Datavetenskap

Petter Andersson
Jonas Karlsson

SoftTrail SMS reader - Utveckling av ett meddelandehanteringssystem

---

Examensarbete

2003:09

SoftTrail SMS reader - Utveckling av ett meddelandehanteringssystem

Petter Andersson
Jonas Karlsson

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.

Jonas Karlsson

Petter Andersson

Godkänd, 3 Juni 2003

Handledare: Stefan Alfredsson

Examinator: Stefan Alfredsson

Sammanfattning

SMS-tekniken är utvecklad för att kunna skicka korta meddelanden mellan användare av mobiltelefoner. Idag används tekniken flitigt och har blivit mycket populär. SoftTrail AB har identifierat ett behov av att kunna föra över dessa meddelanden till en dator och har också utvecklat en prototyp för detta ändamål.

Syftet med detta examensarbete har varit att vidareutveckla den befintliga prototypen till en fungerade programvara. Utvecklingen av programmet har skett med portabilitet och modularitet i åtanke och därför har programspråket Java valts för implementationen. Detta gör att programmets olika delar är fristående och lätt kan användas i andra tillämpningar.

Resultatet är ett program som fungerar med olika tillverkares mobiltelefonmodeller. Överföringen av meddelanden från mobiltelefon till dator sker via telefonens inbyggda modem. Detta görs t.ex. genom datorns serieport-, IR- eller USB-anslutning. Kommunikationen med modemet görs med hjälp av modemkommandon. Meddelanden som hämtas från telefonen är inte direkt läsbara utan måste avkodas innan dom visas som meddelanden i programmet. De avkodade meddelandena presenteras i ett grafiskt gränssnitt.

I rapporten ingår en beskrivning av hur programmet är konstruerat och hur de viktiga delarna är implementerade. Vidare ges en programbeskrivning som redogör för programmets olika funktioner och hur dessa används. Rapporten ger också en kort introduktion till *Multimedia Messaging Service* (MMS) som är en ny teknik för att skicka mer avancerade meddelanden.

Abstract

The SMS technology is a service aimed at sending short messages between mobile phone users. SMS is today frequently used and has become very popular. SoftTrail AB have identified the need of transfering these messages to a computer and have also developed a prototype for this purpose.

The purpose of this Bachelor's project has been to futher develop the prototype to a complete software. The development of the program has progressed with portability and modularity in mind and therefore the implementation has been done in the programming language Java. This makes the different parts of the program very independent and can easily be used with other applications.

The result is a program that handles mobile phones from different manufacturers. The internal modem of the phone is used to transfer messages to the computer, by using a serial, infrared or USB connection. The phone is controlled by sending modem commands. The messages have to be decoded before they can be read and shown as messages in the program. The decoded messages are presented in a graphical user interface.

This report describes how the program is designed and how the important parts have been implemented. Further there is a description of the program that explains its different functions and how they are used. The report also gives a short introduction to Multimedia Messaging Service (MMS) which is the latest technology in mobile messaging.

Tack

Vi vill börja med att tacka Anders Molander som varit vår handledare på SoftTrail för den tid och arbete som lagts ner för att hjälpa oss genomföra detta examensarbete. Vidare vill vi även tacka Stefan Alfredsson som varit vår handledare på universitetet.

Ett stort tack går också till Magnus Forsberg för att ha ritat bilden och de ikoner som finns i programmet samt Stefan Danielsson för möjligheten att testa programmet med hans mobiltelefon. Även ett tack går till Åsa Andersson för sina goda engelskakunskaper.

Slutligen vill vi också tacka våra flickvänner, Eleanor Lundell och Lisa Silo, för korrekturläsning och moraliskt stöd. Vi vill även ge ett extra tack till Eleanor för att ha lånat ut sin mobiltelefon under större delen av projektet.

Innehåll

Figurer

Tabeller

1 Introduktion

Vårt sökande efter ett examensarbete resulterade i en kontakt med företaget SoftTrail i Karlstad. SoftTrail AB är ett nystartat (november 2001) företag inriktat på realtidsprogrammering inom områdena kommunikation och säkerhet samt viss egen utveckling inom SMS-området.

Företaget erbjöd oss att vidareutveckla deras redan befintliga, men dock begränsade prototyp för SMS-läsning. Prototypen utvecklades för att läsa SMS-meddelanden från en Ericsson T68 med IR-uppkoppling. SoftTrail ville även att programmet skulle implementeras i programspråket Java p.g.a dess plattformsoberoende egenskaper [9].

Tjänsten SMS, *Short Message Service*, har funnits ända sedan början av 90-talet och var från början bara tänkt att användas för att skicka korta textmeddelanden mellan mobiltelefoner. På senare tid har dock tekniken utvecklats och idag finns även möjlighet att skicka bilder, ljudsignaler och även formaterad text. Operatörerna tjänar idag mycket pengar på dessa tjänster och det har blivit ett mycket populärt sätt att förmedla information mellan användare.

Uppdragsgivaren har identifierat ett behov av att spara meddelanden från en mobiltelefon till en dator eftersom en mobiltelefons minne är begränsad och gamla meddelanden måste tas bort med jämna mellanrum. På sikt finns fler möjligheter med programmet, t.ex. att läsa MMS eller konfigurera telefonen.

Resterande delen av rapporten är indelad enligt följande. I kapitel 2 beskrivs de olika delar som anses viktiga för att förstå arbetet. Detta innefattar beskrivning av SMS/EMS, Javas stöd för serieportskommunikation. Vidare i kapitlet tas även MVC och modemkommandon upp. Här finns även en kortfattad introduktion till MMS.

Syfte, krav, problemställning och förutsättningar för arbetet tas upp i kapitel 3. I kapitel 4 beskrivs programmets konstruktionslösning, hur programmet är uppbyggt och hur delarna hänger ihop. I kapitel 5 beskrivs hur de väsentligaste delarna av programmet är implementerade samt vilka tester som genomförts på programmet. I kapitel 6 ges en detaljerad beskrivning av programmets funktioner samt hur programmet installeras. Vidare tas även de fel upp som kan uppstå i programmet. Kapitel 7 behandlar de erfarenheter och problem som framkommit av arbetet.

Förslag till framtida utökningar och förbättringar samt en kort beskrivning av resultatet finns här. I appendix finns definitioner av TPDU-parametrar samt en förkortningslista.

2 Bakgrund

I detta kapitel tas delar upp som anses viktiga för att förstå arbetet. I avsnitt 2.1 beskrivs hur ett SMS är uppbyggt, vad som specifierar ett EMS (delavsnitt 2.1.1) samt ett exempel som visar 8- till 7-bitars avkodning (delavsnitt 2.1.2). I avsnitt 2.2 ges en kort introduktion av begreppet MMS. I avsnitt 2.3 beskrivs kort hur kommunikationen med mobiltelefonens inbyggda modem går till. I avsnitt 2.4 tas Javas stöd för serieportskommunikation upp och förklaras. Olika alternativ till SUN:s implementation av detta tas även upp.

För att förstå hur programmet är konstruerat tas den grundläggande idén bakom *Model View Controller* (MVC) upp i avsnitt 2.5.

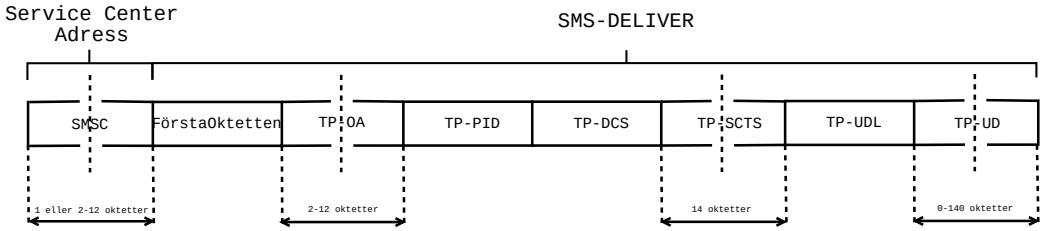
2.1 Short Message Service

Short Message Service (SMS) är en teknik som möjliggör överföring av korta meddelanden från och till en GSM/UMTS telefon. Meddelandena går via en meddelandecentral (eng. Service Central) även kallad *Short Message Service Central* (SMSC). Meddelandecentralen tjänstgör som relä/mellanlagringsstation vilket möjliggör skickandet av ett meddelande även om den mottagande parten inte är aktiv, d.v.s. har sin telefon avslagen. Meddelandecentralen lagrar meddelandet och gör upprepade utsändningsförsök, dessa är dock tidsbegränsade beroende på vilken operatör som står för tjänsten.

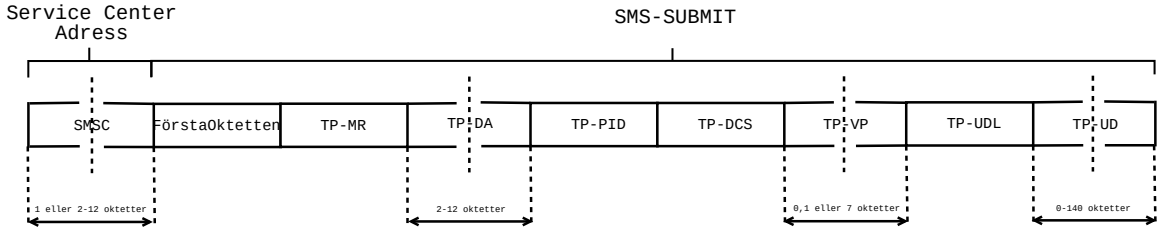
Ett meddelande som skickas via SMS är en mängd parametrar och ett meddelandefält. Detta kan ses som ett meddelande (brev) man skickar iväg instoppat i ett kuvert med viss påskriven information. Kuvertet skickas iväg till en meddelandecentral som undersöker informationen på kuvertet och försöker dela ut det till mottagaren med den postservice som är specificerad i parametrarna på kuvertet. Meddelandet plus de andra parametrarna är tillsammans känt som ett *Protocol Data Unit* (PDU) eller *Transport Protocol Data Unit* (TPDU).

Det finns flera olika typer av PDU vilka innehåller olika parameterkombinationer. De två mest grundläggande PDU:erna som är specificerade i GSM är SMS-DELIVER och SMS-SUBMIT.

Dessa två förser användaren med möjligheten att ta emot respektive skicka ett meddelande. De övriga är till för statusrapportering och överföring av kommandon till en meddelandecentral från telefonen. Parametrarna i de två olika PDU:erna (figurerna 2.1 och 2.2) innehåller information som rör kodning av meddelandefältet, adresser, tidstämpel m.m.



Figur 2.1: Uppställning av SMS-DELIVER



Figur 2.2: Uppställning av SMS-SUBMIT

Första oktetten i en PDU innehåller en mängd flaggor. Dessa flaggor informerar bl.a. om vilken typ av SMS-PDU det rör sig om eller om meddelandefältet börjar med en header. Flaggor indikerar om en statusrapport ska skickas eller om begäran av en sådan ska skickas.

Adressfälten innehåller sändaradressen (TP-OA) eller destinationsadressen (TP-DA) beroende på om det är ett SMS-DELIVER eller SMS-SUBMIT meddelande. Om det är ett SMS-DELIVER tillkommer dessutom en tidstämpel (TP-SCTS) som innehåller tiden och datum då ett meddelande anländer hos meddelandecentralen. Sändaradressen samt tidsstämpel läggs i regel till automatiskt då meddelandet skickas från meddelandecentralen.

