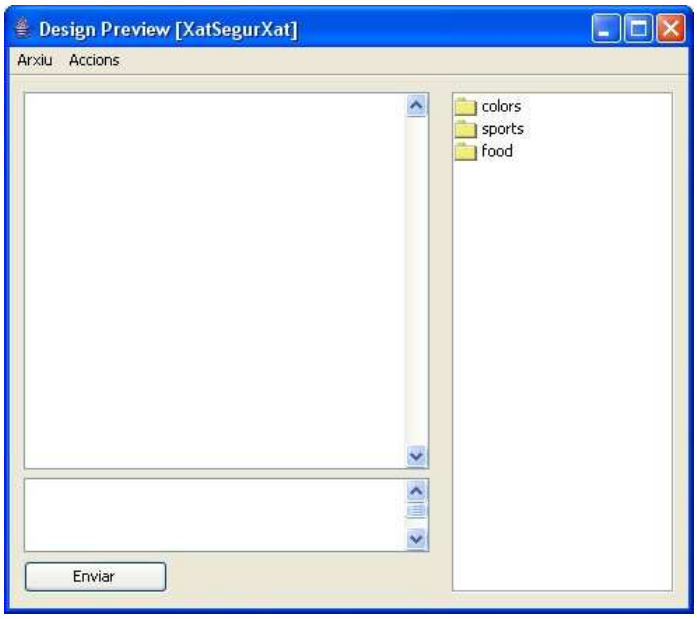

Fig. 33 - Disseny GUI. Finestra de xat.

La part dreta de la finestra permet visualitzar la llista de contactes dins un objecte **Jtree**. Aquest té associat un menú contextual amb accions per iniciar una nova conversa o convidar a l'actual.

3.4.- Diagrames UML.

Tot el què s'ha estat comentant en els apartats anteriors d'aquest capítol es concreta en els següents diagrames UML. Emprant una terminologia pròpia de l'Enginyeria del Programari Orientada a l'Objecte, el primer d'ells

correspondria a l'etapa inicial del disseny tot just acabada l'anàlisi, en què a partir de l'estudi dels casos d'ús s'esbrinen possibles classes d'entitat, mentre que el segon correspondria al diagrama final de l'etapa de disseny. Aquest darrer diagrama ha servit de base a la implementació final de l'aplicació. Respecte a aquest últim, cal indicar el següent:

- En algunes classes –**XatSegurControl**, **XatSegurSessio** i **ThreadParticipant**– no s'han indicat tots els atributs i mètodes, per tal de mantenir el diagrama en uns límits raonables.
- El mateix criteri s'ha aplicat a les classes associades a la interfície gràfica.
- Per motius de claredat del diagrama, algunes classes apareixen repetides.

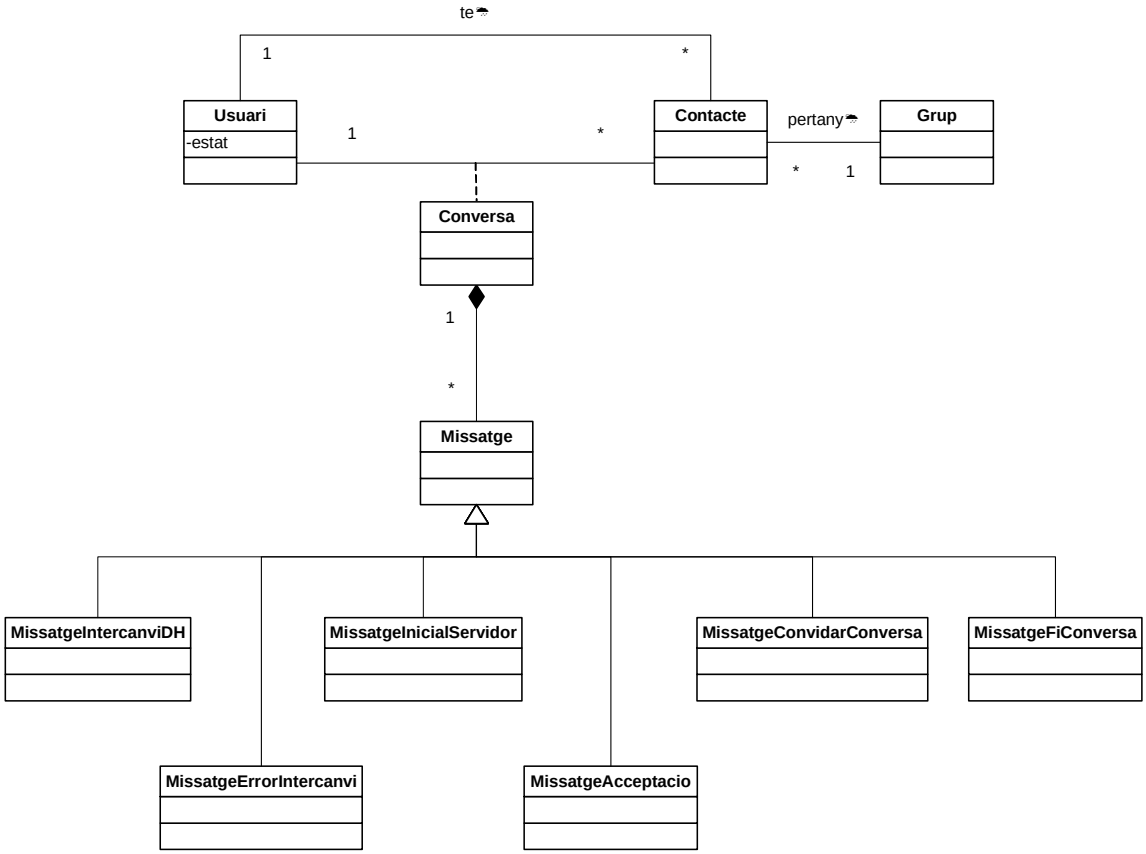

Fig. 34 - Diagrama de classes d'entitat

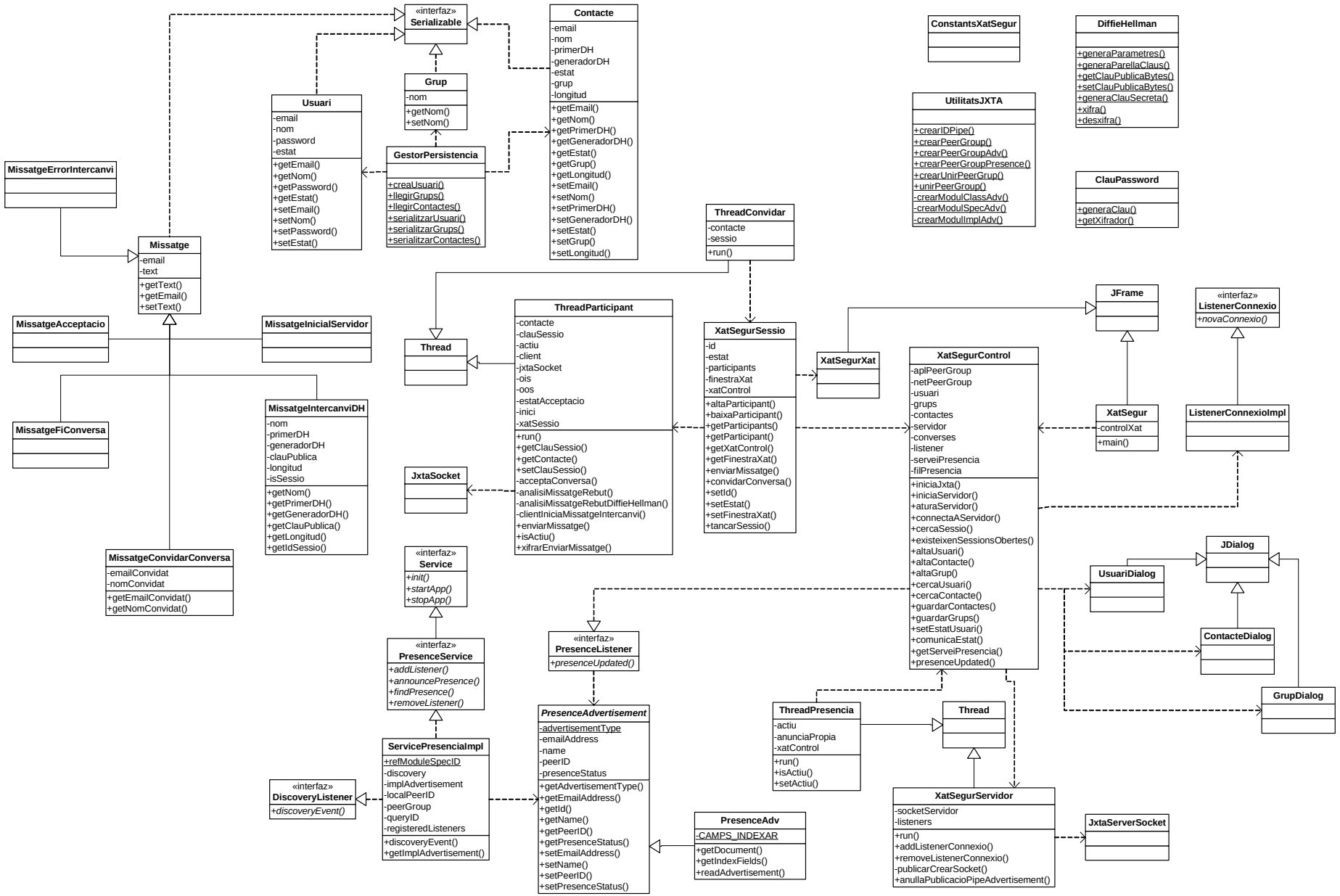

Fig. 35 - Diagrama de classes de disseny.

4.- Aspectes de la implementació.

4.1.- Introducció.

Aquest projecte s'ha desenvolupat en el llenguatge Java emprant la tecnologia JXTA. Les eines utilitzades en el desenvolupament han estat les següents:

- **Entorn de desenvolupament:**
  - **Java 2 SE versió 1.5.0**. Aquesta versió incorpora tot un conjunt de llibreries i eines de seguretat.
  - **NetBeans** versió 5.0[20].
- **D'altres eines i/o llibreries:**
  - **JXTA J2SE 2.3.7**[21], darrera versió de la implementació Java del conjunt de protocols de la plataforma JXTA.
  - **JAVA MAIL 1.4**. En concret, la llibreria **mail.jar** per tal de verificar el format de les adreces de correu electrònic

Les classes de l'aplicació estan organitzades en un conjunt de paquets. Són els següents:

- **`xatsegur.bean`**, que conté les classes d'entitat.
- **`xatsegur.control`**, que conté la classe de control principal –**XatSegurControl**– i el fil que controla l'estat presencial.
- **`xatsegur.dialog`**, amb els formularis d'entrada de dades dels usuaris, grups i contactes.
- **`xatsegur.frame`**, amb la finestra principal i la de xat.
- **`xatsegur.server`**, que conté les classes de la part servidora.
- **`xatsegur.sessions`**, amb les classes que encapsulen les sessions i els participants.
- **`xatsegur.presence`**, amb les classes associades al servei de presència.
- **`xatsegur.utils`**, que conté les classes que implementen els serveis criptogràfics, la persistència, les constants de l'aplicació i d'altres utilitats.
- **`xatsegur.excepcions`**, amb les exempcions pròpies de l'aplicació.

