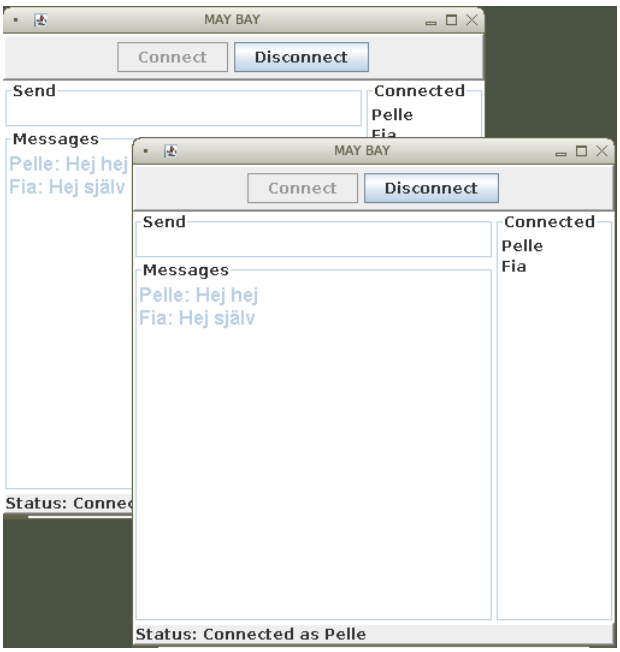
Laboration 2: Ett kommunikationssystem

1 Syfte

- Att arbeta ännu mer med OO-design och programmering mot gränssnitt.
- Undantag och felhantering.
- Trådar.

2 Uppgift

Ni skall implementera ett litet kommunikationssystem bestående av en klient- och en server-applikation. Systemet kan hantera meddelanden (chat) och (enkel) filöverföring mellan användare se figur.



3 Java RMI och Nätverk

Vi kommer att använda Java:s Remote Method Invocation (RMI). RMI gör det möjligt att anropa metoder på objekt i andra JVM:er (potentiellt på andra maskiner). De exakta detaljer om hur RMI fungerar skall vi inte behöva bekymra oss om, i koden ser anropen ut som vanligt (förutom några speciella Exceptions)[1].

I botten på RMI finns ett TCP/IP-nätverk. Vi måste alltså använda IP-adresser och portar för att tala om vart objekt m.m. finns (Servern är ett objekt). För att förenkla kommer vi att simulera nätverket genom att använda den s.k. "loopback" adressen, IP-nummer 127.0.0.1 (kallas också localhost). Innebär att vi kan köra klienten och servern på samma dator[2].

4 Funktionalitet

4.1 Klientapplikationen

Klienten kan göra följande (se figur);

- Klienten kan ansluta sig till Servern (Connect). Då klienten är ansluten kan han/hon skicka meddelanden till andra anslutna klienter. Alla anslutna visas i listan till höger (Connected).
- Klienten kan koppla ner sig (Disconnect).
- Meddelanden skickas genom att man skriver något i det tomma texfältet och trycker Enter. Samtliga anslutna får meddelandet.
- Då man dubbelklickar på någon ansluten i Connected-listan öppnas ett fönster som visar vilka filer man kan ladda ned från personen. Det som listas är filerna i katalogen upload. Om man markerar en fil och klickar Download laddas filen ner (direkt från den andra klienten). Nedladdade filer hamnar i katalogen download.

Klientapplikationerna startas med ett s.k. skript, se runclient.sh och runclient2.sh (fungerar inte på Windows, måste skriva om till .bat filer).

