Datavetenskap

Jan Arenö

Sofia Axelson

addSMS

ett klassbibliotek för hantering av SMS

Examensarbete, C-nivå

2001:23

addSms

ett klassbibliotek för hantering av SMS

Jan Arenö

Sofia Axelson

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Jan Arenö

Sofia Axelson

Godkänd, Juni 06, 2001

Handledare: Stefan Alfredsson

Examinator: Stefan Lindskog

Sammanfattning

Vid sökandet efter examensarbete hittade vi en intressant uppgift hos företaget AddIn som hade behovet av att utveckla ett program i programspråket Java. Detta program skulle användas i företagets interna system för att hantera sändning av korta meddelanden till de anställdas mobiltelefoner inom företaget. Intresset för de korta meddelandena har snabbt spridits över världen och kallas för SMS (Short Message Service). I stort sett alla som idag äger en mobiltelefon har kommit i kontakt med SMS på ett eller annat sätt. De kan skickas med hjälp av telefonen och komma fram på ett par sekunder till mottagaren. Nu erbjuds även tjänsten gratis på flera Internetplatser, bland annat på Passagen och på Telia. Då skickas meddelandena med hjälp av en dator till en mobiltelefon. För att datorn ska kunna skicka meddelanden måste den ha ett modem som kan koppla upp sig mot Internet och ett program som hanterar överföringen. Överföringen av SMS sker med mellanlagringsteknik. Från sändaren via modemet överförs meddelandet först till en meddelandecentral från vilken överföringen av meddelandet levereras vidare till adresserad GSM-telefon. Det är möjligt att sända ett meddelande även om mottagaren inte är tillgänglig. Meddelandet lagras då i meddelandecentralen varifrån upprepade utsändningsförsök görs i maximalt 72 timmar. Ett sådant program var alltså önskvärt hos AddIn och har nu implementerats. Avhandlingen beskriver bakgrunden för att förstå vilka delar som behandlades, så som meddelandenas uppbyggnad och vilka regler som måste följas vid datakommunikation för att överföring ska kunna ske. Modemets funktioner och vilka kommando som behövs för att möjliggöra uppkoppling, läsning och skrivning till modemet och nedkoppling beskrivs. Vidare introduceras programspråket Java, som är ett objektorienterat språk, vilket är det språk som tyngdpunkten av vårt program ligger på. En liten del består av programmeringsspråket C. Programmets design och implementation presenteras sedan i var sitt kapitel. Slutligen utvärderas och sammanfattas arbetet. För att få en bredare förståelse finns ett avsnitt med begrepp och förkortningar som förklaras närmare.

addSMS, a classlibrary to handle SMS

Abstract

This document describes the development of a program in Java. The program is used to send short messages to mobile telephones of the employees of the company AddIn. The interest of these short messages is spread worldwide and is called SMS (Short Message Service). Almost everybody who owns a mobile telephone has been introduced to these messages sometime. They can be sent from your own phone and seconds later be delivered to a phone miles away. Nowadays the service to send SMS is offered free on several Internet sites, such as Passagen and Telia. When using this service the message is sent by a computer to the mobile telephone. To enable a computer to send messages it must have a modem to establish a connection to Internet and a program that handles the delivery. From the sender to the receiver the message is transferred through the message central where it is stored until the receiver is connected to the GSM net. This way the message may be stored in the message central up to 72 hours. This paper describes how this program was implemented and the background behind the technology. It describes how the message is being handled and how the communication with the modem and message central is done.

Furthermore we introduce the Java programming language in which the program mainly is implemented. A small part of the program is implemented in C-code since Java does not support serial communication. The design and implementation of the program is presented in separate chapters, followed by an evaluation and summary of the work.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

Idag kan de allra flesta som har tillgång till en mobiltelefon skicka korta meddelanden, så kallade SMS (Short Message Services), till en annan mobiltelefonanvändare. Det är en tjänst som är tillgänglig genom det digitala mobiltelefonsystemet GSM (Global System for Mobile communications). I Europa och nästan över hela världen har GSM blivit en standard för den nya generationen av mobiltelefoner. Det har öppnat en ny era av digital kommunikation, som förutom tal även inkluderar enklare dataöverföring. Det finns nu även möjligheten att utan mobiltelefonens hjälp skicka SMS på Internet. Några som erbjuder denna tjänst är passagen.se, Mina Sidor på Telia.se och mtnsms.com. SMS som Telia kallar för MobilText och dess uppbyggnad förklaras i avsnitt 2.1.

I sökandet efter examensarbete knöts kontakter med företaget AddIn som erbjöd ett intressant projekt. De ville ha ett programbibliotek utvecklat i programmeringsspråket Java. Programmet skulle kunna skicka SMS-meddelanden via ett modem till en SMS-central. Telias meddelandecentral för MobilText skulle användas. Programmet skulle brukas av företagets anställda genom det interna nätverket för att hantera distribution av SMS.

Resten av denna uppsats är uppdelad enligt följande:

I kapitel 2 beskrivs bakgrunden av det man behöver veta för att förstå hur arbetet har utförts. Kapitel 3 berättar i vilket syfte arbete verkställts och mer specificerat vilka krav som ställts på det. Kapitel 4 visar hur programmets design ser ut och beskriver det är upplagt. I kapitel 5 förklaras hur implementationen har gått till och vad de olika programdelarna gör. I det 6:e och sista kapitlet finns en sammanfattning och en utvärdering av arbetet. Avhandlingen avslutas med ett avsnitt med begrepp och förkortningar som används i de kommande kapitlen.

2 Bakgrund

För att få en förståelse för hur alla bitar hänger ihop beskrivs i detta kapitel de olika delarnas funktionaliteter. SMS-meddelandenas uppbyggnad och olika former beskrivs, de olika operatörernas protokoll presenteras, det vill säga de regler som ska följas vid kommunikationen, och TCP-protokollet (Transmission Control Protocol) förklaras. Nätverket använder sig av protokoll vid kommunikation genom applikation-, transport-, internet- och nätverkslagret. Dessa är bland annat http (HyperText Transfer Protocol), TCP, IP (Internet Protocol) och modem. De olika typer av modem som finns och hur ett modem fungerar tillkännages. SSL (Secure Socket Layer) förklaras och programmeringsspråket Java presenteras. Huvuddelen av vårt program är gjord i Java.

2.1 SMS

SMS-meddelande (Short Message Service) är specificerat av GSM och kan bli upp till 160 tecken långa, där varje tecken är 7 bitar. Paketen är alltså uppbyggda av 140 oktetter, där en 7-bitars teckenkodning används vid överföring av SMS mellan GSM-telefoner. 8-bitars meddelande (max 140 tecken) är oftast inte användbara av telefonerna som textmeddelanden, istället används de till data t ex ”smart”-meddelande som är ett protokoll för att skicka exempelvis grafik och ringsignaler till mobiltelefoner. 16-bitars meddelande (max 70 tecken) används för Unicode (UCS2) -textmeddelanden och är användbara av de flesta telefoner. Unicode är baserad på ASCII-tecken men använder sig utav 16-bitar istället för de ursprungliga 8-bitarna. På detta sätt ger det stöd för de flesta ASCII-teckenuppsättningar. Ett 16-bitars meddelande uppträder på vissa telefoner som flash-SMS, d v s blinkande SMS.

Överföringen av SMS sker med mellanlagringsteknik. Från den sändande GSM-telefonen överförs meddelandet först till en meddelandecentral, SMSC (Short Message Service Center), från vilken överföringen initieras av meddelandet till adresserad GSM-telefon. Därmed är det möjligt att sända MobilText även om mottagaren inte är tillgänglig. Meddelandet lagras i meddelandecentralen varifrån upprepade utsändningsförsök görs i maximalt 72 timmar.

Det finns två sätt att skicka och ta emot SMS-meddelande; genom textform och genom PDU-form (Protocol Description Unit). Textformen (som är oanvändbar på vissa telefoner) är bara kodning av bitströmmen presenterad av PDU-formen. Alfabeten kan skilja sig åt och det finns flera kodningsalternativ när ett SMS-meddelande visas. Om man läser meddelandet på

telefonen, väljer telefonen en lämplig kodning. En dataapplikation som är kapabel att läsa inkommande SMS-meddelanden, kan använda textform eller PDU-form. Om textform används är applikationen bunden till (eller begränsad av) mängden av de förinställda kodningsalternativen. I vissa fall är det inte bra nog. Om PDU-form används kan vilken kodning som helst implementeras.

Ett exempel på hur ett meddelande kan skickas i PDU-form från en mobiltelefon:

Följande exempel visar hur meddelandet ”hellohello” kan skickas i PDU-form från en Nokia 6110:

AT+CMGF = 0 //Sätter PDU-form

AT+CSMS = 0 //Kollar om modem stöder SMS-kommandon

AT+CMGS = 23 //Sänder meddelandet, 23 oktetter (exklusive de två inledande nollorna)

>0011000B916407214365F70000AA0AE8329BFD4697D9EC37

Det finns 23 oktetter i detta meddelande (46 tecken). Första oktetten (”00”) räknas inte, det är bara en indikator av längden. PDU-strängen innehåller följande (tabell 2.1.1):

Oktetter	Beskrivning
00	SMSC-informationens längd. Här är längden 0, vilket betyder att SMSCn lagrad i telefonen skall användas.
11	SMS-meddelandets första oktett.
00	Meddelandereferens. ”00” –värdet låter telefonen själv sätta meddelandets referensnummer.
0B	Adresslängd. Telefonnummerlängd (11).
91	Adresstyp. (91 indikerar internationellt format av telefonnumret).
6407214365F7	Telefonnumret i semioktetter (46701234567). Längden av telefonnumret är udda (11), därför har ett F lagts till som om numret var ”46701234567F”. Om man använder sig av ett annat format (adresstyp 81 istället för 91) skulle det ge telefonnumret oktettsekvensen ”7010325476” (0701234567). Notera att längden då blir 10 (A), vilket är jämnt.
00	PID. Protokollidentifierare.
00	DCS. Datakodningsschema. Detta meddelandet är kodat utifrån 7-bitars alfabetet. Om man har ”02” istället för “00”

	skulle det indikera att Användardata-fältet av detta meddelande skulle tolkas som 8-bits (som används i ”smart”-medelande).
AA	Giltighetsperiod. ”AA” betyder 4 dagar.
0A	Användardatalängd. Meddelandets längd. DCS fältet indikerar 7-bit data, så längden är antalet septetts (10). Om DCS-fältet sattes till 8-bit data eller Unicode, så skulle längden bli antalet oktetter.
E8329BFD4697D9EC37	Användardata. Dessa oktetter representerar meddelandet ”hellohello”.