En aquest capítol es descriuen –en algun cas, com en els temes de criptografia, amb més detall– aquelles qüestions que es creuen destacables en la implementació de l'aplicació ***JXTA Xat Segur***, seguint un criteri fixat per la seva relació amb els següents temes:

- Criptografia.
- Gestor de persistència.
- La classe controladora. Identificació d'usuaris.
- Sessions de xat.
- Servei de presència.
- Interfície d'usuari.

4.2.- Implementació dels aspectes criptogràfics.

Els temes criptogràfics s'han resolt emprant les classes incloses en l'API de Java. Aquestes classes s'utilitzen dins el context d'una classe d'utilitat pròpia que es fa servir en els processos d'intercanvi i generació de claus –en

[20] En el pla de treball presentat a l'inici del semestre s'expressava un dubte referent a l'IDE a utilitzar: NetBeans o Eclipse. Finalment, s'ha emprat NetBeans, bàsicament, per la facilitat que presenta –a criteri de l'autor del TFC– a l'hora de generar les interfícies GUI.

[21] Les darreres versions de la implementació Java de JXTA, a més de les llibreries pròpies de la plataforma n'incorporen d'altres, algunes relacionades amb temes de criptografia com la versió 1.31 de la llibreria Bouncy Castle Cryptography. En la implementació de l'aplicació ***JXTA Xat Segur***, però, no s'han emprat aquestes llibreries, fora de l'ús que en fa la implementació de la plataforma.

iniciar-se la sessió de xat–, i en els de xifratge i desxifratge –en el transcurs de la conversa. La taula següent reprodueix el seu diagrama UML –veure apartat 3.2.3 figura 27–:

DiffieHellman
+generaParametres(longitud : int): DHParameterSpec +generaParellaClaus(p : BigInteger, g : BigInteger, long : int): KeyPair +getClauPublicaBytes(kp : KeyPair): byte[] +setClauPublicaBytes(clPublicaB : byte[]): PublicKey +generaClauSecreta(PrivateKey clPr, PublicKey clPub): SecretKey +xifra(txtClar : byte[], clau : SecretKey, algorisme : String): byte[] +desxifra(txtXifrat : byte[], clau : SecretKey, algorisme : String): byte[]

En la implementació del primer dels mètodes, cridat quan es dóna d'alta un contacte, s'han emprat les següents classes i mètodes de l'API de Java:

- **AlgorithmParameterGenerator**: classe del paquet `java.security` que, com indica el seu nom, permet generar un conjunt de paràmetres per a un determinat algorisme. S'instancia un objecte d'aquesta classe per a Diffie-Hellman mitjançant el mètode `getInstance()`, s'inicialitza emprant el mètode `init()`, i es generen els paràmetres a través de `generateParameters()` que retorna un objecte **AlgorithmParameters**.
- **AlgorithmParameters**: s'usa el mètode `getParameterSpec()` d'aquesta classe del paquet `java.security` per a obtenir l'objecte **DHParameterSpec** que retorna el mètode `generaParametres()`.

Els quatre mètodes següents s'utilitzen en el procés d'intercanvi de claus.

⇒ El mètode `generaParellaClaus()` construeix un objecte **DHParameterSpec** a partir dels paràmetres que se li passen, i utilitza els mètodes `getInstance()` i `generateKeyPair()` de la classe **KeyPairGenerator** per a obtenir-ne una instància i generar la parella de claus.

⇒ `getClauPublicaBytes()` retorna una matriu de bytes emprant els serveis dels mètodes `getPublic()` –de la classe **KeyPair**– i `getEncoded()` –de la interfície **PublicKey**–.

⇒ El mètode `setClauPublicaBytes()` retorna un objecte **PublicKey**. Per a fer-ho, utilitza la classe **KeyFactory** de la qual n'instancia un objecte emprant el mètode `getInstance()`. A partir d'aquest objecte, obté la clau pública mitjançant el mètode `generatePublic()`, al què passa un objecte de la classe **X509EncodedKeySpec** obtingut a partir de la clau pública expressada en bytes.

⇒ `getClauPublicaBytes()` utilitza mètodes de la classe **KeyAgreement** del paquet `javax.crypto`. Aquesta classe permet generar una clau secreta a partir del component privat de la clau d'un usuari i de la clau pública de l'altre.

Per a fer-ho, cal:

- obtenir-ne una instància per a l'algorisme de Diffie-Hellman mitjançant el mètode `getInstance()`,
- inicialitzar-la amb la clau privada a través del mètode `init()`,
- indicar el pas final de l'intercanvi trucant a `doPhase()` passant-li la clau pública,
- i generar la clau secreta cridant al mètode `generateSecret()` indicant-li el tipus de clau a generar.

Finalment, per a xifrar i desxifrar s'usen:

⇒ `xifra()` que retorna una matriu de byte que és el resultat de xifrar el text en clar amb la clau que es passa com a paràmetre. Per a fer-ho empra els serveis de la classe **Cipher**. Actua de la forma següent:

- Obté un xifrador mitjançant el mètode `getInstance()`,
- l'inicialitza en mode ENCRYPT emprant el mètode `init()`,
- i l'utilitza per xifrar cridant al mètode `doFinal()`.

⇒ El mètode `desxifra()` treballa de forma anàloga. El canvi més significatiu respecte l'anterior és la forma en què s'inicialitza el xifrador. Ara es fa en mode DECRYPT ja que es pretén emprar el mètode per a desxifrar.

4.3.- Implementació de la persistència.

Cal fer persistents les dades de l'usuari i dels seus contactes, que poden formar grups. Tenint en compte el tipus d'aplicació, així com la conveniència de fer les dades fàcilment transportables, la persistència s'ha implementat d'acord amb el següent:

- Disposar d'un fitxer amb les dades personals de l'usuari. Aquest fitxer es designa en la forma: *nom_usuari.jxs*.
- Un altre fitxer per emmagatzemar les dades dels grups: *nom_usuari.gxs*.
- Un darrer fitxer on es guarden les dades dels contactes, designat *nom_usuari.cxs*.

Aquests tres fitxers s'emmagatzemen en una carpeta amb el nom de l'usuari, situada en el directori corresponent al *peer* –definit pel seu identificador– en la carpeta de l'aplicació. Aquesta es troba dins el directori personal de l'usuari que executa l'aplicació. La figura mostra aquesta estructura.

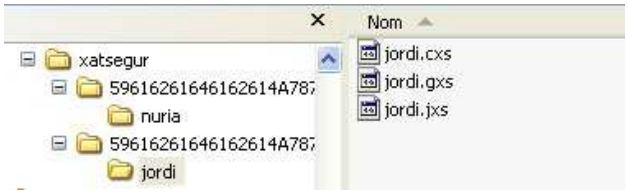

Fig. 36 - Agenda de contactes

La implementació de la persistència s'ha realitzat mitjançant una única classe: **GestorPersistencia**. Els mètodes d'aquesta classe permeten serialitzar els distints objectes –usuari, grups i contactes–, així com recuperar-los de disc. Les dades es guarden xifrades amb una clau obtinguda a partir de la contrasenya de l'usuari, i la mecànica de treball en la serialització és gairebé sempre la mateixa:

- Obtenir un **ObjectOutputStream** associat al fitxer.
- Generar la clau a partir de la contrasenya.
- Obtenir un objecte **SealedObject** xifrant l'objecte a serialitzar amb la clau secreta. L'objecte **SealedObject** encapsula l'objecte que es vol fer persistent de forma serialitzada i xifrada.
- Finalment, emmagatzemar l'objecte en disc.

Naturalment, a l'hora de recuperar les dades el procés és semblant en sentit contrari:

- Cal obtenir un **ObjectInputStream** relacionat amb el fitxer on es troba la informació.
- Generar la clau.
- Recuperar de disc l'objecte **SealedObject** i desxifrar-lo amb la clau secreta per tal d'obtenir l'objecte cercat.

4.4.- La classe controladora. Identificació d'usuaris.

La classe **XatSegurControl** és la classe de control de l'aplicació. En aquest sentit, disposa, entre d'altres, dels atributs que li permeten conèixer la identitat de l'usuari, els seus contactes i les sessions de xat iniciades. Els grups de contactes es guarden en un objecte **SortedSet** que permet fer-ne el recorregut ordenat; els contactes en un **Map** en què la clau és el correu electrònic del contacte, mentre que les sessions s'emmagatzemen en una llista. Com a classe controladora, rep les sol·licituds d'altres classes per a:

- Iniciar el servidor de l'aplicació i aturar-lo.
- Connectar-se a la part servidora d'un altre *peer*.
- Cercar les dades de l'usuari, tant les personals com les dels contactes.
- Realitzar el manteniment d'aquestes dades.

- Fer el manteniment de les sessions.

En la resolució d'aquestes sol·licituds a vegades s'utilitzen els serveis d'altres classes, com és el cas de la persistència.

La classe controladora, com d'altres en l'aplicació, disposa d'un atribut del tipus **PropertyChangeSupport**. Aquesta classe del paquet `java.beans` és una classe d'utilitat que forma part del model de delegació d'esdeveniments de Java. Aquest model permet a les classes publicar esdeveniments i, també, enregistrar-se com a subscriptores dels esdeveniments que es produeixen en d'altres. L'esmentada classe s'utilitza en l'aplicació sempre que es desitja notificar a d'altres –per exemple, les classes que proporcionen les vistes– determinats canvis.

En el cas de la classe controladora, es notifiquen, mitjançant una crida al mètode `firePropertyChange()` associat a l'objecte **PropertyChangeSupport**, els canvis produïts en l'estat presencial de l'usuari i dels seus contactes, a més de qualsevol canvi en els noms dels grups i els contactes. Els objectes que reben notificació dels canvis, és a dir, els que es troben en la llista de *listeners*, són la finestra principal de l'aplicació i les finestres de xat –la subscripció d'aquestes al servei no es realitza, però, des de la classe controladora.

4.4.1.- Identificació d'usuaris en els processos client/servidor.

Ja s'ha esmentat anteriorment que la dada que identifica a un usuari davant la resta d'usuaris és l'adreça de correu. També s'ha indicat que en JXTA tots els recursos d'un *peer* o grup de *peers* es divulguen mitjançant els *advertisements*. En l'aplicació de xat, quan un usuari activa la seva part servidora ha de fer públic l'*advertisement* de la *pipe* sobre la que es crearà el *socket* JXTA. L'anunci conté un identificador de la *pipe*, que es construeix a partir de l'identificador del *peergroup*, del tipus de *pipe* i de l'adreça de correu de l'usuari.

El codi següent mostra el mètode de la classe **UtilitatsJXTA** que crea l'identificador. És trucat des del mètode `iniciaServidor()` de la classe controladora i construeix l'identificador emprant una funció de resum.

```
public static PipeID crearIDPipe(String eMail, PeerGroupID idPeerGroup, String tipus) {
        DigestTool dt = new DigestTool(ConstantsXatSegur.ALG_SHA_256);
        PipeID idPipe = dt.createPipeID((net.jxta.impl.id.UUID.PeerGroupID)idPeerGroup,
                                        eMail, tipus );
        return idPipe;
}
```

De forma anàloga, quan es vol contactar amb un altre *peer*, cal adreçar-se a ell a través d'un procediment semblant. En aquest cas, la crida al mètode anterior és fa des de `creaPublicaPipeContacte()` de la classe de control. Aquest darrer mètode és trucat des de dos llocs: dins la mateixa classe a través de `connectaAServidor()`, i des del mètode `run()` de **ThreadConvidar** quan es vol convidar a una conversa ja iniciada.

