Datavetenskap

Ulf Skogström och Rikard Skogberg

Menybaserade mobiltelefonitjänster

Examensarbete, C-nivå

2000:17

Menybaserade mobiltelefonitjänster

Ulf Skogström och Rikard Skogberg

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.

Ulf Skogström och Rikard Skogberg

Godkänd, 2000-05-30

Handledare: Johan Garcia

Examinator: Stefan Lindskog

Sammanfattning

Dagens mobiltelefoner klarar av mycket mer än att bara förmedla talsamtal. De kan exempelvis användas för att överföra data mellan två datorer eller för att skicka textbaserade meddelanden, så kallade SMS (Short Message Service). Mottagaren av ett SMS-meddelande kan vara någon annan person med en mobiltelefon men det kan också tänkas vara en dator av något slag. Datorn kan då tolka meddelandet och utföra någon tjänst baserat på innehållet i SMS-meddelandet. Exempel på tjänster kan vara att slå på värmen i sommarstugan eller att hämta information på Internet och vidarebefordra den inhämtade informationen till användaren.

Telia Mobile AB har idag en sådant system med ett fyrtiotal fördefinierade tjänster. En användare som vill utnyttja någon av dessa tjänster kan komponera ett SMS-meddelande och skicka detta till en speciell server för att få tjänsten utförd. Ett sådant meddelande måste dock följa en viss syntax så att servern kan tolka det. Det faktum att tjänsterna är olika och kan kräva vissa inparametrar gör att användaren måste känna till syntaxen för varje specifik tjänst. Önskvärt vore att användaren helt slapp tänka på syntaxen för tjänsterna och istället kunde välja tjänst genom menysystemet i mobiltelefonen.

Vårt arbete har resulterat i en lösning på detta problem genom ett system som låter användaren bygga sin egen menystruktur med hjälp av ett grafiskt gränssnitt vid en vanlig persondator. Menystrukturen kan sedan skickas till telefonen där den presenteras i telefonens display. Användaren kan nu bläddra i sitt menysystem och exekvera de tjänster han/hon önskar med några enkla knapptryckningar.

Menubased mobile phone services

Abstract

Contemporary mobile phones can do more than just set up voice calls between people. They can for example be used to transmit data between computers or to send text messages, also known as SMS (Short Message Service). The receiver of such a message can be another person with a mobile phone but it may also be a server. This server can then interpret the message and, based on the content of the message, execute some kind of service. These services can be anything from turning on the heater in a house to collect information on the Internet and then send the collected information back to the user.

At present, Telia Mobile AB provides such a system with about forty predefined services. A user who would like to use any of these services has to compose an SMS and send this to the server to execute the specific service. However, such a message must follow a strict syntax to make it interpretable to the server. Due to the fact that the services are different, and may require parameters as an input, the user has to know the syntax for every specific service. A good solution would be if the user did not have to concern about the syntax at all, but instead was able to browse through his/her services in the mobile phone display.

The result of our project is a system that lets the user design his/her own menu structure through a graphical interface using a standard PC. The menu structure can then be transmitted to the mobile phone, in which it will be presented to the user in the phone's display. This procedure enables the user to browse his/her menu structure and execute any service in a simple manner.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

I takt med att mobiltelefonanvändandet ökat har också önskemålen om att kunna använda telefonerna till annat än bara samtal ökat. Detta har lett till att det numer finns möjlighet att använda sin telefon till att t.ex. överföra data mellan datorer och kommunicera med korta textmeddelanden så kallade SMS-meddelanden (Short Message Service). Ett vidare användningsområde för SMS är att låta SMS-meddelandet i sig vara en signal som triggar en tjänst. En tjänst kan exempelvis vara att slå på värmen i sommarstugan eller att få reda på aktiekurser. Genom att låta en server ansluten till Internet ta emot SMS-meddelanden kan denna sedan utföra önskad tjänst eller hämta önskad information och sedan skicka ett svar till användaren. Detta öppnar nya möjligheter för användandet av mobiltelefoner då dessa kan fungera som en form av fjärrkontroller och fönster mot Internet.

Ett annat sätt att komma åt Internet från en mobiltelefon är det omtalade WAP-systemet. Detta möjliggör en form av web-surfning i mobiltelefoner och innebär att användaren får tillgång till en mängd informationstjänster genom så kallade wml-sidor (jämför html). I dagens tidsdebiterade mobiltelefonisystem kan det dock bli dyrt att koppla upp sig en minut eller två för att bläddra sig fram till någon enkel fråga-svar tjänst. Då kan SMS-baserade tjänster vara ett mycket intressant alternativ på så sätt att ett SMS-meddelande skickas till en speciell server som sedan utför arbetet och skickar tillbaka önskad information. Man behöver alltså inte vara uppkopplad och betala per minut utan betalar då bara för vad det kostar att skicka SMS-meddelandet. Möjligheten att skicka SMS-meddelanden finns i stort sett alla mobiltelefoner och man behöver alltså inte skaffa en ny mobiltelefon för att kunna använda tjänsterna.

För att göra användandet av SMS-baserade tjänster enklare skulle man vilja låta användaren komma åt tjänsterna genom menyn i sin mobiltelefon istället för att komponera ett kryptiskt SMS. Här kommer SIM Application Toolkit (SIMAT), se kapitel 2.2, in i bilden. SIMAT är en standard som gör det möjligt för operatörer att lägga in egna applikationer på användarnas SIM-kort. Man kan då tänka sig en applikation som presenterar en meny av tjänster och låter användaren bläddra sig fram till önskad tjänst. Applikation kan sedan

komponera och skicka SMS-meddelandet som behövs för att trigga tjänsten. SIMAT stöds av merparten av alla mobiltelefoner tillverkade det senaste året och även en del äldre modeller.

1.1 Bakgrund

SIM Application Toolkit (SIMAT) är en del av en GSM-standard som beskriver kommunikationsgränssnittet mellan ett SIM-kort och en mobiltelefon. Det kom till stånd efter önskemål från teleoperatörer för att de skulle kunna skapa egna unika tjänster i mobiltelefonerna utan att mobiltelefontillverkarna skulle behöva ta fram nya modeller av mobiltelefoner för varje ny tjänst. Eftersom det är teleoperatören som äger SIM-kortet som finns i telefonen sågs en möjlighet i att kunna erbjuda dessa unika tjänster till sina kunder genom att lägga egenutvecklade applikationer på SIM-kortet. Dessa applikationer skulle då fungera på vilken mobiltelefon som helst som stöder SIMAT.

Ett SIM-kort innehåller dock en mycket begränsad mängd minne tillgängligt för applikationer. Genom att ladda ned nya applikationer till telefonen när användaren har behov av en ny tjänst kan detta problem kringgås. Det kommer dock att ta relativt lång tid, i storleksordningen minuter, att ladda ner nya applikationer vilket gör att denna lösning blir mindre attraktiv för användarna.

Tack vare att en SIMAT-applikation är så hårt knuten till att sända och ta emot väl definierade meddelanden och att göra menyval är det möjligt att beskriva en tjänst med hjälp av en datastruktur som kan tolkas av en *generell* applikation på SIM-kortet. På så sätt kan en ny tjänst beskrivas och laddas ned enbart i form av en datastruktur vilket innebär att den överförda informationsmängden och likaså nedladdningtiden minskar drastiskt. Nedladdningen till mobiltelefonen sker i form av ett eller flera så kallade SMS-datadownloads som beskrivs ytterligare i Kapitel 2.5.

Telia tillhandahåller något som kallas Unified Services (US), se kapitel 2.6, till vissa kundgrupper. US består av ett antal tjänster som en användare kan utnyttja från sin mobiltelefon. Användaren kan personalisera de tjänster han/hon är intresserad av och ge dem egna namn. Tjänsterna är av de mest skilda slag och exempelvis kan man få information om tåg och flygtider, börskurser, vilka filmer som går på biograferna och få reda på om man vunnit något på sin Bingolott. För att använda en tjänst skriver användaren sitt namn på

tjänsten, och i vissa fall parametrarna som behövs, i ett SMS-meddelande och skickar detta till en speciell US-server i Telias nät. Denna server tolkar SMS-meddelandet, hämtar informationen på Internet och skickar sedan svaret i ett vanligt SMS-meddelande tillbaka till användaren. En användare bestämmer själv vilka av de tillgängliga tjänsterna han/hon vill kunna använda och hur de skall vara konfigurerade, dvs. vilka parametrar som skall behöva anges i mobiltelefonen och vilka som skall vara fördefinierade. En användare kanske ofta reser från Karlstad till Stockholm och anser att inga andra linjer är intressanta att kunna få information om. Användaren kan då konfigurera denna tjänst så att namnet på städerna inte behöver anges varje gång tjänsten nyttjas. Istället anges nu endast tjänstenamnet, tid och datum i SMS-meddelandet.

1.2 Syfte

För att använda en tjänst behöver alltså användaren själv komponera ett SMS-meddelande som kan tolkas av US-servern. Detta kan bli ganska krångligt för användaren om han/hon har många tjänster med många olika parametrar. Målet är att användaren skall kunna välja och personalisera sina tjänster, ordna dem i en trädstruktur i ett enkelt webbaserat gränssnitt och därefter kunna överföra dem till mobiltelefonen. I mobiltelefonen skall motsvarande trädstruktur presenteras som en påbyggnad till telefonens befintliga menysystem. Trädstrukturen finns kvar i telefonen i sin ursprungliga form ända tills en ny trädstruktur byggs och skickas. Genom bläddring i menysystemet kan användaren välja önskad tjänst och mobiltelefonen skall då fråga efter nödvändiga parametrar. När alla parametrar fått värden skall sedan ett SMS-meddelande komponeras och skickas iväg automatiskt. Detta innebär att användaren inte behöver komma ihåg tjänsternas namn och parametrar och inte heller komponera SMS-meddelandet.

1.3 Avgränsning

Vi har delat in arbetet i tre delar.

- Ta fram ett sätt att representera trädstrukturen så att denna kan skickas till en mobiltelefon som en byteträng med hjälp av ett eller flera SMS-datadownloads.

- Bygga en SIMAT-applikation, här kallad tjänsteinterpretator, som tar emot användarens trädstruktur och presenterar denna som en meny i telefonen. Applikationen skall då användaren valt en tjänst ta in nödvändiga parametrar och komponera och sända ett SMS-meddelande till rätt server i Telias nät.

- Ta fram ett enkelt webbaserat gränssnitt som låter användaren ordna sina tjänster i en trädstruktur. Denna struktur skall sedan kunna sändas till användarens mobiltelefon på ett, för användaren, enkelt sätt.

Vårt arbete skall resultera i en funktionsduglig prototyp på systemet. Detta innebär att ovanstående delar skall designas och implementeras med tyngdpunkt på funktionalitet men med lägre krav på delarnas korrekthet och kommersiella användbarhet.