Tabell 2.1.1 SMSC formaterat meddelande

2.2 Operatörskommunikation

2.2.1 UCP

Telias meddelandecentral för MobilText har levererats av det holländska företaget CMG som internationellt är marknadsledande för GSM inom detta teknikområde. CMG har valt att använda protokollet UCP (Universal Computer Protocol) vid kommunikationen mellan meddelandecentralen och den externa applikationen. UCP utvecklades ursprungligen av ETSI (European Telecommunications Standard Institute) för personsökningstjänster. CMG har vidareutvecklat UCP och anpassat det mot de funktioner som finns i GSM. Tele2 (Comviq) använder sig av en tidigare version av UCP än Telia men även utav SMPP (se kapitel 2.2.2)

UCP-protokollet används mellan den egna applikationen och Telias meddelandecentral och är uppbyggt kring transaktioner. Dessa består av en operation och ett returnerat resultat, som bekräftar om operationen accepterats eller ej. Varje transaktion innehåller ett transaktionsnummer som används för att separera olika transaktioner från varandra. Mottagaren av en operation skall kontrollera kommandot och om den erhållna operationen är korrekt returneras ett positivt resultat, en ACK. Om operationen däremot innehåller något fel returneras ett negativt svar, en NACK [3].

Varje UCP-sträng (operation eller resultat) består av komponenterna Start, Header, Datafält, Checksumma och Stopp (se figur 2.2.1.1).

Start	Header	Datafält	Checksumma	Stopp

Figur 2.2.1.1 UCP-strängens komponenter

- Headern innehåller fyra parametrar som separeras med tecknet ”/”. Parametrarna är transaktionsnummer, längd (det totala antalet tecken mellan start och stopp), O eller R (operation eller resultat) och operationstyp. Starttecken och header ser tillsammans ut på följande sätt:

  `<stx> trn/len/O eller R/OT/`

- Datafältet varierar i format och längd beroende på om det är en operation eller ett resultat (O eller R). Fältet består av parametrar, liksom i headern separerade med tecknet ”/”. En operation eller ett resultat kan innehålla tomma parametrar på grund av att parametern är valfri eller inte stöds.

  En extern applikation kan begära att få notifieringar när ett textmeddelande ändrar status. Tre olika statustillstånd finns: mottaget, sparat och raderat och kan kombineras på olika sätt, till exempel mottaget och raderat, mottaget och sparat eller bara mottaget.

  Vid begäran om senarelagd utsändning (delayed_delivery = 1), måste önskad tid för leverering anges. Om begäran ej skett gör meddelandecentralen upprepade försök att leverera ett meddelande i upp till 72 timmar om det från början misslyckas.

  Själva meddelandet kan vara av typen numeriskt meddelande (msg_type = 2), alfanumeriskt meddelande (msg_type = 3) och transparent data (msg_type = 4). Alfanumerisk typ är den som används, vilket är kodat med IA5-tecken. Varje tecken i själva meddelandet översätts alltså enligt en IA5-översättningstabell, till 7-bitars IA5-kod, som sedan representeras av två hexadecimala tal [3].

  Datafältet kan se ut på följande sätt:

  ```
  recievers_phonenr / send_ID // notification or not
  /////// delayed_delivery / if_dd_time_for_delivery
  /////// msg_type /// message ////////////
  ```

- Checksumman beräknas efter att själva meddelandet omvandlats till hexkod genom att addera de hexadecimala värdena för alla tecken mellan STX och ETX enligt IA5-tabellen. Checksumman består sedan av de två minst signifikanta (de två sista) hexadecimala siffrorna. Hela paketet med checksumma och stopptecken på slutet ser ut på följande sätt:

```
<stx>trn / len / O / OT /  recievers_phonenr / send_ID //
notification or not /////// delayed_delivery /
if_dd_time_for_delivery /////// msg_type /// message
////////////// checksum<etx>
```

UCP 50 heter protokollet som CMG använder idag. Det är en vidareutveckling av det ursprungligen definierade UCP-protokollet för personsökartjänster. Operationstypen som finns i UCP-operationens header är en parameter som anger vad operationen avser. Olika operationstyper använder sig av olika parametrar i datafältet.

- Operationstypen OT = 51 initieras av den externa applikationen och används för att skicka textmeddelanden. Det är här möjligt att begära notifiering från meddelandecentralen när meddelandet ändrar status. Då måste en adress och nät anges till vilket notifieringen skall levereras.

  Ett exempel på hur hela UCP-strängen kan se ut för OT = 51 då man skickar ett meddelande kan ses ovan. Resultatet kan se ut på följande sätt:

  ```
  <stx>trn / len / R / 51 / A or N // recievers_Phonenumber
  : msg_centers_timestamp / checksum<etx>
  ```
- OT = 52 initieras av Telias meddelandecentral och används vid mottagning av meddelanden.
- OT = 53 initieras av Telias meddelandecentral och används för att leverera begärd notifiering till applikationen.
- OT = 01 användes tidigare för överföring av meddelande. OT = 01 använder sig av färre parametrar än sina efterträdare OT = 31 och OT = 51. Ett exempel på hur en UCP-sträng kan se ut med denna operationstyp är enligt följande:

  ```
  <stx>trn / len / O / 01 / recievers_IPadr /
  senders_phonenr // msg_type / message / checksum<etx>
  ```

  OT = 01 resultat:

  ```
  <stx>trn / len / R / 01 / A or N / Err_code_if_N /
  checksum<etx>
  ```

2.2.2 SMPP

SMPP (Short Message Peer to Peer Protocol) är det andra protokoll som bland annat Tele2 (Comviq) använder sig av. SMPP-protokollet är ett öppet, standardprotokoll designat för att ge ett flexibelt datakommunikationsgränssnitt för överföring av korta meddelanden mellan en meddelandecentral, som en SMSC, GSM USSD (Unstructured Supplementary Services Data)

Server eller andra typer av meddelandecentraler och ett SMS-applikationssystem, som en WAP Proxy Server, Email Gateway eller andra meddelandegatewayer. Vid användning av SMPP-protokollet initierar ett SMS-applikationssystem, ESME (External Short Message Entity), en applikationslageruppkoppling med en SMSC över ett TCP/IP eller X.25-nätverksuppkoppling. Sedan kan den sända och ta emot meddelanden till och från meddelandecentralen.

SMPP stöder en mängd av tvåvägsmeddelandefunktioner som att :

- överföra meddelande från ett ESME till en eller flera destinationer via meddelandecentralen.
- ta emot ett meddelande från ett ESME via meddelandecentralen från andra SME (mobila stationer).
- fråga efter ett meddelandes status som finns lagrad i meddelandecentralen.
- annullera eller återställa ett meddelande som finns lagrad i meddelandecentralen.
- schemalägga meddelandets leveransdatum och tid.
- välja meddelandetillstånd.(Store and forward, datagram eller transaktionstillstånd).
- bestämma meddelandets leveransprioritet.
- definiera meddelandets datakodningstyp.
- sätta meddelandets giltighetstid.

2.2.3 CIMD

CIMD (Computer Interface to Message Distribution) är protokollet som bland annat Europolitan använder. När ett meddelande som använt sig av CIMD anlänt till meddelandecentralen använder sig meddelandecentralen sin policy av upprepade försök att leverera meddelandet. Om levereringen misslyckas lagras meddelandet i meddelandecentralens databas till försöket lyckas. Om ett permanent fel uppstår eller att giltighetstiden av meddelandet utgår, som är 72 timmar, raderas meddelandet.

Varje meddelande, operation eller svar består av en header, ett datafält och en slutdel av meddelandet. Headern består av starttecken, operationskod och paketnummer på meddelandet:

```
<stx> operationskod : paketnummer <tab>
```

Datafältet består av en lista av parametrar. Dessa är parametertyp och parametervärde:

```
parameter : parametervärde <tab>
```

Slutligen finns en del som består av meddelandets checksumma och stopptecken:

```
checksumma <etx>
```

Hela meddelandet ser alltså ut som följande:

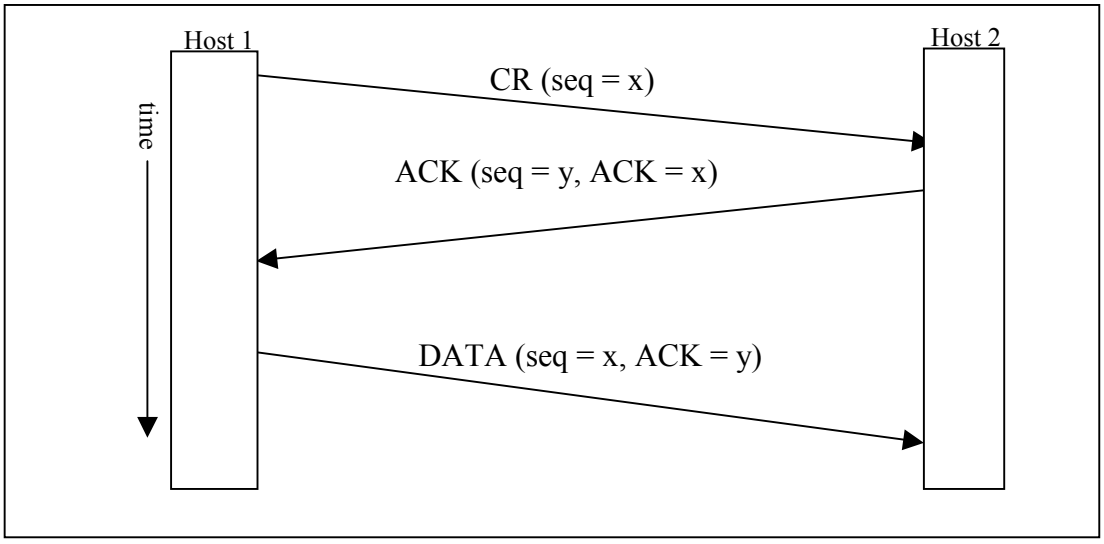
```
<stx> operationskod : paketnummer <tab> parameter :
parametervärde <tab> checksumma <etx>
```

Protokollet använder sig liksom UCP-protokollet av IA5s 7-bitars teckenkodning.

2.3 TCP

Kommunicerande datorer i ett nätverk behöver tala samma språk för att förstå varandra. För att kommunikationen ska fungera så används en uppsättning av regler, ett protokoll. Detta protokoll är en uppsättning regler som på olika nivåer beskriver hur paketering och adressering ska utföras för att ett datapaket ska kunna skickas på nätverket. Det finns olika protokoll till olika typer av nätverk. Ett av dem är TCP (Transmission Control Protocol) [4].

TCP associeras ofta med IP (Internet Protocol). Protokollet bryter ner godtyckligt långa meddelanden till 64 kB-block som skickas på Internet. Det är gjort för att användas ovanpå ett otillförlitligt datagramnätverk, exempelvis IP. Varje byte som har transporterats får sitt eget sekvensnummer. Vid etablering av förbindelser använder sig TCP av ”three-way handshake”, som innebär att vid uppkoppling skickar mottagaren ett svar, en ACK, om att han fått en begäran om uppkoppling. Begäraren skickar därefter ett svar, en ACK, tillbaka om att det mottagits ett svar (se figur 2.3.1).