Protokollidentifieraren (TP-PID) och Data-Kodnings-Schemat (TP-DCS) är två fält som finns i både ett skickat och mottaget meddelande. Protokollidentifieraren är en flagga som används för

en mängd olika syften. Den avgör hur den mottagande enheten eller meddelandecentralen ska hantera meddelandet. Schemafältet indikerar vilket alfabet som meddelandefältet (TP-UD) är kodat i, den vanligaste kodningen är GSM 7-bitars standardalfabet [4] men meddelandefältet kan även vara kodat i 8 eller 16 bitar. Denna flagga indikerar även vilken klass meddelandet har, om meddelandet är komprimerat eller om en ikon hos den mottagande enheten ska knytas till meddelandet.

Meddelandefältet kan innehålla upp till 160 tecken om fältet är kodat med GSM 7-bitars standardalfabet. Detta fält kan även vara i ett speciellt format. Sändaren kan ha inkluderat en header i början av meddelandefältet utöver själva textmeddelandet. En lista på de olika möjliga typerna finns i tabell A.4.

De olika parametrarna i PDU:erna finns beskrivna mer ingående i Appendix A, dock endast för parametrarna som finns i SMS-DELIVER och SMS-SUBMIT.

2.1.1 Enhanced Messaging Service

Enhanced Messaging Service (EMS) är baserad på två standardmekanismer i GSM SMS.

Den första mekanismen är *User Data Header* (TP-UDH). Denna header gör det möjligt att skicka binärdata i ett normalt SMS utöver själva textmeddelandet. Binärdata finns i meddelandefältet vilket innebär att den stjäl en del av de 140 oktetterna som fältet utgör. Varje objekt är identifierat med ett informationselement (IE) i headern. En översiktsbild av meddelandefältet med en header finns att se i figur A.1.

Den andra mekanismen är uppdelade meddelanden (eng. concatenation). Denna funktion tillåter meddelanden längre än 140 bytes. Det blir då möjligt att sätta ihop upp till 255 stycken 140 bytes stora meddelanden till ett enda.
Ett EMS kan innehålla:

- Formaterad text
- Bilder

- Ljud
- Animationer
- Direkt presentation av meddelande

2.1.2 Avkodning av 8-bitars oktetter till 7-bitars data (septetter)

De flesta SMS och EMS har meddelandefältet kodat i GSM 7-bitars standardalfabet. I och med att TP-UD fältet är lagrat i telefonen som 8-bitars oktetter måste dessa avkodas till 7-bitars septetter för att få fram informationen man vill åt.

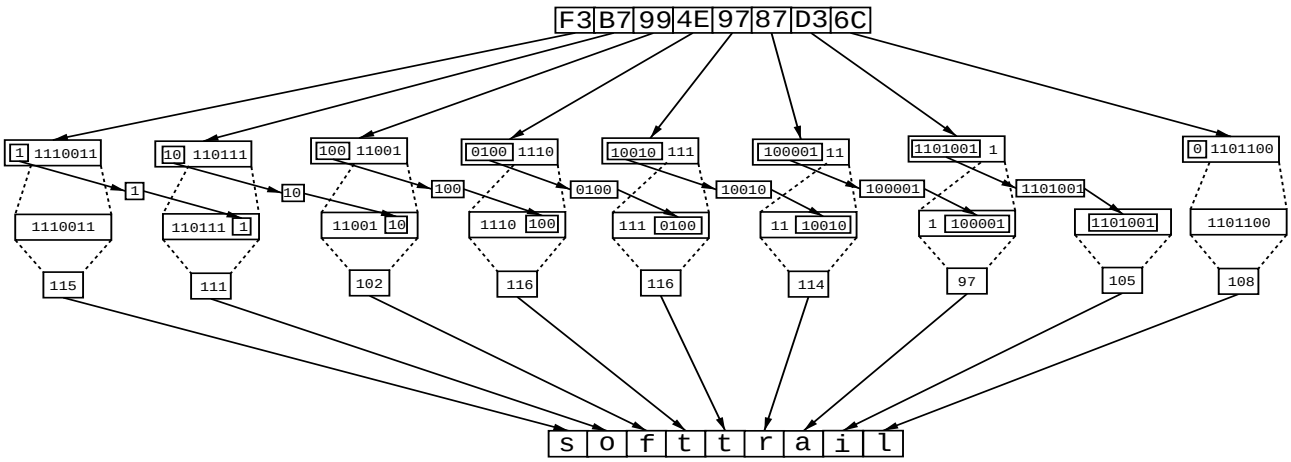
Ett exempel som beskriver avkodning av 8-bitars data till 7-bitars septetter:
Ett meddelandefält innehåller följande oktetter i hexadecimal form:

```
F3 B7 99 4E 97 87 D3 6C
```

Första oktetten blir septett genom att man flyttar bort den mest signifikanta biten. Den borttagna biten placeras före den minst signifikanta biten i den efterföljande oktetten. Den kvarvarande septetten avkodas till lämpligt tecken enligt specifikationen för GSM 7-bitars standardalfabet. Nästa oktett innehåller nu 9 bitar, här flyttas de två mest signifikanta bitarna för att få kvar en septett att avkoda till ett tecken. Dessa bitflyttar med ökning av en bit för varje flytt görs ytterligare 4 gånger. Vid den sjunde flytten tar man sju bitar, dessa läggs inte till den efterföljande oktetten utan kodas till ett tecken direkt. Avkodningen börjar sedan om från början med nästa oktett. Den sista oktetten kan komma att innehålla bitar som inte används, antalet oanvända bitar beror på antalet tecken i meddelandet. I figur 2.3 kan man se avkodningsförloppet steg för steg.

2.2 Multimedia Messaging Service

Multimedia Messaging Service (MMS) är vidareutvecklingen av SMS-tekniken. Syftet med MMS är detsamma som SMS, men skillnaden är att det även går att skicka bilder, ljud och videoklipp



Figur 2.3: Avkodning av oktetter kodade med GSM 7-bitars standardalfabet

samt formaterad text. Dessa egenskaper gör att MMS inte överförs via SMS överföringskanal eftersom denna inte har tillräcklig bandbredd. Överföring sker med *Wireless Application Protocol* (WAP) [15] vilket medför att MMS inte är beroende av t.ex. GSM eller WCDMA för överföringen. MMS kan använda vilken överföringsteknik som helst med WAP kapacitet. WAP Wireless Session Protocol (WSP) används för transporten mellan telefon och meddelandecentralen (MMSC).

En fördel med MMS är att ingen maximal storlek för meddelandet finns specificerad i standarden. Detta för att undvika det problem som SMS har lidit av, där begränsingen med maximalt 140 byte stora meddelanden har varit problematisk. Den maximala storleken på MMS är alltså en implementations- och operatörsfråga. Nokia förutser att meddelanden i det första skedet kommer att vara mellan 30 och 100 kb stora.

MMS stödjer flera olika mediatyper t.ex. JPEG, GIF, AMR samt en hel del andra format. För att skapa interoperatibilitet har några av dom stora företagen som tillverkar mobiltelefoner enats om ett dokument (MMS Conformance Document) som listar den minsta mängd mediatyper som ska stödjas av en MMS-telefon.

Multimedia Messaging Service Center är det nätverkselement som tar emot och skickar meddelandet vidare till rätt mottagaradress. Meddelandet lagras även en tid om mottagaren inte kan

ta emot det direkt. När meddelandet har skickats till rätt mottagare tas det bort från servern. Detta koncept är det samma som för SMSC.

2.3 Modemkommunikation

De flesta mobiltelefoner har ett inbyggt modem för att kunna kommunicera med externa enheter. Vid kommunikationen med modemet används olika modemkommandon som följer Hayes [8] standarden. Exempel på några modemkommandon som används i vårt program för att läsa in SMS-meddelanden från en mobiltelefons minne visas i tabell 2.1.

Modemkommando	**Beskrivning**
AT	Test för att avgöra om telefonen svarar på modemkommandon.
AT+CPMS	Väljer vilket minne som skall användas i telefonen.
AT+CMGL	Hämtar meddelanden från valt minnesområde i telefonen.

Tabell 2.1: Exempel på modemkommandon

Det finns fyra olika typer av modemkommandon. Skrivkommandon (eng. Set) används för inmatning av parametrar till modemet. Exekveringskommandon används för att utföra ett direkt kommando utan parametrar. Läskommandon används när man vill läsa av inställningarna på något kommando och testkommandon används för att ta reda på ett kommandos möjliga parametrar. Ett kommando exekveras med hjälp av följande syntax:

```
AT<kommando>=<parametrar><CR>
```

Ett exempel på hur ett modemkommando kan exekveras:

```
AT+CGMI
ERICSSON
```

```
OK
```

När man skickar ett modemkommando får man som regel ett eko (beroende på hur modemet är inställt) av det man skickar, samt ett svar. Detta svar innehåller någon form av information och avslutas med OK eller ERROR beroende på om kommandot utfördes korrekt eller inte.

2.4 Java Communications 2.0 API (JCA)

I Javas standardbibliotek finns inget stöd för kommunikation med serieportarna i systemet men eftersom det är viktigt att kunna göra detta så används ett tilläggsbibliotek. Det finns flera olika alternativ beroende på vilket operativsystem man kör och under vilken licens man utvecklar programvara.

- Java(TM) Communications 2.0 API. Utvecklad av SUN, fungerar med Windows och Solaris.
- RXTX. Öppen fri mjukvara som fungerar tillsammans med JCA för Solaris där man använder JCA som en plugin till detta. Denna variant fungerar på de flesta UNIX-, MAC OS X-, BeOS-, Win32- och Win CE-system med flera.
- Java(TM) Communications API för OS/2, utvecklad av IBM och fri för registrerade OS/2 användare.

Förutom dessa finns flera andra kommersiella och fria varianter av detta bibliotek för olika typer av plattformar.

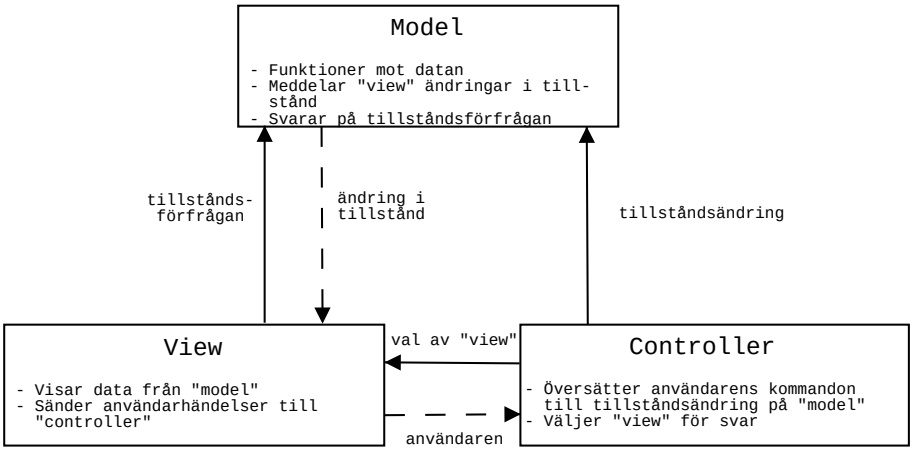
SUN:s JCA är det tilläggsbibliotek som använts för utvecklingen i detta projekt. Då JCA endast har stöd för Microsoft Windows och Solaris så är inte detta en komplett lösning. För att kunna få en plattformsoberoende lösning krävs att det finns ett tilläggsbibliotek för det operativsystem man kör programmet på. Det finns tilläggsbibliotek till i princip alla stora operativsystem där man kan köra en Java VM. T.ex går det att använda RXTX i en Linuxmiljö om man

vill använda programmet där. Tänkbart är också att den version av JCA som utvecklas av SUN kommer att kunna köras på Linux och även andra operativsystem i framtiden.