1.4 Dokumentöversikt

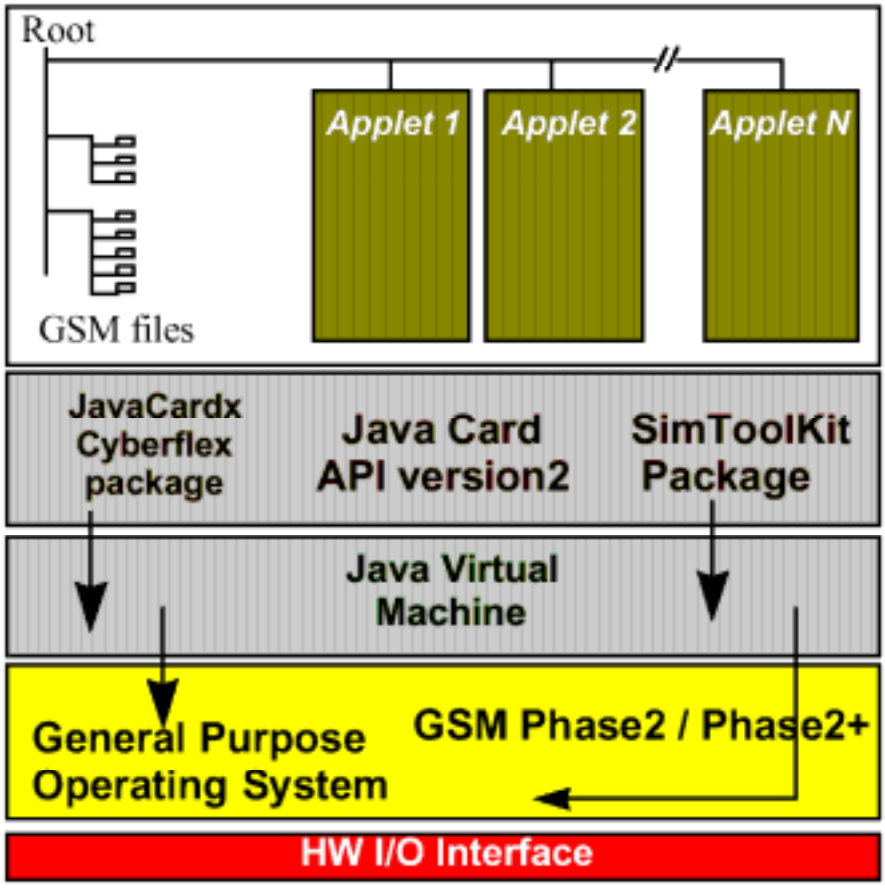
Denna rapport beskriver arbetet med att ta fram delarna beskrivna i kapitel 1.3 och är uppbyggd på följande sätt: Kapitel 1 ger en kort inledande beskrivning av problemet och syftet med vårt arbete. I Kapitel 2 beskrivs de utvecklingsmiljöer vi arbetat i samt de befintliga system som vårt system samverkar med. Vi ger här även en kort beskrivning av SIM-kortet och SIM Application Toolkit. Kapitel 3 beskriver datastrukturens uppbyggnad och funktion. Tjänsteinterpretatorn beskrivs i Kapitel 4. Vi går där igenom dess uppbyggnad och funktion. Det grafiska gränssnittet som används för att kunna bygga en menystruktur online beskrivs i Kapitel 5. Kapitel 6 går igenom ett fullständigt användarexempel från menybyggande till nyttjande av tjänster. En slutsats, där arbetets resultat beskrivs, finns i Kapitel 7. En lista på de förkortningar som används i denna rapport finns i appendix B.

2 Beskrivning av miljön

Eftersom vårt arbete täcker hela kedjan från web-gränssnitt till mobiltelefon-menyer har vi arbetat i flertalet utvecklingsmiljöer och de applikationer vi tagit fram exekverar också i flera olika miljöer. Vårt system samverkar även med flera redan befintliga system och nedan beskriver vi dessa system tillsammans med de olika miljöer vi stött på under arbetets gång.

2.1 SIM-kortet, dess operativsystem och Java Virtual Machine

De SIM-kort vi jobbat med är tillverkade av företaget Schlumberger och har en minneskapacitet på 16 Kb. GSM-delens telefonbok och SMS-minne tar dock upp en hel del utrymme. När utrymmet för dessa delar räknats bort finns endast 7576 bytes kvar för vår applikation, dess arbetsminne och dess filer. Kortets struktur visas i Figur 2.1:



Figur 2.1 SIM-kortets struktur, från Cyberflex Simera Programmers guide v 2.5

Arbetsminnet på kortet är lite speciellt då det inte rör sig om flyktigt RAM utan EEPROM och därför inte rensas då strömmen till kortet stängs av. Det finns heller inget sätt deallokera använt minne och därför måste all minnesallokering ske en gång och därefter aldrig mer. Detta gäller för allt utrymme utom stacken där operativsystemet ser till att minnet kan återanvändas. Dock är stackens storlek kraftigt begränsad och därför skall, helt emot vanlig programmeringspraxis, lokala variabler och metodanrop användas i så liten omfattning som möjligt.

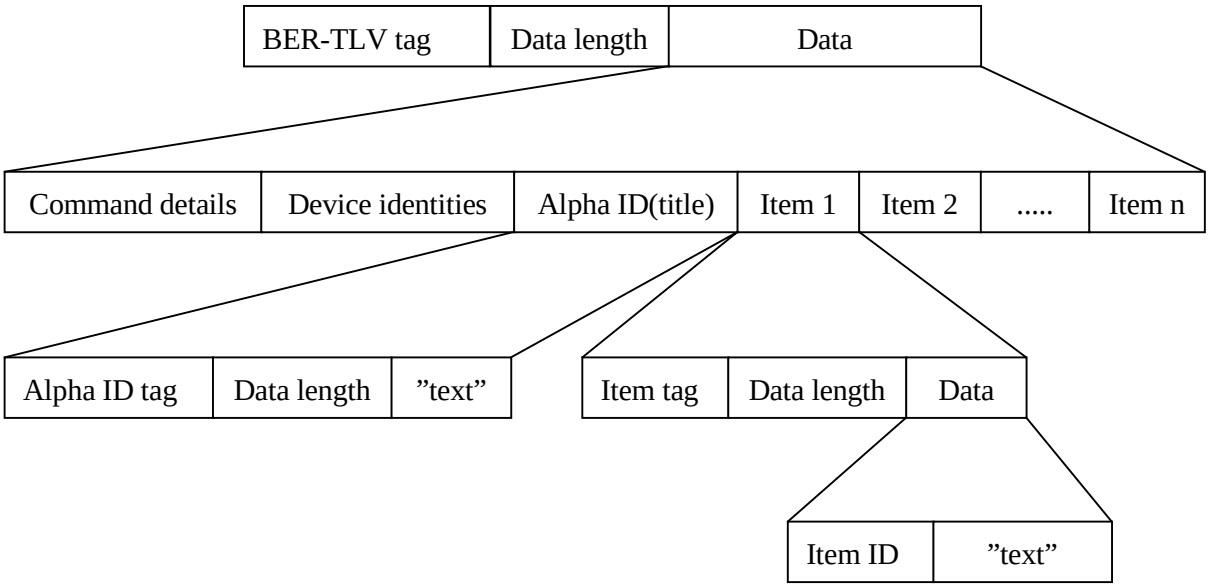
Den virtuella Javamaskin som finns på kortet är också lite begränsad i sin implementation då den till exempel inte stöder alla vanliga primitiva typer utan endast *byte* och *short*. Dessutom kan man ej skapa flerdimensionella arrayer av dessa typer eller av andra objekt. En annan kraftig begränsning är att garbage-collection saknas helt.

2.2 SIMAT – SIM Application Toolkit

Kommunikationen mellan mobiltelefonen och de Java-applikationer som finns på SIM-kortet sker genom ett gränssnitt definierat i SIMAT-standarden [1]. Kommunikationen sker i ett master/slave-förhållande där mobiltelefonen är master och SIM-kortet är slave. Detta innebär att all interaktion sker på initiativ från mobiltelefonen och fungerar på så sätt att telefonen med jämna mellanrum skickar ett STATUS-meddelande till SIM-kortet. Kortet svarar då antingen OK, som betyder att kortet för tillfället är nöjd med sin tillvaro och inte har några meddelanden till telefonen, eller med ett meddelande som talar om att kortet har ett kommando det önskar få exekverat. Kommandot läggs i en, för kortet och telefonen, gemensam kommandobuffert. Telefonen hämtar sedan kommandot från bufferten med kommandot FETCH och exekverar det. Därefter skickar telefonen ett resultat-meddelande till kortet som då i sin tur svarar antingen att ett nytt kommando finns att hämta eller enbart OK.

Alla kommandon är uppbyggda som TLV:er (Tag, Length, Value) och kodas enligt Basic Encoding Rules. Varje kommando består således av en tag, ett längdfält och ett datafält. Datafältet kan i sin tur bestå av ytterligare TLV:er. Exempel på kommandon som specificerats i SIMAT är: Select Item, Get Inkey, Send Short Message, Set Up Call, Play Tone m.fl. För att ytterligare förklara hur ett kommando är uppbyggt visar vi i Figur 2.2 ett exempel på kommandot Select Item. Kommandot ställer användaren inför en valsituation genom att visa en meny med flera val i telefonen. När telefonen visat menyn väntar den på att användaren

skall göra sitt val och sedan svarar telefonen till kortet vilket val som gjorts. Vi förklarar inte vad varje fält står för utan meningen är att visa den principiella uppbyggnaden av ett SIMAT-kommando.



