Pràctiques de laboratori

Un xat distribuït orientat a objectes basat en RMI (3 sessions)

Concurrència i Sistemes Distribuïts

Introducció

L'objectiu d'aquesta pràctica és introduir als estudiants als sistemes distribuïts i al disseny orientat a objectes d'aplicacions distribuïdes. S'utilitzará RMI com middleware distribuït subjacent, però qualsevol altre middleware que suporte objectes distribuïts podria ser utilitzat. L'aplicació seleccionada per a aprendre i practicar els sistemes distribuïts és una aplicació de Xat distribuïda. Existeixen moltes aplicacions de Xat en el mercat i també alguns estàndards. L'objectiu no és desenvolupar una aplicació completa ni seguir un estàndard en particular, sinó centrar-se en un sistema distribuït orientat a objectes clar i senzill. Aquesta pràctica també tracta alguns aspectes importants dels sistemes distribuïts no específics dels sistemes orientats a objectes, com a servei de noms, i el paradigma de client/servidor.

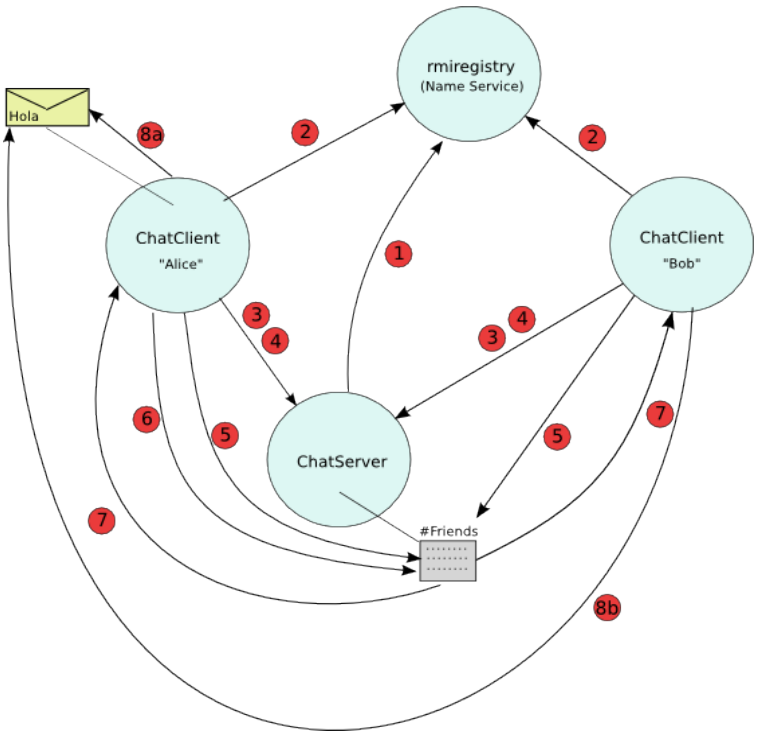
Una vegada completada aquesta pràctica, s'haurà après a:

- Identificar els diferents processos que conformen una aplicació distribuïda.
- Compilar i executar aplicacions distribuïdes senzilles.
- Entendre la funció de servei de noms.
- Entendre el paradigma de client/servidor i la seua extensió orientada a objectes per a dissenyar i implementar aplicacions distribuïdes.

Es necessiten aproximadament tres setmanes per a completar la pràctica. Cada setmana cal assistir a una sessió de laboratori, on el professor pot resoldre qüestions tècniques. També pot ser necessari dedicar temps addicional per a completar la pràctica. Amb aquesta finalitat es poden utilitzar els laboratoris durant els períodes d'accés lliure. També es pot utilitzar el vostre ordinador personal. La pràctica es pot realitzar connectant-se a un escriptori remot dels laboratoris, o bé utilitzant directament el seu ordinador personal. Per a la correcta visualització de la interfície gràfica amb l'escriptori remot, la profunditat de color seleccionada al client de l'escriptori remot ha de ser "Color verdader (24) bits".