2.5 Model View Controller

Model View Controller (MVC) är ett designmönster vars avsikt är att separera ett programs data från sättet det visas/tillträds och kopplingen mellan systemhändelser och hur programmet beter sig.

MVC består av tre komponenttyper (se figur 2.4). Modellkomponenten representerar tillämpningens data och metoder som opererar på datan. Modellen saknar ett användargränssnitt. Kontrollkomponenten översätter användarens handlingar till operationer på modellen. Visningskomponenten uppdateras av modellen och visar ändringarna i data.



Figur 2.4: Uppgifter och relationer hos de tre MVC komponenterna

3 Uppdragsbeskrivning

I detta kapitel beskriver vi syftet med arbetet och uppdragsgivarens krav. Problemställningen tas upp och förutsättningarna för arbetet klargörs.

3.1 Syfte

Syftet med arbetet är att vidareutveckla den befintliga prototypen till fungerande programvara med ett grafiskt gränssnitt. Arbetet ska dessutom utreda befintliga standarder för datakommunikation med mobiltelefoner. Uppdragsgivarens syfte med produkten är att ge företaget reklam för sin inriktning på SMS. Företaget kan även tänka sig att använda delar av programmet i framtida kommersiella produkter. Projektet har också till syfte att påbörja studier på MMS-standarden för att möjliggöra fortsatt utveckling av programmet inom detta område. Programmet ska sträva efter att vara kompatibelt med de flesta stora tillverkarna av mobiltelefoner som t.ex. Ericsson, Nokia och Siemens.

3.2 Problemställning

Enligt uppdragsgivaren finns ett identifierat behov av att kunna spara denna information som är lagrad i mobiltelefoner till en dator genom en serieportsanslutning. Liknande befintliga program visar (enligt SoftTrails egen undersökning) sig oftast sakna funktion för att hämta meddelande samt att de endast stödjer en telefontillverkare.

3.3 Krav

Arbetsgivaren har följande krav på programmet:

- Programmet ska vara skrivet i programspråket Java.
- Programmet ska stödja minst en mobiltelefon från Ericsson, en från Nokia och en från Siemens.

- För att det ska vara lätt att vidareutveckla och ändra i programmet ska det vara modulärt.
- Både skickade och mottagna SMS/EMS ska kunna behandlas.
- Fria händer ges för att formge ett gränssnitt som är lätt att använda.
- Öppen källkod enligt GNU General Public License [7].

Programmet ska i första hand fungera i alla versioner av Microsoft Windows som stödjer Java version 1.4.1 eller högre, men uppdragsgivaren ser gärna att programmet är plattformsoberoende.

3.4 Förutsättningar

Detta avsnitt klargör förutsättningarna som gavs för examensarbetet. Delavsnitt 3.4.1 beskriver den hårdvara som funnits tillgänglig. I delavsnitt 3.4.2 listas den mjukvara som använts i arbetet.

3.4.1 Hårdvara

Uppdragsgivaren har försett oss med den nödvändiga utrustningen för att koppla upp en mobiltelefon till en dator både genom IR-, Serie- och USB-anslutning. De telefoner som har varit tillgängliga under projektet är Eriksson T65 och T68, Nokia 6210 och Siemens ME45.

3.4.2 Mjukvara

- Utveckligen av programmet har skett i JCreator 2.5 [13] som är en utvecklingsmiljö för Java. Anledningen till användningen av detta program var snabbheten och användarvänligheten som erbjöds.
- För att underlätta utvecklingen av programmet samt versionshantering skaffades ett konto på Sourceforge [11] som tillhandahållit *Concurrent Versions System* (CVS) och plats för att lägga upp information om projektet för allmänheten. CVS är ett system som används av många mjukvaruutvecklare för att lagra källkoden i en central position. Genom

att kontrollera hur ändringar är gjorda på den lagrade källkoden tar CVS hand om att sammanfoga ändringar gjorda av olika utvecklare. Den meddelar om det uppstår konflikter med ändringar gjorda av andra utvecklare (dvs. två personer har råkat ändra samma rader vid samma tid). I och med att CVS har en historisk databas så kan en programmerare lätt gå tillbaka till en tidigare version av källkoden.

- Hyperterminal användes för att testa modemkommandon mot telefonerna innan implementering i programmet. Programmet är inkluderat i Microsoft Windows.

- SoftTrails prototyp för läsning av SMS är ett program skrivet i programspråket C++. Programmet har en begränsad funktionalitet samt saknar ett grafiskt gränssnitt. Avkodningen innefattar endast meddelandefältet, ingen hänsyn tas till övriga parametrar. Programmet klarar inte att avkoda EMS.

- För rapportskrivning användes Kile 1.4 [16] som är ett integrerat utvecklingverktyg för Latex i Linux.

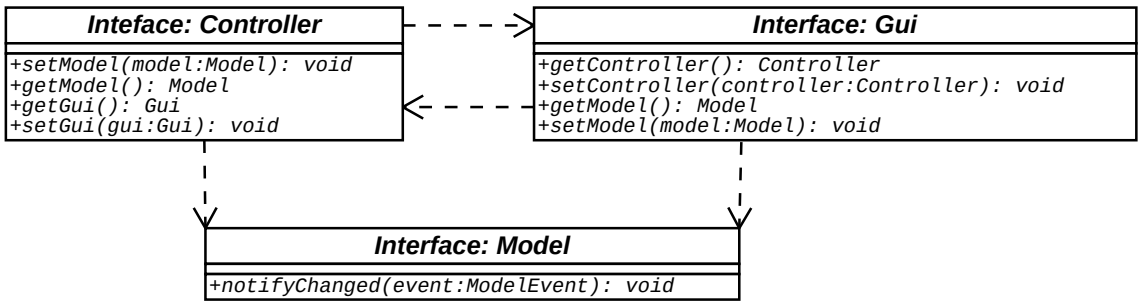
- För installationsscript har Spoon Installer [12] använts som är ett projekt med öppen källkod.

4 Konstruktionslösning

I detta kapitel så beskrivs konstruktionslösningen av programmet, samt hur klasser och programmets delar hör ihop.

4.1 Översiktlig konstruktionslösning

Ett av kraven för tillämpningen var att det skulle bli så modulärt som möjligt. Programmet delades upp i tre gränssnitt (figur 4.1). Grundtanken till denna struktur är inspirerad av *Model View*

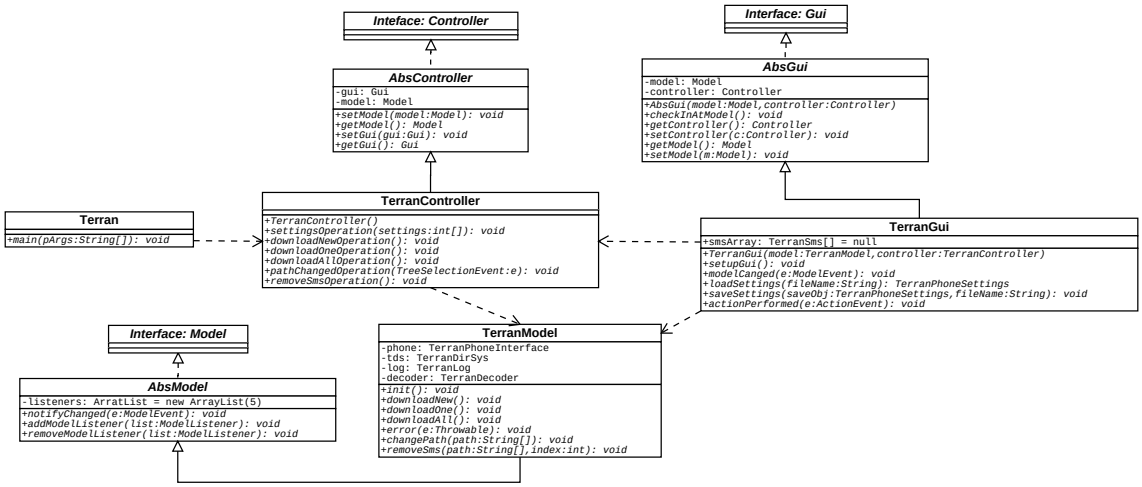


Figur 4.1: Programmets gränssnitt

Controller. Detta möjliggör att programmets olika delar kan utvecklas separat, dessutom underlättar det för att i framtiden kunna göra ändringar.

MVC-funktionaliteten i programmet implementeras i tre abstrakta klasser som implementerar de tre ovan nämnda gränssnitten. Dessa klasser tillhandahåller MVC-funktionalitet för programmets huvudklasser som ärver från dessa. En översiktlig bild av hur programmet är uppbyggt kan ses i figur 4.2.

Modellen får sina operationer från kontrollklassen som fungerar som en adapter mellan guiklassen och modellen. När modellen ändras genereras ett *ModelEvent*. För att guiklassen ska kunna lyssna på dessa händelser finns gränssnittet *ModelListener* som implementeras i denna. Detta gränssnitt definierar endast en funktion *modelChanged* som anropas då modellen har ändrat tillstånd. En översiktsbild av denna meddelandemekanism visas i figur 4.3.



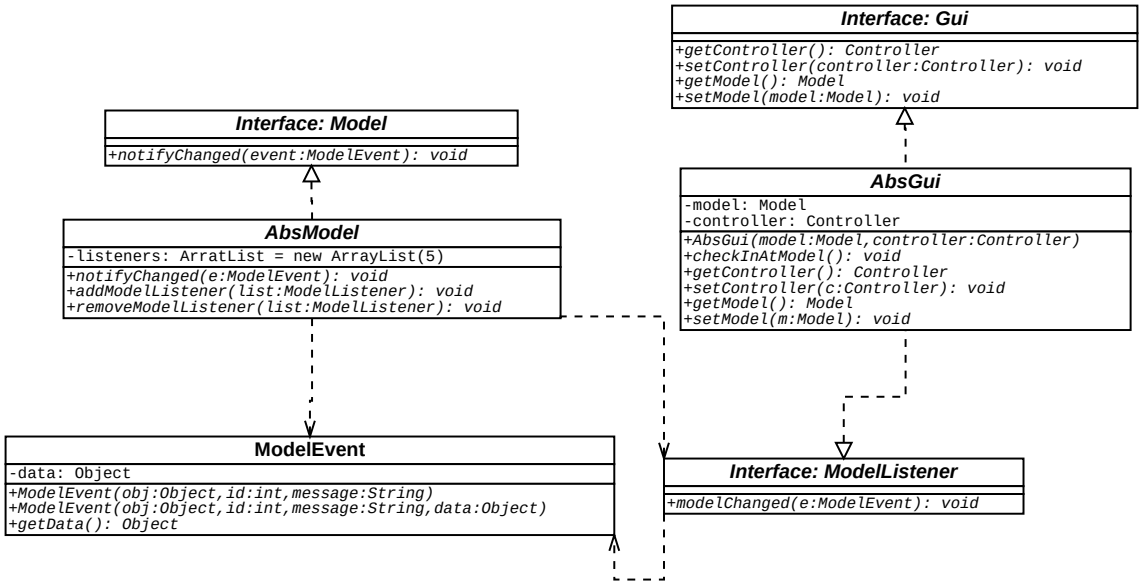
Figur 4.2: Översiktlig programstruktur

4.2 Modellen

Modellen är programmets hjärna. Här finner man kommunikationen med telefonen, filhantering och SMS/EMS avkodning.