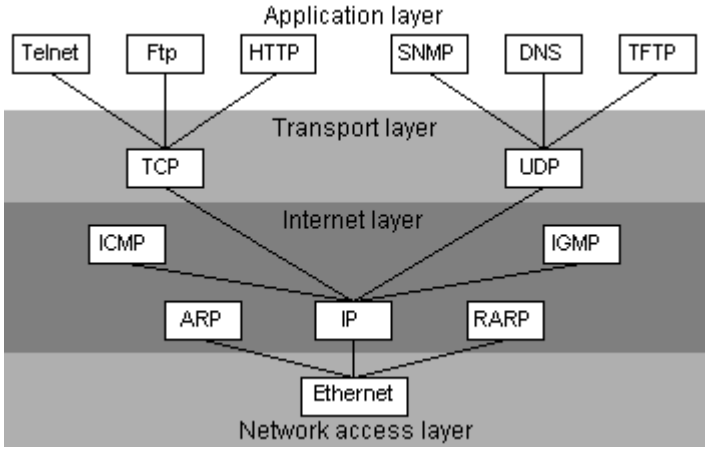


Figur 2.3.1 Three-way handshake

Utvecklingen av TCP/IP började i slutet av 60-talet då DARPA (Defense Advanced Research Projects Agency) fick i uppdrag att utveckla ett nätverk åt BBN (Bolt, Bernek and Newman). Ett år senare fanns ett fungerande nätverk med namnet ARPANET. Det protokoll man använde kallades NCP (Network Control Protocol). Redan 1971 täckte ARPANET en stor del av USA och 1973 kopplades länkar till Europa. Så småningom växte nätverket så att NCP inte längre kunde hantera trafikeringen. Detta ledde till utveckling av TCP/IP. År 1983 bestämde sig DoD (Department of Defense) för att använda TCP/IP för all form av datakommunikation.

I den så kallade OSI-modellen [2, p. 28 - 35], som beskriver hur öppen datakommunikation ska gå till och bestämmer definition av gränssnitt och protokoll, finns sju olika lager som kommunicerar med varandra. Lagren heter applikations-, presentations-, sessions-, transport-, nätverks-, länk-, och fyslagret. Eftersom denna modell inte fanns då TCP/IP skapades skiljer sig TCP/IP-modellen lite ifrån OSI-modellen. TCP/IP-modellen har 4 lager: applikations-, transport-, internet- och nätverks/accesslagret (se figur 2.3.2).

Protokollen TCP/IP har till uppgift att etablera kontakt mellan datorer på Internet och överföra data i små paket, *packets*. IP-protokollet står för adressering av de kommunicerade parterna och TCP-protokollet sköter transporten av data. Till dessa protokoll tillkommer ett applikationsprotokoll, vilket är bundet till en applikation, till exempel ett FTP-program, en webläsare eller en mailklient.



Figur 2.3.2 TCP/IP-modellen

IP motsvaras av OSI-modellens nätverksprotokoll (TCP/IP-modellens internetlager som illustreras ovan). IP-protokollet är kärnan i TCP/IP och kan betraktas som motorn i densamma. Det hanterar felhantering och Segmentation and Reassembly (SAR), det vill säga

en funktion som delar upp data från högre lager före sändning och som på andra sidan sedan sätter ihop dessa till paket för högre lager igen.

TCP är det mest använda protokollet i transportlagret. Transportprotokoll erbjuder en viss service upp till applikationen. TCPs service är att det är pålitligt, förbindelseorienterat och flödesbaserat. Detta innebär att all data som skickas kommer fram, att man måste koppla upp en förbindelse innan sändning och att datan skickas som en ström. TCP har också egenskapen att det har flödeskontrollerade länkar med full duplex, det vill säga protokollet kontrollerar att sändaren inte skickar data snabbare än mottagaren kan ta emot och att de kan skicka till varandra samtidigt. Protokollet har även en stockningskontrollerande egenskap, vilket innebär att om två sändare som sänder till samma mottagare delar på länkar kan det bli paketförluster. TCP kontrollerar detta och sänder i så fall långsammare. Protokollet används i applikationer där det är viktigt att data kommer fram, det vill säga i de flesta administrativa applikationer.

2.4 Modem

Idag börjar modem bli ett självklart tillbehör till datorn. Modemet gör det möjligt att koppla upp sig mot andra nätverk och på så sätt kommunicera med omvärlden. Internet brukar vara den största anledningen till att skaffa modem. När man talar om Internet är det oftast World Wide Web (www) som avses. Andra tillämpningar som finns på Internet är diskussionsgrupper, chat och elektronisk post.

Ett modem gör det möjligt att använda det vanliga telefonnätet för datakommunikation. Datorn arbetar med digitala signaler, ettor och nollor. De digitala signalerna dämpas kraftigt i telefonnätet och måste därför omvandlas till en mer lämplig form, det vill säga analoga signaler.

De flesta telefonlinjer är digitala men linjen till huset är vanligtvis analog, vilket betyder att de är gjorda för att överföra en vågformad signal, vilket är skapat av ljudvågen från människorösten. Signalen ser ut som en sinusvåg av varierande frekvens och amplitud. En digital signal ser ut som en fyrkantvåg, ex +5V är en 1-bit och –5V en 0-bit.

För att sända data från en dator över telefonlinjen omvandlar (modulerar) modemet den digitala signalen, ettor och nollor, från datorn till en tonliknande analog (MOdulation). Tonerna är dock obegripliga för den mottagande datorn så i andra änden översätter sedan ett annat modem från analog till digital signal (DEModultion), det vill säga till de för datorn begripliga ettorna och nollorna. MO-DEM (modem) är alltså ett MOdulator-DEModulator. När modemet används brukas vanligen ett kommunikationsprogram för att använda modemet

och ringa ut på en telefonlinje. När uppkoppling sker mot ett modem presenterar sig modemen för varandra, något som kallas handskakning. När detta sker talar modemen om för varandra vilket protokoll de använder och beroende på förbindelsens kvalitet enas de om en kommunikationshastighet.

När modemen är överens om hastighet etc. är förbindelsen klar för dataöverföring. Förbindelsen består i det här skedet av en vågformad signal, kallad bärvåg. De data som skall sändas vävs sedan in i bärvågen och kan filtreras ut av modemet på andra sidan. Om bärvågen förloras är förbindelsen därmed bruten men kan återtas om den inte varit bruten en länge tid.

Det finns tre typer av modem för en PC: externt, internt och PC-card. Ett externt modem finns utanför datorn och pluggas in till en anslutare på PC:n, till exempel serieporten. Det interna modemet är antingen inbyggt på moderkortet eller ett expansionskort som är insatt i datorn och har en inbyggd serieport. Med andra ord är modemkortet både en serieport och ett modem. PC-card eller PCMCIA (Personal Computer Memory Card International Association) som det tidigare kallades är en speciell typ av expansionskort för i huvudsak bärbara datorer. Ett sådant kort kan innehålla olika typer av funktionalitet varav modem är en. Kortet, som är av kreditkortsformat, sätts in i ett speciellt fack som så gott som alla bärbara datorer idag är utrustade med. Det finns också PC-card som innehåller både modem och nätverkskort. Annars är det huvudsakligen modemets hastighet som avser typen. Modemens hastighet mäts i hur många tusen (1024) bitar per sekund de är kapabla att överföra vilket uttrycks som kbit/s.

Det finns en mängd olika regler och föreskrifter hur kommunikationen skall gå till. För att modem skall kunna kommunicera med varandra oberoende av typ och fabrikat finns det fastställda standarder som modemen måste följa. De vanligaste beteckningarna när det gäller standarder är den så kallade V-serien från de internationella teleförvaltningarnas standardiseringsorgan CCITT (se tabell 2.4.1). Dessa specificerar bland annat hur kommunikationen skall gå till i olika hastigheter. För att kunna använda en viss hastighet måste modemen i båda ändar klara av motsvarande standard.

Vanliga standarder:

Beteckning	Bitar per sekund
V.22	1200
V.32	9600
V.34	28000
V.90	56000

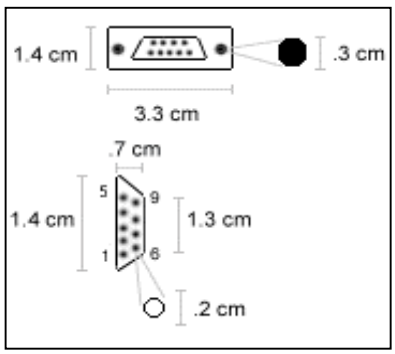
Tabell 2.4.1 Standarder.

De flesta moderna modem stöder datakomprimering, vilket gör att överföringshastigheterna kan öka ytterligare. För att detta ska fungera måste datorn ha en 16550 kompatibel UART (Universal Asynchronous Reciever Transmitter)-kommunikationsport. Ett UART-chip används för att kontrollera porten. Det nya UART-chipet klarar av att hantera data snabbare och behöver inte som sina föregångare skicka IRQ-anrop för varje tecken som mottas.

2.4.1 Seriellporten.

En serieport är en I/O (Input/Output) –komponent. De används för att ansluta mus och modem, så den ena porten kan vara upptagen av musen. De flesta datorer har en eller två serieportar som var och en har en 9-pinnaranslutare (se figur 2.4.1.1). Kontakten var ursprungligen en 25-pinnaranslutare men eftersom i regel bara 9 pinnar utnyttjas så använder man istället en 9-pinnars. Porten följer en standard som kallas RS-232C. Dataprogram kan sända data (bytes) till överföringspinnarna (output) och ta emot bytes från mottagarpinnarna (input). Överföringen av data sker seriellt, 1 bit åt gången, asynkront. Detta görs med hjälp av UART chipet. UART 16550-kretsen stödjer hastigheter upp till 115 kbps. De andra pinnarna är för kontrolländamål.

Serieporten är mycket mer än bara en anslutare. Den konverterar data från parallell till seriell och ändrar den elektriska representationen av data. Inuti datorn flödar databits parallellt, d v s använder många linor på samma gång. Seriellt flöde är en ström av bitar över en ensam lina, så som på serieanslutarens överförings- eller mottagarpinnarna. För att serieporten ska kunna skapa ett sådant flöde måste den konvertera data från parallell (i datorn) till seriell på överföringspinnarna (och omvänt).



Figur 2.4.1.1 Serieports honkontakt

När ett program vill kommunicera med en enhet i datorn används en unik adress som tilldelas vid installationen. Adressen används för att skicka och ta emot de kommandon som behövs för att enheten skall fungera. I/O-portar fungerar som ett gränssnitt mellan processorn och externa enheter. För att kommunicera med dessa portar adresserar man dem på samma sätt som speciella adresser i RAM. Dessa portar accessas emellertid med speciella instruktioner från processorn (ex IN eller UT).