Un Xat distribuït bàsic

El següent esquema mostra una disposició bàsica del xat distribuït. Hi ha un servidor de noms (anomenat *rmiregistry* en RMI), un ChatServer i dos ChatClient (llançats pels usuaris Alice i Bob). Aquests 4 processos conformen un sistema senzill que no obstant això és més complex i interessant que un esquema de client/servidor pur.



L'esquema mostra els passos ordenats que la nostra aplicació segueix quan els usuaris Alice i Bob s'uneixen a un canal de xat denominat "#Friends" i comencen a xatejar. Alice envia un missatge "Hola" i ambdós usuaris ho reben al estar connectats al canal.

A continuació es detallen els passos seguits. Cada pas està marcat amb un cercle roig numerat en l'esquema:

1. ChatServer registra el seu objecte principal "ChatServer" en el servidor de noms.
2. Els dos ChatClients cerquen el objecte "ChatServer" utilitzant el servidor de noms. Note's que els tres processos han d'estar d'acord en el nom de l'objecte "ChatServer".
3. Els clients del Xat es connecten al ChatServer. Per a això creen un objecte "ChatUser" local i el registren en el servidor, per mitjà del mètode *connectUser* de ChatServer.
4. Els clients del Xat pregunten la llista de canals, per mitjà del mètode *listChannels* de ChatServer.
5. Els clientes del Xat s'uneixen al canal "#Friends". Per a això obtenen primer l'objecte ChatChannel del canal sol·licitat (per mitjà del mètode *getChannel* de ChatServer) i s'unixen a dit canal, utilitzant el mètode *join* de ChatChannel. Note's que el primer usuari en un canal també rep una notificació quan el segon client s'uneix. Aquesta notificació de canal no és dibuixada en l'esquema.
6. Alice envia un missatge de xat senzill al canal. Per a això, ChatChannel invoca a

ChatUser per a retransmetre els missatges.

7. A partir d'aquest enviament, el canal retransmet el missatge a tots els usuaris connectats (al canal).
8. Els usuaris quant reben el missatge demanen el seu contingut. 8a és una invocació local, mentre que 8b és una invocació remota.

Note's que encara que el rol dels ChatClient és principalment actuar com a clients de xat, també actuen com a servidors, doncs tenen objectes què són invocats de forma remota. Més concretament, *ChatChannel* invoca a *ChatUser* per a retransmetre els missatges, i qualsevol que necessite veure el contingut d'un missatge donat ha d'invocar l'amo del missatge pel seu contingut. En l'esquema, *ChatClient* pregunta a l'usuari Alice pel contingut del missatge "Hola". Aquest patró d'invocació d'objectes no és habitual en la majoria d'entorns de xat però és bastant complet per a comprendre les aplicacions orientades a objectes distribuïts.

Instal·lacions de laboratori i recomanacions de l'entorn

Es pot utilitzar qualsevol entorn de desenvolupament per a gestionar projectes Java (BlueJ, Eclipse, etc.) o simplement un editor de text senzill, el compilador estàndard per línia de comandaments i la màquina virtual de Java. La nostra recomanació és utilitzar les eines de línia de comandaments, de tota manera s'utilitze un IDE o no per a desenvolupament probablement es necessiten les eines per línia de comandament. Aquestes son normalment més flexibles per a entorns distribuïts.

Si realitza la pràctica connectant-se a l'escriptori remot del laboratoris, cal tindre en compte que la màquina a la que es connecta té un firewall configurat, el qual bloqueja la majoria de ports. Es pot utilitzar els ports **9000-9499** per a les pràctiques de laboratori. Alguns dels programes a utilitzar en la pràctica i l'eina *rmiregistry* necessiten que s'indiqui el port a utilitzar.