4.5.- Sessions i participants.

En l'apartat 3.2.3 s'han indicat les bases del disseny de les sessions de xat. En resum, podem indicar que eren les següents:

- Per la part servidora del *peer*:
  - El mètode `iniciaServidor()` de **XatSegurControl** inicia un nou fil d'execució a través de la classe **XatSegurServidor**, la qual escolta peticions emprant el mètode `accept()` del *socket* JXTA servidor associat a la *pipe* publicada.
  - Quan arriba una petició, el mètode `accept()` retorna un *socket* JXTA. Aquest objecte es passa, després d'instanciar un objecte **XatSegurSessió**, a un nou fil d'execució de la classe **ThreadParticipant** que s'integra en la sessió. A través dels dos *streams* lligats al *socket* esdevé l'intercanvi de claus i s'inicia, si l'intercanvi té èxit, la conversa.
- Quan el *peer* actua com a client i contacta amb un altre, ho fa de la forma següent:
  - Es dirigeix a l'altre *peer* emprant el mètode `connectaAServidor()` de la classe de control. Aquest fet ocasiona la creació d'una sessió i d'un nou fil d'execució que, com en el cas del servidor, s'implementa a través de la classe **ThreadParticipant**.
  - Aquesta classe, crea un *socket* JXTA i, a través de l'*OutputStream*, adreça al servidor un missatge d'inici de connexió.

- En funció de la resposta del servidor, s'iniciarà o no el xat, actuant en aquest sentit de forma anàloga a com s'ha manifestat en cas del servidor.

Per tant, en ambdós casos, ens trobem amb un objecte que representa la sessió i amb un nou fil d'execució que encapsula el contacte amb qui es conversa. A més, la sessió de xat porta associada la finestra des de la què conversarà l'usuari. Aquesta finestra serà visible si l'intercanvi de claus té èxit.

Els aspectes a destacar en la implementació de la classe **ThreadParticipant** són els següents:

- A més dels atributs que emmagatzemen les dades del contacte, les criptogràfiques i les relacionades amb la comunicació, n'existeixen tres associats a la forma d'actuar del fil:
  - **actiu**, que marca l'estat d'activitat;
  - **client**, que indica el paper que pren el fil en la conversa i que marca el missatge inicial a adreçar a l'altra part;
  - **inici**, que permet distingir el primer missatge de la resta.

  També cal esmentar la utilitat de l'atribut **tempsInici**: és la marca de temps que permet controlar el *time-out* associat a l'intercanvi de claus.
- Igual que per a la classe controladora, existeix un atribut del tipus **PropertyChangeSupport**. En aquest cas, els *listeners* són la sessió i la finestra de xat, als què així es pot comunicar tot el què succeeix en el fil.
- Una vegada ha estat activat el fil, aquest treballa de la forma següent:
  - En la primer execució adreça un missatge inicial a l'altra part. Aquest missatge és important, sobretot, en el cas del client, ja que inicia el procés d'intercanvi de claus. En el cas del servidor, s'aprofita aquest missatge per a obrir el canal de comunicació amb l'altra banda.

    Aquesta darrera qüestió és clau ja que la comunicació implementada aprofita la possibilitat que ofereix Java d'enviar objectes serialitzats a través de la xarxa. Ara bé, per a què funcioni adequadament la transmissió, cal establir de forma adient els *object-stream* utilitzats.
  - El fil està permanentment escoltant a l'altra part i, en arribar un missatge, l'analitza.
  - Si es produeix una excepció de *time-out* associada al *socket* i la conversa ha estat acceptada, l'error no es té en compte i continua el fil actiu[22].
  - Quan es tanca el fil, es dóna de baixa el participant en la sessió i, si cal, s'envia una notificació a l'objecte sessió per a què es tanqui si no té més fils actius.
- L'anàlisi esmentada anteriorment es fa un funció del tipus de missatge. Si el missatge rebut és una instància de la classe **MissatgeDiffieHellman** i el fil correspon a un servidor, se segueixen els passos següents:
  - Es comprova si alguna sessió iniciada té el mateix identificador. Si és així, s'esborra el fil de la sessió actual i es passa a la sessió trobada.
  - Es comprova si el client es troba en la llista de contactes. Si és així, sempre s'accepta el xat. Si no s'hi troba, s'adreça un missatge a l'usuari on se li demana que indiqui si accepta o no la conversa. Si no l'accepta, es tanca el fil.
  - Tot seguit, el servidor genera la seva parella de claus, extreu la clau pública del client i adreça a aquest un missatge del tipus **MissatgeDiffieHellman** per a completar l'intercanvi de claus.

  Si el fil és client, en rebre aquest missatge, extreu la clau pública del servidor. Ambdós interlocutors poden calcular, llavors, la clau de sessió i adreçar a l'altra part un missatge d'acceptació.
- Com ja s'ha esmentar en parlar del disseny de l'aplicació, un usuari pot convidar-ne d'altres a participar en una conversa iniciada. Quan ho fa, envia a la resta de participants una instància d'un missatge **MissatgeConvidarConversa** per a què també contactin amb el convidat. En

[22] Teòricament, la classe **JxtaSocket** disposa del mètode `setSoTimeout()` que permet posar un *time-out* indefinit. Aquest mètode, però, no acaba de funcionar correctament d'acord amb les proves realitzades. Aquest fet, juntament amb què no té sentit una espera infinita en aquest cas, ha portat a adoptar la solució esmentada.

rebre aquest missatge, un peer comprova en la seva llista de contactes si existeix el convidat. Si existeix, el convida. Si no és així, crea un contacte temporal per a que pugui participar en el xat.

- Quan un usuari tanca una sessió de xat, envia als altres participants una instància de la classe **MissatgeFiConversa**. Per tant, quan l'interlocutor rep aquest missatge, tanca el fil i ho indica a la sessió.
- Si l'intercanvi de claus no té èxit, qui ho detecta envia a l'altre un missatge indicant que s'ha produït un error en l'intercanvi i tanca el fil d'execució. L'altra part, en rebre el missatge, també tanca el seu fil. Ambdós comuniquen el problema a la sessió, que es tanca si no es detecta la presència de cap participant actiu.
- Naturalment, si de l'anàlisi del missatge rebut resulta que és un missatge normal, es desxifra i es notifica a la finestra de xat que el mostri a l'usuari, la qual cosa es fa, com les altres comunicacions, mitjançant el mètode `firePropertyChange()` de la classe **PropertyChangeSupport**.

Respecte la classe **XatSegurSessio**, els aspectes més destacables a comentar són els següents:

- La classe implementa la interfície **PropertyChangeListener** en estar enregistrada com a *listener* dels canvis que li comuniquen els fils que encapsulen els participants en la sessió.
- Per aquest motiu, la classe ha d'implementar el mètode `propertyChange()` que declara l'esmentada interfície.

  Mitjançant aquest mètode, cridat cada vegada que es produeix un canvi en les propietats escollides dels objectes participants, la sessió rep un objecte **PropertyChangeEvent**. Aquest esdeveniment conté informació del canvi produït i permet, un cop analitzat, prendre les decisions adequades.

4.6.- Els servei de presència.

Al final del paràgraf 3.2.3 dedicat al disseny de les interaccions client-servidor, s'ha fet esment al servei que permet conèixer l'estat presencial dels usuaris. Aquest servei es fonamenta en un dels serveis oferts per la plataforma JXTA, el **Discovery Service**, i el disseny i la implementació de les classes –situades en el paquet `xatsegur.presence`– s'ha pres del llibre sobre JXTA publicat per Brendan Wilson. L'estructura del paquet és la següent:

- Una classe abstracta, **PresenceAdvertisement**, que defineix l'*advertisement* emprat per anunciar la presència.
- La classe **PresenceAdv**, hereva de l'anterior, que permet instanciar els anuncis, formatant-los com a documents XML, i analitzar-los.
- Una interfície, **PresenceListener**, que defineix els mètodes que haurà d'implementar un *listener* associat al servei.
- El servei pròpiament dit, definit com és usual en la plataforma: una interfície que declara els mètodes, **PresenceService**, i la seva implementació, **PresenceServiceImpl**.

Com es comentava en el citat apartat, l'autor del TFC s'ha limitat, en general, a canviar lleugerament el codi per tal d'ajustar-lo a l'aplicació desenvolupada. En algun cas, però, ha calgut fer alguna correcció per assegurar el funcionament. Els canvis han estat els següents:

- En la classe **PresenceAdv**, definir l'atribut de classe `CAMPS_INDEXAR` i el mètode accessor `getIndexFields()`. Sense aquests canvis, el sistema no troba l'*advertisement* publicat i el servei no funciona.
- Simplificar el mètode `findPresence()` en la classe que implementa el servei: n'hi ha prou en fer una descoberta remota i deixar que `discoveryEvent()` faci la resta. Si es fa també una cerca local, es fa la feina per duplicat.

Per tal de poder utilitzar el servei, cal associar-lo al *peer group* de l'aplicació, la qual cosa es fa quan es crea el grup i el *peer* s'uneix a ell. El mètode que acaba realitzant aquesta feina és `crearPeerGroupPresence()` que es troba en la classe **UtilitatsJXTA**. Aquest mètode realitza totes les accions que requereix la plataforma per a especificar un servei –emprant per a algunes mètodes auxiliars:

- crear i publicar l'*advertisement* del mòdul de classe que declara el servei,
- fer el mateix amb l'especificació,

- i amb la seva implementació.

En el darrer cas, cal tenir present una qüestió: és necessari l'element del document XML de l'*advertisement* que conté la URI que indica on es troba el codi amb la implementació, tot i que en el text de Brendan Wilson apareix com opcional.

Un cop s'ha associat el servei al grup, alguna classe de l'aplicació s'ha d'enregistrar com a *listener* i implementar la interfície **PresenceListener** per tal de poder rebre notificacions del servei. La classe que ho fa és la controladora quan s'inicia la plataforma.

4.7.- La interfície d'usuari.

Les classes de la interfície d'usuari estan situades en dos paquets: `xatsegur.dialog` i `xatsegur.frame`. El primer d'ells conté les finestres que permeten fer el manteniment de les dades de l'usuari, els grups i els contactes, mentre que el segon conté la finestra principal de l'aplicació i la finestra de xat.

En la implementació s'han aprofitat les prestacions de disseny gràfic que ofereix l'IDE emprat –NetBeans–, i es podrien destacar els següents aspectes:

- En el què es refereix a les classes del paquet `xatsegur.dialog`:
  - Totes deriven de la classe **JDialog** del paquet `javax.swing`.
  - La comunicació entre el formulari i la classe que l'ha instanciat s'estableix a base d'un valor de retorn –indicador de si s'han acceptat o no les dades introduïdes– i de mètodes accessors que permeten obtenir tant les dades com el valor de retorn.
  - Cada formulari disposa de les eines de validació oportunes: comprovació de valors no nuls, format correcta de les adreces de correu,...
- Respecte al paquet `xatsegur.frame`:
  - Les dues classes principals del paquet són peces bàsiques en el què, en el model MVC, anomenem la vista. Qualsevol canvi en el model, alta d'un contacte, canvi en l'estat de l'usuari o d'un contacte, ..., cal reflectir-lo en la vista.