Figur 2.2 Exempel på uppbyggnad av ett SIMAT-kommando

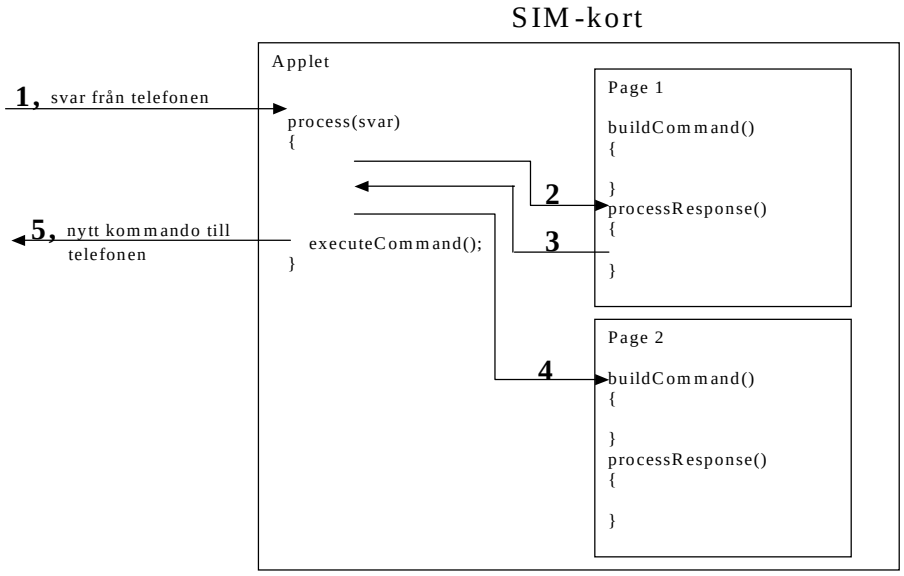
Vi har för enkelhetens skull utelämnat detaljerna i fälten Command Details och Device Identities. Dessa fält innehåller också TLV:er och beskriver vilken typ av kommando som följer och kommandots källadress och destination. I det här fallet är källadressen SIM-kortet och destinationen mobiltelefonen. Som svar när detta kommando utförts skickar telefonen ett Terminal Response-meddelande till SIM-kortet som också det är uppbyggt av TLV:er. I en av dessa TLV:ers datafält finns det Item ID som identifierar valet användaren gjort.

2.3 Java-API

Det API vi använt bygger ut det från Sun utgivna javacard API [2] för utveckling av smartcard-applikationer i Java. Utbyggnaden består i klasser och metoder anpassade för användning i SIMAT-miljö samt för filhantering och dylikt i Schlumbergers operativsystem [3]. Det finns klasser och metoder för att skapa och exekvera de flesta av de i SIMAT specificerade kommandona på ett relativt intuitivt sätt. Grundklassen är klassen Page som

används för att bygga SIMAT-kommandon. Genom att låta egna klasser ärva från klassen Page kan man skapa olika klasser som bygger olika SIMAT-kommandon. Klassen Page har två viktiga metoder: buildCommand och processResponse. BuildCommand-metoden körs för att skapa aktuellt kommando och lägga det på den buffert som telefonen hämtar kommandon från. ProcessResponse anropas då det kommando som byggdes i buildCommand har utförts och telefonen har genererat ett svar på det. Här kan alltså ytterligare bearbetning av resultatet ske, t. ex. kontrollera vilket val en användare gjort så att nästa kommando kan skapas och exekveras. Huvudklassen i en applikation ärver av klassen Applet och här instansieras de av Page ärvda klasserna. Klassen Applet har en viktig metod, process, som anropas då telefonen har lagt någon form av meddelande eller svar på ett SIMAT-kommando på den gemensamma kommandobufferten. Nedan beskrivs de steg som gås igenom då telefonen exekverat ett SIMAT-kommando och väntar på att få exekvera ett nytt (se även Figur 2.3).

1. Appleten får ett svar, från telefonen, på ett utfört SIMAT-kommando genom att appletens process-metod anropas med svaret som inparameter.
2. Appletklassen kontrollerar vilken instans, av de av Page ärvda klasserna, som byggt det just exekverade kommandot och anropar dess processResponse-metod.
3. I processResponse-metoden behandlas svaret och utifrån detta bestäms vad som skall göras härnäst genom att den instans som önskas vara aktiv returneras.
4. Appleten anropar den aktiva instansens buildCommand-metod som bygger det nya kommandot.
5. Appleten kör metoden executeCommand som meddelar telefonen att ett nytt kommando finns att hämta på den gemensamma kommandobufferten.
6. Telefonen exekverar det önskade kommandot, dess svar skickas till appleten och vi är tillbaka på punkt 1.



Figur 2.3 De olika stegen vid exekvering av ett SIMAT-kommando.

2.4 Applet Manager

Applet manager [3] är den programvara vi använder för att ladda ner vår applikation på SIM-kortet. Programvaran kommer liksom SIM-kortet från Schlumberger och är hårt knuten till deras system. Kortet stoppas i en seriellt ansluten kortläsare och därefter används Applet Manager för att administrera filer och applets på kortet. Möjlighet finns att skicka alla former av SIMAT-kommandon till SIM-kortet och logga dess svar. Man kan även spara sekvenser av kommandon i scripts så att man kan återanvända dessa senare. De filer vi har på kortet är skapade genom att vi skrivit ett script som skapar filerna åt oss.

Programmet tillhandahåller också en mobiltelefonsimulator så att man kan testa sina applets utan att stoppa i kortet i en mobiltelefon. Denna simulator har dock visat sig lite bristfällig då det gäller vissa funktioner så vi har oftast valt att testa i en riktig mobiltelefon istället. Då vi testat på detta sätt har vi använt en hård-/mjukvarulösning från Aspects Software [4] som loggar trafiken mellan SIM-kortet och mobiltelefonen. Denna typ av loggning har varit vårt enda sätt att debugga vår applikation då vi inte lyckats använda debuggingmöjligheterna i Symantec Visual Café tillsammans med Applet Manager.

2.5 SMS-datadownload

För att kunna överföra data till SIM-kortet i en mobiltelefon kan man använda sig av något som kallas SMS-datadownload. Skillnaden mellan ett SMS-datadownload och ett vanligt SMS är inte så stor. Egentligen är det bara en flagga i headern till SMS:et som talar om ifall mobiltelefonen skall tolka det som ett vanligt SMS eller som ett datadownload. Om flaggan talar om att det är ett datadownload skickar telefonen SMS:et vidare till SIM-kortet som sparar undan det och kör igång applikationen för vidare behandling. Telefonen visar inte användaren att ett SMS-datadownload mottagits utan det är upp till applikationen att bestämma om användaren skall informeras eller ej. Detta gör att data kan laddas ned till SIM-kortet på ett transparent sätt om så önskas.

Ett SMS-datadownload kan max innehålla 140 stycken 8-bitars tecken men operativsystemet på det SIM-kort vi använder kräver en header på 27 bytes och därför finns alltså endast 113 bytes kvar per SMS-datadownload för data. Headern innehåller information om vilken applikation på SIM-kortet datat är avsett för. Den innehåller även fält för att kunna sekvensnumrera och kryptera meddelanden. Vi använder endast de fält i headern som talar om vilken applikation datat är avsett för men tyvärr är headerns storlek fix och därför kan ej det oanvända utrymmet användas för nyttig data.

Under vårt utvecklingsarbete har vi använt en webbaserad applikation som skickar en inskriven hexadecimal sträng till ett önskat mobiltelefonnummer som ett SMS-datadownload. Denna applikation togs fram av Håkan Blomkvist på Telia Mobile AB efter vårt önskemål.

2.6 US - Unified Services

Unified Services (US) [5] är en samling tjänster som Telia tillhandahåller vissa speciella pilotkunder. De för tillfället tillgängliga tjänsterna finns beskrivna på en hemsida och för att aktivera tjänsterna, d.v.s. göra de möjliga att använda, loggar användaren in på sidan. Där lägger han/hon till de önskade tjänsterna till sin egen så kallade tjänsteportfölj och ger dem kortnamn. Användaren har här också möjlighet att personalisera sina valda tjänster. Personaliseringen innebär att användaren har möjlighet att fördefiniera vissa parametrar som då inte behöver anges i mobiltelefonen när SMS-meddelandet skall komponeras och skickas.

När en användare vill nyttja en tjänst komponerar han/hon ett SMS-meddelande innehållande tjänstens kortnamn samt eventuella parametrar som inte har fördefinierats och skickar detta till en speciell server i Telias nät. När servern tar emot SMS-meddelandet exekveras en viss java-servlet och denna utför sedan den önskade tjänsten.

Tjänsterna kan vara i stort sett vad som helst och nya kan utvecklas på väldigt kort tid (tiden det tar att skriva servleten). Exempel på tjänster som finns för tillfället är allt ifrån Börsinformation till att spela en låt på TMUKMs (Telia Mobile Utveckling Kompetensgrupp Mobile Internet lab) jukebox i fikarummet. För att ytterligare förklara hur det hela fungerar följer ett exempel:

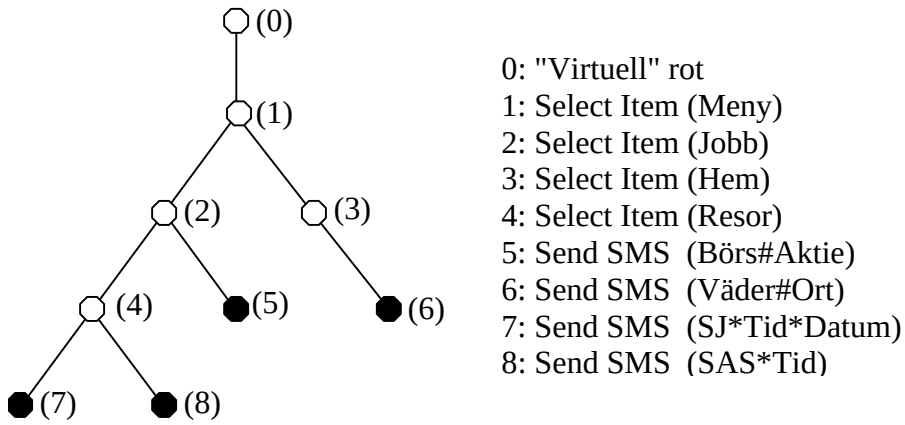
Användare X reser ofta mellan Karlstad och Stockholm och vill därför ha snabb access till tidtabellen för SJs tågresor mellan dessa städer. X går in på US-serverns websida och väljer tjänsten Tågresor. X kallar tjänsten Tåg-KD-STHLM och ger den kortnamnet TKS. Tjänsten Tågresor har 4 parametrar: avreseort, destinationsort, datum och tid. Eftersom X bara är intresserad av Karlstad-Stockholm skriver han in dessa parametrar på sidan och låter datum och tid vara tomma. När sedan X står på stationen i Karlstad den 30 mars 2000 klockan 10 på förmiddagen och undrar när nästa tåg till Stockholm går skickar han ett SMS-meddelande till US-servern med innehållet ”TKS*000330*1000”. US-servern ser att detta SMS-meddelande kommer från X och att X har fördefinierat parametrarna för avreseort och destination till Karlstad respektive Stockholm. En java-servlet startas som går till SJ:s tidtabellsupplysning på Internet och utifrån mottagna och fördefinierade parametrar tar reda på de aktuella tågtiderna. När svaret från SJs databas presenteras plockas den relevanta informationen ut från sidan och US-servern skickar ett eller flera SMS-meddelanden innehållande de efterfrågade tågtiderna tillbaka till användaren X.