4.2.1 Drivrutiner

Tanken är att det ska gå lätt att lägga till drivrutiner för olika telefonmodeller i systemet. För att åstadkomma detta implementerar alla drivrutinsklasser ett interface som specificerar vilka funktioner som måste finnas i en drivrutin för att denna ska fungera korrekt. Eftersom vissa av dessa funktioner samt några andra funktioner inte kommer att vara telefonspecifika finns en generell abstrakt klass som implementerar de mest elementära funktionerna som att skicka och ta emot modemkommandon till telefonen. Drivrutinerna för de specifika telefonerna ärver sedan dessa egenskaper av den abstrakta klassen. Hur dessa klasser hör ihop illustreras i figur 4.4. Med denna lösning kan drivrutiner för nya telefoner på ett enkelt sätt implementeras.



Figur 4.3: Meddelandemekanism

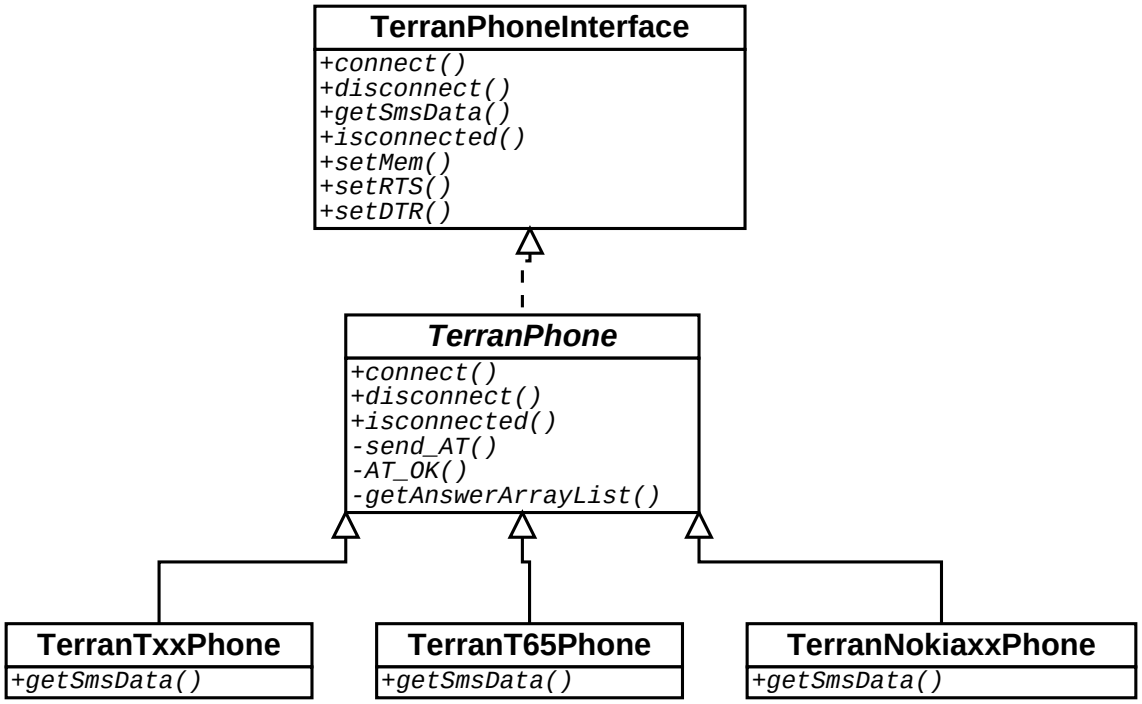
4.2.2 Avkodare

Avkodarklassen klarar att avkoda de två bas PDU:erna, SMS-DELIVER och SMS-SUBMIT inklusive SMSC-fältet som är placerat före de två fälten (se figur 2.1 och 2.2). Även en separat EMS-avkodare är inbyggd här. SMS-avkodaren kodar av och lagrar samtliga fält i de två olika PDU:erna. Ett undantag görs om datafältet innehåller en header, då lagras datafältet precis som det är för att senare avkodas i EMS avkodaren.

Det kan verka lite onödigt att behandla alla parametrar eftersom mycket av informationen som de innehåller saknar värde för programmet, sådant som SMSC-adress, begäran av statusrapport, validitetperiod m.m. Anledningen att dessa behandlas är för att underlätta framtida tillämpningar av avkodaren.

4.2.3 Filhantering

Filhanteringen i programmet består av att kunna öppna och spara SMS/EMS på disken. Även information som berör sökvägen i det grafiska gränssnittet lagras här. Filhanteringen under mod-



Figur 4.4: Klassdiagram för telefondrivrutiner

ellen sköts av klassen *TerranDirSys*.

4.3 Grafiskt gränssnitt

Gui-klassen är det grafiska gränssnittet mot användaren. Här ligger alla komponenter för att visa data samt inmatning av data. Gränssnittet har givits ett utseende likt en e-postklient (t.ex. Microsoft Outlook Express eller KMail) för att göra det lättigenkännligt för användaren. En mer detaljerad beskrivning av användargränssnittets utseende kommer i avsnitt 6.2.

5 Implementation

I detta kapitel förklaras hur de mest väsentliga delarna av programmet har implementerats. Slutligen förklaras vilka tester på programmet som utförts och hur dessa har gått till.

5.1 Drivrutiner

Efter att ha experimenterat med programmet *Hyper Terminal* och prototypen samt tagit reda på vilka modemkommandon som behövdes, påbörjades ett terminalliknande gränssnitt skrivas för drivrutinen som utvecklades. Med detta terminalprogram kunde drivrutinen testas utan att behöva ha något grafiskt gränssnitt kopplat till sig. Detta medförde att drivrutinerna kunde testas fristående.

För programmet har endast en drivrutin skrivits. Å andra sidan fungerar den bra med de flesta telefoner eftersom dessa använder gemensam standard [3] för de flesta elementära funktionerna i telefonen. I inledningen av projektet användes en Ericsson T65 som kommunicerade med COM-porten genom en USB-anslutning. Denna telefon stödjer samma modemkommandon som de flesta andra telefoner för att hämta, spara och ta bort SMS-meddelanden från telefonen. Detta gör att många telefoner på marknaden fungerar med denna drivrutin (vi har testat Ericsson T65/T68, Nokia 6110 och Siemens ME45) men de flesta telefoner innehåller flera egna funktioner som endast kan åtkommas genom telefonspecifika modemkommandon. Om framtida versioner av programmet ska innehålla andra funktioner än de vi har implementerat så kommer fler drivrutiner att behöva skrivas för just dessa funktioner.

Den delen som skickar och tar emot modemkommandon i programmet jobbar helt oberoende av vilken telefon som den kommunicerar med. Vi har här antagit att en telefon med inbyggt modem alltid svarar ”OK” på det grundläggande modemkommandot ”AT”. När en grundläggande uppkoppling mot mobiltelefonen fås kan användaren skicka andra modemkommandon till telefonen. Nedkoppling sker alltid efter varje anrop.

Att skapa en anslutning mot telefonen för att kunna skriva och läsa mot den är väldigt enkelt

med JCA:

1. Ett CommPortIdentifier-objekt skapas som är kopplat mot en COM-port t.ex. COM3.
2. En serieanslutning skapas (SerialPort), som kopplas till CommPortIdentifier-objektet.
3. In- och utströmmar kopplas från SerialPort-objektet till lämpliga strömhanteringsobjekt.
4. Överföringsparametrar sätts för SerialPort-objektet.
5. Läsning och skrivning kan nu ske mot de två strömhanteringsobjekten.

5.2 Avkodare

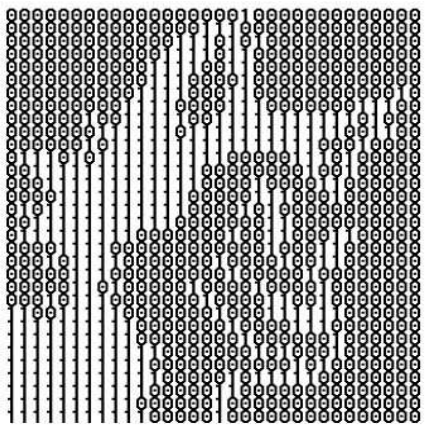
Avkodaren klarar av att avkoda ett SMS eller EMS åtgången. Den data man får från telefonen är en sträng av oktetter där varje oktett är representerad i hexadecimal form. Avkodningen av denna sträng går till på följande vis:

```
TerranSms sms;
TerranDecoder decoder = new TerranDecoder(); // skapar ett objekt av avkodar
try{
	sms = decoder.decodeSMS(pduString); // kodar av oktett sträng
}catch(TerranDecoderException e){
	error(e);
}
```

Vid SMS avkodning returneras ett objekt av typen *TerranSms*. De olika fälten i PDU:t lagras i detta objekt som någon av följande typer: som boolean om det rör sig om enskilda bitar, som ett binärt 8-bitars tal t.ex. TP-PID eller som en textsträng t.ex. TP-UD. Om ett SMS skulle innehålla ett datafält som börjar med en header så krävs det ytterligare behandling. Dessa klassas som EMS, även isärdelade (concatenated) ”vanliga” meddelanden är EMS enligt denna lösning. Avkodning av SMS med header till EMS sker på följande vis:

```
if(sms.hasUdh()){ // Om SMS:ets meddelandefält börjar med en header
	try{
		ems = decoder.decodeEMS(sms);
	}catch(TerranDecoderException e){
		error(e);
	}
}
```

EMS avkodningen returnerar ett objekt av typen *TerranEms* vilket är en subklass under *TerranSms*. Data som lagras vid EMS avkodningen är eventuella index som ett isärdelat meddelande innehåller d.v.s. referens, antal meddelanden som hör ihop och ett sekvensnummer. Eventuella bilder avkodas till en matris bestående av ettor och nollor (figur 5.1), där en en nolla representerar en vit punkt och en etta en svart punkt.



Figur 5.1: En avkodad bild representerad som en matris, 0 = vit punkt, 1 = svart punkt

Vid fel under avkodningen eller om man skickar in ett PDU som inte stöds så genereras ett undantag vilket måste fångas utanför avkodaren. De fält som innebar mest arbete att avkoda var adressfälten samt konverteringen från 8 bitars till 7 bitars septeter. Adressfälten består av tre fält (tabell A.1), ett längd fält som innehåller längden på numret, nummertypsfältet och själva nummerfältet. Det man ska tänka på här är att nummerfältet är representerat i semi-oktetter och

att längden inte är antalet oktetter i nummerfältet utan antalet siffror i numret.

Ett exempel som visar avkodning av adressfältet:

Ett adressfält innehåller följande:

Längd	Typ	Nummer i semi-oktetter
0B	91	6437894471F4

Nummerfältet är lagrat i semi-oktetter vilket betyder att man får flytta om varje par.

```
64 37 89 44 71 F4 -> omflytt ger -> 46 73 98 44 17 4(F)
```

Eftersom längden på numret är ojämnt (0B = 11) så innehåller nummerfältet ett ”trailing F”, detta F finns med bara för att fylla ut så att det blir jämt och har ingen betydelse för nummret. Typen 91 betyder international format. Om typen hade varit 81 (okänt) istället så hade nummerfältet varit 7093481447 (0739844174), här består nummerfältet av 10 (0A) siffror vilket är jämt så här behövs ingen utfyllnad.