    La forma de fer-ho ja s'ha emprat en d'altres parts de l'aplicació: les finestres es constitueixen com a *listeners* dels canvis que es produeixen en els objectes que interessen –usuari, llistes de contactes, participants,... –; implementen la interfície **PropertyChangeListener** i defineixen el mètode `propertyChange()`, amb la qual cosa obtenen informació dels canvis a través de l'objecte **PropertyChangeEvent** que reben en executar-se el mètode.
  - Ambdues classes disposen dels menús –alguns contextuals– que permeten a l'usuari escollir l'acció que vol que realitza l'aplicació.

5.- Manuals d'usuari.

5.1.- Instal·lació i execució de l'aplicació.

Per instal·lar l'aplicació ***JXTA Xat Segur***, només cal descomprimir l'arxiu **JxtaXatSegur.zip** en el directori escollit com arrel de l'aplicació. En fer-ho, apareixeran dins la carpeta **dist** els fitxers necessaris per executar el programa:

- L'arxiu **JAR** que conté les classes compilades: **JxtaXatSegur.jar**.
- La carpeta **lib** que conté les llibreries que utilitza l'aplicació –entre elles les de la plataforma JXTA.

La figura mostra aquest fet.

Fig. 37 - Resultat instal·lació JXTA Xat Segur

Per tal de posar en marxa el programa, cal situar-se en el directori de la instal·lació i executar la comanda:

javaw –jar "JxtaXatSegur.jar"

o bé, situar-se en l'arrel i executar:

javaw –jar "cami_fins_el_fitxer\JxtaXatSegur.jar"

En fer-ho, apareixerà la finestra que indica que l'aplicació s'està iniciant:

Fig. 38 - Pantalla inicial

5.2.- Manual d'ús.

5.2.1.- Configuració de la plataforma JXTA.

Per a què el programa funcioni adequadament, la plataforma JXTA ha d'estar correctament configurada. La implementació Java de JXTA proporciona una eina que es posa en marxa la primera vegada que s'inicia la plataforma en un determinat directori.

Com es pot comprovar en la figura 39, la configuració abarca tres aspectes:

- El bàsic, en el qual cal indicar el nom del *peer* i proporcionar una contrasenya.

  En validar la configuració, es crea un certificat i una clau privada associats al *peer*, a menys que s'importi un certificat existent.
- L'avançat, on es configura:
  - Si el *peer* actua com a *rendezvous*, *relay* i/o *proxy*.
  - Els paràmetres de TCP, entre ells el port emprat en la comunicació.
  - Els paràmetres HTTP.
- Els *rendezvous* i *relays peers* que usarà el *peer*.

  Es pot proporcionar una relació de *peers*, o bé optar per indicar les URI on es poden trobar les llistes actualitzades. El més usual és optar per la segona opció.

La figura esmentada mostra la configuració per un *peer* normal que es troba, per exemple, darrera un tallafocs i/o encaminador NAT[23].

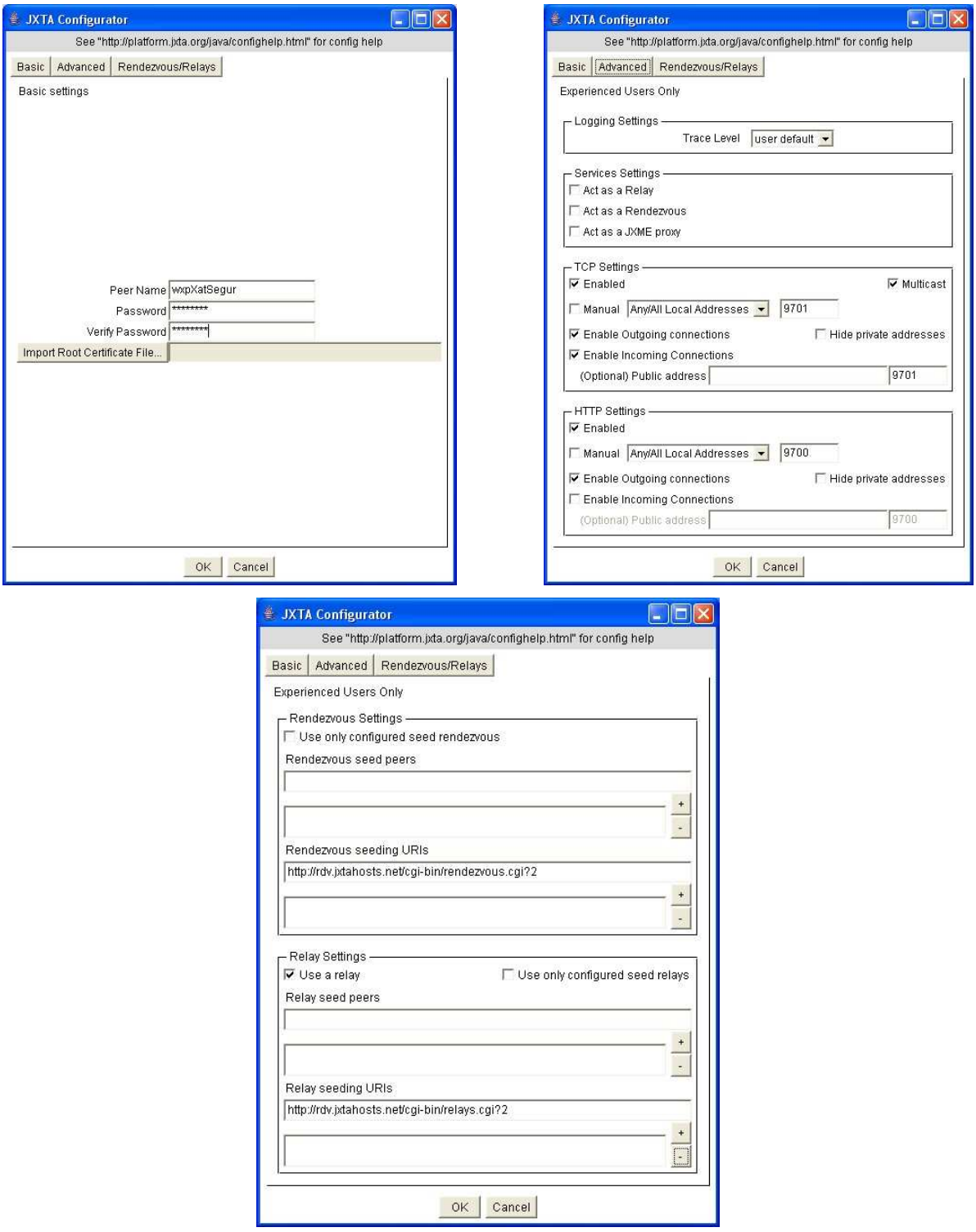

Fig. 39 - Configuració JXTA

Cal esmentar que si es desitja provar l'aplicació establint una conversa entre dos usuaris que estan en la mateixa màquina, cal prendre les següents precaucions:

- Instal·lar i executar l'aplicació en dos directoris distints.

[23] Com el document actual s'ha de mantenir dins uns límits raonables, en d'altres situacions, es recomanar consultar a l'adreça http://platform.jxta.org/java/confighelp.html.

- Configurar per a cada instància números de port diferents tant per TCP com per HTTP.

Quan la plataforma ha estat configurada, es crea en el directori on s'executa l'aplicació la carpeta **.jxta** que conté:

- El fitxer **PlatformConfig**, document XML amb la configuració de la plataforma.
- El directori **cm** on es situa la cau del *peer* configurat, amb una carpeta per cadascun dels grups als què pertany el *peer*.

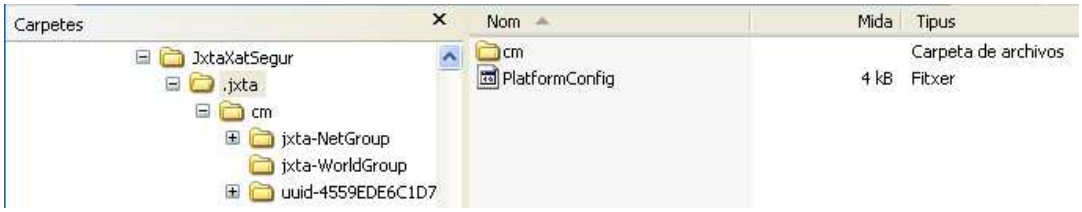

Fig. 40 - Estructura de directoris de la configuració JXTA.

Finalment, cal indicar que un cop la plataforma ha estat configurada es pot reconfigurar de dues formes: esborrant l'estructura anterior, o bé creant en el directori **.jxta** un fitxer buit amb el nom de **reconf**.

5.2.2.- Inici de l'aplicació. Alta de l'usuari.

Quan un usuari inicia l'aplicació, li apareix al cap d'una estona –si té adequadament configurada la plataforma JXTA– la pantalla de la figura adjunta per tal de validar-se.

Fig. 41 - Connexió usuari

Si l'usuari ja ha estat donat d'alta en el sistema, introdueix les dades de connexió i se li presenta la pantalla principal de l'aplicació amb la seva relació de contactes.

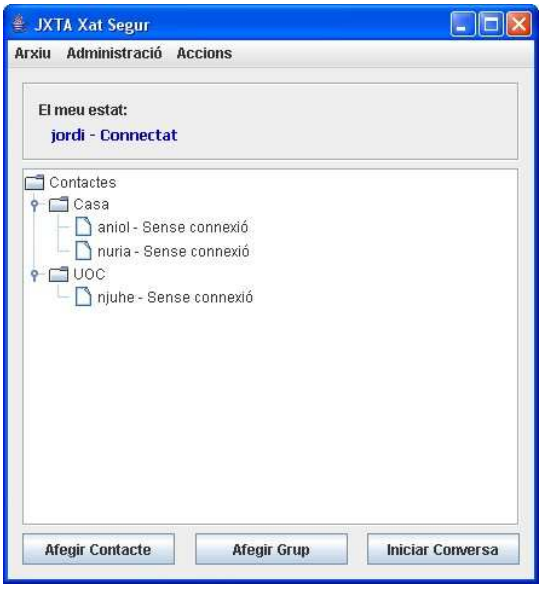

Fig. 42 - Finestra principal

En cas de ser la primera vegada que interactua amb l'aplicació, en intentar validar les seves dades, el sistema detecta que no existeix l'usuari i li ofereix la possibilitat d'alta.

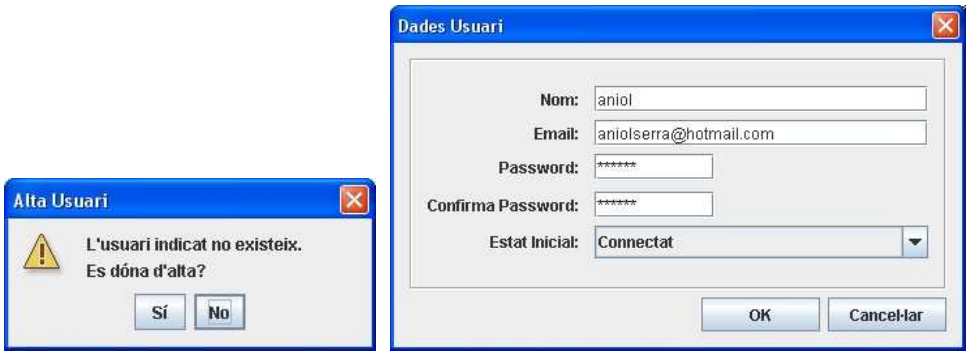

Fig. 43 - Alta usuari

El programa comprova el format de l'adreça i les contrasenyes i, si és el cas, dóna d'alta l'usuari, creant en la carpeta del peer un directori amb el nom de l'usuari on guarda les seves dades personals i la seva agenda de contactes. La informació s'emmagatzema xifrada amb una clau que deriva de la contrasenya i es pot transportar, si es desitja, a una altra màquina: només cal traslladar la carpeta amb el nom de l'usuari.

