# Datakommunikation
# &
# Datornät

5DV065

# Distrubuerad chat-server

Fredrik Johansson (c04fjn@cs.umu.se)
Mattias Edin (id12men@cs.umu.se)

Lärare och handledare:

Jerry Eriksson, (jerry@cs.umu.se)
Anders Broberg, (bopspe@cs.umu.se)
Thomas Johansson, (thomasj@cs.umu.se)
Eric Jönsson, (ericj@cs.umu.se)

# Innehållsförteckning

## Problemspecifikation

I den här laborationen ska vi implementera en chattserver, chattklient och dess komuniktaion med ett givet protokoll. Chattservern och chattklienten ska kunna köras på olika maskiner där designen av detta system lämnas till oss förutsatt att komunikationen följer det givna protokollet. Implementationen enligt labb-specen ske i Java.[1]

## Åtkomst och användarhandledning

Systemet vi har utvecklat är uppbyggt av en klient-applikation och en server-applikation. Klient-applikationen används för att ansluta till en server, servern startas med server-applikationen. Alla klienter som är anslutna till samma server kan skriva meddelanden som syns för all anslutna. Båda applikationerna återfinns i:

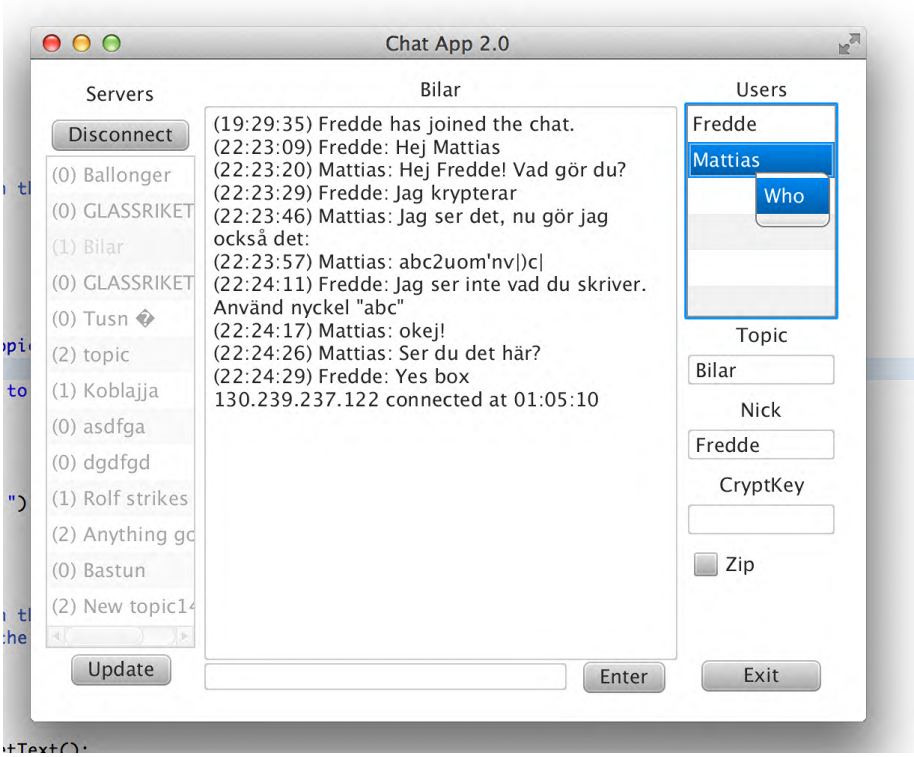
*~/edu/dod/lab1/*

Server-applikationen startas med kommandot:

*java -jar ChatServer.jar [IP till DNS] [Port till DNS] [Topic]*

Klient-applikationen så startas den med kommandot:

*java -jar ChatClient.jar*




[1] Umeå Universitet *Distribuerad chatt-server*. http://www8.cs.umu.se/kurser/5DV065/HT13/lab1/index.html (Hämtad 2013-05-21)

Ett användargränssnitt öppnas vid exekvering av *ChatClient.jar*. Till vänster finns en lista med tillgängliga *chatt-servar*, ange ett användarnamn i fälten till höger, högerklicka på valfri server och välj *connect*. Önskas det annsluta till en server som inte är med i listan över tillgängliga servrar, ange då önskat användarnamn i fältet till höger och skriv *connect [IP]:[Port] (ex: connect 192.168.0.1:1996)* följt av ett *enter* eller knappen *Enter*.