Konverteringen av meddelandefältet till 7-bitars alfabetet löstes på följande sätt:

```
private String decode2Septets(String octets,int nSep) throws TerranDecoderException{
  String msg=new String();
  Binary bin;
  int nBit = Binary.BIT7;
  enQbits(EMPTY_Q_POST);

  while(msg.length()<(nSep)){
     try{
        try{//Konverterar HEX oktett till binärt tal
           bin = new Binary(octets.substring(0,ONE_OCT),Binary.HEX_BASE,Binary.EIGHT_BITS);
        }catch(NumberFormatException e){
           throw new TerranDecoderException(e.getMessage(),e);
        }
        //lägger ut bitar som ska läggas till nästa oktett på kön
        enQbits(bin.toSubBin(Binary.BIT7,nBit++));
        //Kodar av resterande bitar samt de bitarna som finns på kön till ett tecken
```

```
        msg = msg + getLetterFromSeptet(Binary.merge(bin.toSubBin(nBit),deQbits()));

        //Om man har gjort ett varv
        if(nBit==Binary.BIT0){
           //Kodar av de bitarna som finns på kön till ett tecken
           msg = msg + getLetterFromSeptet(deQbits());
           nBit = Binary.BIT7;
           enQbits(EMPTY_Q_POST);
        }
        octets = octets.substring(ONE_OCT);//tar bort oktett
     }catch(StringIndexOutOfBoundsException e){
        throw new TerranDecoderException(e.getMessage(),e);
     }
  }
  flushQ();//rensar kön
  return msg;
}
```

Ihopsättning av särdelade meddelanden sker inte i avkodaren p.g.a. att man behöver avkoda samtliga meddelanden för att veta vilka som hör ihop. Det finns inga garantier att de ligger ordnade efter varandra i telefonen. Funktionen som slår ihop dessa meddelanden lades som en statisk funktion i klassen *TerranEms*. Ihopsättningen fungerar på följande vis: innan man anropar funktionen måste alla aktuella meddelanden sorteras m.h.a. de tidigare nämnda index. Till den statiska funktionen skickas en array av EMS. Dessa EMS är sorterade efter sekvensnummer och består av sammanhörande meddelanden. Det som sammanfogas är meddelandefältet samt eventuella bilder. Tidsstämpeln i det nya meddelandet sätts till den tid som det senast anlända meddelandet har.

Avkodaren saknar en del funktioner vilket gör att den inte är fullständig. Det som saknas implementerades inte eftersom uppdragsgivaren inte tyckte det var nödvändigt för den första versionen. Det som fattas i avkodaren är:

- Avkodarn saknar en 8 samt 16 bitars avkodning
- Komprimerade SMS kan inte avkodas

- De enda EMS som kan avkodas är de som innehåller bilder (samt ”vanliga” isärdelade meddelanden som klassas som EMS i denna lösning)

5.3 Filhantering

För filhanteringen så implementerades ett virtuellt filsystem. Detta består av klasserna *TerranDirSys*, *TerranDir* samt *TerranSms*. De två sistnämnda har implementerats så att de går att skriva till disken som instanser. Detta görs i Java genom att implementera gränssnittet *Serializable*. Detta är implementerat på följande sätt:
TerranDirSys har en instans av *TerranDir* som kallas root. *TerranDir* består av två länkade listor, en för objekt av typen *TerranDir* och en för objekt av typen *TerranSms*. Detta innebär att även objekt av typen *TerranEms* kan lagras, eftersom det är en subklass till *TerranSms*. Även funktioner för att utföra operationer på dessa listor finns här. Root-objektet kan ses som en direkt avbild av katalogsystemet som visas för användaren i det grafiska gränssnittet. Fördelen med detta system är att det endast blir en fil att spara, nämligen root-objektet i *TerranDirSys*.

5.4 Test

För att testa att programmet klarar de krav som ställts i kravspecifikationen har tester utförts med tre olika typer av anslutningar och telefoner. Testning av programmet har även skett på olika operativsystem och datorer. De operativsystem som programmet har testats på är Windows 98, ME, 2000 och XP. Tester på andra operativsystem har inte kunnat ske p.g.a. tidsbrist.

Testningen har skett fortlöpande under projektets arbetsgång på en Ericsson T65 som är den telefon som varit tillgänglig under hela projektet. De övriga telefoner som testats har endast varit tillgängliga i korta perioder.

Eftersom programmet utåt endast ser en serieport så spelar inte valet av anslutningstyp någon roll för funktionen utan beror endast på operativsystemet. Därför kan inte något specifikt test göras för varje typ av anslutning, dock har programmet fungerat lika bra för alla de typer som

kunnat testats.

Inledningsvis togs olika typer av PDU-strängar fram för att testa med. Dessa genererades med hjälp av en mobiltelefon och nerladdades samt sparades med ett terminalprogram. Vissa av meddelandena modifierades för att testa vissa specialfall. Detta gjordes för att testa att avkodaren upptäcker fel och eventuellt känner igen PDU-strängar som inte stöds av programmet t.ex. ett datafält kodat i 16 bitars textformat.

En brist som upptäcktes vid testning är att programmet låser sig om man bryter anslutningen till telefonen under nerladdningsfasen. Mer om detta tas upp i avsnitt 7.1.

6 Programbeskrivning

I detta kapitel görs en funktionsbeskrivning av programmet. I avsnitt 6.1 så beskrivs hur man går till väga för att installera programmet. Avsnitt 6.2 ger en detaljerad bild av användargränssnittets olika funktioner. Slutligen kommer ett avsnitt som förklarar felhantering.

6.1 Installation

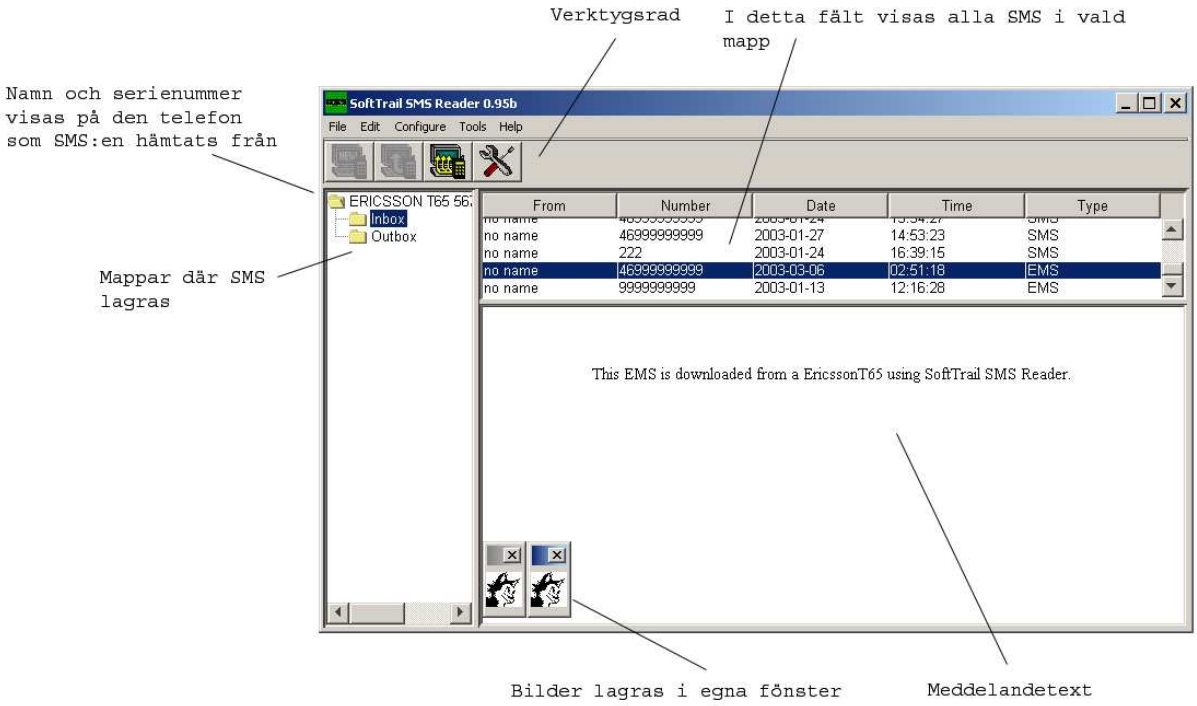
För att kunna köra SoftTrail SMS reader krävs det att man har en version av Java Runtime Environment (JRE). Programmet har inte körts på versioner av JRE tidigare än 1.4.1. Detta gör att det inte kan garanteras att programmet fungerar på tidigare versioner av JRE. I Microsoft Windows installeras programmet genom att köra ett installationsscript. Tyvärr går det bara att köra den nuvarande versionen av programmet i Microsoft Windows. Ett alternativ till scriptfilen är att ladda ner källkoden från CVS servern på Sourceforge via projektets hemsida [10].

6.2 Användargränssnitt

Det grafiska användargränssnittet är uppbyggt med hjälp av Swing-komponenter [9]. För att användaren lätt ska kunna känna igen sig är utseendet likt en e-postklient. Huvudfönstret (figur 6.1) är uppdelat i tre huvudsakliga delar. Trädvyn visar de mappar som SMS-meddelanden sparas till, roten på detta träd visar namnet på den telefon som för tillfället är öppen. Den andra delen är en tabell-liknande lista som visar alla SMS-meddelanden i den valda mappen samt avsändare, nummer, tid, datum samt vilken typ det är. Slutligen finns en vy där det meddelande som är valt i tabellen visas. De bilder som finns infogade i EMS-meddelanden visas i egna fönster i nedre vänstra hörnet av denna vy. Huvudfönstret innehåller även en verktygsrad som innehåller programmets huvudsakliga funktioner.

File -> Open

Funktionen *Open* (figur 6.2) ger användaren möjlighet att öppna en skapad katalogstruktur som är genererad från en specifik telefon. Vid öppnandet återskapas den katalogstruktur som tillhör



Figur 6.1: Huvudfönster

den valda telefonen. Hur dessa filer skapas beskrivs under *Tools -> Download All*.

File -> Exit

Valet *Exit* avslutar programmet.

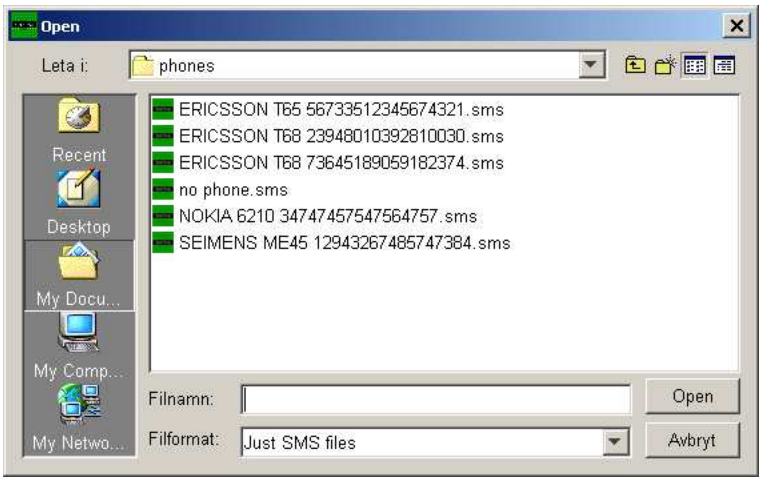
Edit -> Delete

Delete tar bort ett eller flera markerade SMS från listan med SMS.