5.2.3.- Entorn de treball: finestra principal.

La finestra principal de l'aplicació, figura 44, conté a més dels botons Afegir Contacte, Afegir Grup i Iniciar Conversa, tres menús a través dels quals l'usuari pot emprendre tot un conjunt d'accions: modificar les dades personals, canviar el seu estat presencial, salvaguardar la llista de contactes, realitzar el manteniment dels seus contactes i iniciar una conversa.

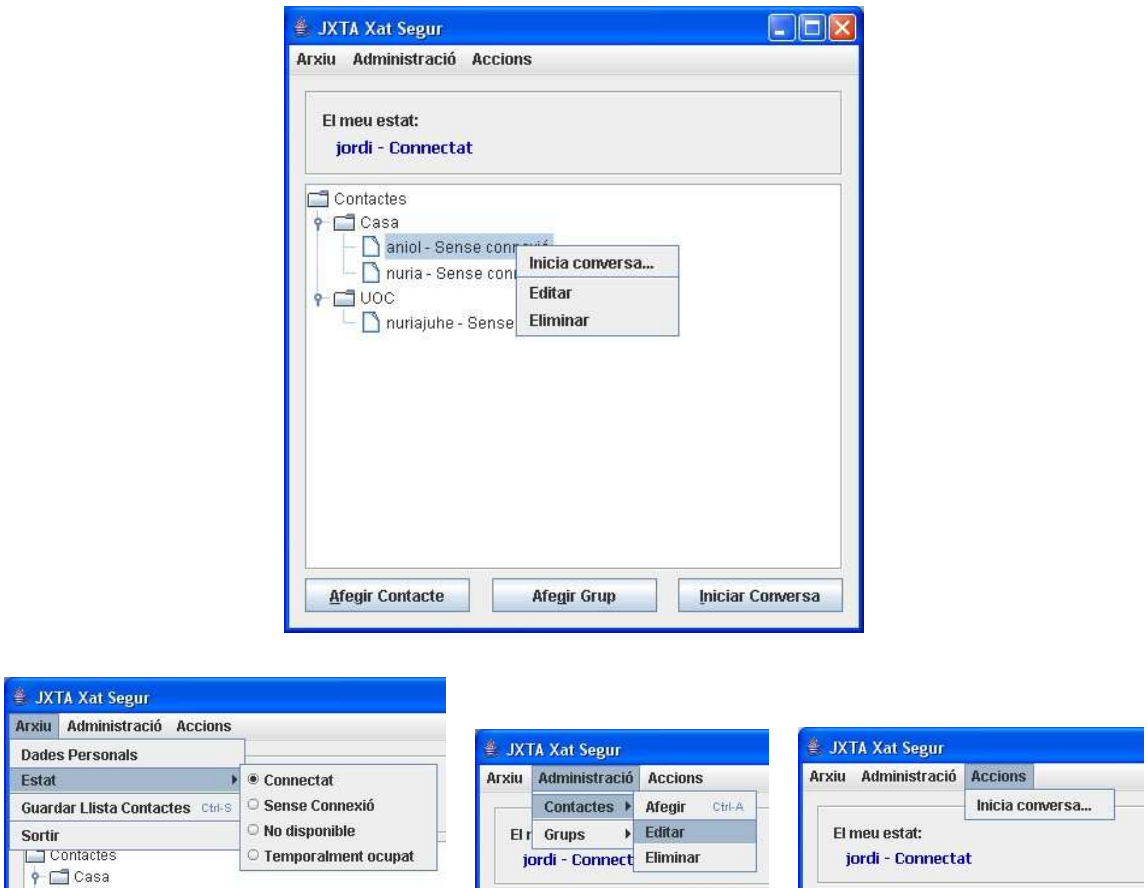

Fig. 44 - Entorn de treball

A més, si escull un element en l'arbre de contactes, es desplega un menú contextual que li permet editar o esborrar l'element, i iniciar una sessió de xat quan l'element elegit és un contacte.

5.2.4.- Manteniment de grups i contactes.

Per donar d'alta un grup, només cal clicar en el botó Afegir Grup, o bé escollir en el menú Administració l'opció Grups i en ella Afegir. En fer-ho, apareixerà la finestra que es mostra en la figura 45. Introduïm el nom del grup i premem OK. El grup apareixerà tot seguit en la llista de contactes.

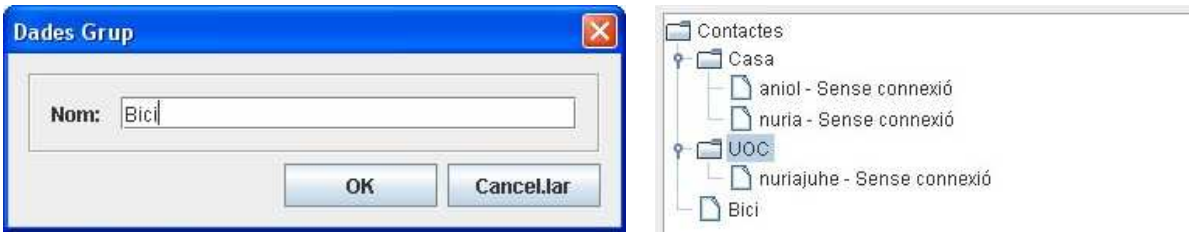

Fig. 45 - Alta grup

Si el què es vol és donar d'alta un contacte, es fa clic en Afegir Contacte o s'escull Afegir dins el submenú Contactes del menú Administració, amb la qual cosa el sistema presenta el formulari d'alta. S'omplen les dades dels distints camps i es clica en OK, fent que causa que el contacte es doni d'alta. En algun cas, es pot observar que el sistema trigui una estona en fer-ho, fet que s'explica perquè en el procés d'alta es produeix la generació dels paràmetres de l'intercanvi de claus –nombre primer i polinomi generador.

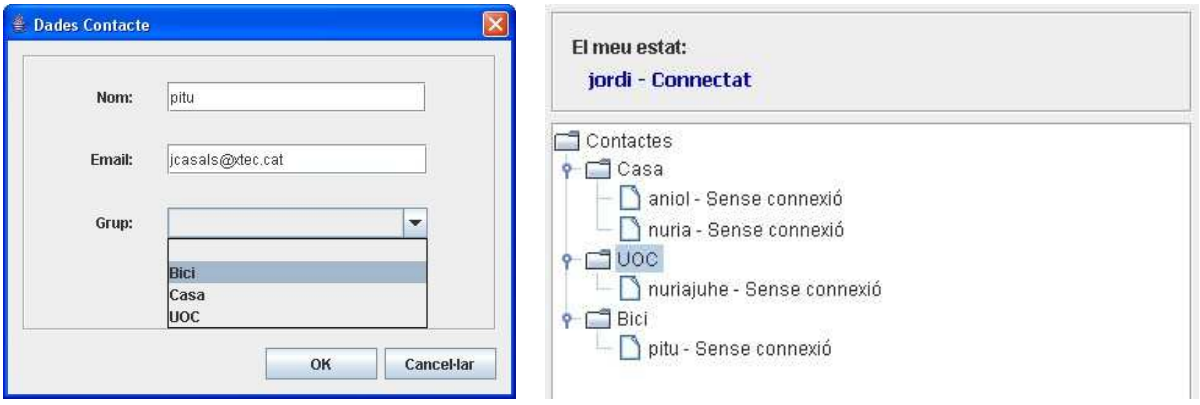

Fig. 46 - Alta contacte

Quan es desitja editar o esborrar un grup o un contacte, la forma d'actuar és la següent:

- Escollir l'element en la llista de contactes –si és mitjançant el botó dret del ratolí, apareix el menú contextual.
- Elegir, a través del menú Administració o del menú contextual, l'opció desitjada.
- En cas de voler editar, el sistema presenta el formulari adequat al tipus d'objecte escollit: només cal modificar les dades i validar els canvis.
- Si es vol esborrar l'element, el sistema demana confirmació. En confirmar, esborra el grup o el contacte.

Tot seguit els canvis es reflecteixen en les finestres obertes. Cal tenir present, però, una qüestió important: els canvis es produeixen només en la memòria del sistema i es fan persistents en tancar l'aplicació. Si es desitja desar en disc els canvis introduïts, cal fer-ho emprant l'opció Guardar Llista Contactes en el menú Arxiu.

5.2.5.- Sessions de xat.

5.2.5.1.- Iniciar conversa.

Quan es vol iniciar una conversa, només cal escollir un contacte en la llista i clicar en el botó Iniciar Conversa, o bé escollir l'opció del mateix nom en el menú Accions o en el menú contextual –que apareix si hem elegit el contacte emprant el botó dret del ratolí. Després de breus instants, en què s'obren els canal de comunicació i es produeix l'intercanvi de claus, apareix la finestra de xat. La figura 47 mostra les dues pantalles corresponents a una sessió entre dos usuaris que han establert contacte.

El missatge que es vol enviar a l'altra part s'introdueix en l'àrea de text que es troba sobre el botó Enviar. En acabar, es clica en aquest –o es prem Alt-E– amb la qual cosa el missatge es xifra i s'adreça als participants en la sessió.

La finestra de xat conté, també, dos menús, Arxiu i Accions, a través dels quals l'usuari pot: emmagatzemar la conversa, tancar la sessió, iniciar una nova sessió o bé convidar a un altre contacte a participar en la conversa. A més, en clicar en un element de la llista de contactes, apareix un menú contextual a través del qual es poden realitzar les dues darreres accions esmentades. La figura 48 permet observar tot el què s'acaba d'indicar.

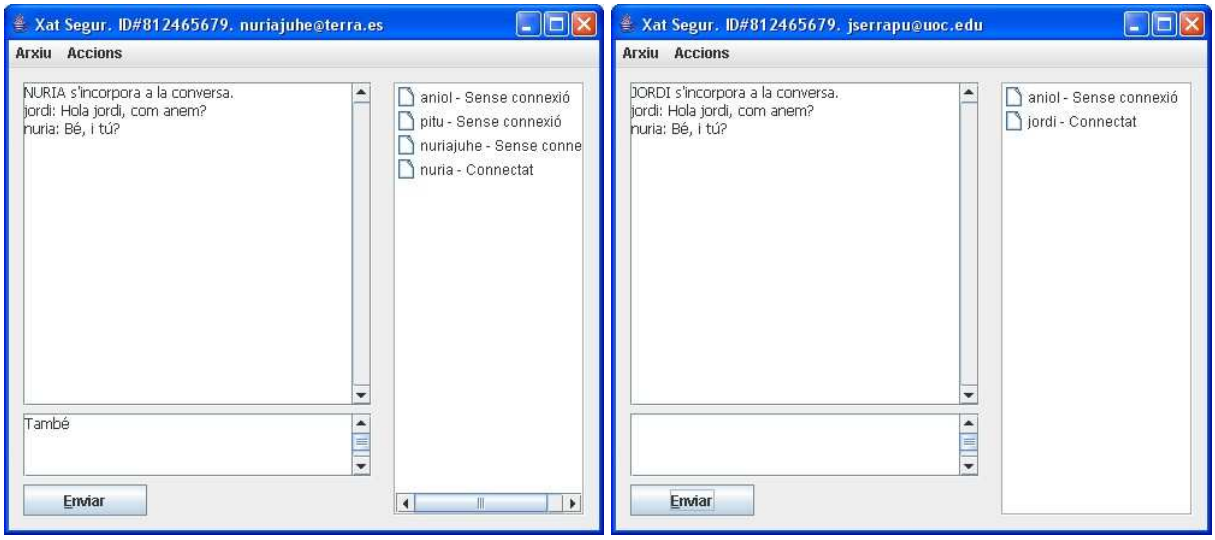

Fig. 47 - Sessió de xat

Fig. 48 - Opcions finestra de xat

5.2.5.2.- Iniciar un nou xat. Convidar a un xat.