Eftersom datorn behöver kommunicera med varje seriell port, måste operativsystemet veta att varje seriell port existerar och vart den är, d v s dess I/O-adress. Det behöver också veta vilken linje (unikt IRQ-nummer/avbrottsnummer) den seriella porten måste använda för att begära service från datorns processor, så att denne vet vilken enhet som begärt en förfrågan. Den frågar efter service genom att sända ett interrupt (avbrott) på denna linje. När ett IRQ genereras hanteras det av en Interrupt Controller som distribuerar avbrottet vidare till processorn efter en viss uppsatt prioritetsordning. Varje seriell port måste alltså lagra både dess I/O-adress och dess Interrupt ReQuest-nummer: IRQ.

Den seriella drivrutinen (mjukvara) upprätthåller en tabell som visar vilken I/O-adress som överensstämmer med vilken serieport. Denna mappning av namn (till exempel linux serieport ”ttyS1”) till I/O-adresser (och IRQ:er) kan båda sättas och överblickas av ”setserial”-kommandot i linux. Detta sätter inte I/O-adressen och IRQ:n i hårdvaran. Alltså vilken fysisk port som överensstämmer med exempelvis ”ttyS1” beror både på vad den seriella drivrutinen tycker och vad som är satt i hårdvaran.

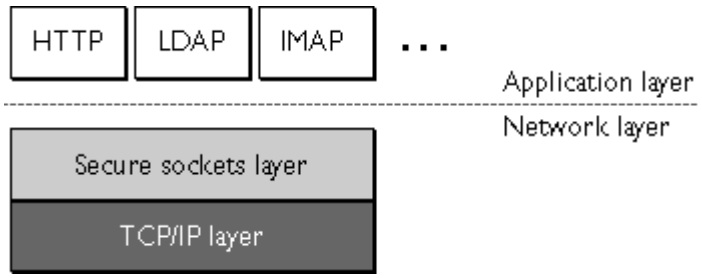
2.4.2 Modemkommandon

Kommandon skickas till modemet från kommunikationsmjukvaran över samma seriella ledare som används för att sända data. Kommandona är kryptiska och korta ASCII-kommandon där alla kommandosträngar inleds med bokstäverna AT som är en förkortning av ATtention. De flesta modem använder en mängd AT-kommandon. Ett exempel är ATZ&K3<enter>, som innehåller två kommandon: Z och &K3 och används för att aktivera hårdvaruflödeskontroll (RTS/CTS) mellan datorn och modemet. Kommandosträngen avslutas med ett enterslag. Ett annat exempel är ”ATDT05412345” för att ringa numret 05412345. DT är en förkortning av Dial Tone. Det finns ungefär hundra olika kommandon. När kommunikationsmjukvaran startas, sänder den först en initieringssträng av kommandon till modemet för att konfigurera det. Alla kommandon skickas på den ordinarie datalinjen före modemet ringer (eller tar emot ett samtal).

Från det att ett modem är kopplat till ett annat modem (on-line tillstånd), går allt som sänds från datorn till modemet direkt till det andra modemet och tolkas av modemet som ett kommando. Det finns ett sätt att komma ur detta tillstånd och gå tillbaka till kommandotillstånd där allting som skickas till modemet tolkas som ett kommando. Datorn sänder bara ”+++” med en speciell tidslucka före, mellan varje + -tecken och efter det sista. Om denna tidslucka är korrekt återgår modemet till kommandoform. Ett annat sätt för att göra detta är genom en signal på en viss modemkontrollinje. Det finns många gemensamma AT-kommandon, men olyckligtvis finns det några olika variationer av dem så det som fungerar för ett modem är inte säkert att det fungerar för ett annat modem.

2.5 SSL

SSL (Secure Socket Layer) är ett öppet, icke-patentskyddat protokoll utvecklat av Netscape för överföring av dokument via Internet. Det är det vanligaste sättet att leverera kryperad överföring av data mellan webbrowsrar och webbservrar. SSL bygger på att använda nycklar för att kryptera data som överförs över SSL-uppkopplingen. Både Netscape Navigator och Internet Explorer stöder SSL och många webbsidor använder protokollet för att leverera hemlig användarinformation, så som kreditkortsnummer. Webbsidor som begär en SSL-uppkoppling börjar med `https` istället för `http`. SSL skapar en säker uppkoppling mellan en klient och en server över vilken data kan sändas säkert. Från och med version 3.0 så blev protokollet en godkänd standard av Internet Engineering Task Force (IETF) under namnet TLS (Transport Layer Security) [5].



Figur 2.5.1 SSL.

SSL-protokollet körs ovanpå TCP/IP och under högre lagers protokoll som till exempel HTTP eller IMAP (se figur 2.5.1).

SSL-protokollet inkluderar två delprotokoll: SSL-recordprotokollet och SSL-handshakeprotokollet. SSL-recordprotokollet definierar formatet som används för att överföra

data. SSL-handshakeprotokollet använder SSL-recordprotokollet för utbyte av en serie meddelanden mellan en server och en klient när de sätter upp en SSL-uppkoppling första gången. Detta utbyte av meddelande är designat för att underlätta följande aktioner:

- Presentera (Authenticate) servern för clienten.
- Tillåta klienten och servern att samla de kryptografiska algoritmerna som de båda behöver.
- Valfri presentation (authenticate) av klienten till servern.
- Använda nyckel krypteringsteknik för att generera delad säkerhet.
- Sätta upp en krypterad SSL-uppkoppling.

Målen med SSL-protokollet är:

1. Kryptografisk säkerhet (Cryptographic security)
   SSL ska användas för att upprätta en säker uppkoppling mellan två parter.
2. Praktisk genomförbarhet (Interoperability)
   Oberoende programmerare ska kunna utveckla applikationer som använder SSL som sen framgångsrikt ska kunna byta kryptografiska parametrar utan kunskap av varandras kod.
3. Uttänjbarhet (Extensibility)
   SSL söker för att ta fram en konstruktion i vilken en ny ”public key” och större delen av krypteringsmetoderna kan integreras. Detta innebär två delmål: att förebygga behovet av att skapa ett nytt protokoll (och riskera introduktionen av möjlig svaghet) och att undvika behovet av att implementera ett fullständigt nytt säkerhetsbibliotek.
4. Relativ effektivitet/kompetens (Relative efficiency)
   Kryptografiska operationer tenderar till att vara mycket processorintensiva, särskilt ”public key”-operationer. Av denna anledning har SSL-protokollet integrerat ett valfritt session-hanteringsschema för att minska antalet uppkopplingar som behöver sättas upp från början. Hänsyn ska tas för att minska nätverksaktiviteten.

2.6 Java

Java är ett objektorienterat programmeringsspråk som med avseende på format, syntax och språkliga konstruktioner är likt C++. En skillnad mellan Java och C++ är att Java saknar möjlighet att använda pekare och adresser.

Java är designat för att vara så enkelt som möjligt. Som sagt används inga pekare, men automatisk minneshantering och stöd för undantagshantering finns. Ett delprogram, en Garbage Collector (GC), tar hand om minnesfrigöring. Det sveper över minnet då och då och ser om det finns områden som ingen använder men som är allokerade. Dessa deallokeras då av GC.

Java exekveras och används på klientsidan i ett distribuerat datasystem. Ett Javaprogram laddas normalt över från en webserver till en klient (browser) via http-protokollet och därmed i ett grafiskt gränssnitt, men det finns också möjlighet att skriva vanliga textbaserade applikationer. Eftersom ett Javaprogram som laddas ner innehåller främmande kod, som exekveras i datorn, omges javakoden av stränga säkerhetsrestriktioner. Detta för att omöjliggöra att det förändrar systemet som det exekverar i. Enkelt uttryckt har ett Javaprogram ingen möjlighet att förändra något annat än lokala variabler i programmet. Då man laddar hem ett Javaprogram kontrollerar den Java-runtime-miljö man har att programmet är säkert och om det inte förefaller helt säkert vägrar miljön att exekvera det. Trots detta finns det inga garantier att Java är 100% säkert, men jämfört med andra programmeringsspråk kan det ändå betraktas som mycket säkert [1].

Det finns fyra olika typer av program man kan skriva med Java:

- *Java Applets*, är små, ”fullständiga” program som kompileras innan man exekverar dem. Exekveras från ett annat program (normalt en webbläsare). Körs i en browser eller med appletviewer. En Java applet har ingen main-metod utan har en uppsättning metoder som anropas av webbläsaren. Metoderna har namn som init, destroy, start och stop som antyder när metoderna anropas. En Java applet har ett utseende som visas på den yta som den tilldelats av webbläsaren. Utseendet på denna yta definieras av appletens paint-metod, som innehåller anrop som grafiskt ritar upp ytans utseende. Ett Javaprogram kan jämföras med ett C/C++ -program.
- *Content handlers, Protocol handlers*, är en speciell typ av program som t ex används för att en browser dynamiskt skall kunna använda nya datatyper som t ex QuickTime-filmer, PhotoCD-data och MIDI. Detta används för att en browser skall

kunna behandla datatyper den inte har programvara för att hantera, genom att man laddar ner denna typ av funktion samtidigt med datan.

- *Servlets*, är ett generellt, plattformsoberoende och serveroberoende sätt för att utöka och anpassa tjänster på servern. En servlet laddas ifrån den lokala disken på servern eller ifrån en annan server. Den kan kopplas till en viss HTML-sida så att den startas när en viss sida refereras och kan också kopplas till en viss typ av http-begäran eller till en viss socket-port.
- *Javaapplikationer*, är självständiga program som exekveras som fristående program men som dock kräver en JVM (Java Vitual Machine), ett mjukvaruskal, för att kunna köras. Innehåller alltid en main-metod i den huvudklass som man startar applikationen med.

I motsats till C/C++ är Java helt objektorienterat, det vill säga kompilatorn kräver att allting är objekt. Globala data och fristående funktioner tillåts inte. Alla klasser i Java härstammar direkt eller indirekt från objektklassen. I förhållande till C/C++ är Java dessutom från grunden utformat för att arbeta i en distribuerad miljö. Det finns inbyggt stöd för både TCP/IP och HTTP [1].

Klasserna i Java representerar en viss objekttyp, det vill säga hur objekt av klassen skall representeras och vilka metoder de skall stödja, med andra ord objektens egenskaper och beteende. En klass kan jämställas med att man definierar en egen typ, så som heltal (int), tecken (char) och så vidare. Utifrån en klass skapas sedan objekt av denna klass, där ett objekt då är en unik förekomst, en instans, av denna klass. Klasser är alltså de begrepp som definieras i programkoden och utifrån de definierade klasserna i ett program kan sedan objekt skapas. En klassdeklaration påbörjas med nyckelordet class följt av namnet på klassen.