Configure -> Settings

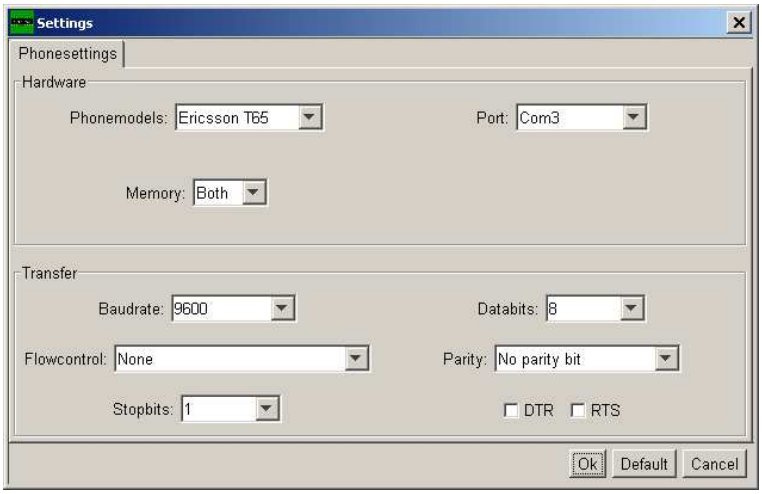
Vid valet *Settings* öppnas en dialogruta (figur 6.3) där man kan välja alternativ för olika parametrar. De parametrar som kan ändras är följande:

- *Phonemodel* - Här ställer användaren in den telefonmodell som ska användas.
- *Port* - Detta val talar om för programmet vilken COM-port som telefonen är kopplad till.
- *Memory* - Här anges från vilket minne i telefonen som programmet laddar ner SMS från.



Figur 6.2: *Open* dialogfönster

- *Baudrate, Databits, Flowcontrol, Parity, Stopbits, DTR, RTS* - Alla dessa är överförings-parametrar. De förinställda värdena är de vanligaste och fungerar ofta utmärkt utan ändring.



Figur 6.3: Dialogfönster för konfigurering

Tools -> Download All

Download All ger användaren möjlighet att ladda ner alla SMS som finns lagrade i den anslutna telefonen. Meddelandena hämtas från det minne som är inställt i *Settings*. Dessa SMS placeras

i antingen *Inbox* eller i *Outbox* beroende på om det är ett mottaget eller skickat SMS. Denna katalogstruktur sparas automatiskt i en fil med ett namn som är sammansatt av telefonens modell, tillverkare och serienummer. Om samma telefon använts vid tidigare nerladdningar kommer de nya meddelanden att läggas till och de gamla att finnas kvar. Dubbletter tas det ingen hänsyn till utan dessa sparas som två eller flera olika meddelanden.

Help -> About

Vid valet *About* (figur 6.4) så visas ett informationsfönster med information om programmet.

Figur 6.4: Om programmet

6.3 Felsökning/Loggning av fel

Alla undantag som inträffar i programmet loggas i en textfil (log.txt) under programmets hemkatalog. Dessa fel kan bl.a. vara:

- Felmeddelanden från telefondrivrutinen.
  - ”No Such Port” - Inträffar när inte porten finns tillgänglig på systemet.
  - ”Port In Use” - Inträffar när porten används av en annan tillämpning.
  - ”Unsupported Comm Operation” - Inträffar om inställningar satts som inte stöds av serieanslutningen.
  - ”Nullpointer” - Odefinierat fel men troligen har ett fel i överföringen inträffat. Se loggfil för mer information.
  - ”No Such Memory” - Minnestypen stöds inte av telefonen.
  - ”Error getting list of SMS” - Överföringen avbruten eller något fel har inträffat under överföringen av SMS-meddelanden från telefonen.
- Felmeddelanden från avkodaren.
  - ”SMS is not a submit or deliver” - Fel typ av SMS-PDU.
  - ”Cant decode adressfield” - Fel adressfält i PDU:n.
  - ”Adress type not supported” - Adresstypen stöds inte av dekodern.
- Felmeddelanden vid in- och utdata.
  - ”Invalid file type” - Fel vid öppnade av SMS-fil. Filen måste innehålla ändelsen ”.sms”.
  - ”Error when opening file” - Fel vid öppnande av fil. Troligtvis fel vid läsning av fil från disk.

Förutom att felen loggas så visas även felmeddelandet i en dialogruta (figur 6.5).

Figur 6.5: Exempel på felmeddelande

7 Slutsatser

7.1 Erfarenheter och problem

Många problem har uppstått under arbetet med programmet men dessa har vi kunnat lösa på ett eller annat sätt.

Det första problemet som uppstod var svårigheten med att få tag på standarder som behövdes för dekodern och modemkommunikationen. Till slut fick vi tag på dessa genom *European Telecommunications Standards Institute* (ETSI) som tillhandahåller sina standarder gratis genom deras hemsida. Mycket information har dessutom kunnat hittas på telefontillverkarnas hemsidor samt genom vår handledare som tillhandahållit viss information.

Ett implementationsproblem som uppkom i slutet av arbetet var hur man skulle kunna identifera ett meddelande för att inte dubbletter ska lagras. Problemet ligger i att det inte görs någon skillnad på skickade och mottagna meddelanden när de lagras i programmet. Eftersom skickade meddelanden inte har en tidstämpel kan inte denna parameter användas för att avgöra om ett meddelande redan är nerladdat. Möjliga lösningar på detta kan vara att strukturera om filhanteringen eller att någon form av checksumma kan genereras för varje meddelande.

Ett annat problem, som upptäcktes när vi testade att bryta anslutningen under nerladdning av meddelanden, är att lästråden låser sig. Detta inträffar eftersom programmet försöker att läsa från serieporten, men eftersom ingen data finns att läsa väntar tråden till dess att data fås. Lösningen på detta problem hade varit att skriva om de delar som används för kommunikationen med telefonmodemet så att dessa använder händelsehanterare istället.

En önskan från uppdragsgivaren var att programmet skulle vara plattformsoberoende vilket vi inte lyckades fullt ut med. Detta för att serieportskommunikation inte är implementerad i Javas standard API och det tilläggsbibliotek som vi använt oss av har inte varit plattformsoberoende. Detta har varit ett stort problem i vårt arbete. Ett annat problem är att det finns krav på hur man lägger filerna som tillhör JCA. Filen *comm.jar* måste ligga på samma enhet som programmet och *javax.comm.properties* måste ligga i samma katalog som *comm.jar* annars hittas inga

portar i systemet. En lösning på problemet som vi har använt oss av men som gör programmet plattformsberoende är att lägga till några rader som initierar drivrutinen för serieporten direkt i koden. Dessa rader behöver läggas till:

```
String drivernamn = ''com.sun.comm.Win32Driver'';
try{
       CommDriver driver = (CommDriver) Class.forName(drivername).newInstanc
       driver.initialize();
}catch(Exception e){}
```

En annan skillnad mellan olika operativsystem är att COM-portarnas namn skiljs åt. I Microsoft Windows heter det t.ex. *COM3* medan detsamma i Linux heter */dev/ttyS2*.

7.2 Möjlig utökning och förbättring

En viss utökning och förbättring av programmet kommer troligtvis att göras efter att examensarbetet är avslutat. De punkter som ligger närmast till hands och som vi tycker är högt prioriterade är:

- Skapa plattformsoberoende.
- Mer hjälp för användaren i gränssnittet.
- Händelsestyrd serieportkommunikation.
- Möjlighet att spara bilder i något bildformat.
- Möjlighet att skapa egna kataloger.
- Flera nerladdningsfunktioner, t.ex. ladda ner enbart olästa SMS.
- Möjlighet att kunna ta bort SMS i samband med nerladdning.
- Integrera adressboken.

- Skapa och skicka SMS/EMS med hjälp av telefonen.
- Implementera stöd för MMS.
- Möjlighet att konfigurera telefonen i programmet.

7.3 Kort sammanfattning av resultatet

Vi tycker att vi lyckats väl med att uppnå de krav som ställdes på programmet. Programmet har nu ersatt den befintliga prototypen. Ett önskemål var att programmet skulle stödja telefoner tillverkade av de tre stora tillverkarna Ericsson, Nokia och Siemens. Detta visade sig vara enklare än befarat eftersom alla dessa använder sig av en gemensam standard för de viktigaste funktionerna. Många telefoner har dock, utöver denna standard, egen funktionalitet som utökar standarden. Om man vill använda denna utökade funktionalitet i programmet blir man tvungen att utveckla särskilda drivrutiner för dessa telefoner.

Referenser

[1] Ericsson Mobile Communications AB. *Mobile Phone T68 Developers Guidelines. AT Commands Online Reference, EN/LZT 108 5194 R1A*. Ericsson Mobile Communications AB, Augusti 2001.

[2] Ian Darwin. *Java Cookbook*. O'Reilly, 2001.

[3] European Telecommunications Standards Institute. *ETSI TS 100 585 version 7.0.1 Release 1998. Digital cellular telecommunications system (Phase 2+); Use of Data Terminal Equipment - Data Circuit terminating; Equipment (DTE - DCE) interface for Short Message Service (SMS) and Cell Broadcast Service (CBS)*, Juli 1999.

[4] European Telecommunications Standards Institute. *ETSI TS 100 900 version 7.2.0 Release 1998. Digital cellular telecommunications system (Phase 2+); Alphabets and language-specific information*, Juli 1999.

[5] European Telecommunications Standards Institute. *ETSI TS 123 140 version 3.0.1 Release 1999. Universal Mobile Telecommunications System (UMTS); Multimedia Messaging Service (MMS), Functional description; Stage 2*, Mars 2000.

[6] European Telecommunications Standards Institute. *ETSI TS 123 040 version 4.6.0 Release 4. Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); Technical realization of the Short Message Service (SMS)*, Mars 2002.

[7] Free Software Foundation Inc. *GNU General Public License (GPL)*. http://www.gnu.org/licenses/gpl.html, Juli 2001. (hämtad 2003-05-21).

[8] Stig Jensen och Arne Gjelstrup och Valention Berti. *Datakommunikation*. Liber AB, 2000.

[9] Mary Campione och Kathy Walrath och Alison Huml. *The Java Tutorial Third Edition*. Addison-Wesley, 2001.

[10] Open Source Development Network (OSDN). *SoftTrail SMS reader - projekthemsida*. http://sourceforge.net/projects/terranproject. (hämtad 2003-05-21).

[11] Open Source Development Network (OSDN). *Sourceforge*. http://sourceforge.net. (hämtad 2003-05-21).

[12] Open Source Development Network (OSDN). *Spoon Installer - projekthemsida*. http://sourceforge.net/projects/spoon-installer. (hämtad 2003-05-21).

[13] Xinox Software. *JCreator 2.5*. http://www.jcreator.com. (hämtad 2003-05-21).

[14] WAP Forum, http://www.wapforum.org. *WAP MMS Architecture Overview. WAP-205-MMSArchOverview-20010425-a*, April 2001.

[15] WAP Forum, http://www.wapforum.org. *Wireless Application Protocol Architecture Specification. WAP-210-WAPArch-20010712*, Juli 2001.

[16] Jeroen Wijnhout. *Kile 1.4*. http://soliton.science.uva.nl/ wijnhout/Kile. (hämtad 2003-05-21).

A Definition av TPDU parametrar

Följande utdrag är en sammanfattning av de parametrar som finns i SMS-SUBMIT och SMS-DELIVER [6].

A.1 TP-Message-Type-Indicator(TP-MTI)

Message-Type-Indicator är ett tvåbitarsfält, som är placerat i bit nummer 0 och 1 i första oktetten hos alla olika PDU:s.
Kan ha följande värden:

bit1	bit0	Meddeladetyp
0	0	SMS-DELIVER(i riktning SC till MS)
0	0	SMS-DELIVER REPORT(i riktning MS till SC)
1	0	SMS-STATUS-REPORT(i riktning SC till MS)
1	0	SMS-COMMAND(i riktning MS till SC)
0	1	SMS-SUBMIT(i riktning MS till SC)
0	1	SMS-SUBMIT-REPORT(i riktning SC till MS)
1	1	Reserverat

Om en MS mottar ett TPDU med ett reserverat värde i TP-MTI skall den behandla meddelandet som om det vore ett SMS-DELIVER men spara meddelande exakt som det mottogs.