El programari proporcionat

Inicialment, s'ha proporcionat el codi binari del DistributedChat perquè l'alumne puga comprovar la funcionalitat del xat distribuït que es desitja implementar. Descarrega l'arxiu "DistributedChat.zip" del material de la pràctica i descomprimisca-ho. Veurà que conté un conjunt de fitxers de tipus .class (és a dir, fitxers amb codi binari Java). D'estos fitxers ens interessen els programes **ChatClient.class** i **ChatServer.class**. Els altres arxius són interfícies i classes necesàries per a aquests programes i tots ells conformen un xat distribuït bàsic.

Per a poder executar ChatServer i ChatClient, es necessita una instància de **rmiregistry** ja en funcionament de manera que els nostres programes de xat puguen utilitzar-lo com a servidor de noms. Els servicis de noms, com s'ha explicat en classe, permeten registrar un nom simbòlic per a un objecte remot i associar-li una referència, de manera que puga ser localitzat pels clients. Per a començar rmiregistry, només cal executar-lo en una terminal o consola. Important: ha d'executar-ho **en el mateix directori on es troben les classes .class de DistributedChat**.

```
rmiregistry 9000
```

El paràmetre 9000 implica arrancar rmiregistry en el port 9000. Si no es proporciona aquest paràmetre, rmiregistry s'arrancarà en el seu port per defecte 1099. Cal recordar que en els laboratoris del DSIC ha d'utilitzar ports dins del rang 9000-9499.

NOTA: Si al executar aquesta ordre apareix un error del tipus "Port already in use", possiblement és degut a que altre alumne ha llançat una instància sobre la mateixa màquina virtual. En eixe cas, torne a llançar el *rmiregistry* amb altre nombre de port, per exemple 9100.

Per a començar un ChatServer, ha d'executar-se el següent comandament **en el mateix directori on es troben les classes compilades .class** (els paràmetres nsXX fan referència a name-server):

```
java ChatServer nsport=9000 myport=9001
```

Aquesta instrucció comença un ChatServer en el port 9001, i que utilitza el port local 9000 quan necessita connectar-se al rmiregistry. El port local per defecte per a ChatServer i ChatClient és 9001, i el port per defecte per a rmiregistry és 9000, d'aquesta manera la instrucció anterior té el mateix efecte que aquesta instrucció més senzilla:

```
java ChatServer
```

Per a començar un ChatClient, es poden proporcionar els mateixos paràmetres. Per exemple:

```
java ChatClient nsport=9000 myport=9002
```

Aquesta instrucció llança un ChatClient en el Port 9002 i s'utilitza el port local 9000 per a connectar-se a *rmiregistry*.

A més, es pot proporcionar també un paràmetre addicional (*nshost*) per a especificar l'amfitrió on el servidor de noms està corrent. Per exemple:

```
java ChatClient nshost=192.168.1.1 nsport=9000 myport=9002
```

Amb aquests paràmetres, aquest ChatClient intentarà trobar un rmiregistry executant-se en el port 9000 de l'ordinador 192.168.1.1. En aquest exemple hem suposat que el rmiregistry es troba en eixa màquina, però en general caldrà descobrir quin és el nom de l'ordinador, la qual cosa es pot fer fàcilment usant, per exemple, el comandament **ipconfig** des d'una terminal en Windows, o bé **ifconfig** en Linux.

Si el procés rmiresgistry s'ha llançat a la màquina local, no és necessari indicar el nom del host. És a dir, seria suficient amb:

```
java ChatClient myport=9002
```

Aquest ChatClient atendrà connexions al port 9002. Recordem que si volem tindre més d'un client o servidor corrent al mateix ordinador físic, cal especificar un port diferent al port per defecte 9001, assignant un port nou per cada client que es llance.

Hi ha un paràmetre addicional per a ChatClient i ChatServer, i es el nom de l'objecte ChatServer. Per defecte es "TestServer". Es pot modificar aquest nom utilitzant l'argument "server" en ambdós programes. Per exemple, les següents instruccions comencen un ChatServer i un ChatClient que utilitzaran un objecte ChatServer anomenat "NewServer".

```
java ChatServer server=NewServer
java ChatClient server=NewServer myport=9002
```

Important: En aquesta pràctica s'assumeix que rmiregistry i ChatServer son executats al mateix ordinador.

