Laboration 2: Ett kommunikationssystem

1 Syfte

- Att arbeta ännu mer med OO-design och programmering mot gränssnitt.
- Undantag och felhantering.
- Referens- och värdeanrop i distribuerade system.

2 Uppgift

Ni skall implementera ett litet kommunikationsystem bestående av en klient- och en serverapplikation. Systemet kan hantera meddelanden (chat) och (enkel) filöverföring mellan användare se Figur 1 och 2.

3 Bakgrund

Eftersom applikationen är distribuerad måste det finnas något sätt att kommunicera (över ett nätverk). För att göra det hela lite enklare kommer vi att köra alla applikationer på samma dator. M.h.a. en s.k. loopback-adress (ip 127.0.01) kan vi simulera anrop över nätet.

Vi kommer att använda Java's Remote Method Invocation (RMI). RMI gör det möjligt att anropa objekt (metoder) i andra JVM:er (potentiellt på andra maskiner). I princip innebär det att objekt kan öppna (ibland slumpvisa) portar och ta emot anrop på dessa. De exakta detaljer om hur RMI fungerar skall vi inte behöva bekymmra oss om[1].

4 Kravspecifikation

4.1 System arkitektur

Server implementerar gränssnittet IServer som använda av Client. Client implementerar IClient som används av Servern och IPeer som används mellan klienter (peer2peer). See Figur 3. Samtliga gränssnitt är "Remote" eftersom metodanropen sker med RMI.

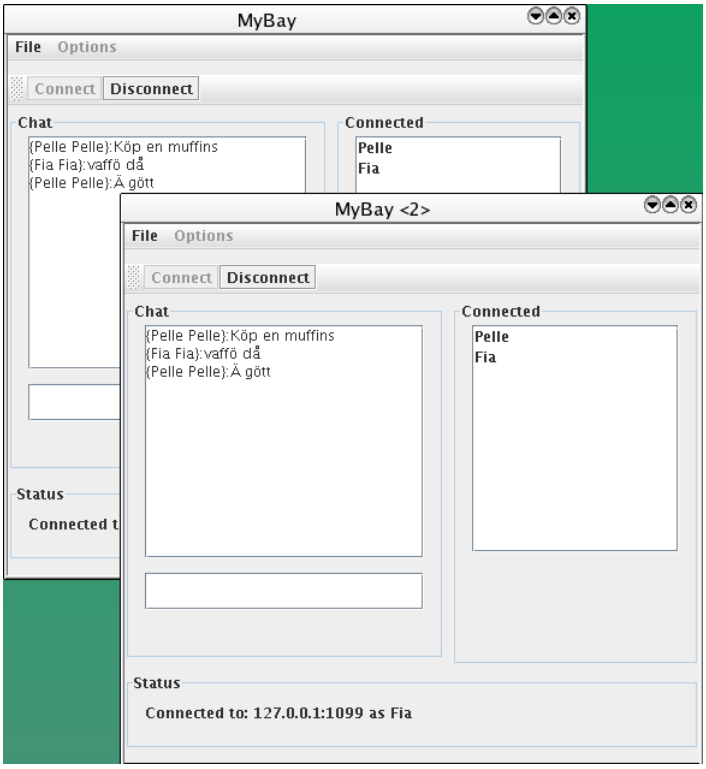
4.2 Funktionalitet

4.2.1 Klientapplikationen

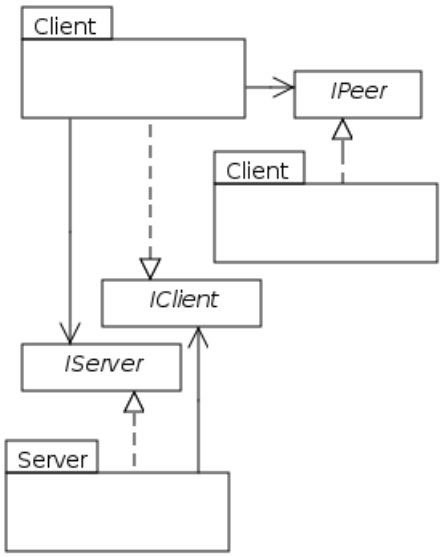
Klienten kan göra följande (se Figur 1);

- Klienten kan ansluta sig till Servern (Toolbar > Connect). Då klienten är ansluten kan han/hon skicka meddelanden till andra

[1] Förhoppningsvis...RMI har blivit mycket lättare att använda i Java >= 1.5. Se upp med gamla artiklar på nätet om; rmic, stubs, skelletons, e.t.c. Det mesta skall fungera utan att vi skall behöva göra något speciellt.