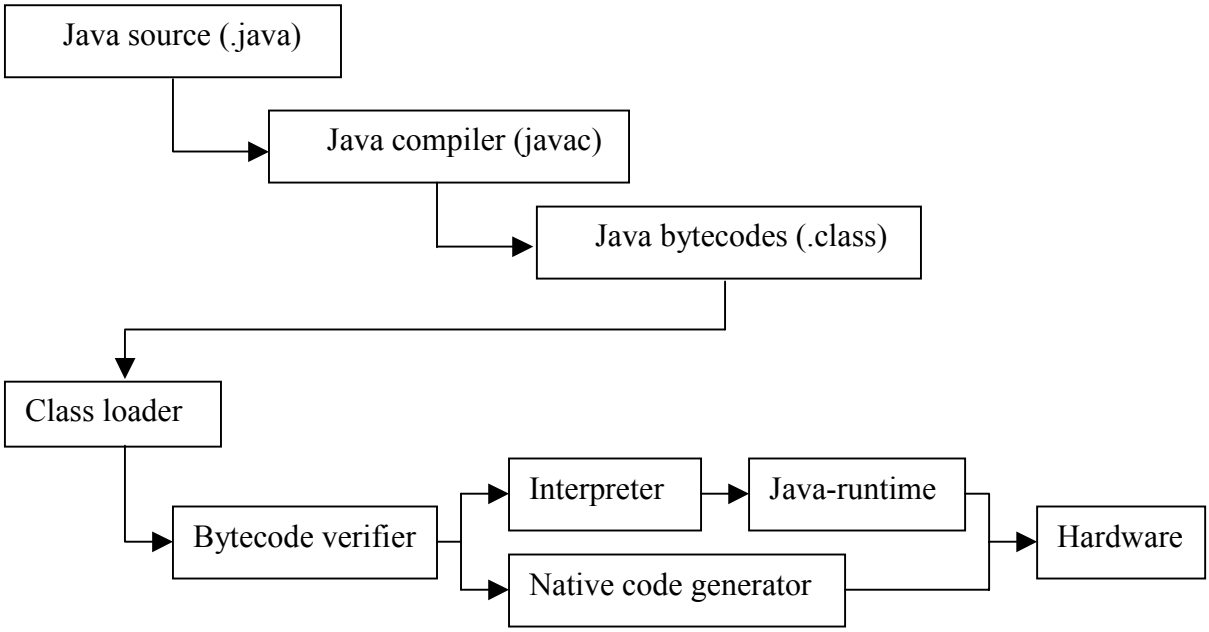
Gränssnitt (interface) är en teknik för att definiera abstrakta typer som endast innehåller abstrakta metoder samt konstanter. Ett gränssnitt definieras för att beskriva en uppsättning metoder som en klass måste ha för att ha en viss egenskap. De klasser som har dessa egenskaper väljer sedan att implementera detta gränssnitt. Den stora skillnaden mellan ett gränssnitt och en klass är att klassen är en blandning av specifikation och implementation, medan ett gränssnitt endast är en specifikation. Nyckelordet interface används framför namnet på gränssnittet.

Javaprogram skrivs på samma sätt som vanliga program och sparas i filer vars namn avslutas med ”.java”. Dessa program kompileras till ett maskinoberoende format (Java Byte Codes) som gör att ett program kan exekveras på vilken dator som helst, som har en Java-

runtime-miljö (The JVM). Java är alltså oberoende av processor, operativsystem och användargränssnitt.

Strukturen på Java.

För att förklara strukturen av Java kan förhållandet mellan olika komponenter i Java visas med följande bild:



Figur 2.6.1 Javakompilering.

Bilden (figur 2.6.1) visar ett Javaprogram (”x.java”) som kompileras med ”javac” så att en fil med Java Byte Code skapas, denna fil heter då ”x.class”. Då man skall exekvera programmet t ex när en browser refererar det, laddas det hem via nätet, laddas in av en Class Loader som lämnar det vidare till en Byte Code Verfier som kontrollerar att innehållet verkar ok. Efter detta tas det om hand av vederbörliga funktioner, beroende på om det är ett Javaprogram eller en Java Applet. Sista steget är att exekvera programmet.

Exempel på program/verktyg som finns i JDK (Java Developers Kit) är:

- appletviewer, är ett program som används för att exekvera ett Javaprogram utan att man använder en browser.
- Java, är en interpretator som används för att exekvera kompilerade Javaprogram.
- Javac, är Javakompilatorn som används för att översätta Javaprogram till bytecodes.
- Javadoc, är ett verktyg som kan användas för att skapa API-dokumentation i HTML-format.

- Javah, används om man vill använda C/C++ -kod i Javaprogram.
- Javap, är en ”dissemblator” som kan användas för att översätta bytecodes till Javakod.

2.6.1 Java Native Interface

Java Native Interface (JNI) är en del av JDK i Java och ett så kallat ”native programmeringsinterface”. Detta betyder att man kan använda sig av programmeringskod som är skrivet i andra språk än Java. Genom att använda JNI så kan du få din kod att bli portabel mellan olika språk och plattformar.

JNI tillåter Java kod som körs genom Java Virtual Machine (JVM) att operera med applikationer och bibliotek skrivna i andra språk, exempelvis C/C++ och assembler. Utöver det tillåter Invocation API dig att bädda in Java Virtual Machine i dina nativeapplikationer.

Man använder sig utav JNI för att skriva nativemetoder som hanterar de situationer som inte kan hanteras enbart med hjälp av programspråket Java. Exempelvis kan du behöva använda det vid följande fall :

- Javas standardbibliotek kanske inte kan hantera de plattformsspecifika finesser som kan finnas.
- Om man redan har ett bibliotek eller en applikation i ett annat språk som man vill göra tillgänglig till Javaapplikationer.
- Man vill implementera en liten del tidskritisk kod i ett lågnivåspråk, såsom assembler, och sedan låta en Javaapplikation anropa dessa funktioner.

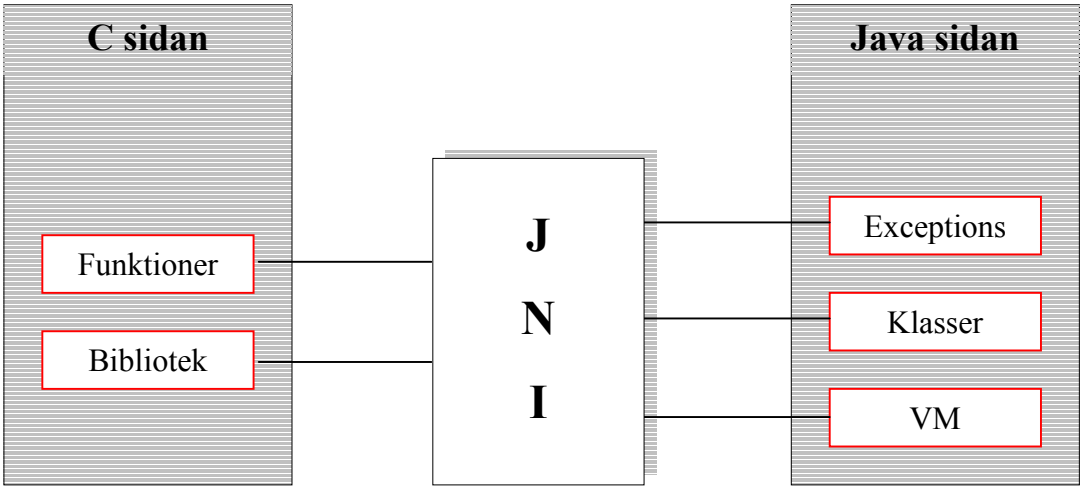
Genom att programmera genom JNI så kan du använda nativefunktioner till att lösa flera operationer. Bland annat till att lösa problem som löses bäst utanför Javamiljön.

JNI tillåter även nativefunktioner att hantera Javaobjekt på samma vis som Javakod hanterar dessa klasser. En nativefunktion kan skapa nya objekt, inklusive arrayer och strängar, och sedan använda dessa objekt för att uppnå sitt mål. Nativefunktionen kan även uppdatera objekt som är nyskapade eller som är inskickade i nativefunktionen. Dessa uppdaterade objekt är eller kan göras tillgängliga för Javaapplikationen. Både Java och nativesidan kan skapa, uppdatera och hantera Javaobjekt och sen dela med sig till den andra sidan.

Nativemetoden kan lätt anropa Javametoder. Ofta så har man redan löst ett specifikt problem i Javakoden och nativefunktionen behöver inte deklarera funktionaliteten igen utan kan anropa Javafunktionen direkt ifrån sig själv.

JNI tillåter programmeraren att använda fördelarna med Java i sina nativefunktioner. JNI klarar bland annat av att fånga och kasta undantag (exceptions) från nativefunktionen och

hantera dessa i Javaapplikationen. Nativefunktioner har också möjligheten att använda JNI för att göra runtime type checking.



Figur 2.6.1.1 JNI-kopplingen mellan C och Java

3 Syfte och problemställning

Syftet med detta projekt på företaget AddIn var att utveckla ett program som har som funktion att skicka SMS via ett modem till en SMS-central. Från början fanns kravet att även skicka via SSL och TCP/IP, men efter närmare undersökningar ansågs ett sådant abonnemang hos Telia vara för dyrt för detta projekt. Detta bestämdes en tid in i arbetet med projektet, så en början på SSL och TCP/IP-klassen implementerades, men är inte fullständigt. Ett klassbibliotek i programspråket Java implementerades för att kunna köra programmet i Linux. På grund av att Java inte har något generellt stöd för modem kommunikation (kommunikation med serieporten), implementerades det i C för att få stöd för modem i Linux. För att sätta ihop C-koden och Javakoden användes sedan Java Native Interface (JNI). Det mjukvarukrav som ställdes var att skapa ett fungerande J2SE-implementation (Java 2 Standard Edition). De olika operatörerna, Telia, Tele2 (Comviq) och Europolitan använder sig av olika protokoll vid kommunikation av SMS och måste därför specificeras för varje operatör. I första hand var kravet att implementera stöd för Telias protokoll UCP. Telia använder sig av tre olika telefonnummer för att koppla upp sig med beroende på hur många meddelanden som ska skickas. Detta skulle hanteras av programmet.