Activitat 0: començar a xatejar

En aquest cas s'ha d'aprendre com xatejar utilitzant el Xat distribuït proporcionat. Per a xatejar necessitem almenys 2 usuaris (Alice i Bob). Hem de seguir els passos indicats al apartat anterior, es a dir:

- Pas 1: iniciar *rmiregistry*
- Pas 2: iniciar un ChatServer
- Pas 3: iniciar un ChatClient per a Alice. En aquest pas arranquem un primer ChatClient en el mateix ordenador on estan executant-se la resta de programes. Com el ChatServer iniciat en el pas anterior utilitza el port 9001, cal arrancar el *ChatClient* de Alice en un port diferent (per exemple, el port 9002).
- Pas 4: iniciar un ChatClient per a Bob. En este pas s'inicia un segon client de xat per a Bob. Aquest client pot executar-se en la mateixa màquina on hem llançat els processos anteriors (assignant-li un port diferent, per exemple el port 9003), o bé en altra màquina diferent.

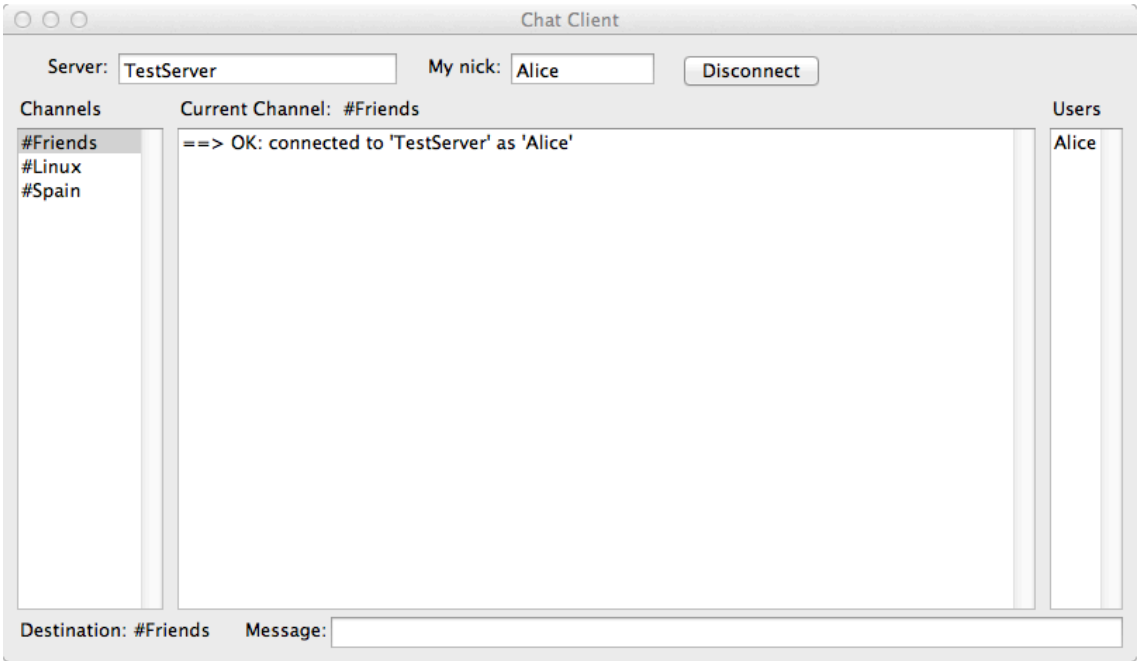
Per a llançar un ChatClient que es connecte al servidor de xat d'una altra màquina diferent (per exemple, l'ordinador del grup veí), caldrà llançar el client especificant l'ordinador on s'està executant el rmiregistry del grup veí. Per exemple, podem assumir que és l'ordinador 192.168.1.2 i que han llançat rmiregistry també en el port 9000. El nou ChatClient el llançarem en aquest cas amb el port 9004.

```
java ChatClient nshost=192.168.1.2 nsport=9000 myport=9004
```

- Pas 5: xatejar un poc.