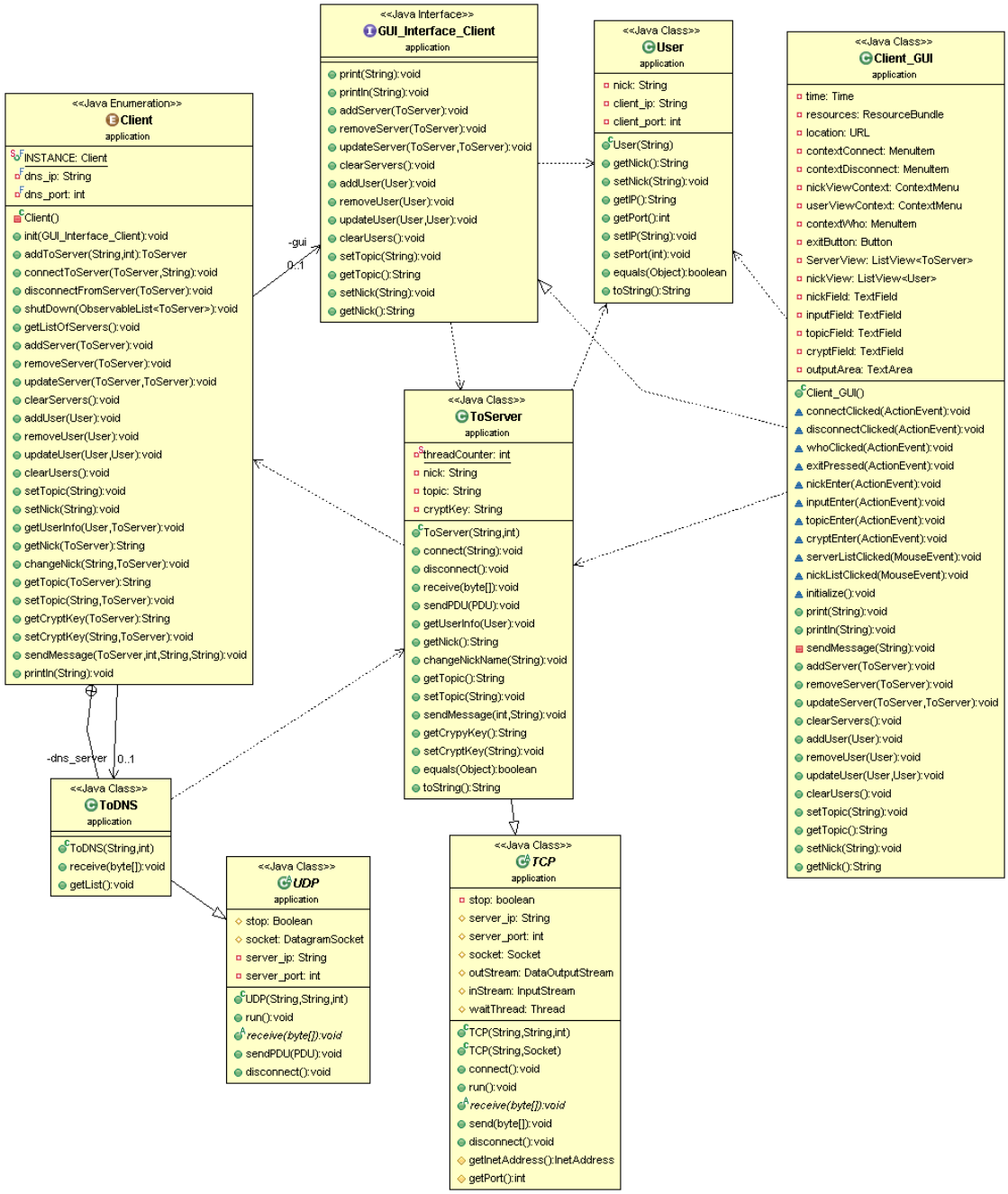
Efter lyckad anslutning visas alla klienter som är anslutna till servern i listan till höger. Under listan visas även aktuellt ämne, denna ruta går att ändra och där med välja ett nytt ämna på servern. Det går även att byta användarnamn i rutan för användarnam. Om användaren vill kryptera sina meddelanden finns det ett fält även för det, har en eller flera användare anngett samma krypteringsnyckel så kan bara de se vad de skriver. Bockrutan under krypterings fältet reglerar om meddelandena ska skickas komprimerade eller ej.