Tot seguit es mostra el resultat d'haver escollit l'opció Inicia conversa. Es pot observar com es mantenen dues converses a través de dues finestres independents: la de l'esquerra mostra la nova conversa, mentre que la de la dreta mostra l'estat de la conversa iniciada anteriorment.

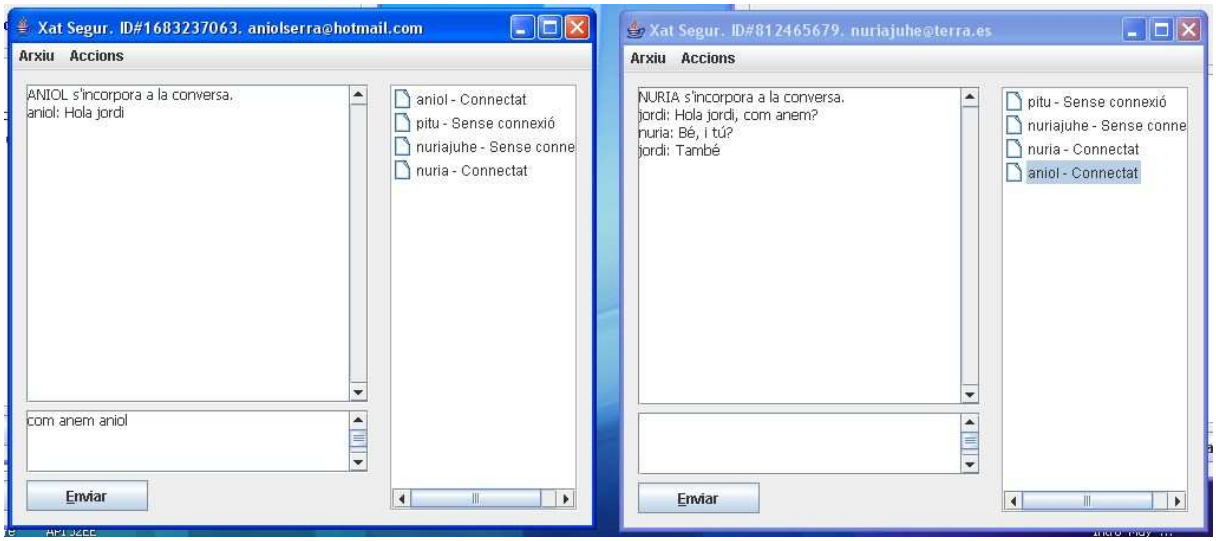

Fig. 49 - Dues converses simultànies

Com s'ha indicat, la finestra de xat presenta la possibilitat de convidar a un altre contacte a una conversa ja iniciada. Per a fer-ho, un dels participants en la sessió convida al nou participant, la qual cosa causa que automàticament la resta de participants faci el mateix ja que el programa genera els missatges de convit

oportuns. La figura 50 presenta la concreció d'aquest fet: en *jordi* convida a l'*aniol* a la conversa que manté amb la *núria*, amb la qual cosa al cap d'una estona el nou participant s'incorpori a la conversa.

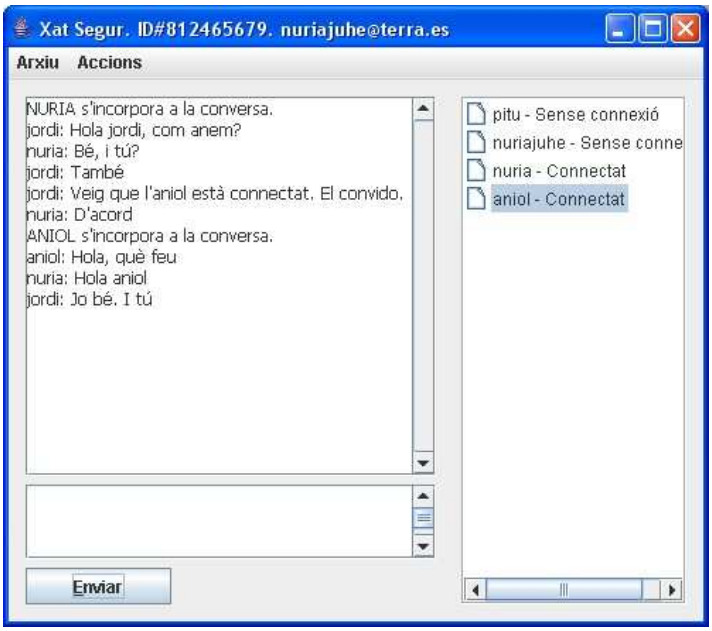

Fig. 50- Conversa amb tres participants

5.2.5.3.- Guardar i tancar sessions.

Si una sessió de xat es vol salvar en disc, es pot fer mitjançant les opcions Desa i Anomena i desa... del menú Arxiu. La segona mostra sempre un quadre de diàleg que permet escollir el lloc i el nom del fitxer de salvaguarda, mentre que la primera només mostra el quadre si el sistema no té la referència del fitxer: si aquesta existeix, emmagatzema el contingut de la conversa en el fitxer referenciat.

Finalment, per a tancar una sessió de xat tan sols cal elegir l'opció Tanca Sessió del mateix menú. En fer-ho, s'adreça als altres participants un missatge de fi de conversa i es dóna aquesta per finalitzada. La figura següent mostra com veu un usuari com van tancant les sessions els altres participants en la conversa.

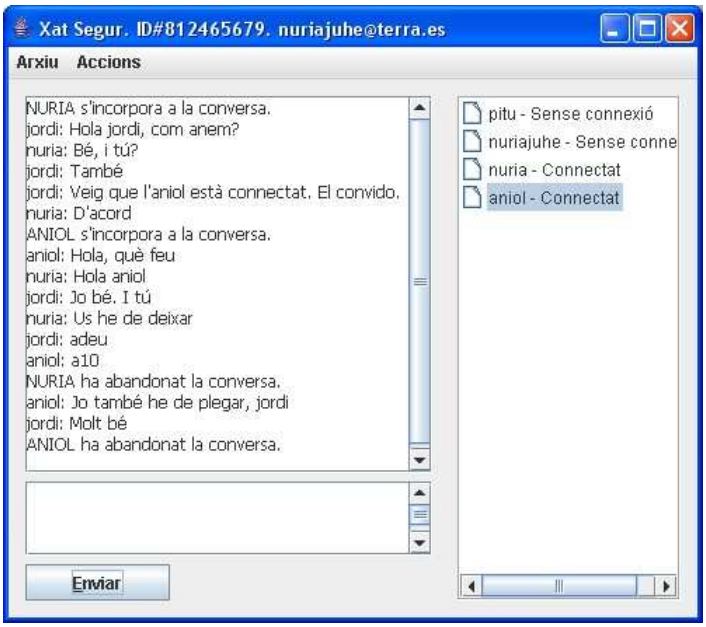

Fig. 51 - Tancar sessió

6.- Proves de funcionament.

Els aspectes desenvolupats han estat tots provats i s'ha comprovat que funcionen correctament. L'apartat anterior proporciona mostres del funcionament de l'aplicació, per la qual cosa en aquest ens limitarem a indicar quines han estat les pràctiques de comprovació realitzades, així com a mostrar d'altres resultats obtinguts.

Les proves efectuades han estat les següents:

- S'ha comprovat que la generació i la transmissió de les claus funciona correctament, així com els processos de xifratge i desxifratge. Aquestes proves s'han realitzat amb l'ajut de les eines de depuració que incorpora el IDE emprat en el desenvolupament (veure figures 53 a 55).
- També s'ha comprovat que es generen de forma correcte tant les dades de l'usuari com les dels seus contactes, i que aquestes s'emmagatzemen xifrades en els fitxers que contenen les dades associades a un determinat usuari –dades personals, grups i contactes–.
- S'ha provat que el funcionament de la interfície d'usuari pel que es refereix al manteniment –modificacions i baixes– de les llistes de contactes és l'esperat.
- S'ha comprovat que l'aplicació funciona en l'entorn d'una única màquina. Per a poder realitzar aquesta prova, ja s'ha indicat que, d'acord amb els requeriments de la plataforma JXTA, cal tenir l'aplicació instal·lada en dos directoris diferents i configurar-la de forma que empri ports distints.
- També s'ha comprovat que funciona en l'entorn d'una LAN.
- En ambdós casos, s'ha comprovat que és possible mantenir més d'una conversa, i que poden participar més de dos participants en un mateix xat.
- S'ha comprovat que el funcionament del servei de presència dels usuaris és correcte.

Les figures que segueixen permeten observar resultats de proves realitzades que no s'han mostrat en el capítol anterior. La primera figura visualitza el funcionament del servei de presència. Les finestres principals de dos usuaris connectats mostren el seu estat presencial i els dels seus contactes.

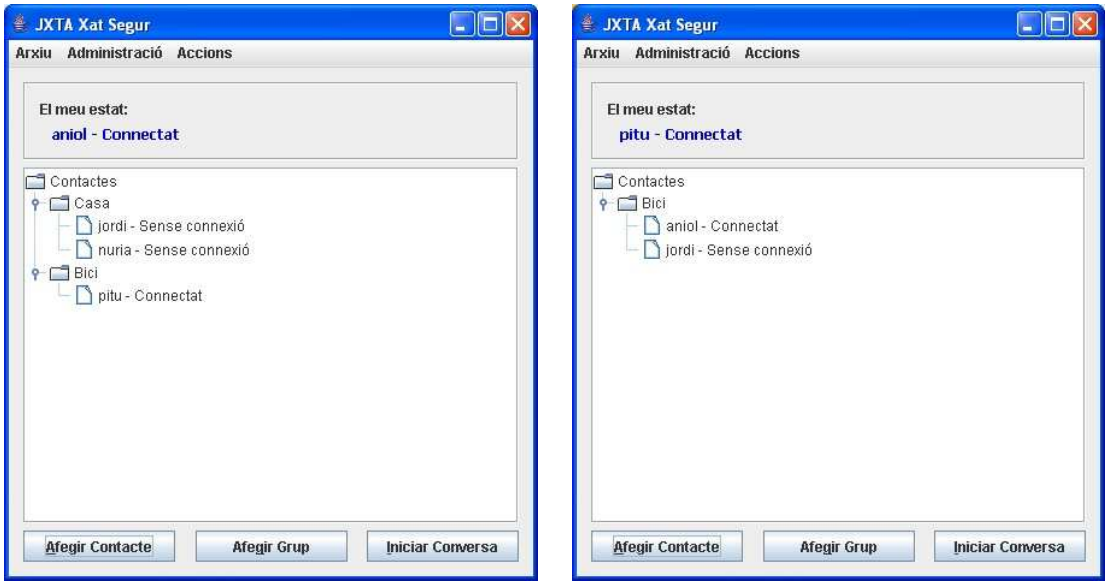

Fig. 52 - Estat presencial

Quan un usuari inicia conversa amb un altre, esdevé primerament l'intercanvi de claus. Les figures següents, una obtinguda des de l'IDE NetBeans –corresponent a l'usuari *pitu* (A)– i l'altre des de Eclipse –on executa el xat l'usuari *aniol* (B)–, mostren la clau de sessió que s'ha obtingut com a producte de l'intercanvi.