La figura de la pàgina següent mostra la finestra del ChatClient. En ella es poden observar tres àrees. Hi ha una línia superior per als paràmetres de servidor. A continuació, hi ha una àrea central amb barres de desplaçament vertical (dividida en tres parts) i una tercera àrea en la part inferior, on es poden escriure missatges.

La línia superior té 2 camps de text, un per a indicar el nom del ChatServer al que el client es vol connectar i un altre on cal escriure el sobrenom de l'usuari. En aquest cas ha d'utilitzar Alice i Bob en cadascun dels ChatClient iniciats. Una vegada escrits ambdós camps el ChatClient pot connectar-se al ChatServer. Si la connexió es produeix, s'observarà un missatge d'èxit.



Una vegada es fa doble-clic sobre un canal el ChatClient s'uneix a aquest canal eixint del canal en què s'estiguera prèviament. Si es fa un clic a un usuari d'aquest canal no es deixa el canal actual, solament es canvia la destinació del missatge. D'aquesta manera s'envia un missatge privat a l'usuari.

Activitat 1: Obtindre el proxy del servidor

En esta activitat i següents aprendrem com està constituït el Xat distribuït orientat a objectes i quines invocacions d'objecte ocorren quan es realitzen les activitats bàsiques del xat.

Té a la seua disposició una implementació parcial de l'aplicació ChatClient, anomenada ChatClientCSD, que està inclosa en **DistributedChatSources**.

Descarrega del material de la práctica l'arxiu "DistributedChatSources.jar" i descomprima'l.

En esta activitat i les següents haurà d'actualitzar la classe ChatClientCSD segons anem indicant, per a aconseguir una funcionalitat semblant a la de ChatClient. En este cas, el component d'interfície d'usuari està totalment implementat, però en canvi part de la funcionalitat del client està pendent d'implementar.

La primera tasca que ha de realitzar ChatClientCSD consistix a buscar l'objecte ChatServer, utilitzant el servidor de noms, i així obtindre la seua referència amb què poder invocar-ho. Recordem que prèviament el servidor ChatServer haurà registrat el seu objecte principal ChatServer en el servidor de noms.

Instruccions

En la classe *ChatClientCSD*, implemente en el mètode `doConnect` el següent:

1.a. Obtinga una referència al servidor de noms, utilitzant el mètode `getRegistry` de la classe *`LocateRegistry`*.

1.b. Busque l'objeto `ChatServer` en el servidor de noms, utilitzant el mètode *`lookup`* de la interfície *`Registry`*. Tinga en compte que el nom amb què s'ha registrat *ChatServer* en el servidor de noms el tenim guardat en la variable `serverName`. A més, guarde la referència remota obtinguda en la variable `srv`, que ha sigut prèviament definida al principi de la classe *ChatClientCSD*. Com podrà observar, dita variable és de tipus `IChatServer`, que és una interfície que estén de `Remote`.

1.c. Tracte les excepcions que es puguen produir. En concret, si no es pot trobar `rmiregistry` en el host i/o port indicat; i si el nom d'objecte indicat no està registrat en el servidor de noms. Per a això, pot descomentar les línies de tractament d'excepcions que s'han inclòs en el cpdi.

Amb açò haurem obtingut un proxy de l'objecte `ChatServer`, per la qual cosa des de *ChatClientCSD* podrem realitzar invocacions remotes a dit objecte, utilitzant per a això els mètodes definits en `IChatServer`.

NOTA: per a ajudar-li a realitzar esta activitat, en el codi de la classe ChatServer, en el seu mètode work() , pot observar com ChatServer registra el seu objecte principal `ChatServer` en el servidor de noms.
Pot també tirar-li una ullada al codi de la classe ChatConfiguration per a veure quins són els paràmetres de configuració de l'aplicació i els seus valors per defecte.