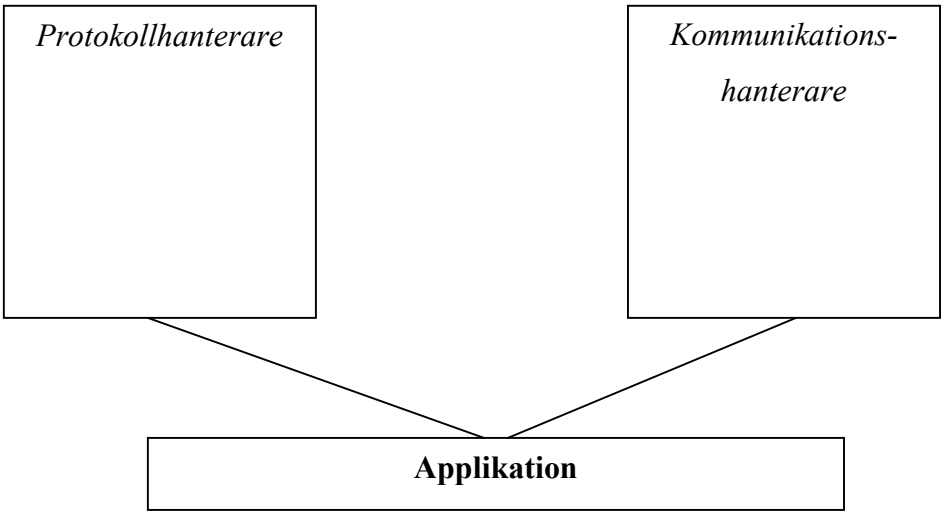
En testapplikation, som använder sig av samtliga funktioner i klassbiblioteket implementerades också. Applikationen tar emot ett meddelande och skickar det till av användaren angivna mottagare.

Telias ”Handbok för applikationsutvecklare” [3] användes som stöd vid skapandet av klassbiblioteket.

4 Design

Klassbiblioteket utvecklades för att vara så modulärt som möjligt. Biblioteket delades upp i två delar (figur 4.1), en del som tar hand om kommunikationen till och från meddelandecentralen och en som har hand om själva SMS-paketet. Eftersom endast stöd för modem samt protokollet UCP är implementerat så består biblioteket endast av dessa två huvudklasser med respektive interface.

UCP-klassen behöver även en klass för att konvertera mellan ASCII och IA5.



Figur 4.1 Klassbiblioteket delades upp i två delar

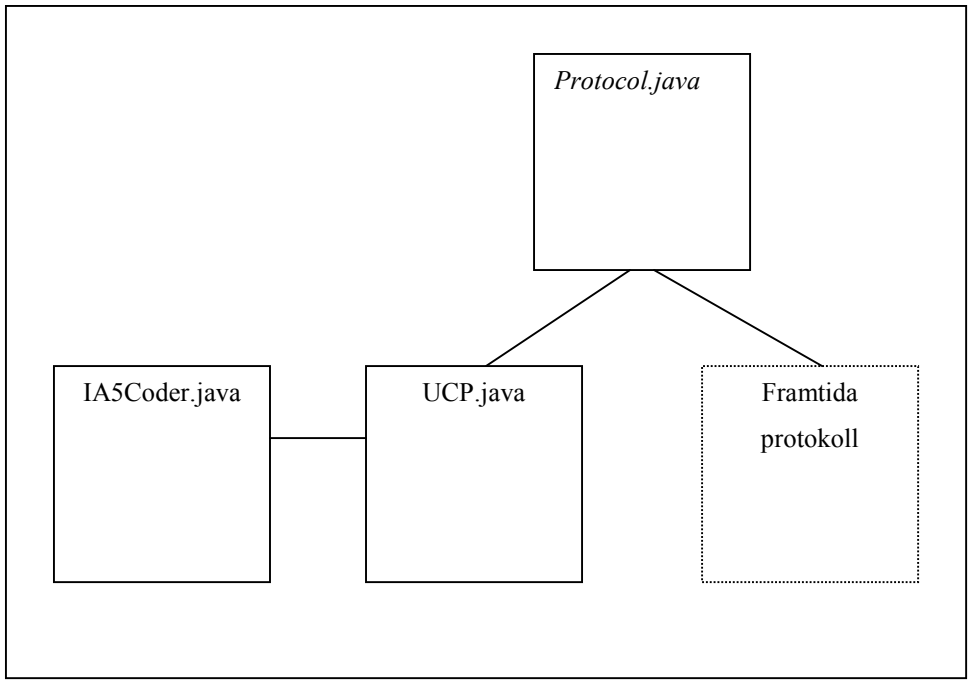
4.1 Protokollhanteraren

Protokollhanteraren (figur 4.1.1) består av en UCP klass, med en tillhörande IA5Coder samt ett generellt interface som används av UCP och ska användas av alla framtida klasser.

Eftersom de olika SMS-protokollen skiljer sig på många sätt kan inte alla representeras av en enda klass. Istället finns det ett interface som sätter vissa begränsningar på hur en ny protokollklass får implementeras. Interfacet Protocol definierar de funktioner som krävs för att en klass från Kommunikationshanteraren skall kunna hantera objektet, exempelvis setResponse() och setMessage().

Klassen UCP hanterar UCP-protokollet [3] genom att erbjuda ett sätt att ställa in parametrarna i SMS-paketet. Klassen kan även hantera svaren från meddelandecentralen och svara på om paketet blev levererat vid sändning eller ej.

Det är tänkt att IA5-kodaren ska gå att hänga på framtida protokollklasser som använder sig av IA5-kodning. IA5Coder kan både koda och avkoda från IA5-tecken.



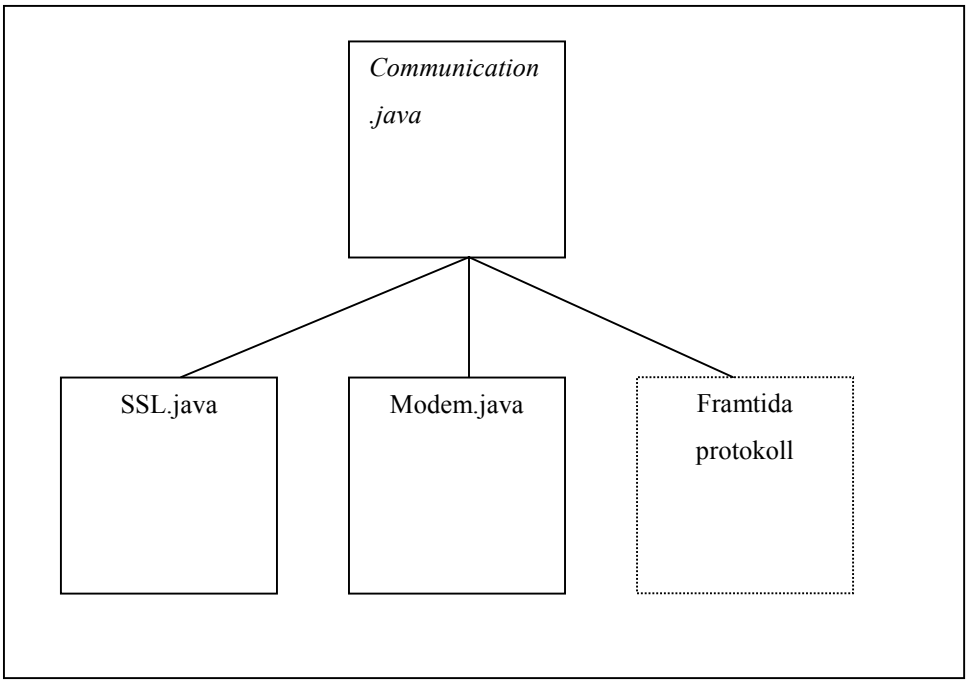
Figur 4.1.1 Design av Protokollhanteraren

4.2 Kommunikationshanteraren

Kommunikationshanteraren har i uppgift att sätta upp en fysisk förbindelse med meddelandecentralen. Den skickar även SMS-paketet i den form som Protocolklassen erbjuder.

Kommunikationshanteraren klarar av att hantera vilken protokollklass som helst så länge den bygger på Protocol-interfacet.

Kommunikationshanteringsdelen består av två klasser och ett interface (figur 4.2.1). Klassen Modem är den klass som till fullo är implementerad till Linux och stödjer att man över telefonnätet kopplar upp sig till meddelandecentralen med hjälp av modem. Klassen SSL ska ha stöd för att kommunicera med meddelandecentralen via Internet och SSL. Den är dock inte fullt utvecklad, utvärderad och testad då AddIn anser att priserna för en sådan tjänst är för höga hos Telia för detta projekt.



Figur 4.2.1 Design av Kommunikationshanteraren

4.3 Testapplikationen

Eftersom det är tänkt att klassbiblioteket skall integreras med andra program så är en enkel testapplikation i kommandoform ett sätt att nå funktionaliteten i klassbiblioteket från andra applikationer utan att behöva göra en ny implementation.

Ett grafiskt gränssnitt hade givit bättre och lättare funktionalitet för en klientanvändare, men när programmet är tänkt att ligga på servern och anropas från andra program så är ett grafiskt gränssnitt helt onödigt.

De inställningar som måste göras för att kommandoraden inte skall bli allt för lång har försökts minimerats.

5 Implementation

Detta kapitel behandlar hur klassbiblioteket implementerades och utvecklar hur problemen löstes i vissa klasser, funktioner och kodblock. Strukturen på klasserna kommer inte tas upp så mycket utan bara diskuteras ytligt.

Först behandlas Interfacen, sedan UCP följt av modemklasserna och testapplikationen.

5.1 Interfacen

Det finns två interface (se 2.6) i addSMS klassbibliotek. Ett för protokoll och ett för kommunikation.

Protokollinterfacet specificerar vad som minst krävs av en protokollklass. Framför allt så är det för att en kommunikationsklass skall kunna veta att en viss funktion finns, till exempel setResponse().

Kommunikationsinterfacet specificerar en generell kommunikationsklass. De som implementerar en klass som använder sig av kommunikationsinterfacet måste ha med vissa metoder. Till exempel så måste varje klass ha en initieringsfunktion, en uppkopplingsfunktion, en sändfunktion och en nedkopplingsfunktion. Funderingar fanns på att ha en läsfunktion men eftersom detta nästan alltid sker efter skrivning så ansågs det vara bättre att sändfunktionen tog hand om och sände svaret till protokollklassen.

Båda interfacen är abstrakta, vilket innebär att en klass som använder interfacet måste implementera funktionerna. Interfacen har försökts gjorts så små och lätthanterliga som möjligt, men ändå ha all funktionalitet som behövs för nya eventuella klassimplementationer.

5.2 UCP-klassen

Denna javaklass bygger på SMS-protokollet UCP (*se kapitel 2.2.1*) och har implementerats med stöd för sändning av SMS. Eftersom tillgången till att ta emot meddelande från meddelandecentralen inte funnits så har ett val gjorts att inte implementera stöd för operationstyperna 52 och 53 (notifiering från meddelandecentralen). Dock stöds den äldre operationstypen 01, vilket medför att man kan ta emot meddelanden från meddelandecentraler som skickar med denna operationstyp. Dock skulle UCP klassen kunna modifieras på några få ställen för att även kunna hantera de operationstyper som ej stöds i nuläget.