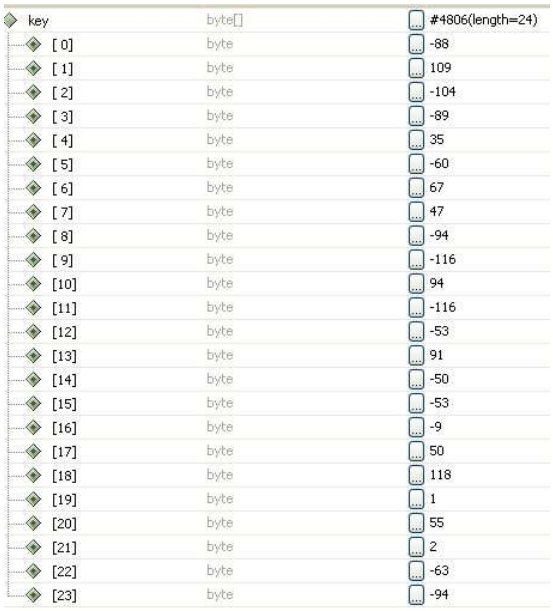

Fig. 53 - Clau obtinguda per l'usuari A

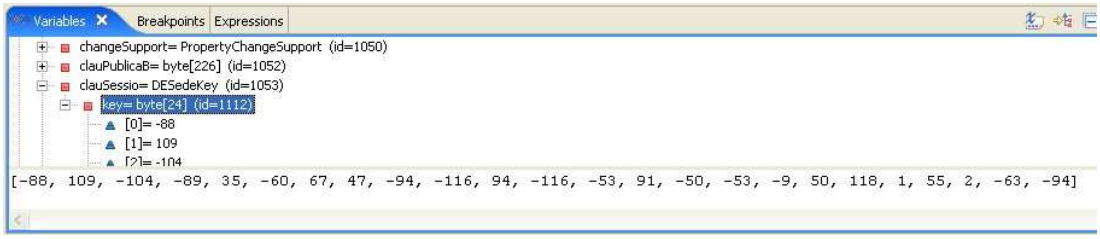

Fig. 54 - Clau obtinguda per l'usuari B

Si tot seguit, en *pitu* envia a l'*aniol* el missatge "Hola aniol, com estàs?", aquest missatge es xifra amb la clau de sessió que s'han intercanviat i s'envia xifrat. La part superior de la figura 55 mostra el text abans i després de ser xifrat en el mètode `xifrarEnviarMissatge()` de la classe **ThreadParticipant**, mentre que en la part inferior de la figura s'observa el missatge que ha rebut l'*aniol* abans de ser desxifrat i després de ser-ho –dins el mètode `analisiMissatgeRebut()`.

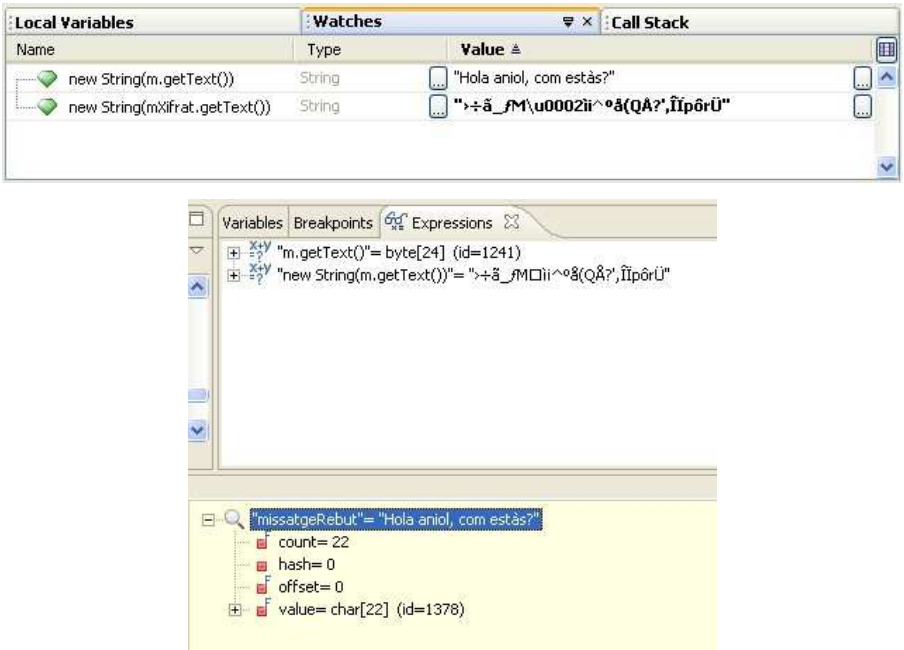

Fig. 55 - Resultat de xifrar i desxifrar un missatge

7.- Conclusions i comentaris finals.

Com a conclusió final podem indicar que gairebé tots els objectius que es van plantejar en iniciar el TFC –recollits en el Pla de treball– s'han pogut complir. El producte obtingut satisfà les funcionalitats previstes i, allò què en aquell moment era una decisió a prendre –emprar o no JXTA–, ha acabat essent un aspecte fonamental del treball.

El descobriment i l'anàlisi d'aquesta plataforma ha esmerçat molts dels esforços dedicats al treball. Ha valgut, però, la pena, tot i que en algun cas –en concret, el servei de presència– ha costat bastant trobar quin era l'origen del problema i, per tant, la via de solució.

En la segona de les PAC s'esmentava que s'intentaria fer una prova del producte connectant a dos usuaris via Internet. Finalment, no ha estat possible realitzar aquesta prova que, per altra banda, és on es podria verificar la plena utilitat de la plataforma. Hi ha, per tant, una sèrie de qüestions sobre JXTA que resten pendents de comprovació[24]. El treball hauria de continuar en aquesta línia si no fos que el temps s'ha exhaurit. A més, l'experiència viscuda indica que, en general, cada nou pas realitzat sobre la plataforma, ha necessitat de esforç i de temps –quan s'han presentat algunes dificultats en les primeres proves– abans no s'ha acabat d'ajustar el producte.

Respecte a JXTA, cal indicar que com altres projectes que es nodreixen de les aportacions voluntàries de múltiples desenvolupadors, a vegades sembla avançar a batzegades. Tot i que les versions van apareixent a un bon ritme –en iniciar el TFC feia poc que estava disponible la versió 2.3.6 i el treball ha acabat implementant-se a partir de la versió 2.3.7–, hi ha projectes al voltant de la plataforma que no acaben de quallar. En aquest sentit, cal esmentar que des dels primers temps de la plataforma hi ha iniciat un projecte anomenat precisament *presence*[25] amb l'objectiu d'ajudar en la descoberta de l'estat d'altres *peers* en la xarxa. En un primer moment es va pensar en aprofitar aquest projecte en el control de l'estat presencial. Aquest projecte, però, fa temps que està aturat.

Els comentaris finals es referiran a aquelles millores que caldria incorporar en el programa i que no s'han pogut implementar degut al temps disponible.

S'ha esmentat en l'apartat 2.1.2 que el procés d'intercanvi de claus de Diffie-Hellman és vulnerable a l'atac de l'home al mig. Cal recordar que, en aquest atac, quan dos usuaris **A** i **B** realitzen l'intercanvi, un tercer usuari **C** intercepta el component públic de la clau de **A** ($\mathbf{M_a}$) i el substitueix per un component propi ($\mathbf{M'_a}$). El mateix fa amb l'altre usuari **B**. Així aconsegueix tenir una clau compartida amb cadascun dels usuaris i, per tant, pot interceptar –i manipular si ho desitja– el missatges que **A** i **B** s'adrecen a través d'un canal que creuen protegit. La vulnerabilitat es deu a què els participants no s'autentiquen. Per tant, una possible prevenció davant l'atac és la utilització de signatures digitals amb l'objectiu d'autenticar els interlocutors.

Una forma plausible de portar a la pràctica aquest procés d'autenticació amb l'ús de signatures digitals podria ser la següent:

- Crear una **autoritat de certificació** (CA –Certification Authority–) associada al *peergroup* **JXTA Xat Segur**. Aquesta CA es vincularia a una adreça IP pública des de la qual, via web, es descarregaria l'aplicació.
- A l'hora de descarregar el programa, el futur usuari hauria de proporcionar l'adreça de correu que posteriorment faria servir com a identificadora i, en validar la petició de descàrrega, la CA emetria un certificat digital associat a l'adreça. Aquest certificat i la clau privada relacionada s'instal·larien en l'equip de l'usuari.
- Cada usuari de l'aplicació disposaria d'un magatzem de claus –*keystore*– amb les seves claus i certificats. A més, a través de l'aplicació, podria realitzar processos d'importació/exportació en el magatzem. També disposaria d'un altre magatzem amb els certificats de les autoritats de certificació en qui confia.
- Amb tota aquesta infraestructura, el procés d'intercanvi de claus es realitzaria d'acord amb les següents pautes:
  - El client, en iniciar l'intercanvi, signaria un resum del missatge amb la seva clau privada, en el què incorporaria el seu certificat.

[24] Algunes podrien afectar a la configuració de la plataforma, tot i que la configuració explicada ha estat comprovada a través de l'eina **jxta-shell** disponible en la web de la plataforma.

[25] Aquest projecte disposa d'una adreça web pròpia: http://presence.jxta.org/servlets/ProjectHome. No hi ha, però, pràcticament activitat.

- En rebre el missatge inicial, el servidor llegiria el certificat i comprovaria si la CA signant és de confiança, cercant el certificat de la CA en el magatzem adient.
- En cas de ser-ho, comprovaria la signatura. Si aquesta és vàlida, continuaria el procés trametent al client la resposta al missatge inicial d'intercanvi. Aquesta resposta aniria signada, ara amb la clau privada del servidor, i incorporaria el seu certificat.
- Naturalment, el client, en rebre la resposta, actuaria igual que el servidor. Si tot es trobés correcte, es completaria de forma satisfactòria l'intercanvi.
- Si en qualsevol instant es detectés algun problema en la cadena de certificats o en les signatures, no es validaria l'intercanvi i s'avortaria el xat.

Finalment, un altre aspecte a considerar en aquest capítol de conclusions finals és la forma d'implementació del servei associat a l'estat presencial dels usuaris. És clar que la implementació actual com a servei lligat al *peergroup* **JXTA Xat Segur**, és difícil de mantenir quan creix el nombre de *peers* del grup. En aquest cas, caldria pensar en d'altres possibilitats, entre les quals podem esmentar:

- Segmentar el grup, cosa que representaria dotar al programa de la possibilitat de crear grups de *peers* segons la conveniència de l'usuari.
- Implementar l'estat presencial d'una altra forma: disposant, per exemple, en cada *peer* d'un *socket* especial dedicat al servei, al què se li poguessin adreçar peticions per part dels usuaris que tenen el *peer* com a contacte.