Comprovació

1. Compile des de el terminal tots els arxius .java amb la instrucció:

```
javac *.java
```

2. Llance a execució `rmiregistry` i `ChatServer`. Recorde que deuen ser llançats al mateix directori on estiga el codi de `ChatClientCSD`.

3. Llance a execució `ChatClientCSD`, indicant el nombre de port del servidor de noms (en paràmetre `nsport`) i facilitant un nombre de port no utilitzat encara (en paràmetre `myport`), perquè siga utilitzat pel cliente de xat. Per exemple:

```
java ChatClientCSD nsport=9000 myport=9005
```

3. Veurà que apareix la interfície gràfica del client. Comprove el seu funcionament actual.
a) A l'indicar un nick i polsar en `Connect`, haurà d'eixir un missatge del tipus "`Server has no channels`".
b) Si no s'indica cap nick i es polsa en `Connect`, haurà d'eixir un missatge del tipus "`Nick cannot be empty`".

c) Si no s'indica cap nom de Server i es polsa en `Connect`, haurà d'eixir un missatge del tipus "`Server name cannot be empty`".

Qüestió: *Esbrine en quines classes del projecte **DistributedChatSources** es detecten i imprimixen tots estos missatges indicats.*

Activitat 2: Connectar-se a ChatServer i obtindre la llista de canals

La segona tasca a realitzar consistix en el fet que els clients del Xat es connecten al ChatServer, utilitzant el mètode *connectUser* de ChatServer. Una vegada connectat, els clients pregunten la llista de canals, i dita llista serà el que tornarà el mètode `doConnect`. Este mètode, recordem, haurà sigut invocat per la interfície d'usuari (classe `ChatUI`) al polsar-se el botó "`Connect`". La llista obtinguda es mostrarà en la interfície.

Nota: El mètode `connectUser` requerix que se li passe un objecte de tipus ChatUser, que conté el nick o malnom de l'usuari i un objecte de tipus "MessageListener", que permet tractar els missatges rebuts per l'usuari.

Instruccions

2.a. Crea un nou objecte `ChatUser`. El nick o malnom a utilitzar s'ha subministrat anteriorment com a paràmetre d'entrada del mètode `doConnect`. Ha de passar com "MessageListener" al propi *ChatClientCSD*.

2.b. Utilitzant el mètode `connectUser` de `srv`, connecte el client al ChatServer. Llance una excepció si la connexió no s'ha realitzat. La connexió no es durà a terme sempre que el malnom ja estiga en ús.

2.c. Obtinga la llista de canals, usant el mètode *`listChannels`* de ChatServer. Assigne dita llista al contingut de la variable `channels`.

Comprovació

Assegure's de que està en execució tant `rmiregistry` com `ChatServer`. Ha d'haver-los llançat en el mateix directori on es tinga *ChatClientCSD*.
1. Recompila i llança a execució ChatClientCSD (com en l'activitat anterior).

Nota: pot llançar `ChatClientCSD` des de l'entorn de programació (per exemple, des de BlueJ) o bé des del terminal. En el primer cas, haurà de proporcionar al main els paràmetres d'entrada, per exemple: { `"nsport = 9000", "myport = 9003"}`.
Si el llança des del terminal, per evitar problemes de versions de Java, haurà tornar-lo a compilar amb la instrucció:

```
javac *.java
```

2. Comprova el funcionament actual de la interfície gràfica del client:
a) A l'indicar un nick i polsar en `Connect`, haurà d'eixir la llista de canals i un missatge del tipus "`OK: connected to 'TestServer' as XX`", sent XX el nick que s'haja indicat.

b) El botó de "`Connect`" ha passat a mostrar "`Disconnect`". Comprova que al polsar-ho es desconnecta a l'usuari del canal.

c) Si es polsa en algun canal, es mostrarà un missatge d'error del tipus "`BUG. Tell professor there are no users`". Açò és perquè encara no s'ha implementat el codi corresponent per a unir l'usuari al canal.