A.2 TP-More-Messages-to-Send(TP-MMS)

More-Messages-to-Send är informations elementet som SC använder för att informera MS om att det finns ett eller flera meddelanden hos SC som väntar på att bli skickade till MS.
TP-MMS är ett enbitsfält placerat i bit nummer 2 i första oktetten hos ett SMS-DELIVER och även i SMS-STATUS-REPORT.
Kan ha följande värden:

Bit2	
0	Fler meddelanden väntar på MS i denna SC
1	Inga fler meddelanden väntar på MS i denna SC

A.3 TP-Reject-Duplicates(TP-RD)

TP-Reject-Duplicates är ett enbitsfält placerat i bit nummer 2 i den första oktetten i ett SMS-SUBMIT och kan ha följande betydelse:

Bit2

0 Instruerar SC att acceptera ett SMS-SUBMIT av ett SM som fortfarande hålls av SC, vilket har samma TP-MR och samma TP-DA som föregående skickade SM från samma OA.

1 Instruerar SC att inte acceptera ett SMS-SUBMIT av ett SM som fortfarande hålls av SC, vilket har samma TP-MR och samma TP-DA som föregående skickade SM från samma OA.

A.4 TP-Validity-Period-Format(TP-VPF)

TP-Validity-Period-Format är ett tvåbitsfält placerat i bit 3 och 4 i första oktetten hos ett SMS-SUBMIT.

Kan ha följande värden:

bit4	bit3	Meddeladetyp
0	0	TP-VP-fältet är inte inkluderat
1	0	TP-VP-fältet är inkluderat i relativt format
0	1	TP-VP-fältet är inkluderat i förhöjt (eng. enhanced) format
1	1	TP-VP-fältet är inkluderat i absolut format

A.5 TP-Status-Report-Indication(TP-SRI)

TP-Status-Report-Indication är ett enbitsfält placerat i bit nummer 5 i första oktetten hos ett SMS-DELIVER.

Kan ha följande värden:

Bit5

0 En statusrapport skall inte returneras till SME

1 En statusrapport skall returneras till SME

A.6 TP-Status-Report-Request(TP-SRR)

TP-Status-Report-Request är ett enbitsfält placerad i bit nummer 5 i första oktetten hos ett SMS-SUBMIT och SMS-COMMAND.

Kan ha följande värden:

Bit5

0 En status rapport är inte begärd

1 En status rapport är begärd

A.7 TP-User-Data-Header-Indicator(TP-UDHI)

TP-User-Data-Header-Indicator är ett enbitsfält placerat i bit nummer 6 i första oktetten hos samtliga sex PDU:s.

Kan ha följande värden:

Bit6

0 TP-UD fältet innehåller endast ett textmeddelande

1 Början av TP-UD fältet innehåller en header som tillägg till textmeddelandet

A.8 TP-Reply-Path(TP-RP)

TP-Reply-Path är ett enbitsfält placerat i bit nummer 7 i första oktetten hos både SMS-DELIVER och SMS-SUBMIT.

Kan ha följande värden:

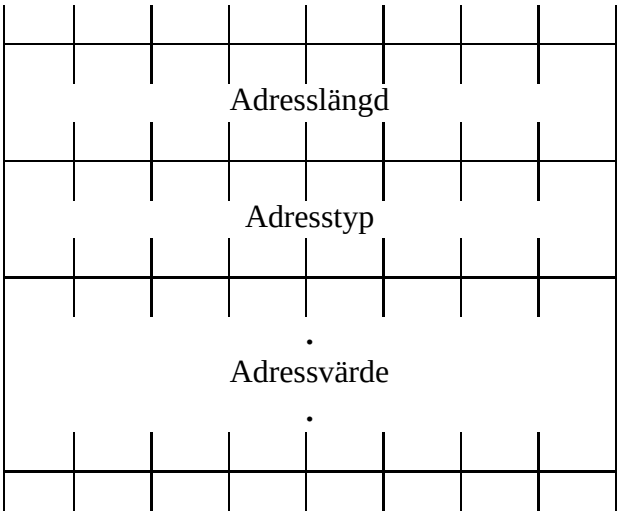
Bit7

0 TP-Reply-Path-parametern är inte satt i detta SMS-SUBMIT/DELIVER

1 TP-Reply-Path-parametern är satt i detta SMS-SUBMIT/DELIVER

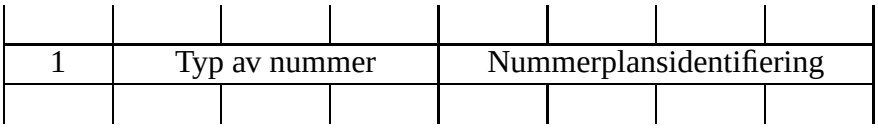
A.9 TP-Originating-Address(TP-OA)

TP-Originating-Address innehåller adressen till avsändaren av det aktuella SMS:et.
Detta adressfält består av följande delfält: En adresslängd som är en oktett lång, ett adresstypsfält som även det är en oktett lång och ett adressvärdesfält av varierande längd som kan ses i tabell A.1.



Tabell A.1: Uppställningen av TP-OA

Adressfältslängden är ett heltal av antalet användbara semi-oktetter i adressvärdesfältet. Adresstypsfältets uppställning kan ses i tabell A.2.



Tabell A.2: TP-OA Adresstypsfält

Typ av nummer fältet finns i bitarna 6-4 och kan ha följande värden:

bit6	bit5	bit4	Typ av nummer
0	0	0	Okänt
0	0	1	Internationellt nummer
0	1	0	Nationellt nummer
0	1	1	Nätverksspecifikt nummer
1	0	0	Abonnentnummer
1	0	1	Alfanummeriskt
1	1	0	Förkortat nummer
1	1	1	Reserverat för utbyggnad

Nummerplansidentifieringsfältet finns i bitarna 3-0 och kan ha följande värden:

bit3	bit2	bit1	bit0	Nummerplansidentifiering
0	0	0	0	Okänt
0	0	0	1	ISDN/telfon nummerplan
0	0	1	1	Data nummerplan
0	1	0	0	Telex nummerplan
0	1	0	1	SC specifikt nummer plan
0	1	0	0	SC specifikt nummer plan
1	0	1	0	Nationellt nummerplan
1	0	0	1	Privat nummerplan
1	0	1	0	ERMES nummerplan
1	1	1	1	Reserverat för utbyggnad

Alla andra värden är reserverade

Adressvärdesfält är representerat av antingen semi-oktetter eller Alfanummeriskt.

Den maximala längden av ett adressfält(adress längd, adress typ och adress värde) är 12 oktetter.

A.10 TP-Destination-Address(TP-DA)

TP-Destination-Address är kodat på samma sätt som TP-OA, se A.9.

A.11 TP-Protocol-Identifier(TP-PID)

TP-Protocol-Identifier är informationselementet som antingen hänvisar till att ett ”higher layer protocol” används, eller indikerar ”interworking” med en viss typ av ”telematic device”. TP-PID består av en octet och kan vara satt till följande värden:

0 bit7	bit6	Användning
0	0	Bitarna 0-5 är satta enligt definition nedan
0	1	Bitarna 0-5 är satta enligt definition nedan
1	0	Reserverad
1	1	Bitarna 0-5 är satta för SC specifikt behov

Om fallet är att bit 7 = 0 och bit 6 = 0 så:

bit5	Användning
0	Ingen ”Interworking”, men SME-till-SME protokoll
1	”Telematic interworking”

Om fallet är att det är ”telematic interworking” så anger ett fembitars mönster i bitarna 0-4 olika typer av ”telematic” enheter.

4 . . . 0	Enhet
0 0000	Implicit - enhetstypen är specifik i den här SC, eller kan vara avgjord beroende på adressen
0 0001	Telex(eller teletex till telex format)
0 0010	Grupp 3 telefax

0 0011	Grupp 4 telefax
0 0100	Taltelefon(d.v.s. konvertering till tal)
0 0101	ERMES(European Radio Messaging System)
0 0110	Nationellt ”Paging” system(känt hos SC)
0 0111	Videotex(T.100[20]/T.101[21])
0 1000	Teletex, ”carrier unspecified”
0 1001	Teletex, i PSPDN
0 1010	Teletex, i CSPDN
0 1011	Teletex, i analogt PSTN
0 1100	Teletex, i ISDN
0 1101	UCI(Universal Computer Interface)
01110..01111	Reserverade
1 0000	En meddelandehanteringsanordning (känd av SC)
1 0001	Alla offentliga X.400-baserade meddelandehanteringssystem
1 0010	Internet Elektronisk post
10011..10111	Reserverade
11000..11110	SC specifika värden, användning baserad på inbördes överenskommelse mellan SME och SC
1 1111	En GSM/UMTS mobil station. SC konverterar SM från det mottagna TP-DCS till någon annan DCS som stöds av MS

Om bit 5 har värdet 1 i ett SMS-SUBMIT PDU så indikerar det att SME är en telematic enhet av typ som är indikerad i bitarna 0-4 och begär att SC ska konvertera SM till ett format som passar en sådan enhet. Om destinationen är ISDN måste SC även välja en lämplig serviceindikator för att ansluta en sådan typ av enhet.
Om bit 5 har värdet 1 i ett SMS-DELIVER PDU så indikerar det att SME är en ”telematic device” av en sådan typ som är indikerad i bitarna 0-4.

Om bit 5 har värdet 0 i ett SMS-DELIVER PDU så identifierar värdet i bitarna 0-4 vilket SM-AL protokoll som används mellan SME och MS
Notera att för ett enkelt MS till SC SM överföring så är PID satt till ett värde av 0.

I fallet att bit 7=0 och bit 6=1 så är bit 0-5 satta enligt följande:

5 0	Enhet
0 00000	Short Message Type 0
0 00001	Replace Short Message Type 1
0 00010	Replace Short Message Type 2
0 00011	Replace Short Message Type 3
0 00100	Replace Short Message Type 4
0 00101	Replace Short Message Type 5
0 00110	Replace Short Message Type 6
0 00111	Replace Short Message Type 7
001000..01110	Reserverade
0 11110	Enhanced Message Service (förlegad)
0 11111	Return Call Message
100000..111011	Reserverade
1 11100	ANSI-136 R-DATA
1 11101	ME Data download
1 11110	ME De-personalization Short Message
1 11111	(U)SIM Data download

A.12 TP-Data-Coding-Scheme(TP-DCS)

TP-Data-Coding-Scheme fältet indikerar DCS av TP-UD fältet och kan indikera en meddelandeklass. Ett reserverad kod ska antas vara GSM standardalfabet (samma som värdet 00000000)

av den mottagande enheten. Oktetten är ordnad i kodningsgrupper (se tabell A.3) vilka är indikerade i bitarna 4-7.

A.13 TP-Service-Center-Time-Stamp(TP-SCTS)

Service-Center-Time-Stamp är informationselementet som SC meddelar en mottagande MS om ankomsttiden av meddelandet hos SC. Fältet är angivet i semi-oktetter och representerar den lokala tiden på följande sätt som visas nedan.

	År	Månad	Dag	Timme	Minut	Sekund	Tidszon
Antal tecken: (semi-oktetter)	2	2	2	2	2	2	2

Tidszonen indikerar differensen uttryckt i kvartar av en timme mellan den lokala tiden och GMT. I den första av de två semi-oktetterna, representerar den första biten (bit 3 hos den sjunde oktetten av TP-SCTS) det algebraiska tecknet hos denna skillnad(0=positiv, 1=negativ).