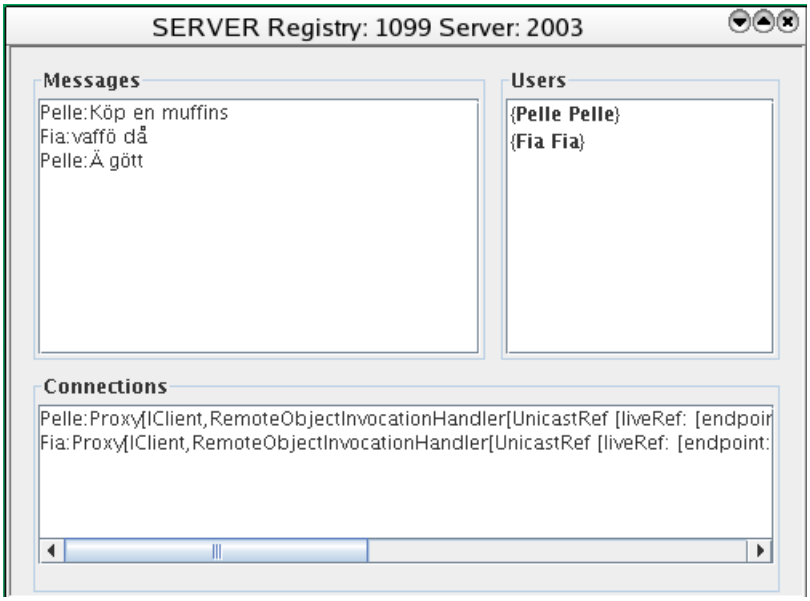
Figur 1: Två klienter.



Figur 3: Systemdesign.

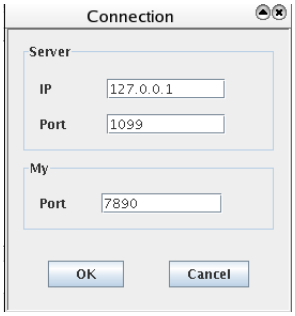
anslutna klienter. Alla anslutna visas i listan till höger (Connected).

- Klienten kan koppla ner sig (Toolbar > Disconnect).
- Då klienten ansluter blockeras Options-menyn (disabled), man kan t.ex. inte ändra anslutning då man är ansluten.
- Meddelanden skickas genom att man skriver något i det tomma texfältet och trycker enter. Samtliga anslutna får meddelandet (broadcast). Textfältet för meddelanden kan inte användas då man är nerkopplad (disabled).
- Under menyn Options kan man få upp tre olika dialogrutor. Dessa skall fungera, se figurer.
- (BONUSPOÄNG) Då man klickar på någon ansluten i Conneced-

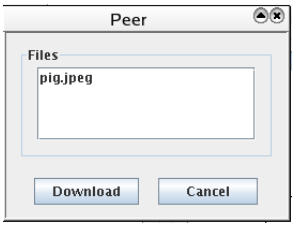


Figur 2: Server.

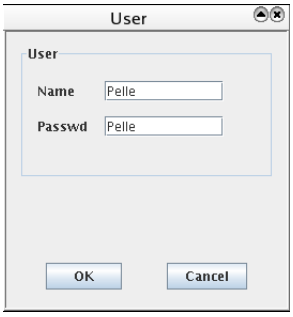
listan öppnas ett fönster som visar vilka filer man kan ladda ned från personen se Figur 5. Det som listas är filerna i katalogen upload i Figur 7. Om man markerar en fil och klickar Download laddas filen ner (direkt från den andra klienten). Nedladdade filer hamnar i katalogen download.