3 Datastruktur

När användaren skapat en trädstruktur, i det grafiska gränssnittet, måste denna överföras till telefonen i någon form av datastruktur. Datastrukturens uppbyggnad är av högsta betydelse för vårt arbete då dess effektivitet i praktiken avgör om lösningen blir praktiskt genomförbar. Skulle en enkel meny behöva tiotals SMS-datadownloads för att överföras skulle inte tjänsten bli användarvänlig nog. Därför har vi försökt banta informationsmängden så mycket som möjligt utan att för den skull behöva minska funktionen. Ett SMS-datadownload kan max innehålla 140 stycken 8-bitars tecken men operativsystemet på SIM-kortet kräver en header på 27 bytes (se Kapitel 2.5) och därför finns alltså endast 113 bytes kvar per SMS-datadownload för datastrukturen.

3.1 Menyuppbyggnad

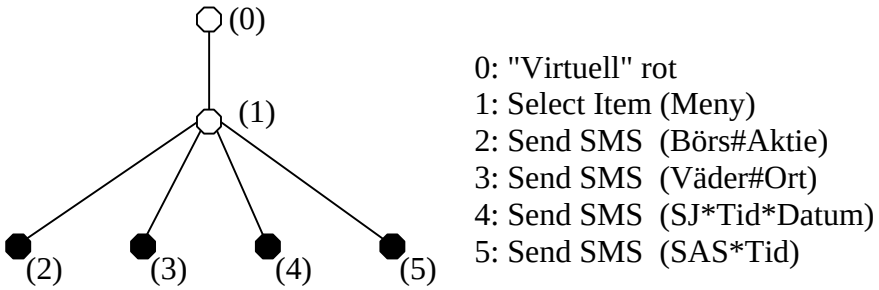
I Figur 3.1 har en användare skapat en enkel menystruktur.



Figur 3.1 Exempel på menystruktur

Vi har valt att låta ett träd bestå av två olika sorters noder, valnoder och tjänstenoder. Valnoderna används för att presentera namnen på de olika val användaren kan göra i noden, d.v.s. namnen på valnodens barn. Antalet barn en valnod kan ha är för tillfället begränsat till tio. Detta är en avvägning gjord med tanke på användarvänlighet och minnesutnyttjande men det maximala antalet kan lätt ändras i källkoden om det skulle lanseras telefoner med större displayer eller SIM-kort med mer minnesutrymme. Möjligheten finns att lägga alla

tjänstenoder under en och samma valnod. Detta resulterar i en plattare menystruktur (se Figur 3.2).



Figur 3.2 Exempel på en platt menystruktur

Det är alltså endast löven i trädet som är tjänstenoder som leder till att ett SMS-meddelande skickas till US-servern. I Figur 3.1 är alltså noderna 7,8,5 och 6 tjänstenoder och de övriga valnoder.

3.2 Kodning av datastrukturen

Den information som behövs för att tjänsteinterpretatorn skall kunna hantera en nod är:

- Valt nodnummer
- Typ av kommando

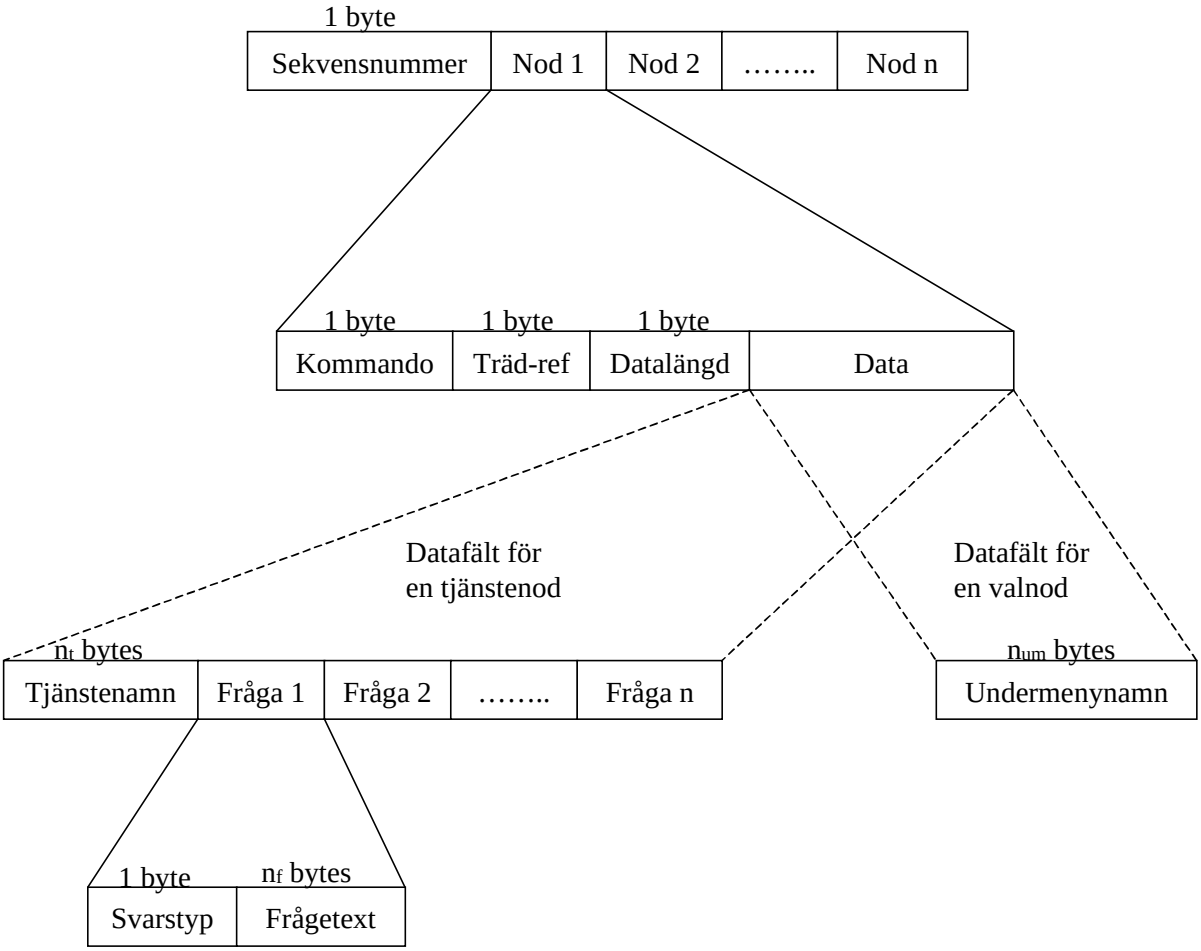
Kommandon kan egentligen vara alla möjliga kommandon som kan exekveras i SIMAT men då syftet med vårt system är att presentera en meny och låta användaren exekvera US-tjänster behöver vi endast använda två olika SIMAT-kommandon. Dessa är SelectItem som presenterar en valsituation och motsvarar valnoderna och SendSMS som skickar ett SMS-meddelande och därmed motsvarar tjänstenoderna. De olika kommandona representeras av varsin byte-kod, 0x01 för SelectItem och 0x02 för SendSMS.

Beroende på vilken typ kommandot är behövs också:

- Information om vilka noder som är den aktuella nodens "barn", gäller valnoder.
- Information om vilka frågor som skall ställas till användaren och av vilken typ svarsvärdet kan vara (endast siffror eller alla tecken i GSM-alfabetet), gäller tjänstenoder.

Efter att ha tittat på ett tidigare förslag på utformning av datastrukturen [6], utarbetat av Telia Mobile AB under 1998, och funderat en del själva beslutade vi oss för följande lösning:

Datastrukturen kodas på följande sätt:



Figur 3.3 Datastrukturens uppbyggnad

Förklaringar på beteckningar:

Sekvensnummer — Ett enkelt sekvensnummer för numrering av SMS-datadownload. Då användarens menystruktur skall beskrivas i en datastruktur kan denna bli större än 113 bytes och den får därmed inte plats i ett SMS-datadownload utan måste delas upp i flera. Sekvensnumrets första fyra bitar talar om hur många delar som skall komma totalt och de sista fyra bitarna talar om vilken del det är. Delningen sker mellan två noder och sekvensnumret ökas för varje meddelande. Detta gör att varje del av datastrukturen följer strukturen ovan.

Kommando	Typ av kommando, SelectItem eller SendSMS.
Träd-ref	Nodnummer på förälder.
Datalängd	Längden, i bytes, på efterföljande data.
Data	Nodens data.

Tjänstenoder:

Tjänstenamn	Det namn som visas i menyn.
Svarstyp	Typ av svar som önskas, numeriskt eller alfanumeriskt. '*' innebär att användaren bara kan använda siffror när svaret skrivs in i mobiltelefonen. '#' innebär att användaren kan använda hela GSM-alfabetet när svaret skrivs in i mobiltelefonen.
Frågetext	Den hjälptext som visas när användaren skall mata in en parameter.

Valnoder:

Undermenynamn	Det namn noden har i menyn.

För att ytterligare visa med vilken information en nod representeras i datastrukturen, visas nedan datastrukturen för menyträdet i Figur 3.1 i tabellform.

(Nod-nummer)	Kommando	Träd-ref	Data-längd	Data
1	1	0	4	Meny
2	1	1	4	Jobb
3	1	1	3	Hem
4	1	2	5	Resor
5	2	2	10	Börs#Aktie
6	2	3	9	Väder#Ort
7	2	4	12	SJ*Tid*Datum
8	2	4	7	SAS*Tid

Tabell 3.1 Datastruktur från Figur 3.1 i tabellform

Då datastrukturen skickas i ett SMS-datadownload skickas den som en enda lång byteträng och nedan visas byteträngen för menyträdet i Figur 3.1.

1,1,0,4,M,e,n,y,1,1,4,J,o,b,b,1,1,3,H,e,m,1,2,5,R,e,s,o,r,2,2,10,B,ö,r,s,#,A,k,t,i,e,2,3,9,V,
ä,d,e,r,#,O,r,t,2,4,12,S,J,*,T,i,d,*,D,a,t,u,m,2,4,7,S,A,S,*,T,i,d
data = 79 bytes (av max 113 per SMS-datadownload)

Överföringen sker i form av hexadecimala värden men för enkelhetens skull har vi valt att visa strängen okodad.

4 Tjänsteinterpretator

Idén med tjänsteinterpretatorn är att användaren inte skall behöva komma ihåg kortnamn och parametrar till sina US-tjänster utan istället kunna välja tjänst i en meny i mobiltelefonen och få frågor angående parametrarna. Detta innebär att då användaren väljer och personaliserar sina tjänster så får han/hon även möjligheten att lägga in dessa under olika grenar i ett menyträd. Då användaren är färdig skickas detta menyträd över till telefonen i form av den tidigare beskrivna datastrukturen. Tjänsteinterpretatorn avkodar meddelandet och lagrar informationen i två filer på SIM-kortet. Dessa filer medför att det aktuella menyträdet finns kvar i telefonen ända tills användaren byggt och skickat ett nytt.