A.14 TP-Validity-Period(TP-VP)

Relativt format

TP-VP innefattar ett heltal med storlek av 1 oktett. Detta heltal anger validitetsperioden räknad från tiden då SC mottar meddelandet. Tiden representeras på följande sätt:

TP-VP värde	Validitetsperiod
0-143	(TP-VP + 1) * 5 minuter
144-167	12 timmar + ((TP-VP - 143) * 30 minuter)
168-196	(TP-VP - 166) * 1 dag
197-255	(TP-VP - 192) * 1 vecka

Absolut format

TP-VP är 7 oktetter stort representerat i semi-oktetter och anger den absoluta tiden för validitets

Kodnings grupp 7..4	Användning av övriga bitar
00xx	Generell datakodning Bit 5..0 indikerar följande: Bit5 0 Inte komprimerad text 1 Komprimerad text Bit4 0 Bit 1 och 0 är reserverade och innehåller ingen meddelandeklass 1 Bit 1 och 0 har meddelande klass: Bit1 Bit0 Meddelandeklass 0 0 Klass 0 0 1 Klass 1 Default betyder: ME-specifikt 1 0 Klass 2 Sim specifikt meddelande 1 1 Klass 3 Default betyder: TE-specifikt Bit 3 och 2 indikerar vilket alfabet som används: Bit3 Bit2 Alfabet 0 0 Defult alfabet 0 1 8 bitars data 1 0 UCS2(16bit) 1 1 Reserverat
0100..1011	Reserverade kodningsgrupper
1100	Meddelandet väntar indikations grupp: Bortse från meddelandet. Bitarna 3..0 är kodade exakt som grupp 1101, men med bitarna 7..4 satta till 1100 kan mobilen bortse från innehållet i meddelandet och endast presentera indikationen till användaren
1101	Meddelandet väntar indikations grupp: Spara meddelandet. Text inkluderat i användardatan är kodad i det defaulta alfabetet. Mobilen skall spara texten hos SMS meddelandet utöver att sätta indikationen. Bit 3 indikerar indikationsbetydelse Bit3 0 Sätt indikationen inaktiv 1 Sätt indikationen aktiv Bit 2 är reserverad och satt till 0 Bit1 Bit0 Indikationstyp 0 0 Voice Mail-meddelande väntar 0 1 Faxmeddelande väntar 1 0 Electronic Mail-meddelande väntar 1 1 Annat meddelande väntar
	fortsätter

Kodnings grupp 7..4	Använding av övriga bitar
1110	Meddelandet väntar indikations grupp: Spara meddelandet. Kodningen av bitarna 3..0 är den samma som för grupp 1101 med undantaget är att texten inkluderad i TP-UD är kodad i den okomprimerade UCS2 alfabetet. Text inkluderat i användar datan är kodad i det defaulta alfabetet.
1111	Datakodning/Medelande klass Bit 3 är reserverad och satt till 0 Bit2 Meddelandekodning 0 Default alfabet 1 8-bit data Bit1 Bit0 Indikationstyp 0 0 Klass 0 0 1 Klass 1 Default betyder: ME-specifikt 1 0 Klass 2 Sim specifikt meddelande 1 1 Klass 3 Default betyder: TE-specifikt

Tabell A.3: TP-DCS kodningsgrupper

periodens slut. Detta fält är identiskt med TP-SCTS.

”Enhanced” format

TP-VP är 7 oktetter stor. Alla 7 ska var inkluderade även om inte alla nödvändigtvis används. Den första oktetten indikerar hur de övriga 6 används. Alla reserverade och oanvända bitar eller oktetter måste vara satta till 0.

Octet 1, TP-VP funktionalitetsindikator

Bit 7(förlängnings bit)	Betydelse
1	TP-VP är förlängt till ytterligare en oktett
0	Inga fler TP-VP funktionalitetsindikatorförlängningar följer

Bit 6(Singel shot SM) är satt till 1 om det krävs.

Bit 5, 4 och 3 är reserverade.

Bit2 Bit1 Bit0 Validitetsperiodsformat

0	0	0	Ingen validitetsperiod specifierad
0	0	1	Validitetsperioden är specificerad likt relativt format. Följande oktett innehåller ett TP-VP värde likt det relativa formatet.
0	1	0	Validitetsperioden är representerat likt relativt format i ett heltal. Efterföljande octet innehåller ett TP-VP värde 0-255 och står för 0-255 sekunder. Ett TP-VP-värde lika med 0 är odefinierat och reserverat för framtida bruk.
0	1	1	Validitetsperioden är representerat i semi-oktetter. Följande 3 oktetter innehåller den relativa tiden givet i timmar, minuter och sekunder.
100..111			Reserverade

A.15 TP-User-Data-Length(TP-UDL)

Om TP-User-Data är kodat med GSM 7-bitars standardalfabet, ger TP-UDL ett heltalsvärde av antalet septetter inom TP-UD. Om TP-UD innehåller ett TP-User-Data-Header fält, då är värdet av TP-UDL summan av antalet septetter i TP-UDH(inklusive eventuella fyllnads bitar) och antalet septetter i TP-UD som följer(se fig). Om TP-UD är kodat i 8-bitars data eller UCS2 så ger TP-UDL en heltals representation av antalet oktetter i TP-UD. Om ett UDH fält är inkluderat så anger TP-UDL ett heltalsvärde för antal oktetter i TP-UDH och det följande TP-UD(se fig). Om TP-UD är kodat med komprimerat GSM 7-bitars standardalfabet eller komprimerat 8 bit data eller komprimerat UCS2 så ger TP-UDL ett heltalsnummer av antalet oktetter inom TP-UD efter komprimering. Om ett TP-UDH-fält är inkluderat så anger TP-UDL summan av antalet oktetter i den okomprimerade TP-UDH och antalet oktetter i den komprimerade TP-UD som följer(se figur A.1).

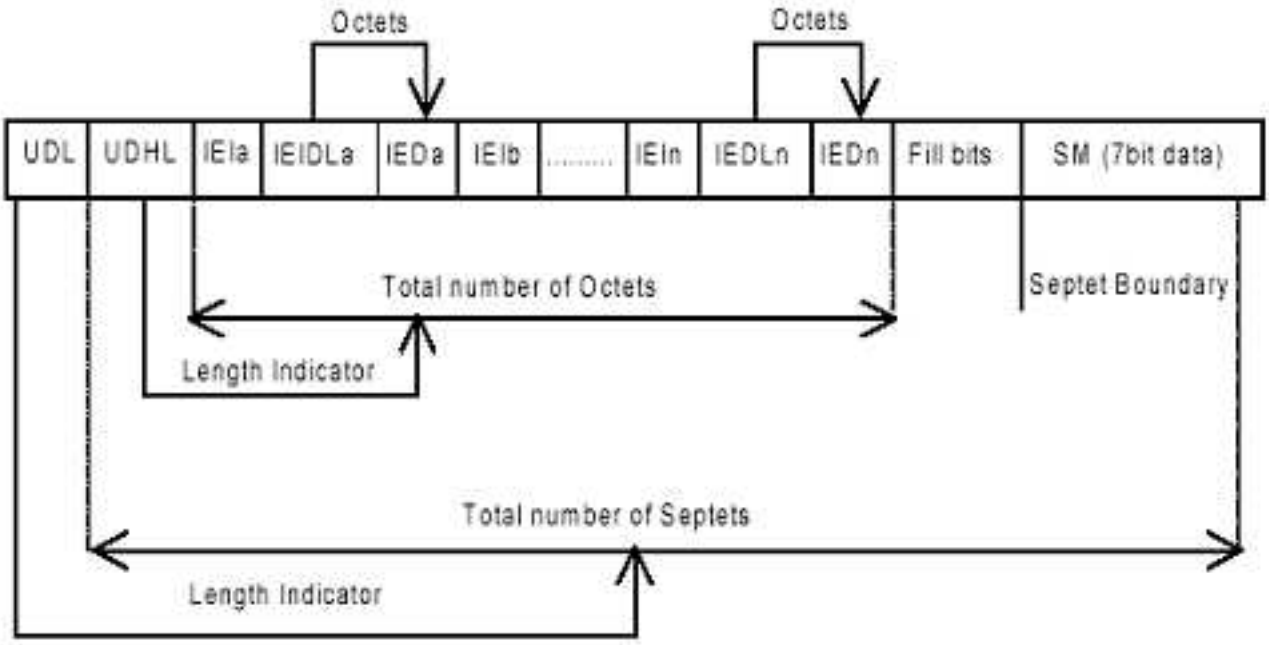
A.16 TP-User-Data(TP-UD)

TP-User-Data kan bestå av enbart ett SM eller så kan det ha en header som tillägg till textmeddelandet beroende på vad som är satt i TP-UDHI.
Om TP-UDHI är satt till 0 så innehåller TP-UD enbart ett textmeddelande, där datan kan vara kodat med 7 bit (standardalfabet), 8 bit data eller 16 bit (UCS2) data.
Men om TP-UDHI är satt till värdet 1 så innehåller TP-UD en header i följande ordning, med start från första oktetten i TP-UD fältet:

Fält	Längd
Längden av Header	1 octet
Information-Element-Identifierare ”A”	1 oktett
Längden av Information-Element-Data ”A”	1 oktett
Information-Element-Data ”A”	0 till ”n” oktetter
Information-Element-Identifierare ”B”	1 oktett
Längden av Information-Element-Data ”B”	1 oktett
Information-Element-Data ”B”	0 till ”n” oktetter
Information-Element-Identifierare ”X”	1 oktett
Längden av Information-Element-Data ”X”	1 oktett
Information-Element-Data ”X”	0 till ”n” oktetter

I figur A.1 kan man se uppställningen av TP-UDL och TP-UD som är komprimerat med GSM 7 bitars standardalfabet. UDHL fältet är den första oktetten i TP-UD.

Kodning av Information-Element-Identifierare oktetten kan ses i tabell A.4. Längden av Information-Element-Data är representerat av ett heltal som anger antalet oktetter inom det tillhörande Information-Element-Data, i denna längd så är inte längdoktetten medräknad.



Figur A.1: TP-UD 7 bitars kodning med UDH [6]

B Förkortningar

API Application Programming Interface

CVS Concurrent Versions System

EMS Enhanced Message Service

ETSI European Telecommunications Standards Institute

GSM Global System Mobile Telecom

JRE Java Runtime Environment

JCA Java Communication API

MMS Multimedia Messaging Service

MS Mobile Station

MVC Model View Controller

PDU Protocol Data Unit

SC Service Central

SM Short Message

SMS Short Message Service

SMSC Short Message Service Central

TPDU Transport Protocol Data Unit

UMTS Universal Mobile Telecommunications System

USB Universal Serial Bus

WAP Wireless Application Protocol

WCDMA Wideband Code Division Multiple Access

Värde(hex)	Betydelse
00	”Concatenated” meddelande, 8 bitars referens nummer
01	Special SMS meddelandeindikator
02	Reserverat
03	Värdet används inte
04	8-bitars adress
05	16-bitars adress
06	SMSC kontrollparametrar
07	UDH källindikator
08	”Concatenated” meddelande, 16 bitars referensnummer
09	Wireless Control Message Protocol
0A	Textformat
0B	Fördefinerat ljud
0C	Användardefinierat ljud(iMelody max 128 bytes)
0D	Fördefinerad animation
0E	Stor animation
0F	Liten animation
10	Stor bild(32*32 = 128 bytes)
11	Liten bild(16*16 = 32 bytes)
12	Variabel bild
13	User prompt indicator
14-1F	Reserverad för framtida EMS
20	RFC 822 E-Mail Header
21-6F	Reserverade för framtidabruk
70-7F	(U)SIM Toolkit Security Headers
80-9F	SME till SME specifik användning
A0-BF	Reserverad för framtidabruk
C0-DF	SC specifik användning
E0-FF	Reserverad för framtida bruk

Tabell A.4: Kodning av Information-Element-Identifierare oktetten