Figur 4: Menyvalen Options > Connections



Figur 5: Dialogrutan med fillista.



Figur 6: Menyvalen Options > User

4.2.2 Serverapplikationen

Grundläggande serverfunktionalitet (mer behövs, ingår i uppgiften att reda ut);

- Klienter skall kunna registrera sig (registrerade klienter läggs i en lista) och avregistrera sig (strykas ut listan). Alla klienter måste reg-

Figur 7: Menyvalet Options > Directories

istrera sig under ett unikt användarnamn. Inget krav på kontroll av lösenord.

- (BONUSPOÄNG) En klient måste kunna få direkt kontakt med en annan klient (kallas då Peer). Detta därför att all filöverföring skall ske direkt mellan klienter (finns inga filer på servern). Kallas ofta peer2peer (P2P) nätverk.

4.3 Systemdesign

4.3.1 Parallellism

Detta är ett parallell system, flera klienter kan samtidigt kontakta servern. Varje RMI-anrop skapar normalt en egen tråd!

Tidskrävande operationer skall köras i egna trådar (Swing worker för filnerladdning). Uppdateringar av GUI skall köras i Swings event-tråd (invokeLater).

4.3.2 Inkapsling

Se upp med by reference och by value (representation exposure), inte bra om klienter kan komma åt saker på servern by reference (d.v.s. skall inte kunna ändra saker på servern). Se dessutom till att allt "håller sig" på sin abstraktionsnivå.

4.3.3 Felhantering

Felhanteringen är ganska komplex. Många olika undantag kan uppstå. Mycket viktigt är att tänka igenom vad som skall ske då ett visst undantag uppstår. Kan felet åtgärdas (på någon nivå)? Vem hanterar felet; Klient eller server? Om vi delar upp undantagen i två kategorier.

1. Icke-krascher. T.ex. klienten försöker ensluta sig till en server med ett namn som redan är upptaget.

2. Krascher. Klienten kraschar vad gör servern?

Ett knippe fel (krascher simuleras med Ctrl c i terminalen man startade applikationen från, se vidare Exekvering):

1. Klient försöker ansluta till server som inte är igång.

2. Klient skickar null-parameter till servermetod?

3. Server returnerar null, skall vi tillåta sådant? Undantag?

4. Server kraschar, vad gör klienten?

4.3.4 Defensiv programmering

Viktigt! Använd defensiv programmering (RuntimeExceptions) för att fånga programmeringsfel d.v.s. kontrollera inparametrar m.m. Vid konstigheter kastar man IllegalArgumentException eller IllegalStateException. Tänk pre och post villkor!

4.3.5 Exekvering

För att köra systemet startar man tre terminaler och kör sedan skripten runclient.sh, runclient2.sh och runserver.sh i respektive. Titta igenom

skripten så att paketnamn, sökväger m.m. är ok (måste troligen ändra lite i dessa).

När ni börjar med filöverföringen finns ett skript buildfiletest.sh som skapar en miljö för test av filöverföring (skapar kataloger, kopiera filer, gör ett run-skript...)

4.4 Design klient

Generellt samma som i föregående lab (förenklad MVC). GUI:et visas som paketet view nedan. Gränssnitten mellan Client och användare är markerade med ??? i bilden nedan (ingår att ta fram dessa).