Applikationen läser sedan i dessa filer för att kunna bygga upp menyn efterhand som användaren bläddrar i den. När användaren vill exekvera en tjänst knappar han sig fram till den önskade tjänsten i sin egen meny och väljer tjänsten. Om parametrar behövs får användaren frågor om dessa. När alla frågor besvarats skickas ett SMS-meddelande till US-servern som utför tjänsten. Denna fråga-svar-skicka sekvens blir likartad oavsett vilken tjänst som skall utföras och därför kan samma generella applikation på SIM-kortet användas för att få olika tjänster exekverade på US-servern.

4.1 Beskrivning av funktionalitet

4.1.1 Avkodning av datastruktur

När datastrukturen laddas ner i telefonen (via SMS-datadownload) avkodas den av applikationen och sparas undan i två filer på SIM-kortet. En del läggs i en Recordfil som innehåller information om en nods kommandotyp (selectItem eller sendSMS), d.v.s. om noden är en valnod eller en tjänstenod. Den innehåller även information om nodens trädreferens, längden på datat som tillhör noden samt till sist information om hur stor offseten är till nodens data som ligger lagrat i en binärfil. Binärfilen innehåller alltså enbart alla nodernas datafält (se Figur 4.1). Vi har gjort på detta sätt därför att längden på olika noders data varierar och det blir då svårt att ge varje record en bestämd storlek. Vår recordfil kan innehålla maximalt 47 stycken 5 bytes stora records. Denna begränsning har vi satt på grund

av det begränsade minnesutrymmet på kortet men vi tror att 47 noder skall vara mer än tillräckligt för de flesta användare.

I Figur 4.1 visas filernas utseende då interpretatorn processat datastrukturen från Figur 3.1.

Recordfil

RecordNr	Data
1	01 00 04 00 00
2	01 01 04 00 04
3	01 01 03 00 08
4	01 02 05 00 0B
5	02 02 0A 00 10
6	02 03 09 00 1A
7	02 04 0C 00 23
8	02 04 07 00 2F

Binärfil

Data
MenyJobbHemResor
Börs#AktieVäder#Or
tSJ*Tid*DatumSAS*
Tid

Figur 4.1 Filernas struktur på SIM-kortet

Varje Record representerar en nod, är 5 bytes stor och innehåller följande information:

1 Kommando

2 Träd-referens

3 Datalängd

4-5 Datats offset i binärfilen

4.1.2 Exekvering av olika kommandon

Vår applikation består av tre stycken huvudklasser:

- ServiceInterpreter som är själva appletklassen.
- SelectItem som skapar SIMAT-kommandot selectItem som, när det exekveras i mobiltelefonen, visar en valsituation för användaren.
- PrepareSMS som ställer frågor till användaren, med hjälp av SIMAT-kommandot getInput, och sedan skapar SIMAT-kommandot sendSMS som skickar ett SMS-meddelande till US-servern.

I klassen ServiceInterpreter finns en instans av SelectItem och en av PrepareSMS. Instanserna kallas selectItemPage respektive prepareSMSPage och dessa återanvänds gång på

gång då vi inte kan skapa nya objekt under körning på grund av de i Kapitel 2.1 nämnda minnesproblemen. Klasserna selectItem och PrepareSMS ärver från klassen Page och i ServiceInterpreter finns en variabel, currentPage, som definierar vilken av de två instanserna som är aktiv för tillfället.

I ServiceInterpreter finns också en metod som heter process. Denna metod anropas så fort telefonen har något åt SIM-kortet, till exempel ett svar på ett SIMAT-kommando eller ett SMS-datadownload. I vårt fall gör inte process-metoden så mycket mer än att ta emot SMS-datadownload innehållande datastrukturen. Då ser process-metoden till att datastrukturen avkodas och sparas undan på rätt sätt. Om däremot ett svar på ett SIMAT-kommando kommer till process-metoden anropar den i sin tur en metod som kallas processResponse i det aktiva objektet. Denna metod får då ta hand om telefonens svar. I klasserna SelectItem och PrepareSMS är det de två metoderna buildCommand och processResponse som är det viktigaste och dessa förklaras därför nedan för respektive klass.

4.1.2.1 SelectItem

När buildCommand anropas i SelectItem kontrolleras först nodnumret för den nod användaren valt och denna nods information läses ut ur filerna. Sedan gås alla, i datastrukturen, efterföljande noder igenom för att se om någon nod har den valda noden som förälder (detta ses på Trädreferensen). Från de noder som hittats extraheras den beskrivande texten (datafältet exklusive eventuella frågor) och denna läggs i en lista av menyval som skall presenteras för användaren. Den nuvarande nodens namn sparas som en så kallad Alpha-Identifier, denna blir sedan menyns rubrik, för det kommande SIMAT-kommandot. När all information skaffats läggs sedan SIMAT-kommandot SelectItem, med rätt Alpha-Identifer och lista över aktuella val, på kommandobufferten.

När processResponse anropas i SelectItem kontrolleras vilket svar som fåtts från telefonen, d.v.s. vilket nodnummer användaren valt. Den valda noden läses ut ur recordfilen och beroende på vilken typ den är sätts currentPage-variabeln i ServiceInterpreter till antingen selectItemPage eller prepareSMSPage.

4.1.2.2 PrepareSMS

BuildCommand-metoden för PrepareSMS ska se till att alla frågor som noden har presenteras för användaren en i taget och att svaren på dessa frågor sparas undan så att de finns

tillgängliga när den till slut ska bygga och skicka SMS:et till US-servern. Eftersom exekveringen av buildCommand-metoden avbryts varje gång en fråga ställs, beroende på att en fråga är ett SIMAT-kommando i sig, måste metoden, varje gång den körs, kontrollera hur många av nodens alla frågor som redan har blivit besvarade och utifrån detta antingen konstruera en ny fråga eller om alla frågor redan är besvarade, bygga och skicka SMS:et. När SMS:et skickats iväg måste alla variabler som håller räkning på antalet frågor respektive antalet besvarade frågor nollställas så att samma instans av klassen PrepareSMS kan användas igen.

När processResponse anropas i prepareSMS kontrolleras om det fortfarande finns obesvarade frågor kvar. Om det finns det sätts ServiceInterpreter.currentPage till prepareSMSPage, så att nästa fråga kan ställas. Om alla frågor redan besvarats är tjänsten exekverad och då sätts currentPage i ServiceInterpreter till selectItemPage så att den tidigare valsituationen kan visas igen.

5 Grafiskt gränssnitt

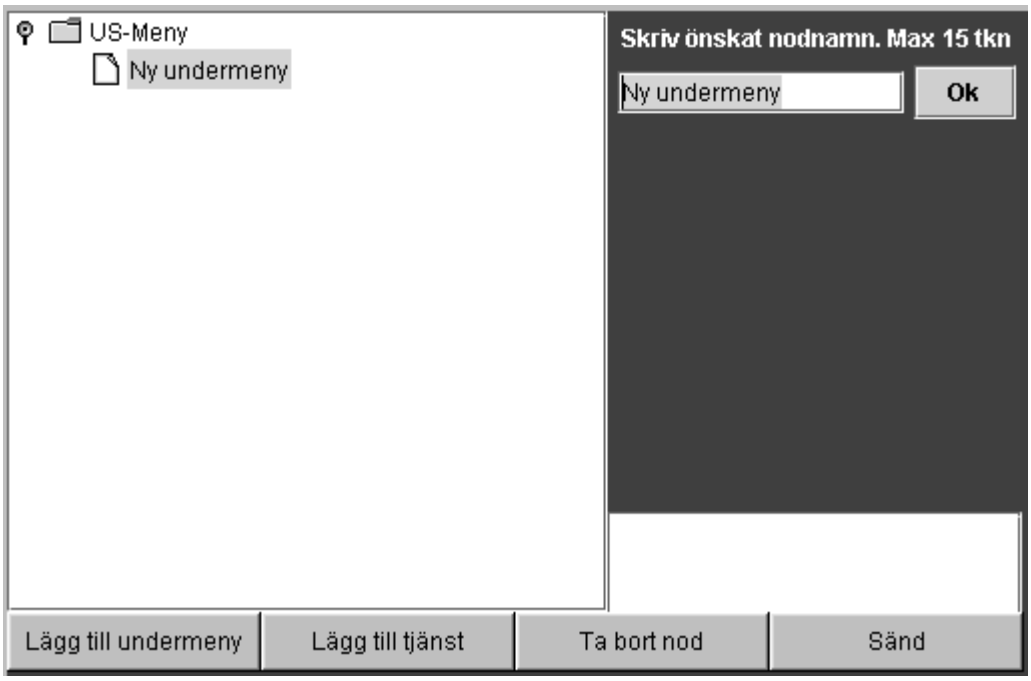
För att systemet skall bli tillräckligt användarvänligt krävs att en användaren kan bygga och sända sin egen menystruktur på ett enkelt och intuitivt sätt.

Efter att en användare loggat in på US och personaliserat sina tjänster finns möjligheten att komma till en webbsida som erbjuder ett grafiskt gränssnitt för att bygga en menystruktur som sedan kan skickas till telefonen. Meningen med detta gränssnitt är att det ska göra det enkelt för en användare att bygga och sända menystrukturen.

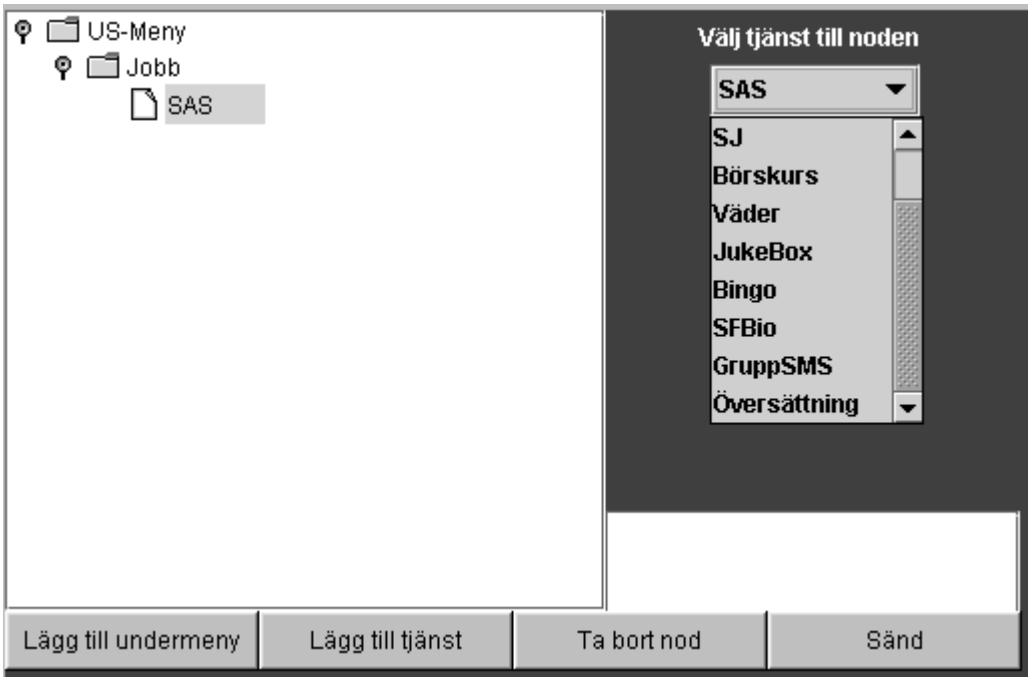
5.1 Beskrivning av funktionalitet

Gränssnittet består av en Java-applet som startas från användarens US-sida. Appleten får som inparametrar användarens mobiltelefonnummer och en lista med de tjänster som användaren har i sin tjänsteportfölj. Om en användare har skapat en menystruktur tidigare får appleten även denna, i form av den i Kapitel 3 beskrivna datastrukturen. Denna används för att bygga menystrukturen i gränssnittet så att användaren kan utgå från sin gamla menystruktur vid skapandet av en ny.