Qüestió: *Quan els clients del Xat pregunten la llista de canals, què proxies s'empren? Quins mètodes d'eixos proxies? S'envia algun objecte com a paràmetre o valor de retorn d'eixos mètodes? Per a què?*

Activitat 3: Unir-se a un canal

La tercera tasca a realitzar consistix en el fet que els clients del Xat es connecten al canal seleccionat a través de la interfície gràfica. Per a això, obtenen primer l'objecte `ChatChannel` del canal sol·licitat (per mitjà del mètode *`getChannel`* de ChatServer) i s'unixen a dit canal, usant el mètode *`join`* de *ChatChannel*.

Pot observar-se que en la classe `ChatUI` s'ha implementat el mètode *`onChannelSelected`*(), al qual es crida cada vegada que es selecciona un canal en la interfície gràfica. I dins d'este mètode es crida al mètode `doJoinChannel` de la classe *ChatClientCSD*, que és on realitzarem la connexió al canal seleccionat.

En esta activitat realitzarem també l'eixida del canal indicat, en el mètode `doLeaveChannel` de *ChatClientCSD*, per a així completar l'operativitat de la interfície gràfica respecte a la selecció de canals.

Instruccions

3.a En el mètode `doJoinChannel` de la classe *ChatClientCSD*, implemente la unió al canal indicat en la variable `ch`. Pot consultar la interfície *IChatChannel* per a determinar el mètode i paràmetres corresponents a utilizar.

3.b En el mètode `doLeaveChannel` de la classe *ChatClientCSD*, implemente l'eixida del canal indicat en la variable `ch`. Pot consultar la interfície *IChatChannel* per a determinar el mètode i paràmetres corresponents a utilitzar.

Comprovació

Recompile i llance a execució diversos *ChatClientCSD* (com es realitzava en les activitats anteriors), i utilitze en cadascun un usuari (nick) diferent. Comprove el funcionament actual de l'aplicació:
a) Si es polsa en algun canal, apareixerà l'usuari en la llista de Users d'eixe canal. I com "*Current Channel*" figurarà el canal seleccionat.
b) La llista d'usuaris d'un canal ha d'actualitzar-se de forma apropiada conforme els usuaris s'unisquen o abandonen els canals.
c) Si s'escriu text en la casella de "Message", no s'envia cap missatge perquè encara no està implementada esta funcionalitat.

Qüestions: *Quan un client del Xat s'unix a un canal, el canal avisa als altres usuaris del canal.*
1) Quin missatge li arriba a cada usuari del canal?
2) S'ha creat algun objecte per a l'enviament del missatge? Quin o quins?
3) A quines classes i mètodes de dites classes s'ha cridat? Explique detalladament els passos que es seguixen perquè el canal avise als altres usuaris de la unió d'un nou usuari, indicant: (i) Classe i mètode que s'invoca; (ii) si es tracta d'un objecte remot (i, per tant, es fa ús del seu proxy); (iii) paràmetres del mètode (i si són per valor o referència).

Activitat 4: Enviament d'un missatge al canal

La quarta tasca a realitzar consistix en el fet que els clients del Xat puguen enviar missatges de xat al canal en què es troben. El client construïx un objecte de tipus `ChatMessage` i l'envia al canal corresponent. El canal s'ocupa de retransmetre el missatge a tots els usuaris connectats a eixe canal. Per la seua banda, cada usuari, al rebre un missatge, demana el seu contingut i el mostra en la seua interfície gràfica.

Instruccions

4.a En el mètode `doSendChannelMessage` de la classe *ChatClientCSD*, implemente l'enviament del missatge al canal destí corresponent. Pot consultar la interfície *IChatChannel* per a determinar el mètode i paràmetres corresponents a utilitzar.

4.b Observe com s'implementa el reenviament del missatge a tots els usuaris connectats al canal en la classe *ChatChannel*.