Vill en användare veta mer om en annan användare som är anslutet till samma server, högerklicka då på användaren i fråga i listan till höger och välj *WhoIs* detta resulterar i att användarinformation om vald användare skrivs ut.

För att skicka meddelande i chatten skriv i meddelande fältet längst i mitten på gränssnittet och avsluta med *enter*, eller klicka på knappen *Send*.

# SystembeskrivningKlient

Klassdiagramm för Klient




Klientens centrala del i systemet är klassen *Client* denna klass är den klass som sköter all kommunikation mellan användargränssnittet *(GUI_Client)* och de olika servrar *(ToServer)* som klienten har valt att ansluta sig till. *Client* klassen har en nästlad klass som heter *ToDNS,* och den håller i all kommunikation mot *DNS-servern (Namnservern)*.

*ToDNS* klassen skickar bland annat *Alive-meddelanden* till *DNS-servern* så den vet att servern inte har kopplat ner. All kommunikation mellan *ToDNS* och *DNS-servern* sker via *UDP-protokollet*, där av ärver *ToDNS* av klassen *UDP*. *ToDNS* genererar meddelanden i form av *PDU-paket* (använder sig av klassen *PDU_Factory*), plockar ur de bytes som ska skickas och passar de till klassen *UDP. ToDNS* tar även emot bytes från *UDP* och gör om dem till *PDU-paket* för att sedan läsa av dem och agera där efter.

*UDP* ansvarar för det första för att skicka de bytes som den får av *ToDNS* till *DNS-servern*, och för det andra att ta emot bytes från *DNS-servern* och passa de vidare till *ToDNS*.

*ToServer* ärver av *TCP* då den kommunikationen mot servern ska ske via *TCP-protokollet.* Lika som *ToDNS* så skapar den *PDU-paket* (med hjälp av *PDU_Factory*), skickar och tar emot bytes från *TCP* för att beroende på innehåll i packeten agera på olikasätt.

*TCP* och *UDP* ärver båda av klassen *Thread*, vilket är nödvändigt då det behövs en tråd per ansluten server så systemet kan ta emot data från olika håll samtidigt.