5.2.1 Att skapa ett UCP-objekt

När ett UCP-objekt skapas är det en del saker man måste tänka på. Eftersom flera UCP-meddelanden kan skickas i samma uppkoppling till meddelandecentralen så måste varje UCP-objekt ha ett unikt transaktionsnummer. Detta sätts med hjälp av konstruktorn.

```
UCP myUCPpackage = new UCP(transactionNumber);
```

Detta är viktigt eftersom meddelandecentralen anger vilket transaktionsnummer som den ger svar på och om dessa inte överensstämmer så kan det hända att svaret från meddelandecentralen inte godkänns.

```
myUCPpackage.delayDelivery(true,  "DDMMYYhhmm");
```

Om man vill senarelägga utsändningen av meddelandet så görs det på ovanstående sätt. UCP-klassen kollar inte formatet på tiden utan om den är fel så kommer man få ett felmeddelande då paketet skickas iväg.

```
myUCPpackage.notification(true,  "IP", "npid", "trigger");
```

Om möjligheten finns till att ta emot notifikationer från meddelandecentralen (vanligen genom TCP/IP) så görs det på ovanstående sätt. Observera att IP, npid och trigger inte finns som färdiga statiska variabler som kan hämtas eftersom detta inte är fullständigt implementerat i klassen. Om strängarna formateras på rätt sätt så kan denna funktion användas [3].

Både notifiering och senarelagd utsändning stöds inte av OT 01, men eftersom dessa kan sättas innan kommer UCP-klassen inte generera något fel.

```
myUCPpackage.setMessage(OT,"message","fromPhone","toPhone");
myUCPpackage.setMessage("message","fromPhone","toPhone");
```

För att sätta meddelandet används setMessage() som ovan. Det finns två implementationer av denna funktion. I den ena anges operationstyp medan i den andra antar att det är operationstyp 51.

5.2.2 Hur svaret från meddelandecentralen sätts

Då ett meddelande sänds iväg genom en kommunikationsklass så skickas även UCP-klassen med. Kommunikationsklassen vet att den skall anropa funktionen setResponse(String) för att sätta svaret, så det hanteras helt av klassbiblioteket.

setResponse(String) kontrollerar att svaret från meddelandecentralen är helt korrekt och om det är ett ACK eller NACK. Dock är det upp till applikationen att sedan fråga vad meddelandecentralen svarat.

5.3 IA5-kodaren

IA5-kodaren har förmågan att både koda och avkoda IA5-tecken. Nedan beskrivs förloppet av en kodning av ett meddelande som sedan avkodas.

```
IA5Coder coder = new IA5Coder();
String msg = new String("Koda detta meddelande");
String codedmsg;
String uncodedmsg;

// Koda meddelandet och lagra det i codedmsg variabeln
codedmsg = coder.encode(msg);

// Avkoda meddelandet och lagra det i uncodedmsg variabeln
uncodedmsg = coder.decode(codedmsg);
```

Observera att msg hela tiden är i sin ursprungsfom. Encode()-funktionen tar emot en sträng och returnerar ut en ny sträng i kodad form.

IA5-kodaren är baserad på ASCII-tabellen och eftersom den inte specificerar tecknen som ligger i kodområdet 128-255 så hanteras därför ovanliga tecken som å,ä och ö olika beroende på teckentabellen i datorn. Om det kommer ett tecken som inte stöds av IA5 så ersätter kodaren det med ett blanksteg utan att generera något fel.

5.4 Modemklasserna

Modemklasserna består utav tre klasser. Dels så implementeras Communicaton-interfacet (*se kap 4.2)* och även två modemspecifika klasser, Modem.java och Modem.c. Modem.java innehåller fyra stycken nativefunktioner som används för att sköta kommunikationen med C-

klassen som i sin tur tar hand om den seriella kommunikationen. Det som ställde till mest problem under utförandet var att koppla ihop C- och Javakoden med Java Native Interface (JNI) (*se kap. 2.6.1*) och att kontrollera att kommunikationen med modemet fungerar.

För att få stöd för modem i Linux var det seriella stödet i C tvunget att implementeras, eftersom Java inte har stöd för någon seriell kommunikation.

Modem.java består av fyra nativefunktioner. En för att initiera modemet, en för att ringa upp och koppla upp en uppkoppling, en för att skicka ett SMS-meddelande och slutligen en för att koppla ned från meddelandecentralen.

Koden med nativefunktionerna kompilerades och med hjälp av javah genererades en C-header som definerar hur C-filen kommer att se ut.

```
javah –jni Modem
```

5.4.1 C-filen

All kommunikation med den seriella porten sker över en linuxpipe. Hur implementationen för initiering, uppringning, läsning och skrivning i följande underkapitel löstes kommer att presenteras.

5.4.1.1 Initiering

Initieringen av modemet krävde en hel del arbete. För att göra inställningarna på modemet behövdes en termios struct. Eftersom C-interfacet till modemet skulle vara så litet och enkelt som möjligt så kan endast de flaggor som anses behövas sättas.

Modemhastigheten sattes endast till 9600 för både ingående och utgående trafik, detta för att öka chanserna att modemet klarar av det. Eftersom implementationen gjordes för anpassning för Linux så användes hardware flowcontrol. Vi såg även till att fildeskriptorn inte var blockerande.

C-koden testar också om modemet ekar tillbaka kommandon, detta för att veta hur kommandon skall skickas och läsas senare under exekveringen.

5.4.1.2 Uppringning

För att kunna ringa upp så måste modemet svara på ett ATZ-kommando som talar om att det är redo för kommunikation. Numret slås med ett ATDT-kommando och programmet avvaktar på svar från modemet. Vissa modem svarar med ett ”RING” eller ”RINGING” medan det väntar på att andra sidan skall svara. C-koden är inställd på att acceptera max 6 stycken ringsignaler innan det anser att det inte är något svar. Det är dock inte alla modem som ger ett

”RING” utan endast svarar då det kommer ett CONNECT eller BUSY. På grund av detta så har en timer lagts på hur lång tid det maximalt får ta innan det kommer ett svar från modemet. Denna ligger på 60 sekunder.

5.4.1.3 Läsning från modemet

C-filen behöver två olika sätt att läsa från modemet. Ett sätt då den vet vad som kommer att returneras och ett sätt för att läsa till ett visst tecken. Tecknen läses ett efter ett tills dess att ett förväntat antal tecken läses in eller ett förväntat tecken kommer.

Eftersom mycket kan gå fel och man kan tro att ett visst svar kommer, så måste tiden som man får vänta på tecken begränsas. Detta löstes genom att sätta en actionlistener som väntar på en SIGALRM-signal.

5.4.1.4 Skrivning till modemet

Skrivningen till modemet sker på liknande sätt som läsningen. Tecknen skrivs ett efter ett tills dess att alla tecken skrivits. Även här finns det en timer som signalerar om det tagit för lång tid, det vill säga att modemet inte tar emot tecknen.

Om modemet har ekot på, så ser skrivfunktionen också till att ekot läses in från bufferten.

Vid skrivning av själva SMS-meddelandet så anropas en nativefunktion som returnerar ett svar från meddelandecentralen. Detta medför dock att meddelandecentralen måste svara för att det ska gå bra. Om meddelandecentralen inte ger ett svar kan man inte skilja på om sändningen gick fel eller om det var så att inget svar erhölls. Hänsyn till meddelandecentraler och protokoll som inte ger svar, har inte tagits då vi antar att nästan alla protokoll använder sig av transaktioner.

5.4.2 Skapa ett Shared Library

För att JNI ska kunna fungera så måste man ladda in C-filen som ett Shared Library eller som ett Dynamic Link Library (DLL) i Windows. Dessa bibliotek möjliggör att de kan delas av flera program samtidigt. Dessa program delar på exakt en fysisk kopia av biblioteket och behöver inte göra en egen kopia av koden. Detta görs på olika sätt i de olika operativsystemen och eftersom koden är anpassad till Linux så gås förloppet där igenom.

För att få ett Shared Library i Linux så måste följande kompileringssteg göras.

```
gcc –c –fPIC –I/usr/java/jdk1.3.0_02/include
    –I/usr/java/jdk1.3.0_02/include/linux Modem.c
gcc –shared –Wl,-soname,libmodem.so –o libmodem.so Modem.o
ld  –G –f/usr/lib/libmodem.so Modem.o –o libmodem.so
```

Först kompileras koden med jni.h som ligger i javas hemkatalog/*include* samt /include/*operativsystem*. Parametern –c i gcc kompilerar men länkar inte nativekoden utan genererar en Modem.o -fil istället och –fPIC genererar positionsoberoende kod, lämplig för att användas vid dynamisk länkning.

För att kunna länka filen med andra exekverbara filer används parametern –shared.

Ld kommandot länkar ihop filerna till ett shared library.

För att exekvera applikationen gör man som vanligt i java.

```
java Modem
```

5.5 Testapplikationen

Testapplikationens stöd för att använda sig utav UCPs bibliotekfunktioner begränsas av modemuppkopplingen. Detta innebär att funktioner som notification() inte har utvärderats och testats ordentligt.

```
SYNTAX
java test [FLAGS] ”phoneToSMSC” “message” “toPhone1” […]

MÖJLIGA FLAGGOR
[-s “SendFromNumber”]
[-n “IP” “NPid” “Trigger”]
[-d “DeliveryTime”]
[-o “Operationtype”]
[-c “/dev/ttySX”]
```

Testapplikationen kan ta emot ett meddelande som parameter och skicka det till oändligt antal mottagare. Tänk dock på att de olika operatörerna brukar ha olika nummer beroende på hur många meddelanden som skall skickas.

5.5.1 De olika flaggorna

Testapplikationen har stöd för en del inställningar, till exempel kan man sätta avsändarnumret, notification, operationstyp och senarelagd utsändning. Nedan kommer en förklaring på hur man använder de olika flaggorna.

Om numret som skall stå i mottagarens display önskas bestämmas kan detta göras med flaggan **–s**. Flaggan följs av en sträng som specificerar numret som skall visas. Bokstäver kan tyvärr inte visas utan kommer generera ett syntax error från meddelandecentralen.

Notificationfunktionen är inte helt utvecklad vilket medför att parametrarna till flaggan blir lite underliga. IP-parametern skall innehålla ipadressen utan punkter och utfylld med nollor följt av portnumret som notifieringen skall skickas till. IPadress 192.87.25.9 och port 5000 blir således 1920870250095000, 16 numeriska siffror som är den maximala längden på parametern. NPid parametern kan antingen vara 0339 (Datapak) eller 0539 (Internet). Sedan kommer trigger-parametern. Denna parameter kan vara ett värde mellan 0 och 7 och specificerar vad det är som kommer generera en notifiering.

Parameter	**Mottaget**	**Sparat**	**Raderat**
0	Nej	Nej	Nej
1	Ja	Nej	Nej
2	Nej	Nej	Ja
3	Ja	Nej	Ja
4	Nej	Ja	Nej
5	Ja	Ja	Nej
6	Nej	Ja	Ja
7	Ja	Ja	Ja

Tabell 5.5.1.1 Möjliga värden på triggerparametern.

För att senarelägga utskickningen av ett SMS så används **–d** flaggan. Flaggan följs av tiden då meddelandet skall skickas. Tiden har ett bestämt format som den skall stå i vilket är DDMMYYhhmm. Tiden kan som längst ligga 72 timmar efter det att meddelandet skickas till meddelandecentralen, vilket är den tid som meddelandecentralen kan bevara meddelande i systemet.