Comprovació

Recompile i llance a execució diversos *ChatClientCSD*, usant un usuari diferent per a cada client. Comprove el funcionament actual de l'aplicació:
a) Si un usuari envia un missatge al canal, en la seua interfície d'usuari ha de mostrar-se dit missatge.
b) A més, també ha d'aparéixer en la interfície de tots els usuaris connectats al canal on s'ha enviat el missatge.

Qüestions: *Com rep un usuari un missatge? Quins mètodes i classes s'invoquen? Identifique en quina part del codi es procedix a analitzar el tipus de missatge rebut per part d'un usuari i a mostrar dit missatge per pantalla.*

Activitat 5: Enviament d'un missatge privat a un usuari

Esta tasca consistix en el fet que els clients del Xat puguen enviar-se missatges privats directament. El client emissor construïx un objecte de tipus `ChatMessage` i l'envia a l'usuari destí corresponent.

Instruccions

5.a En el mètode `doSendPrivateMessage` de la classe *ChatClientCSD*, implemente l'enviament del missatge a l'usuari destí corresponent. Pot consultar la interfície *IChatUser* per a determinar el mètode i paràmetres corresponents a utilitzar.

5.b Observe en el mètode `messageArrived` de la classe *ChatClientCSD* com l'usuari tracta els missatges privats rebuts.

Comprovació

Recompile i llance a execució diversos *ChatClientCSD*. Comprove que si un usuari envia un missatge a un altre usuari del canal al què està connectat (fixe's que en "Destination" s'indique el nom de l'usuari destí), este missatge només es reflectix en les interfícies d'estos dos usuaris, però no en les d'altres usuaris connectats al seu mateix canal.

Qüestions: *Com rep un usuari un missatge privat? Com sap que el missatge és privat (i no del canal)? Quins mètodes i classes s'invoquen? Identifique en quina part del codi es procedix a analitzar el tipus de missatge rebut per part d'un usuari i a mostrar dit missatge per pantalla.*

Activitat 6: Ampliació de la funcionalitat → ChatRobot

Si ha resolt amb èxit totes les activitats anteriors, enhorabona!, haurà aconseguit desenvolupar la mateixa funcionalitat que el codi utilitzat en l'Activitat 0.

En aquesta última activitat ampliarem la funcionalitat del Xat. Es desitja implementar un ChatRobot, un procés encarregat de connectar-se a un servidor donat i connectar-se al canal indicat. Cada vegada que un usuari es connecta a aquest canal, el `ChatRobot` li ha de saludar enviant un missatge "Hola sobrenom" al canal, sent "sobrenom" el nick empleat per l'usuari que s'ha unit al canal.

Per a això, complete la implementació del ChatRobot en l'arxiu ChatRobot.java que es proporciona amb la pràctica.

Qüestions

1) Expliqueu quins objectes i quines operacions son invocades cada vegada que un usuari es connecta al canal "#Friends", assumint que el ChatRobot ja està connectat a dit canal.

2) Justifica l'afirmació, "si llancem més d'una aplicació *ChatClient* o *ChatRobot* a la mateixa màquina, deurien tindre valors *myport* distints. Si només es llança una aplicació per màquina no és necessari modificar dit valor"

3) La ubicació del servidor de noms (nsport, nshost) deu ser coneguda per tots, però els clients no necessiten conèixer el host ni el port que correspon al *ChatServer*. Indica la raó.

4) Assumint que *rmiregistry* i *ChatServer* estan en una màquina A, i que llancem l'aplicació *ChatRobot* en una màquina B, i una aplicació *ChatClient* en una màquina C, i sabent que estes dos aplicacions indiquen la ubicació del servidor de noms llançat en la màquina A, descriga gràficament:

a) on s'ubiquen cadascun dels següents objectes (en quina màquina)

b) indique si s'empren una o diverses instàncies de cada objecte

c) explique com poden comunicar-se les instàncies entre si, és a dir, com s'obtenen les seues referències, quin tipus d'invocacions es realitzen (locals o remotes), etc.