Om en användare ej har skapat någon menystruktur tidigare visas endast trädets rot. Användaren har nu möjlighet att lägga till undermenyer (se Figur 5.1) och tjänster (se Figur 5.2) i det tomma trädet. Undermenyerna kan namnges efter eget önskemål medan namnen på tjänsterna bestäms av de kortnamn användaren givit tjänsterna vid personaliseringen.



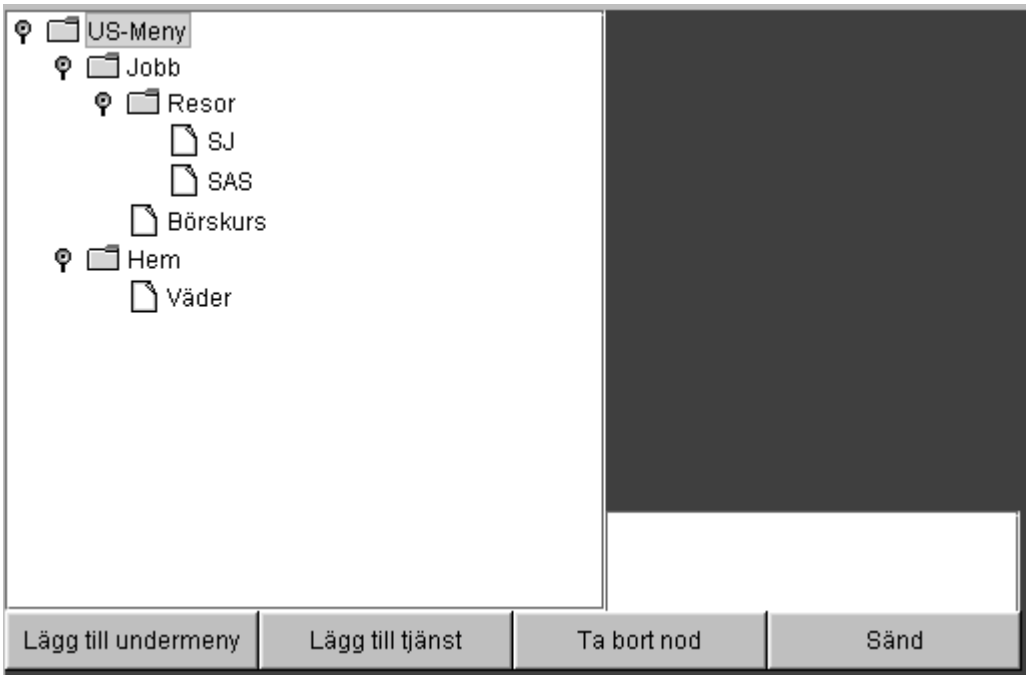
Figur 5.1 Namngivning av undermeny



Figur 5.2 Val av tjänst

När användaren är nöjd med sitt träd trycker han/hon på sänd-knappen. Då skapas datastrukturen för det byggda trädet och denna skickas vidare till en servlet som dels sparar den i US-databasen och dels skickar den till användarens mobiltelefon. Om datastrukturen skulle vara större än ett SMS-datadownload delas den upp i flera delar innan den skickas.

I figuren nedan har en användare byggt upp sin meny enligt exemplet i Figur 3.1.



Figur 5.3 Menystrukturen från figur 3.1

Om trädet som användaren byggt innehåller fler än det maximalt tillåtna antalet noder får användaren ett meddelande om detta. Detsamma gäller om trädet har tomma undermenyer d.v.s. undermenyer som inte har vare sig andra undermenyer eller tjänster under sig. När datastrukturen till slut skickas iväg vet vi att den är korrekt och att den uppfyller de av tjänsteinterpretatorn ställda krav på storlek och uppbyggnad.

5.2 Samverkan mellan det grafiska gränssnittet och US-servern

Det är en hel kedja av applikationer och system som det grafiska gränssnittet måste samverka med. Nedan beskrivs kortfattat hur det hela fungerar.

Allt startar när en användare loggat in på US för att personalisera sina tjänster och därefter beslutat sig för att bygga en menystruktur. Då trycker han/hon på en knapp på US-sidan som kör igång en Java-servlet, USMenuTool, som får användarens userid med som inparameter. USMenuTool hämtar med hjälp av userid användarens mobiltelefonnummer, befintliga datastruktur och en lista på de tjänster användaren har i sin tjänsteportfölj ur US-databasen. USMenuTool anropar därefter ytterligare en servlet, StartMenuTool, och skickar med de parametrar som just lästs in från US-servern. StartMenuTool kör i sin tur igång Java-appleten,

MenuBuilder, med det grafiska gränssnittet. MenuBuilder får även den inparametrarna mobiltelefonnummer, datastruktur och tjänstelista. Här kan man fundera över varför inte USMenuTool själv startar MenuBuilder utan istället anropar StartMenuTool som sedan kör igång den. StartMenuTool verkar ju inte göra någonting av nytta. Detta är helt sant och förklaringen är att vi under utvecklingen av MenuBuilder skapade StartMenuTool för att kunna testköra mot appleten. StartMenuTool innehåller speciell HTML-kod som gör det möjligt att köra MenuBuilder i en browser som inte stöder javas grafik-API swing [7]. För enkelhetens skull lät vi då USMenuTool, som tagits fram av Håkan Blomkvist på Telia Mobile AB, anropa vår testservlet. Utvecklingsarbetet kunde då ske på två fronter samtidigt.

När användaren trycker på sändknappen i MenuBuilder för att skicka sin nya menystruktur till telefonen startar appleten ännu en Java-servlet, SendAndStoreMenu. Denna servlet tar emot mobiltelefonnumret och den nya datastrukturen som inparametrar. Datastrukturen sparas undan i US-databasen och körs därefter in i en splittermetod. Splittermetoden analyserar datastrukturen och delar upp den i flera delar, om så är nödvändigt, samt lägger till nödvändig headerinformation (se Kapitel 2.2) och sekvensnummer till varje del. Dessa delar skickas sedan en och en tillsammans med mobiltelefonnumret till den sista Java-servleten, SMSDatadownload, som skickar iväg dem till telefonen.

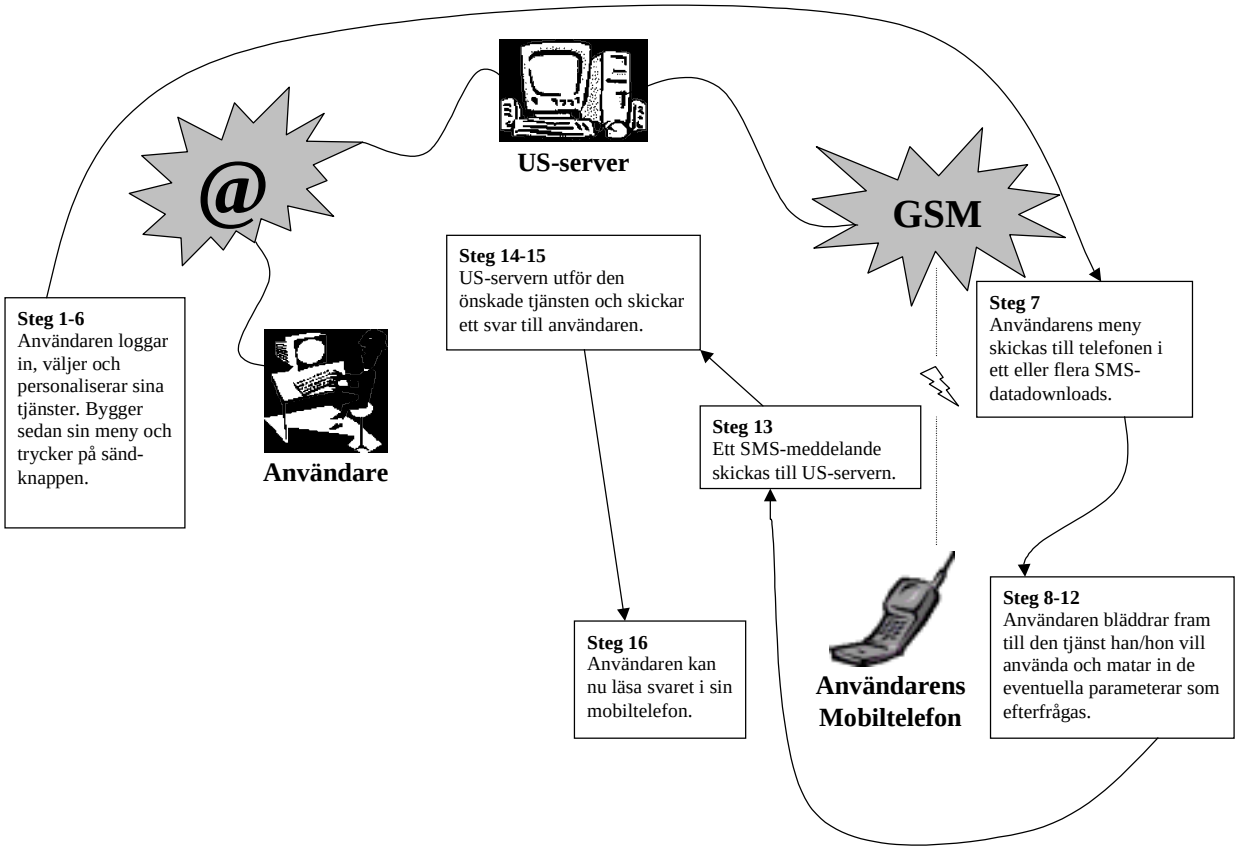
Även SendAndStoreMenu och SMSDatadownload servletarna har tagits fram av Håkan Blomkvist på Telia Mobile AB. Splittermetoden som används i SendAndStoreMenu har vi dock skrivit själva.