4.2 Serverapplikationen

Grundläggande server-funktionalitet (mer behövs, ingår i uppgiften att reda ut);

- Klienter skall kunna registrera sig (registrerade klienter läggs i en lista) och avregistrera sig (stryk ur listan). Alla klienter måste registrera sig under ett unikt användarnamn (lösenord saknas)

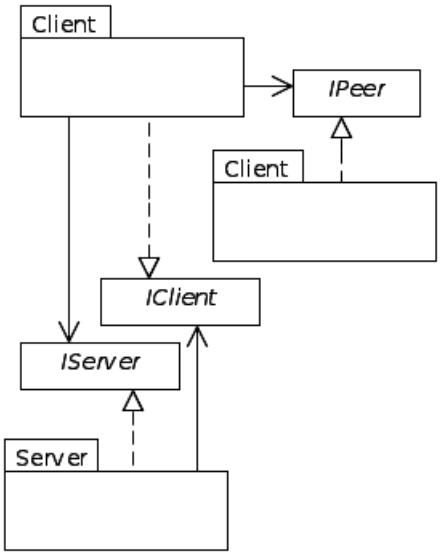
[1] Förhoppningsvis...RMI har blivit mycket lättare att använda i Java >= 1.5. Se upp med gamla artiklar på nätet om; rmic, stubs, skelletons, e.t.c. Det mesta skall fungera utan att vi skall behöva göra något speciellt.

[2] Att köra RMI i verkligheten, mellan flera datorer kan vara trassligt, problem med Proxy-servrar m.m.

- 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 peer2peer (P2P) nätverk.

Servern saknar helt GUI. Det är lämpligt att låta servern logga allt som händer (skriva ut i terminalen). Servern startas med ett skript, se runserver.sh

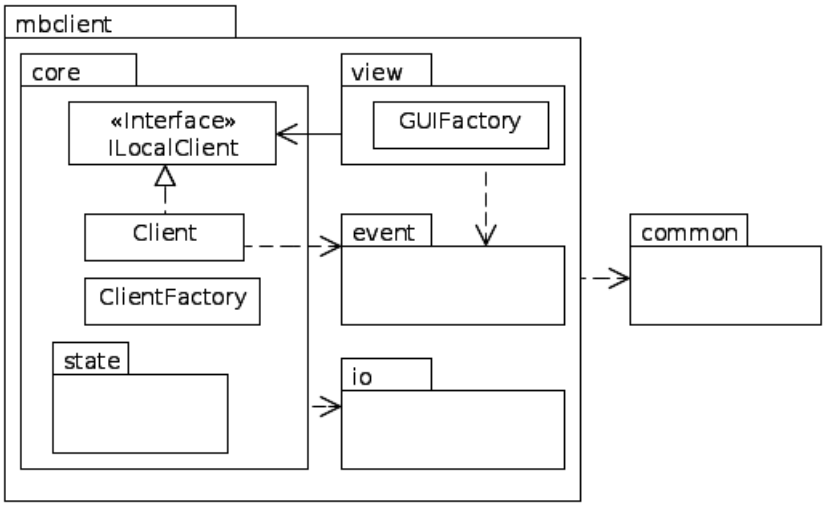
5 Systemarkitektur



Server implementerar gränssnittet IServer som används av Client. Client implementerar IClient som används av Servern. IPeer används mellan klienter vid filnedladdning. Samtliga gränssnitt "extends Remote" eftersom metodanropen sker med RMI. Samtliga metoder i gränssnitten måste ha "throws RemoteException".

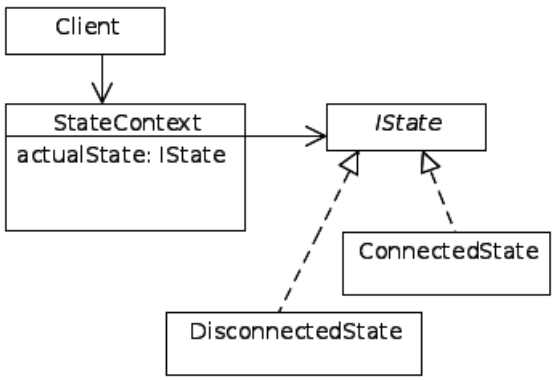
6 Designmodell för Klienten

Alla klasser och kopplingar visas inte.