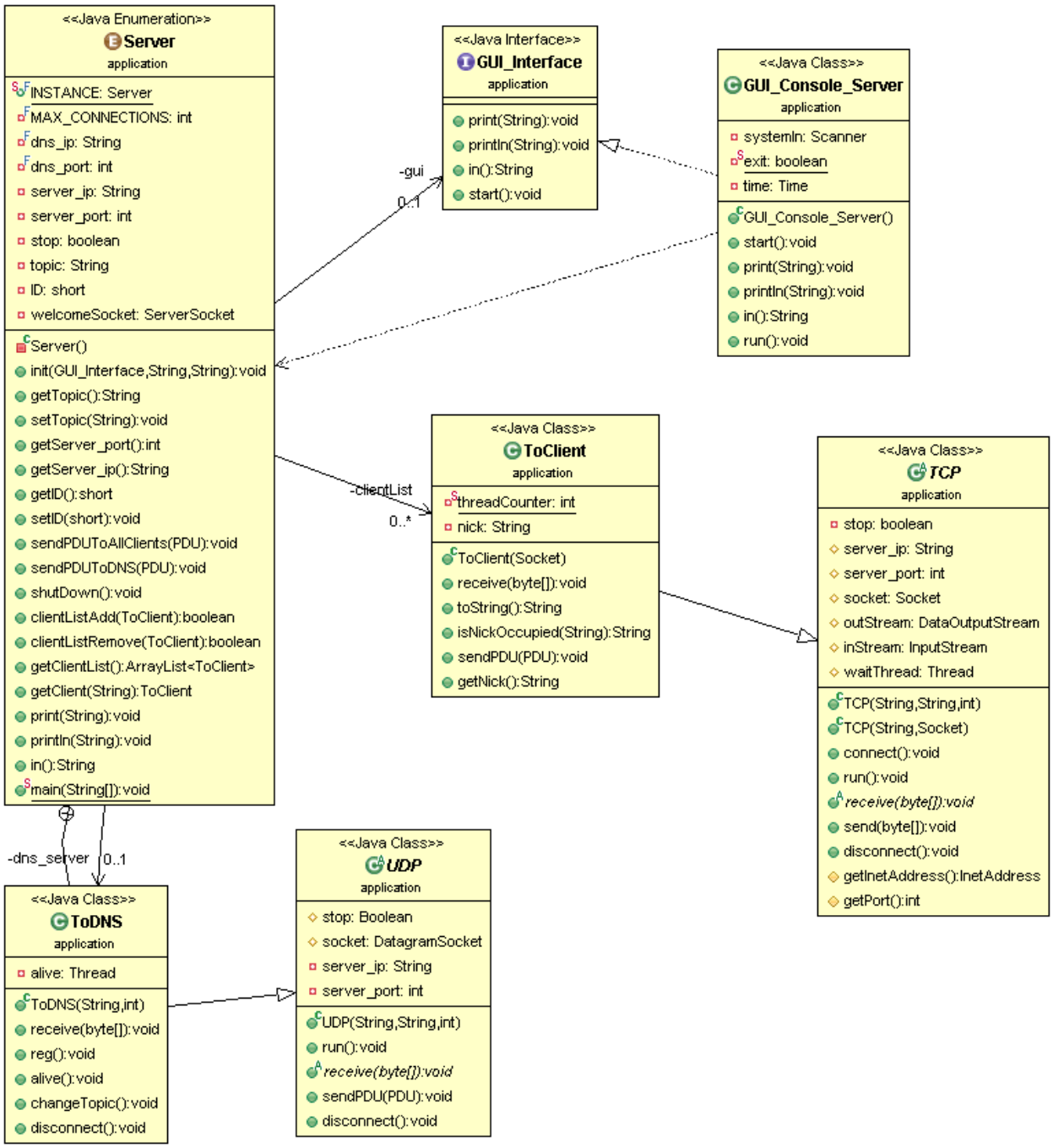
Klassen *UDP_Factory* är en klass med överägande statiska metoder för att generera *PDU-packet* i olika former och innebörd.

*GUI_Client* står för hela användargränssnittet och innehåller en lista med tillgängliga servrar *(ToServer)* och en lista med anslutna användare *(User)*.

*User* innehåller information om en användare, i huvudsak användarnamnet. Informationen i en *User* uppdateras då användaren av klienten begär extra information om den användaren.

## Server

Klassdiagram för server:




Servern är uppbygd på samma sätt som Klienten med undantag för gränssnittet. Gränssnittet på servern är konsolbaserat och där med utan grafiskt användargränssnitt och att paketen som skickas har annan form och innehåll.