6 Fullständigt användarexempel

En användare som vill kunna komma åt och använda Telias US-tjänster måste först skaffa ett konto på Telias US-server, detta kan dock inte en vanlig konsument göra i dagslägetutan det är förbehållet vissa pilottestare. Om sedan användaren skaffar ett speciellt SIM-kort med vår applikation på kan han/hon använda vårt system till fullo. Ett användarexempel som visar hela kedjan av händelser beskrivs stegvis nedan, se även Figur 6.1.

1. En användare loggar in på US-sidan.
2. Bland de US-tjänster som finns väljer användaren ut några som han/hon vill kunna använda och lägger dessa i sin tjänsteportfölj.
3. Användaren har nu möjlighet att personalisera, d.v.s. fördefiniera vissa parametrar, sina tjänster i tjänsteportföljen om han/hon så önskar.
4. När användaren är klar med sin personalisering kan han/hon trycka på en knapp i US-fönstret som kör igång det grafiska gränssnittet.
5. Användaren bygger sin menystruktur.
6. När användaren är nöjd med sin menystruktur och vill få den överförd till sin mobiltelefon trycker han/hon på sänd-knappen i det grafiska gränssnittet.
7. Datastrukturen för menystrukturen skickas till telefonen med hjälp av SMS-datadownload.
8. Telefonen tar emot SMS:en, ser att det är datadownloads, och vidarebefordrar dem till applikationen på SIM-kortet.
9. Applikationen, d.v.s. tjänsteinterpretatorn, avkodar datastrukturen och sparar undan den i två filer.
10. Användaren får ett meddelande i sin mobiltelefondisplay som säger ”Meny mottagen”.
11. Användaren kan nu under menyn ”Telia US” i sin telefon bläddra mellan sina tjänster.
12. Då användaren väljer en av tjänsterna får han/hon frågor om de parametrar som behövs, han/hon svarar på dessa och när den sista frågan besvarats komponeras ett SMS-meddelande.
13. Det komponerade SMS-meddelandet skickas automatiskt US-servern.
14. US-servern tar emot detta meddelande, kontrollerar avsändarnummer, tjänstnamn och parametrar och utför sedan den önskade tjänsten.

15. När tjänsten utförts komponerar US-servern ett eller flera svarsmeddelanden som skickas till användarens mobiltelefon som vanliga SMS-meddelanden.
16. Användaren kan nu läsa dessa SMS-meddelanden genom mobiltelefonens ordinarie funktioner.
17. Om användaren känner att han/hon vill uppdatera sin menystruktur är det bara att logga in på US-sidan och börja om från steg 1. Användaren kan utgå från sin gamla meny vid skapandet av en ny. När den nya menyn skickas till telefonen ersätts den gamla automatiskt.



Figur 6.1 Fullständigt användarexempel

7 Slutsats

Vårt arbetet har resulterat i ett system som uppnått de mål, och som uppfyller de krav, som ställdes på det i projektets början. Systemet skall dock ses som en prototyp och kan ej anses vara helt fritt från felaktigheter.

En sak vi kommit fram till är att exekveringsmiljön på SIM-kortet ej verkar vara avsedd för att hantera såpass dynamiska och interaktiva applikationer som vår. Detta då vi varit tvungna att ta till extrema åtgärder som till exempel egen minneshantering för att få det hela att fungera. Vi trodde länge att arbetsminnet på kortet rensades då användaren avslutade applikationen eller åtminstone då användaren stängde av mobiltelefonen. En kvarleva från detta är att vi använder oss av filer i relativt stor utsträckning. Det är ju egentligen inte nödvändigt då en global variabel, i vårt fall, är minst lika beständig som en fil. Problemen med minneshanteringen uppdagades dock inte förrän i slutskedet av projektet då vår applikation var i stort sett färdig och såpass stor att minneskapaciteten började tryta vid längre testkörningar. Funderingar har funnits på att eliminera filhanteringen, och därmed eventuellt öka prestandan, men filer har visat sig vara värdefulla i andra avseenden. En recordfil är till exempel ett bra substitut till flerdimensionella arrayer som ju ej stöds i Java-API:n. Kontentan av detta blev att vi beslöt oss för att behålla filhanteringen för att slippa göra stora tidskrävande förändringar i koden, trots ett litet avkall på prestandan som följd.

Ett annat problem vi haft under utvecklingsarbetet beror på det faktum att våra relativt processorkrävande operationer gör att SIM-kortet inte alltid hinner skicka den typ av keep-alive signaler som det skall skicka till telefonen för att bekräfta att kortet fortfarande fungerar. Detta ledde till konstiga fel som var svåra att spåra. Problemet har vi löst genom att låta SIM-kortets operativsystem då och då, relativt ofta faktiskt, få lite tid till detta genom att vi kör metoden APDU.waitExtension. Denna metod är därför relativt ofta använd i programkoden och det enda den gör är egentligen att lämna över kontrollen till operativsystemet en kort stund.

Projektet har följt tidsplanen och flutit utan administrativa problem. Då vi haft problem av mer teknisk karaktär har kunnig handledning alltid funnits till hands både från Universitet och Telia Mobile UKM i Karlstad.

Vi vill härmed också passa på att tacka våra handledare på Universitet och TMUKM samt övrig personal på TMUKM för ett trevligt bemötande och sakkunnig handledning.

8 Referenser

[1] ETSI (European Telecommunications Standard Institute), Digital cellular telecommunications system (Phase 2+); Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM-ME) Interface (GSM 11.14 version 8.0.1 Release 1999), ETSI, 1999-07

[2] Sun, Javacard 2.0 Specification, http://java.sun.com/products/javacard

[3] Schlumberger, Cyberflex Simera Development Kit, Schlumberger, 1998

[4] Aspect Software, http://www.aspect-software.com, 2000

[5] Annette Andersson, Konceptet US, Telia Mobile AB 1/0062-FCPA605185, 1999-01-22

[6] Stellan Emilsson, Nerladdning av en SIMAT-baserad tjänst via SMS till en mobiltelefon, Telia Mobile AB, 1998-09-11

[7] Sun, Java 2 SDK Standard Edition, http://java.sun.com/products/jdk/1.2/index.html

A Programkod

Detta appendix innehåller programkoden för vårt system, indelat i två huvudblock. Ett för tjänsteinterpretatorn, som körs i mobiltelefonen, och ett för appleten som utgör det grafiska gränsnittet.