- core, är modellen med själva klienten. Klienten implementerar ILocal-Client som används av view.
- view, är hela GUI:et. Finns en fabrik som bygger GUI:et.
- event, innehåller EvenBus.
- io, hanterar läsning och skrivning av filer.
- common, paketet innehåller kod som delas av klient och server (gränssnitten ovan m.m.)

Klienten kan uppenbart vara i två olika tillstånd, uppkopplad och nedkopplad. Utifrån detta skall klienten använda designmönstret State. Klienten byter alltså tillstånd genom att byta objekt (ändrar referens till annat objekt). Klassen StateContext håller reda på de två tillstånden (objekten) samt det aktuella tillståndet. RMI hanteringen kommer att ske i klasserna Connected och DisconnectedState (klienten startar alltid i nedkopplat läge). Se vidare nedan.



7 Designmodell för Servern

Själva servern består bara av en enda klass. Dessutom finns en Main-klass för att starta server. Klasser som servern är beroende av finns i mbcommon-paketet.

8 Parallellism

Detta är ett parallell system, flera klienter kan samtidigt kontakta servern. Varje RMI-anrop skapar en egen tråd på mottagande sida! Se över så att servern är trådsäker. Servern behöver dessutom med jämna mellanrum rensa ut "döda" klienter. Använd Java:s Timer och TimerTask klasser.

Om inkommande anrop på klientsidan skall vidare upp till GUI:et får man byta från RMI-tråden till Swing-tråden (allt i GUI:et skall ritas av denna). Man får alltså "lämna" över från en tråd till en annan. Använd SwingUtilities. Tidskrävande operationer skall köras i egna trådar (SwingWorker för filnerladdning).

9 Felhantering

Systemet skall tåla att applikationer kraschar, t.ex. skall en klient kunna ansluta sig på nytt efter det att den kraschat. Felhanteringen blir ganska komplex. Ett knippe undantag (krascher simuleras med Ctrl c i terminalen man startade applikationen från);

1. Klienten försöker ansluta sig till en server med ett namn som redan är upptaget.
2. Klienten kraschar vad gör servern?
3. Klient försöker ansluta till server som inte är igång.
4. Klient skickar null-parameter till servermetod? Kan den? Får den?
5. Server returnerar null, skall vi tillåta sådant? Undantag?
6. Server kraschar, vad gör klienterna?

10 Kom igång

Det finns många sätt att arbeta på, detta är ett ...

1. Finns två påbörjade projekt, hämta och importera i Eclipse. Se till att mappen common är länkad från server-projektet till klientprojektet, se Build Path.
2. Det finns en test i klienten och en implementerad "ping"-metod i servern. OBS! Att testen behöver en "runtime" konfiguration, se kommentar i filen. Försök få testen att fungera.

3. Gör så att klienten kan ansluta sig till servern. Skapa en metod i IServer för detta och implementera på server-sidan. På klientsidan; avvakta med State-mönstret, koda direkt i klientklassens metoder. Testa!

4. Gör så att klienten kan skicka ett meddelande som når alla (just nu bara sig själv). Ni måste skapa en metod i IClient som servern kan använda. Implementera direkt i Client. Testa! Nu kan ni börja fundera mer generellt vilka metoder som skall finnas i de olika gränssnitten.

5. Faktorera om kodens på klientsidan så att klienten använder state-mönstret. All RMI hantering kommer att hamna i de olika tillstånden för state mönstret.

6. Lägg till metod för metod (utom filöverföring) så att allt fungerar. Om möjligt skapa tester.

7. Se över felhanteringen, testa att krascha applikationer.

8. Bestäm vilka metoder som behövs i IPeer. och implementera filöverföringen. Mycket finns färdigt, det gäller att koppla ihop det hela, se PeerDialog.

11 Redovisning

Som tidigare labbar.

Inlämningsdatum Se kurssida.