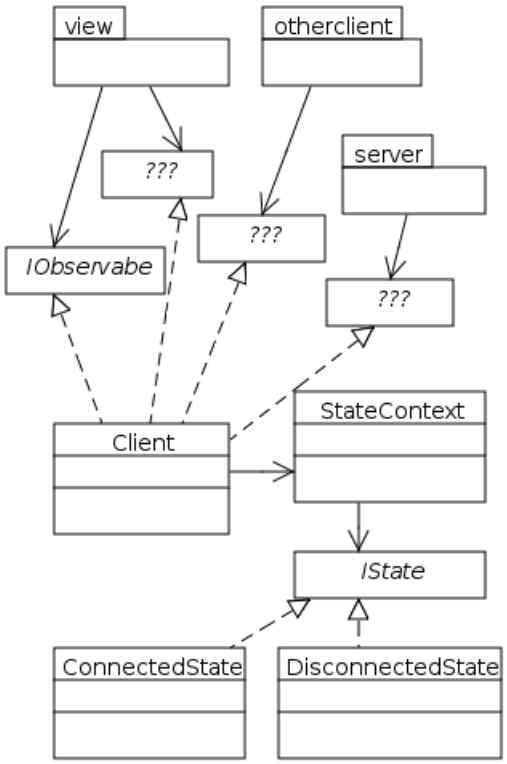
Klienten kan uppenbart vara i två olika tillstånd, uppkopplad och nedkopplad. Utifrån detta skall klienten använda designmönstret State, se Figur 8. Klienten byter alltså tillstånd genom att byta objekt (ändrar referens till annat objekt). Klassen StateContext håller reda på alla tillstånd (objekt) samt det aktuella tillståndet. RMI hanteringen läggs i klasserna Connected och DisconnectedState (klienten startar alltid i nedkopplat läge).

4.4.1 Felhantering

Antag t.ex. att ett kontrollerat undantag uppstår i Connected state i Figur 8. Skall vi hantera det där? Skicka det vidare via StateContext via Client och vidar till gui:et eller tunnla det direkt till GUI:et?

Om man har långa anropskedjor kan det vara lättare att "tunnla" undantaget. Detta tillsammans med try..finally rekommenderas.

Försök att få felhanteringen så konsekvent som möjlig, eller följ någon princip, "vi gör alltid så här...". Använd failure atomicity.



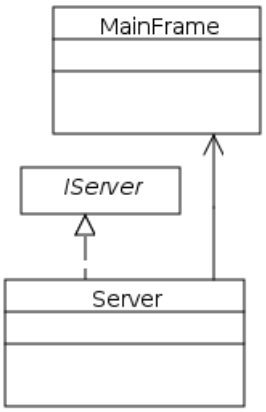
Figur 8: Design klient.

4.5 Design Server

Serven är mycket enkel. Serverklassen har en direkt referens till MainFrame som bara dumpar ut data. Allt RMI kod finns i Main-klassen (syns ej på bild).

5 Utvecklingsprocess

1. Hämta hem startkod för klient och server från kursidan (alltså två projekt) och importera. Projekten delar kod. Den delade koden finns i serverns common mapp. För att klienten skall kunna använda koden måste man länka mappen till klientprojektet. Eventuellt är länken bruten. För att återställa denna;



Figur 9: Server.

Markera klient projektet > Build path > Link Source > Browsa till serverns common mapp, välj (namn fixas automatiskt) > Next > Finish. Samma källkod visas nu i både servern och klientens common-mapp.

2. Det finns en färdig test i klientprojektet.
   a) Öppna en terminal och gå till Serverns projektmapp.
   b) Starta server m.h.a. skriptet runserver.sh (i en terminal).
   c) Kör JUnit-testen TestServer i klientprojektet. Se till att testen har en korrekt Runtime configuration, se kommentarer i testfilen.
3. Arbeta nu med chatfunktionaliteten. Vilka metoder behövs i IServer och IClient (vad vill klienten göra med servern och vise versa?)? Skissa tänkbara metoder i aktuella gränssnitt (hög abstraktions nivå).
4. Implementera och testa en metod i taget. Tänk på att inkommande data måste pushas ut till GUIn. Delegera lågnivåarbetet med RMI till klasserna i State-mönstret.
5. Fortsätt nu med klienten gräsnsnitt mot GUI:et. Skapa ett interface ILocalClient. Vilka metoder behövs. Implementera och testa. Koppla slutgiltigen ihop med GUI:et.
6. Gå igenom felhanteringen, vad händer då t.ex. servern kraschar. Se 4.3.3. Använd en Timer eller dylikt för att rensa ut döda klienter.
7. (BONUSPOÄNG) Implementera filöverföringen.

6 Redovisning

Som tidigare labbar.

Inlämningsdatum Se kurssida.