## Lösningens begränsningar

Vi hittar inga begränsningar i programmet i förhållande till laborationsspecifikationen men något som kan tänka sig vara begränsning i programmets helhet som en chatt är till exempel att det inte finns något sätt att ge en användare administrativ-behörigheter för ett visst chattrum. Det här gör att alla anslutna användare kan ändra på chattrummets ämne. Lika så finns det ingen möjlighet att blockera störiga användare. För att förhindra ovälkommna användare skulle man även kunna behöva ett lösenord för chattrummet.

## Problem och reflektioner

## Problem

Under implementationen har vi stött på ett antal olika problem och det som vi anser vara det största problemet vara hur själva kommunikationen mellan gränssnittet och programmet ska fungera. Det här har vi löst genom att implementera en del av programmet utifrån ett designmönster somkallas *singleton* där vi har valt att sätta klienten som en enum-klass och GUI:t skapas i klienten. På det här sättet kan både klient- och GUI-klasserna hitta varandra och komunicera med varandra.. *Singleton* designen handlar om att begränsa instansieringen av ett visst objekt som i det här fallet är klient- och serverklassen. I vår lösning vill vi att det endast ska finnas ett objekt av klienten respektive servern och tillämpningen av detta designmönster tillsammans som att göra klasserna till enum passade bra.. Med hjälp av *Singleton* mönstret tillsammans med enumklasser löste vi tvåvägskommunikationen.

Vi valde i vår lösning att implementera ett grafiskt gränssnitt och för att göra detta på enklast sätt använde vi oss utav *JavaFX* tillsammans med *JavaFX Scene Builder* som är ett plugin till Eclipse. *JavaFX* förenklar produktionen av gränssnittet genom att tillhandahålla ett grafiskt gränssnitt för utvecklingen med “drag-and-drop” möjlighet. Det här gjorde att det grafiska gränssnittet enkelt kunde byggas med lite kodning men det här medförde även ett nytt problem som vi inte hade kunskap utav.. Problemet var att *JavaFX* inte tillåter att andra trådar i programmet ändrar på attribut i den. Det här kunde lösas genom att låta GUI:t som är en *JavaFX*-klass själv göra den ändringen “när den har tid” med hjälp av en runlater-metod.

# Reflektioner

## Hur skalbart är detta system?

Vi skulle säga att vårt program skalar “bra” eftersom att belastningen på systemet beroende på antal anslutna användare är proportionerligt i förhållande till varnadra. Anledningen till att det är så beror på att vår implementation använder sig till mesta dels av datastrukturer som har en proportionerlig tidskomplexitet. Till exempel finns en lista med alla anslutna klienter i servern där servern kan behöva gå igenom listan element för element och detta gör att programmets komplexitet är proportionerlig mot antal anslutna klienter.

## Vad skalar bra?

Om servern hade varit implementarad med hash-tabeller istället för listor skulle till exempelvis servern kunna hitta varje klient mycket snabbare. Vår implementation bygger på att servern etablerar en TCP-anslutning till varje klient och delar upp varje sådan anslutning i var sin tråd. Det här skulle man kunna säga skalar “dåligt” ifall det skulle bli ett stort antal anslutna klienter och serverns hårdvara inte klarar av det. Om programmet istället skulle vara implementerad med enbart kommunikation med hjälp av UDP- istället för TCP-protokollet och inte dela upp varje anslutning i trådar. På det här sättet skulle servern kanske kunna hantera fler klienter.

## Vad skalar dåligt?

Om vi skulle implementerat en lista med alla anslutna klienter som också skulle vara sorterad skulle programmet skala dåligt eftersom att datorn skulle behöva sortera listan varje gång det ansluter en ny klient.

## Fundera kring säkerheten för systemet ur olika synvinklar.

Vi annser att det finns två typer av säkerhet i systemet. Dels säkerhet för att användarna ska få en bra användarupplevelse och inte få chatten utnyttjad av andra användare, och dels säkerhet mot användare som försöker påverka systemet utifrån, till exempel med hjälp av tredjeparts program.