Eftersom ett SMS inte kan tas emot och andra operationstyper än 51 och 01 inte stöds så är denna parameter ganska onödig. Den finns med för att ha ett stöd för framtida utvecklingar av klassbiblioteket och för att visa i kod hur det kan gå till att skapa UCP-paket med olika operationstyper. Om operationstyp 01 används så kan inte flaggorna –n och –d användas.

Den sista flaggan, **-c** används för att specificera vilken seriellport som modemet finns på. Utan flaggan antas modemet finnas på /dev/ttyS0 (COM1) porten. När porten specificeras

skall man i Linux se till att porten fungerar och att sökvägen till den anges, exempelvis ”/dev/ttys2” (COM3).

6 Utökning och sammanfattning av implementationen

Detta kapitel går igenom vilka brister som finns i addSMS klassbibliotek och vad som kan förbättras. Meningen är att ge förslag på vad som skulle kunna göras om biblioteket skulle vidareutvecklas. Efter det följer en sammanfattning av implementationen.

6.1 Hur man skulle kunna utöka biblioteket

I biblioteket finns det en början på implementation för SSL. Eftersom det inte fanns möjlighet att kunna testa och få den i helt fungerande skick utan en alltför stor kostnad så är det ovisst om den fungerar. SSL-klassen använder sig utav Java Secure Socket Extension (JSSE) 1.0.2 vilket är ett externt javabibliotek som hanterar just SSL-sessioner. Detta bibliotek är fullt kompatibelt med SSL v3 och TLS 1.0 och tillhandahas från Sun. SSL-klassen skulle kunna utvecklas om resurser finns och med hjälp av den skulle även UCP-klassen kunna utvecklas och dess stöd för notifiering och mottagande av meddelande från meddelandecentralen.

Stöd för andra protokoll skulle också kunna utvecklas. Flera av operatörerna har stöd för olika protokoll i sina meddelandecentraler och de flesta protokollspecifikationer går att få tag på via Internet, operatören eller protokollutvecklaren.

UCP-klassen och modemklasserna går att utveckla. UCP klarar av att skicka meddelanden till både minicall och fax, detta skulle kunna gå att utveckla på ett enkelt sätt. Modemklasserna är för närvarande anpassade för just UCP och Telias meddelandecentral. Eventuellt skulle man vilja ha lite mer frihet till att sätta fler inställningar själv, men detta hann tyvärr inte implementeras. Eventuellt skulle de nuvarande inställningarna vara till problem med andra protokoll än UCP, och då är en inställningsmöjlighet av nativedelen av Modemklasserna en nödvändighet.

6.1.1 Stöd för Windows

Klassbiblioteket är som nämnts tidigare anpassad för Linux enbart, detta för att det var det som efterfrågades. En möjlig vidareutveckling skulle vara att göra stöd för modemdelen även i Windows.

Sun erbjuder en seriell lösning till Java som går under biblioteksnamnet Java Communication API eller java.comm. Detta är ett färdigt bibliotek som lätt kan implementeras med nuvarande modemhantering. Dock är även Suns seriellkommunikationslösning plattformsberoende, och endast till Windows, så stödet för flera plattformar måste ligga inom applikationen, det vill säga att filer som används måste vara beroende av vilket operativsystem som körs vid exekveringen.

6.2 Sammanfattning av implementationen

Under implementationen av klassbiblioteket har det varit en del problem. Det första problemet var att Telias support tog alldeles för lång tid på sig att svara, så frågor om hur man kommunicerade med meddelandecentralen fick testas fram. Det var också en hel del problem med C-koden. Dels så fanns det ingen kunskap om hur den skulle kompileras för att kunna integrera den med Javakoden och dels så var det svårt att få modemet att svara i Linux.

7 Kort sammanfattning av projektet

Projektet har inte inneburit några större problem. Givetvis har man kört fast i kodningen men det snarare på grund av okunnighet än av problem. En lite för stor optimism när det gäller tidsplanen har tagits men det medförde bara att stöd för ytterligare tillägg inte implementerades.

IA5-strängshanteraren skapade lite problem då ASCII-tabellerna inte såg ut så som förväntat. Men även detta var inte något större problem som inte kunde lösas relativt snabbt.

Sammarbetet med handledarna både på skolan och på företaget har varit god och snabb hjälp med de få problem som uppstått har erbjudits. Arbetet har lösts på halvfart under 20 veckor, men på grund av olika schema har det varit lite problem med att samordna arbetstiden.

Begrepp och förkortningar

A

ARPA : *Advanced Research Projects Agency*. Amerikansk forskningsorganisation som bl a skapade ARPANET.

ARPANET : *Advanced Research Projects Agency Network*. Nätverk skapat av ARPA. Detta är föregångaren till Internet.

ASCII : *American Standard Code for Information Interchange*. Ett format som används för att representera tecken och styrkoder med binära värden. Detta format används både inom kommunikation och internt i en dator.

B

BBN : *Bolt, Bernek and Newman*. Ett konslutföretag från Cambridge. Ville i slutet av 60-talet ha ett nätverk som skulle använda sig av minidatorer, som skulle fungera som växlingsnoder till värddatorerna som skulle anslutas i nätverket.

C

CCITT : De internationella teleförvaltningarnas standardiseringsorgan.

CIMD : *Computer Interface to Message Distribution*. Protokoll som används för att hantera SMS. Används bland annat av Europolitan.

D

DARPA : US *Defense Advanced Research Projects Agency*.

DoD : *Department of Defense*. Är en instutition som var med och utvecklade Internet.

E

Email Gateway : En meddelandegateway.

ESME : *External Short Message Entity*. SMS-applikationssystem.

ETSI : *European Telecommuncations Standard Institute*. Hanterar europeisk standard för telekommunikation.

F

FTP : *File Transfer Protocol*. Ett protokoll ovan TCP/IP som används för att överföra filer. Det är ett effektivt protokoll som bara utför den mest grundläggande typen av filhantering. FTP tillhör applikationslagret i OSI-modellen.

G

GC : *Garbage Collector*. Ett delprogram i programmeringsspråket Java, som tar hand om

minnesfrigöring.

GSM : *Global System Mobile telecom*. Europeisk standard för mobiltelefoni.

GSM USSD Server : *Unstructured Supplementary Services Data*. En meddelandecentral.

H

HTTP : *HyperText Transfer Protocol*. Det protokoll som används för att transportera allting på WWW. Detta är ett ”lager 5-7”-protokoll som arbetar ”uppe på” TCP/IP.

HTML : Hyper Text Markup Language.

I

IA5 : IA5 är en teckenuppsättning som mestadels används inom datakommunikation. Varje IA5-tecken består av 7-bitar (till skillnad från ASCIIs 8-bitar) vilket medför att det kan beskriva 128 unika tecken.

Internet : Ett stort nätverk som täcker hela världen. Internet består egentligen av flera små nätverk som tillsammans bildar ett stort. Det har sitt ursprung ur ARPANET som skapades i slutet av 60-talet.

Invocation API : Tillåter en nativeklass att ladda in Java Virtual Machine.

IETF : *Internet Engineering Task Force*.

IP : *Internet Protocol*. Grunden i TCP/IP, ett protokoll som arbetar i nätverkslagret.

IP-adress : Den typ av adresser som används inom TCP/IP och därmed Internet. Adressen är en 32 bitars adress som består av fyra komponenter. Ofta ser man adressen i textform – ”aa.bb.cc.dd”, där ”aa”, ”bb”, ”cc” och ”dd” är siffror mellan 1 och 255. Adressen består av två delar, där den ena delen är en nätverksdel som adresserar rätt nätverk och den andra delen är en ”host”-del som adresserar rätt dator inom nätverket.

IRQ : *Interrupt ReQuest*. Avbrottsförfrågan. Avbrott som levereras till processorn från olika enheter (t ex tangentbordet) som vill få någonting utfört.

J

Java : Ett objektorienterat programmeringsspråk. Likt C++.

JDK : *Java Developers Kit*. En utvecklingsmiljö som stöder programmeringsspråket Java.

JNI : *Java Native Interface*. Ett standardiserat programmeringsinterface för att möjliggöra för Javakod att operera med applikationer och bibliotek skrivna i andra språk, så som C, C++ och assembler.

J2SE : *Java 2 Standard Edition*. Utvecklingsmiljö för mjukvaruapplikationer som kan köras på olika datorer och servrar.

JVM : *Java Virtual Machine*. Ett mjukvaruskal.

N

NCP : *Network Control Protocol*. Protokoll som användes i ARPANET, men som ersattes av TCP.

O

OSI : *Open System Interconnection*. Modell för hur öppen datakommunikation skall gå till. Används för definition av gränssnitt och protokoll.

P

PCMCIA: *Personal Computer Memory Card International Association*. (PC-kort). En speciell typ av expansionskort för i huvudsak bärbara datorer. Ett sådant kort kan innehålla olika typer av funktionalitet varav modem och nätverkskort är två av dem.

PDU : *Protocol Description Unit*. En form att skicka SMS på. Ett annat är textform.

R

RS-232C : Standard för serieport.

S

SAR : *Segmentation and Reassembly*. En funktion som delar upp data från högre lager före sändning och som på andra sidan sedan sätter ihop dessa till paket för högre lager igen.

SME : Mobila stationer.

SMPP : *Short Message Peer to Peer Protocol*. Protokoll som används för att hantera SMS. Används bland annat av Tele2 (Comviq).

SMS : *Short Message Services*. Tjänst i GSM med vilken man kan skicka korta meddelanden.

SMSC : *Short Message Service Center*. En meddelandecentral.

SSL : *Secure Socket Layer*. Krypteringsfunktioner för trafik över Internet.

T

TCP : *Transmission Control Protocol*. Används oftast tillsammans med IP. Protokoll i transportlagret.

U

UART : *Universal Asynchronous Reciever Transmitter*. Ett chip.

UCP : *Universal Computer Protocol*. Protokoll för kommunikation mellan meddelandecentralen och den externa applikationen.

UCS2 : Unicode.

W

WAP Proxy Server: En meddelandegateway.

www : *World Wide Web*.

X

X.25 : Ett paketförmedlande publikt nät för datakommunikation. Kommunikation mellan dator och paketförmedlande nät. ITU-T-protokollstandard för anslutning till paketförmedlande nät. Anger också teknik för paketförmedlande nät.

Referenser

[1] Eriksson Hans-Erik. *Programutveckling med Java*. Studentlitteratur, första upplagan, 1997.

[2] Tanenbaum S. Andrew. *Computer Networks*. Prentice-Hall, tredje upplagan, 1996.

[3] Telia: *Handbok för applikationsutvecklare*. Version 4.4, 2000.

[4] RFC793: Transmission Control Protocol.

[5] RFC2246: Transport Layer Security.