EDA095 Projekt, YARTMUTE

Yet Another Real-Time Multi-User Text Editor

Av:
Elias Gabrielsson, adi10ega
Niklas Strandberg, adi10nst
Johan Svensson, adi10jsv
Aron Söderling, adi10aso

Bakgrund

Projektet gjordes som en del av kursen Nätverksprogrammering EDA095 vid LTH VT13 lp2 och bestod av att implementera en texteditor i java, som flera användare kan editera dokument samtidigt, likt Google Docs. Syftet med projektet var att få ett tillfälle att tillämpa den kunskap som man går igenom under kursens gång.

Kravspecifikation

Implementationen ska ej ha stöd för någon slags formatering (ex. fetstilt, kursivt, listor av olika slag m.m.) utan är begränsad till att visa en textfil i klarttext. Det skall dock finnas stöd för olika textstandarder (UTF-8, ascii etc). Användare skall även kunna ladda upp filer till servern och ändra befintliga filer i realtid med andra användare.

Modell

Paket:

- client - Innehåller client-klasser
- common - Innehåller gemensamma klasser för kommandon och loggning.
- common.toclient - Innehåller kommandoklasserna som skickas från servern till klienterna
- common.toserver - Innehåller kommandoklasserna som skickas från klienter till servern
- server - Innehåller server-klasser
- server.exceptions - Innehåller några exceptions som används av programmen

Viktiga klasser:

- *ServerDocHandler* och *ServerDoc* – *ServerDocHandler* har som uppgift att hantera dokument som ändras av användare. Ett öppet dokument representeras med ett objekt av klassen *ServerDoc* som i sin tur innehåller bland annat versionsnummer, en *Writer* för skrivning till filen, filens namn samt filens innehåll i klartext representeat

som en lång sträng. En viktig funktion i *ServerDocHandler* är att centralt styra vilka dokument som är öppna så att två användare som jobbar med samma dokument använder samma *ServerDoc*.

- *ServerSocketHandler* – *ServerSocketHandler* är en klass som implementerarinterfacet *Runnable* vilket gör den trådbar. Klassen har som uppgift att representera varje användare dels med själva Socket-objektet men även med vilket dokument som klienten har öppet för tillfället.
- *ClientSocketWriter* och *ClientSocketReader* – Dessa klasser har som uppgift att skriva till respektive läsa från socketen till servern.
- *ClientDoc* – *ClientDoc* är klientens representation av dokumentet som klienten har öppnat.
- *CommandFactory* – *CommandFactory* är lite av en kärna i programmet då den ansvarar för att tolka alla kommandon som skickas från både klient och server samt att kommandona exekveras. Klassen följer **Factory**-mönstret inom OMD vilket innebär att man matar in en sträng och får ut ett nytt objekt.
- *Command och alla dess subklasser* – *Command* är en klass som i sin tur har två subklasser kallade *ServerCommand* och *ClientCommand*. Dessa klasser är abstrakta och har även de subklasser som utgör kommandon för servern respektive klienten. Klasserna på serversidan följer **Command**-mönstret inom OMD då både superklasserna har en metod kallad *execute()* som utför kommandot i fråga.

En session går i stora drag till så här:

Servern är igång och lyssnar på en port.

Klienten startas och ett GUI visas.

Klienten ansluter till servern. Servern skapar även den en ny tråd för anslutningen och börjar lyssna efter kommandon. Klienten skapar en ny tråd för anslutningen och skickar ett ”GetFileList”-kommando till servern. Därefter lyssnar klienten efter ett svar.

Servern tar emot ”GetFileList”-kommandot som en sträng och kör det genom *CommandFactory*. Ett *”GetFileList”*-objekt skapas på servern och exekveras. Servern svarar då genom att skapa ett *”SendFileList”*-objekt och omvandlar det till en sträng. Denna sträng skickas till klienten.

Klienten tar emot *”SendFileList”*-kommandot och kör det genom *CommandFactory*. Klienten har nu alla tillgängliga filer.

Användaren väljer att öppna en fil och skickar då ett ”*Open”*-kommando till servern. Servern tar emot det och *ServerDocHandler* kontrollerar om filen är öppen av en annan användare. Är filen öppen så görs en markering i tråden för klienten i fråga att just detta dokument är öppet. Serverns svarar med att skicka ett SFILE-objekt som innehåller en textrepresentation, ett versionsnummer m.m. av filen i fråga.

Klienten tar emot SFILE-objektet som en strängrepresentation och konverterar därefter det till ett SFILE-objekt. Texten visas i textfältet i GUI:t och användaren kan nu börja ändra i dokumentet.

Varje gång användaren skriver något i textfältet så skickas ett WRITE-kommando till servern. Detta innehåller information om var text läggs till (uttryckt i rader och kolumner), vad som ska

stå där, användaren id-nummer samt vilket versionsnummer användaren hade innan ändringen.

Då en annan användare gör en ändring i dokumentet skickar servern ett UPDAT-kommando till alla klienten som är anslutna med dokumentet i fråga öppet. Både WRITE och UPDAT har samma format då de i grund och botten ändrar samma sak. Hela dokumentet skickas endast då klienten hämtar dokumentet för första gången.

Användarhandledning

Om du tänker köra servern själv, öppna kommandprompten och skriv java -jar server.jar [portnr] [folder]. folder är mappen med editerbara textfiler. OBS: Filerna måste avslutas med \END\.

När du har en server tillgänglig, exekvera client.jar för att starta klienten. Nu bör en ruta med GUI:t synas. För att ansluta till en server så använder du dig av drop-down-menyn längst upp till vänster och väljer sedan ”Connect to Server”. Skriv in ip/hostname och port enligt formatet HOSTNAME:PORT och tryck därefter enter. Du bör nu vara ansluten till en server.

Klienten har redan hämtat en lista över vilka filer som du kan redigera och ska nu visa “Open file”-dialogen. Du får nu upp en ruta där du kan välja bland filerna som finns på servern. Välj någon av dem och tryck ”OK”. Filen bör nu vara öppen och du kan börja skriva i den.

Utvärdering

Överlag är vi nöjda med projektet. Vi lyckades uppnå det viktigaste kravet nämligen att flera användare kan modifiera text i samma fil i realtid. Vi lyckades dock inte implementera ett stöd för olika textstandarder samt funktionalitet för att ladda upp filer. Dock lärde vi oss väldigt mycket om synkronisering och realtidslösningar vilket vi tyckte var väldigt givande inför framtiden.

Textstandarder hade kanske gått att implementera. Det enda som egentligen hindrar oss från att implementera olika textstandarder är klassen *JTextArea* som används för att visa texten i den öppna filen samt för att editera. *JTextArea* använder sig av UTF-8 för att visa texten och stödjer inget annat format. Då alla klienter använder programmet så kommer UTF-8 tecken vara det enda som skickas och därför är implementationen i sig onödigt komplicerad.

Funktionaliteten att ladda upp filer hade gått att implementera med lite arbete. Dock fanns det inte tid för oss att implementera denna funktionalitet så det fick tyvärr strykas då vi ansåg att den grundläggande funktionaliteten var viktigare.

Upplägget med att ha ett projekt är, för en kurs som denna, en väldigt bra idé då man får en helt annan förståelse för kursens innehåll. Dock känns projektet lite överväldigande då man får ha i åtanke att vi även har fem laborationer i samma kurs att tänka på. Dessutom finns ingen tydlig avgränsning i projektet och vi vet inte riktigt vad som förväntas av oss.

Programlistor

Projekthemsidan är http://users.student.lth.se/adi10nst/EDA095/. Här finns länkar för nedladdning av programmen samt källkod.