Glossari.

Terme	Descripció
Advertisement	Descripció d'un recurs –*peer*, *peer group*, *pipe*,... – disponible en una xarxa P2P. En JXTA es representen mitjançant documents XML.
AES	Acrònim de *Advanced Encryptyon Standard*. Criptosistema basat en l'algorisme de Rindjael que constitueix el nou estàndard en les xifres simètriques de bloc. Xifra blocs de 128 bits amb una clau que pot tenir una longitud de 128, 192 o 256 bits.
Atac de força bruta	Atac contra un criptosistema que consisteix en provar totes les possibles combinacions de la clau fins trobar la correcta.
Atac d'home al mig	Atac contra un protocol d'intercanvi en què l'atacant es situa entre els dos interlocutors, intercepta els missatges d'autenticació i els substitueix per altres canviant les claus públiques, de forma que es pot produir una suplantació si els interlocutors no comproven –per exemple, amb l'ajut de signatures digitals– l'autenticitat de les claus.
Autoritat de certificació	Persona o organització de confiança que emet certificats.
CBC	Acrònim de *Cipher Bloc Chaining*. Mode de xifratge de bloc en què els blocs s'encadenen de forma que el xifratge d'un bloc depèn de l'anterior, amb un bloc inicial aleatori que no cal que sigui secret.
Certificat de clau pública	Estructura de dades, emesa per una autoritat de certificació, que conté un nom d'usuari i la seva clau pública.
CFB	Acrònim de *Cipher Feedback*. Mode de xifratge de bloc que utilitza indirectament el xifrador de bloc, amb la qual cosa la longitud dels blocs del text a xifrar pot ser més petita que la dels blocs del criptosistema.
Clau de sessió	Clau simètrica, coneguda per ambdues parts, que es genera amb l'objectiu de protegir un determinat intercanvi d'informació. Per tal de protegir-la de possibles atacs, s'utilitza criptografia de clau pública en la generació.
Criptografia de clau pública	Criptosistema en què una parella d'usuaris es pot comunicar confidencialment sense necessitat de compartir una clau secreta, en disposar cada usuari d'una clau pública, a l'abast de tothom, i una de privada que només coneix ell.
Criptografia de clau simètrica	Criptosistema en què els dos interlocutors comparteixen la clau, i la fan servir tant per a xifrar com per a desxifrar.
DES	Acrònim de *Data Encryptyon Standard*. Criptosistema de xifratge simètric de bloc desenvolupat per IBM i l'administració americana que xifra dades especejant-les en blocs de 64 bit i utilitzant una clau de 56 bits.
Diffie-Hellman	Protocol d'intercanvi de claus que permet a dos usuaris pactar una clau secreta compartida sobre un canal insegur.
ECB	Acrònim de *Electronic Code Book*. Mode de xifratge de bloc en el què el xifratge dels blocs és independent l'un de l'altre i es realitza amb la mateixa clau.
Endpoint	En JXTA, abstracció de les interfícies de xarxa emprades tant en enviar com en rebre dades
Funció hash	Funció que proporciona a la sortida un resum de longitud fixa calculat a partir d'una entrada consistent en un missatge arbitràriament llarg.
Funció hash segura	Funció que compleix dues condicions: és unidireccional i resistent a les col·lisions -és computacionalment molt difícil obtenir un missatge **M'≠M** tal que **h(M')=h(M)**.
Funció de resum de missatge	Veure funció *hash* segura.

Terme	Descripció
Funció unidireccional	Funció en què la sortida és fàcil de calcular a partir de l'entrada, però l'entrada és difícil de calcular a partir de la sortida.
Gnutella	Aplicació d'intercanvi de tot tipus de fitxers que segueix el model P2P pur.
IDEA	Acrònim de *International Data Encryption Algorithm*. Criptosistema de xifratge de bloc que xifra blocs de text de 64 bits de longitud mitjançant una clau de 128, realitzant vuit iteracions idèntiques i una transformació de sortida.
Infraestructura de clau pública	Conjunt de maquinari, programari, persones, polítiques i procediments necessaris per a crear i gestionar certificats digitals basats en criptografia de clau pública.
JXTA	De l'anglès *juxtapose*. Projecte iniciat per Sun Microsystems l'any 2001 que especifica un conjunt de protocols no propietaris que serveixen de marc de referència en les comunicacions a través de xarxes P2P. Aquests protocols poden ser implementats en qualsevol llenguatge i per a qualsevol sistema operatiu, en estar basats en un estàndard com XML.
JXTA ID	Forma lògica d'adreçament que constitueix la manera uniforme de fer referència a diferents recursos –*peer*, *pipe*, *peergroup*, *advertisement*,...– en la plataforma JXTA. La implementació Java de referència de la plataforma utilitza UUID de 128 bit i permet que cada *peer* els generi de forma aleatòria
Napster	Aplicació per a compartir fitxers en format MP3 que fou una de les primeres en l'àmbit de les xarxes P2P. Napster utilitzava un servidor central que administrava les llistes de cançons i permetia als *peers* localitzar-les. Localitzat el fitxer, la descàrrega es realitzava de *peer* a *peer* sense intervenció del servidor. El Napster primitiu va haver de tancar per infringí la normativa sobre drets d'autor. Actualment, és un servei de pagament.
NAT	Acrònim de *Network Address Translation*. Sistema que permet que els ordinadors d'una LAN tinguin accés transparent a l'exterior sense haver de tenir tots ells adreces IP públiques, gràcies a què l'encaminador s'encarrega, mitjançant taules, de mapar les adreces privades a les públiques possibilitant l'accés des de la intranet a l'exterior. A més, és possible protegir la intranet, permetent només connexions d'entrada a determinades màquines o bé aquelles connexions que són la resposta a d'altres que s'han iniciat en la intranet.
OFB	Mode de xifratge de bloc en què el vector inicial es realimenta directament al resultat del xifratge de bloc.
P2P	Acrònim de *Peer-to-peer*, model de xarxa que no té clients ni servidors fixos, sinó tot un conjunt de nodes que es comporten alhora com a clients i com a servidors dels altres nodes de la xarxa.
P2P híbrid	Model de xarxa P2P en què un *peer*, abans de contactar amb un altre, ho fa amb el servidor per tal de cercar dades respecte la identitat de l'altre peer, els serveis que ofereix o bé per qüestions de seguretat. Quan el *peer* té les dades del seu interlocutor, contacta amb ell directament.
P2P mixt	Model de xarxa P2P en què existeixen equips –anomenats *super-peers*– que contenen informació que els altres *peers* no tenen i de la qual en poden disposar quan la necessiten.
P2P pur	Model de xarxa P2P en què tots els nodes són iguals, en no existir cap servidor central.
Peer	En una xarxa P2P, una entitat capaç de desenvolupar alguna tasca útil i de comunicar els resultats de l'execució a una altra entitat de la xarxa, bé sigui de forma directa o indirectament.
Peergroup	Conjunt de *peers* que en una xarxa P2P ofereixen una sèrie de serveis comuns.
Pipe	Canal virtual de comunicació emprat per a que un *peer* estableixi contacte amb altres *peers*.

Terme	Descripció
PKI	Acrònim de *Public Key Infrastructure*. Veure Infraestructura de clau pública.
Rendezvous peer	*Peer* que en una xarxa P2P permet als *peers* més simples descobrir-ne d'altres i, també, els serveis que ofereixen.
Relay peer	*Peer* que en una xarxa P2P actua com a encaminador que permet que els *peers* més senzills puguin comunicar-se encara que es trobin protegits per tallafocs o situats darrera dispositius NAT.
Router peer	Veure *relay peer*.
SETI@home	SETI (Search for Extraterrestrial Intelligence) és un experiment científic que utilitza ordenadors connectats a Internet –emprant computació distribuïda– en la recerca d'intel·ligència extraterrestre.
Signatura digital	Valor que l'usuari calcula amb la seva clau privada a partir del text que vol signar, i que pot ser comprovat posteriorment per una altre usuari fent servir la clau pública del signant, cosa que permet confirmar que només el pot haver signat el posseïdor de la clau privada.
Socket	Punt que permet, mitjançant una adreça que l'identifica, accedir als serveis de comunicació en l'àmbit d'un protocol de transport.
Triple DES	Protocol de xifratge triple que fa servir el DES com a base per a obtenir un sistema amb un espai de claus superior.
XML	Acrònim de *eXtensible Mark-up Language*. És una especificació que defineix una sintaxi i unes regles sobre l'ús d'etiquetes per a estructurar la informació.

Bibliografia.

Llibres:

Domingo, J.; Herrera, J.; Rifà, H. (2004) *Criptografia.* (2a ed.) UOC

Gradecki, J.D. (2002) *Mastering JXTA.* (1a ed.) John Wiley & Sons.

Herrera, J.; Garcia, J.; Perramón, X. (2004) *Seguretat en xarxes de computadors.* (1a ed.) UOC

Horstmann, C.S.; Cornell, G. (2006) *Core Java 2 Volumen I - Fundamentos.* (7a ed.) Prentice Hall.

Horstmann, C.S.; Cornell, G. (2006) *Core Java 2 Volume II – Advanced Features.* (4a ed.) Prentice Hall.

Menezes, A. ; Van Oorschot, P.; Vanstone, S. (2001) *Handbook of applied cryptography.* (5a ed.) CRC Press. Disponible a http://www.cacr.math.uwaterloo.ca/hac/

Millán, R.J. (2006) *Domine las Redes P2P.* (1a ed.) Creaciones Copyright

Pastor, W.C. ; Sarasa, M.A. (1998) *Criptografia digital. Fundamentos y Aplicaciones.* (1a ed.) Prensas Universitarias de Zaragoza

Richardson, J. ; [et al.] (2005) *Profesional Java 2 v5.0.* (1a ed.) Anaya Multimedia

Wilson, B.J. (2002) *JXTA.* (1a ed.) New Riders. Disponible a http://www.brendonwilson.com/projects/jxta-book/

Articles i altres documents:

Bonilla, A.; Meler, J. (2004) *Aplicaciones distribuidas: P2P.* Disponible a http://studies.ac.upc.edu/FIB/CASO/seminaris/2q0304/M9.pdf

Brookshier, D. (2003) *The Socket API in JXTA 2.0.* Disponible a http://java.sun.com/developer/technicalArticles/Networking/jxta2.0/

Diffie, W.; Hellman, M.E. (1976) *New Directions in Cryptography.* Disponible a http://www.cs.jhu.edu/~rubin/courses/sp03/papers/diffie.hellman.pdf

Gong, L. (2001) *JXTA: A network programming environment.* Disponible a http://www.jxta.org/project/www/docs/JXTAnetworkProgEnv.pdf

JXTA Lecture. Disponible a http://users.cs.cardiff.ac.uk/Ian.J.Taylor/DistributedSystems/PPTSlides/JXTA.ppt

Schuler, F. (2005) *Etude et utilisation des technologies des P2P.* Disponible a http://schuler.developpez.com/articles/p2p/

Sun Microsystems. (2005) *JXTA v2.3.x: Java Programmers Guide.* Disponible a http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

Traversat, B. ; [et al.] (2003) *Project JXTA v2.0 Super-Peer Virtual Network.* Disponible a http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf

Pàgines web:

Projecte JXTA. http://www.jxta.org/

The practice of peer-to-peer computing. http://www-128.ibm.com/developerworks/library/j-p2pcol.html