Mot det förstnämnda finns det inget skydd i systemet, användare får utnyttja det, i den utsträckning att de använder de inbyggda funktionerna, hur mycket de vill utan att riskera konsekvenser. Något som skulle öka säkerheten i denna synpunkt är till exempel att implementera möjligheten att ge användare rättigheter och medel för att koppla ner eller tysta användare som utnyttjar systemet, eller hålla ordning i chatt-rummen.

När det gäller det sistnämnda så är det svårt att sabotera systemet utifrån. Detta då systemet vi har utvecklat bara behandlar paket som har förbestämd struktur och innehåller koder som inte är känt av allmänheten. Systemet har även möjlighet att kryptera meddelanden med unika nycklar vilket gör att bara de användarna med samma nyckel kan läsa texterna som skickas, och skulle

strömen med information mellan systemen vara avlyssnad så kommer det vara svårt att förstå texten som skickats.

## Val av programmeringsspråk för de olika delarna i systemet.

Vi använde oss uteslutande av *Java* för att programmera systemet, men för användargränssnittet använde vi ett externt program som heter *Scene Builder* som ger en grafisk yta för att designa användargränssnitt. Allteftersom vi byggde upp gränssnittet så skapade *Scene Builder* HTML-kod som vi länkades till våran *Java*-programmerade *GUI-klass*.

## Kommentarer kring de använda protokollen.

De protokoll som vi har annvänt oss av är *UDP* och *TCP*. Där *UDP-protokollet* används för kommunikation mot *DNS-servern (Namn servern)*, medans *TCP-protokollet* används för kommunikation mellan *chatt-klienterna* och *chatt-servern*.

Med *TCP-protokollet* skapas en länk mellan två system och en mer säker kommunikation kan ske. Detta är bra då det ska skickas mycket text mellan klienten och servern som behöver komma fram i sinn helhet. Medans kommunikationen med *UDP-protokollet* mot *DNS-servern* inte är lika kontinuerlig så behövs inte samma förbindelse upprättas.

## Binärt protokoll VS textbaserat protokoll?

Ett binärt protokoll kan vara snabbare i den bemärkelsen att programmet som använder sig av ett binärt protokoll kan exekvera olika operationer direkt från informationen där ett textbaserat protokoll kräver mer analysering och i vissa fall ett agerande från användaren. En annan fördel med binärt protokoll kan vara att det är mycket svårt för skadlig programvara att tyda informationen i ett binärt protokoll eftersom att det inte behöver framgå hur informationen ska användas. Ett textbaserat protokoll har den fördelen att den information som utbytes kan läsas av användaren där informationen i ett binärt protokoll inte är användbart för användaren.

## Kan protokollet förbättras och i så fall hur?

Vid avlyssning av dataströmmar mellan systemen skulle en mer avancerad kryptering göra det svårare att tolka innehållet.

# Testkörningar

## Server

```
Received PDU JOIN from Stina:
  ClientList
  -Stina
Sent PDU NICKS to Stina
Sent PDU UJOIN to all clients

Received PDU CHNICK from Stina
Sent PDU UCNICK to all clients
  ClientList:
  -Stina

Received PDU JOIN from Stina:
  ClientList:
  -Stina
  -Lars
Sent PDU NICKS to Stina
Sent PDU UJOIN to all clients

Received PDU CHNICK from Stina.
Sent PDU UCNICK to all clients
  ClientList:
  -Lars
  -Bert

Received PDU CHNICK from Bert.
Sent PDU UCNICK to all clients
  ClientList:
  -Lars
  -Lars1

Received PDU QUIT from Lars1.
Sent PDU ULEAVE to all clients

Received PDU QUIT from Lars.
Sent PDU ULEAVE to all Clients!
```

## Klient

```
Sent PDU JOIN to server
Received PDU NICKS from server
  ClientList:
  -Lars
  -Stina
Received PDU UJOIN to server

Send PDU CHNICK to server
Received PDU UCNICK to server
  ClientList:
  -Lars
  -Lars1

Received PDU ULEAVE from server
  ClientList:
  -Lars1

Sent PDU QUIT to server
```