A.1 Programkod för tjänsteinterpretatorn

A.1.1 ServiceInterpreter.java

```
/*******************************************
*  (c) Copyright Telia Mobile AB, created 2000 *
*      All rights reserved                     *
*******************************************/

import javacard.framework.*;
import javacardx.framework.*;
import sim.toolkit.*;

/** The main class that executes proActive Commands, forwards Terminal responses
    and handles the menu downloads.*/
public class ServiceInterpreter extends Applet
                               implements ST, Constant, MenuConstants,
                                          GSMCharacter
{
    /** The page used for submenu-nodes.*/
    public static SelectItem selectItemPage = new SelectItem();

    /** The page used for service-nodes.*/
    public static PrepareSMS prepareSMSPage = new PrepareSMS();

    /** Used to keep track of which page that is active.*/
    public static Page currentPage;

    /** Used to store information about the current node.*/
    byte[] currentRecord = new byte[RECORD_SIZE];

    /** Used to store information about the next node.*/
    byte[] newRecord = new byte[RECORD_SIZE];

    /** Used to store control information, the first byte represents
        the total number of nodes and the next two bytes represents
        the length of the recordfile.*/
    public static byte[] controlArray = new byte[3];

    /** Used to keep track of which node that is selected.*/
    public static byte selectedItem;

    /** Used to tell whether the menu has been initialized or not.*/
    boolean pagesCreated = false;

    /** Used to tell whether the menu is valid or not.*/
    boolean menuOK = false;

    /** The text presented when entire menu is downloaded.*/
    static byte[] downloadText = {M,e,n,y,SP,m,o,t,t,a,g,e,n}; //NOTE not final!!!
```

```
/** The page used to display the downloadText.*/
public static Page downLoadPage = new DisplayText(downloadText);

/** Not used.*/
ProactiveElement[] customElement = {};

/** Temporary buffer for SMS-datadownload.*/
static byte[] readBuffer  = new byte[114];

/** Temporary record for SMS-datadownload.*/
static byte[] recordData = new byte[5];

/** Registers the applet in the card´s registerfile.*/
public ServiceInterpreter()
{
    register();
}

/** Processes the messages from ME, In case of a MENU_SELECTION
    it shows the menu (if it exists). In case of a SMS-datadownload
    it stores it in files. In case of terminal Response it just calls
    the active page´s processResponse.*/
public void process(APDU apdu)
{
    switch(apdu.getBuffer()[ISO.OFFSET_INS])
    {
        case APDU_ENVELOPE:

            if(apdu.getBuffer()[ISO.OFFSET_CDATA] == MENU_SELECTION)
            {
                if(menuOK)
                {
                    if(!pagesCreated)
                    {
                        FileSystem.selectFile(MASTER_FILE_ID);
                        APDU.waitExtension();
                        FileSystem.selectFile(DIRECTORY_FILE_ID);
                        FileSystem.selectFile(CONTROL_FILE_ID);
                        APDU.waitExtension();
                        CyberflexOS.readBinaryFile(controlArray, NO_OFFSET,
                                                   NO_OFFSET, (short)3);
                        APDU.waitExtension();

                        pagesCreated = true;
                    }

                    selectedItem = apdu.getBuffer()[ITEM_ID_POSITION];
                    APDU.waitExtension();
                    FileSystem.selectFile(DIRECTORY_FILE_ID);
                    FileSystem.selectFile(RECORD_FILE_ID);
                    CyberflexOS.readRecord(currentRecord, NO_OFFSET,
                                           selectedItem, ABSOLUTE,
                                           RECORD_SIZE);

                    switch(currentRecord[INFOTYPE])
                    {
                        case IT_SELECT_ITEM:

                            currentPage = selectItemPage;
                            break;

                        case IT_SEND_SMS:
                            break;
                    }

                    APDU.waitExtension();

                    if(currentPage.buildCommand() == ST_SUCCESS)
```

```
            {
                ProactiveCommand.executeCommand();
            }
        }
    }

    if(apdu.getBuffer()[ISO.OFFSET_CDATA] == SMS_PP_DOWNLOAD)
    {
        //Getting TPDU offset
        short offset = ProactiveServices.getTPDUSmsDownload();
        //Moving to TP_OA length field
        offset++;
        //Getting TP_OA length
        short TP_OA_Length = (short)apdu.getBuffer()[offset];
        //Skipping TON/NPI field
        offset++;
        //Skipping TP_OA field
        offset = (short)(offset + (TP_OA_Length+1)/2);
        //Skipping: TP-PID(1),TP-DCS(1),TP-SCTS(7),TP-UDL(1),UDHL(1),
        //IEI(1),IEIDL(1)
        offset = (short)(offset + 13);
        //Moving to CPL byte 1
        offset++;
        APDU.waitExtension();
        //Getting Command Packet Length(1st byte)
        byte CPL1 = apdu.getBuffer()[offset];
        //Moving to CPL-second byte
        offset++;
        //Getting Command Packet Length(2nd byte)
        byte CPL2 = apdu.getBuffer()[offset];
        //Calculating CPL avoiding twocomplements
        short CPL = (short)(CPL2 & 0xFF);
        //Moving to Command Header Length
        offset++;
        //Getting Command Header Length
        short CHL = apdu.getBuffer()[offset];
        //Calculating secured data length
        short dataLength = (short)(CPL - CHL - 3);
        //Skipping SL(1), RL(1), KIC(1), KID(1), TAR(3), CNTR(5),
        //PCNTR(1), CER(8)
        offset = (short)(offset + 21);
        //Moving to beginning of secured data
        offset++;

        for(short i=0;i<dataLength;i++)
        {
            APDU.waitExtension();
            readBuffer[i] = apdu.getBuffer()[offset+i];
        }

        writeToFiles(dataLength);
        pagesCreated = false;
        menuOK = false;
        if(readBuffer[0]%16 == readBuffer[0]/16)
        {
            menuOK = true;
            currentPage = downLoadPage;
            APDU.waitExtension();
            if(currentPage.buildCommand() == ST_SUCCESS)
            {
                ProactiveCommand.executeCommand();
            }
        }
    }

    break;

case APDU_TERMINALRESPONSE:
```

```
            currentPage = currentPage.processResponse();

            APDU.waitExtension();

            if(currentPage != null)
            {
                if(currentPage.buildCommand() == ST_SUCCESS)
                {
                    APDU.waitExtension();
                    ProactiveCommand.executeCommand();
                }
            }
            break;

        case APDU_CONFIGURE:

            ProactiveServices.configure(apdu, customElement);
            break;

        case APDU_STATUS:
            break;

        case APDU_UPDATE_RECORD:
            break;
    }//switch
}//method

/** Returns true.*/
public boolean select()
{
    return true;
}

/** Does nothing at all.*/
public void deselect()
{
}

/** Creates an instance of ServiceInterpreter upon installation onto card.*/
public static void install(APDU apdu)
{
    new ServiceInterpreter();
}

/** Takes a piece of datastructure, examines it and either creates a new
    menutree or expands the current.*/
private static void writeToFiles(short dataLength)
{

    short dataStructureOffset = 1;//just to skip sequencenumber
    byte currentRecordNb;
    short binaryFileOffset;

    FileSystem.selectFile(MASTER_FILE_ID);
    FileSystem.selectFile(DIRECTORY_FILE_ID);
    APDU.waitExtension();

    /*********************************************************
    * The sequencenumber is of the form 0xXY where X states the *
    * total number of SMS-datadownloads to recieve and Y states *
    * the current SMS-datadownload´s sequencenumber.            *
    *********************************************************/
    if(readBuffer[0]%16 == 0x01) //new tree
    {
        //resetting tree-data
        currentRecordNb = 1;
        binaryFileOffset = 0;
```

```
        }
        else
        {
            FileSystem.selectFile(CONTROL_FILE_ID);
            APDU.waitExtension();
            CyberflexOS.readBinaryFile(controlArray, NO_OFFSET, NO_OFFSET,
                                       (short)3);
            currentRecordNb = (byte)(controlArray[CLF_NO_OF_NODES] + 1);
            binaryFileOffset = (short)((controlArray[CLF_BINARY_FILE_SIZE_1] << 8)
                                       | controlArray[CLF_BINARY_FILE_SIZE_2]);
        }

        do{
            APDU.waitExtension();
            //Write Strings to Binary File
            FileSystem.selectFile(BINARY_FILE_ID);
            CyberflexOS.writeBinaryFile(readBuffer, (short)(dataStructureOffset+3),
                                        binaryFileOffset,
                                        (short)readBuffer[dataStructureOffset+2]);

            APDU.waitExtension();
            //Write Other Data to Record File
            recordData[INFOTYPE] = readBuffer[dataStructureOffset];
            recordData[TREE_REF] = readBuffer[dataStructureOffset+1];
            recordData[DATA_LENGTH] = readBuffer[dataStructureOffset+2];
            recordData[DATA_OFFSET_1] = (byte)(binaryFileOffset / 256);
            recordData[DATA_OFFSET_2] = (byte)(binaryFileOffset % 256);
            binaryFileOffset = (short)(binaryFileOffset +
                                       (short)readBuffer[dataStructureOffset+2]);
            dataStructureOffset = (short)(dataStructureOffset + 3 +
                                          (short)readBuffer[dataStructureOffset+2]);

            APDU.waitExtension();
            FileSystem.selectFile(RECORD_FILE_ID);
            CyberflexOS.writeRecord(recordData, NO_OFFSET, currentRecordNb,
                                    ABSOLUTE, RECORD_SIZE);
            currentRecordNb++;
        }while(dataStructureOffset < dataLength);

        controlArray[CLF_BINARY_FILE_SIZE_2] = (byte)(binaryFileOffset & 0x00FF);
        controlArray[CLF_BINARY_FILE_SIZE_1] = (byte)(binaryFileOffset >> 8);
        controlArray[CLF_NO_OF_NODES] = (byte)(currentRecordNb - 1);
        FileSystem.selectFile(CONTROL_FILE_ID);
        APDU.waitExtension();
        CyberflexOS.writeBinaryFile(controlArray, NO_OFFSET, NO_OFFSET, (short)3);
    }
}
```

A.1.2 SelectItem.java

```
/*********************************************
*  (c) Copyright Telia Mobile AB, created 2000 *
*      All rights reserved                     *
*********************************************/

import javacard.framework.*;
import javacardx.framework.*;
import sim.toolkit.*;

/** The class that handles submenu-nodes*/
public class SelectItem extends Page implements GSMCharacter, ST, MenuConstants
{
    /** Used to store the current node´s information.*/
    byte[] currentRecord;

    /** Used to store the next node´s information.*/
    byte[] newRecord;

    /** Used to keep a temporary node´s information.*/
    byte[] tempRecord;

    /** The head of the menu.*/
    ProactiveElement head;

    /** Used to keep track of the current node´s number.*/
    byte recordNo;

    /** Used as a temporary item for insertion into the list.*/
    ProactiveElement item;

    /** The page to be returned when a response is processed.*/
    Page returnPage;

    /** The list of items.*/
    ProactiveElement[] itemList;

    /** A persistent list of items, used just to not loosing memory
        when nulling elements in itemList.*/
    ProactiveElement[] persistentItemList;

    /** An ordinary counter-variable.*/
    byte i;

    /** The text to show as the head of the menu.*/
    byte[] alphaIDText;

    /** The offset to the current node´s datafield in the binaryfile.*/
    short itemTextOffset;

    /** The text to show for the item.*/
    byte[] itemText;

    /** Used to keep track of the number of items to select between.*/
    byte noOfItems;

    /** Used to keep track of how many records that has been checked.*/
    byte recordsChecked;

    /** Just initalizes all variables.*/
    public SelectItem()
    {
        currentRecord = new byte[RECORD_SIZE];
        APDU.waitExtension();
        newRecord = new byte[RECORD_SIZE];
        APDU.waitExtension();
```

```
    itemText = new byte[MAX_NODENAME_LENGTH];
    APDU.waitExtension();
    itemList = new ProactiveElement[MAX_NO_OF_ITEMS];
    persistentItemList = new ProactiveElement[MAX_NO_OF_ITEMS];
    APDU.waitExtension();
    for(i = 0; i < MAX_NO_OF_ITEMS; i++)
    {
        itemList[i] = new ProactiveElement(ITEM_TAG, (short)itemText.length);
        APDU.waitExtension();
    }
    for(i = 0; i < MAX_NO_OF_ITEMS; i++)
    {
        persistentItemList[i] = new ProactiveElement(ITEM_TAG,
                                                     (short)itemText.length);
        APDU.waitExtension();
    }
    alphaIDText = new byte[MAX_NODENAME_LENGTH];
    APDU.waitExtension();
    head = new ProactiveElement(ALPHA_IDENTIFIER_TAG,
                                (short)(alphaIDText.length));
    APDU.waitExtension();
    tempRecord = new byte[RECORD_SIZE];
    APDU.waitExtension();
}

/** Scans the datastructure for children of the current node, all children are
    added to itemList and the proactive command selectItem is built and put on
    the commandbuffer.*/
public byte buildCommand()
{
    APDU.waitExtension();
    recordNo = ServiceInterpreter.selectedItem;
    APDU.waitExtension();
    FileSystem.selectFile(DIRECTORY_FILE_ID);
    APDU.waitExtension();
    FileSystem.selectFile(RECORD_FILE_ID);
    APDU.waitExtension();
    CyberflexOS.readRecord(currentRecord, NO_OFFSET, recordNo, ABSOLUTE,
                           RECORD_SIZE);
    APDU.waitExtension();

    APDU.waitExtension();
    FileSystem.selectFile(BINARY_FILE_ID);
    APDU.waitExtension();
    CyberflexOS.readBinaryFile(alphaIDText, NO_OFFSET,
                    (short)(256*currentRecord[DATA_OFFSET_1] +
                            currentRecord[DATA_OFFSET_2]),
                    (short)currentRecord[DATA_LENGTH]);

    for(i = 0; i < MAX_NO_OF_ITEMS; i++)
    {
        itemList[i] = persistentItemList[i];
        APDU.waitExtension();
    }


    APDU.waitExtension();
    head.setLength((short)(currentRecord[DATA_LENGTH]));
    APDU.waitExtension();
    head.setValue(alphaIDText, NO_OFFSET, (short)currentRecord[DATA_LENGTH]);
    APDU.waitExtension();

    noOfItems = (byte)0;
    APDU.waitExtension();
    recordsChecked = (byte)0;
    APDU.waitExtension();
    do
    {
```

```
            APDU.waitExtension();
            FileSystem.selectFile(RECORD_FILE_ID);
            APDU.waitExtension();
            CyberflexOS.readRecord(tempRecord, NO_OFFSET,
                                   (byte)(recordNo+recordsChecked+1),
                                   ABSOLUTE, RECORD_SIZE);
            APDU.waitExtension();
            if(tempRecord[TREE_REF] == recordNo)
            {
                APDU.waitExtension();

                itemTextOffset = (short)(256*tempRecord[DATA_OFFSET_1] +
                                         tempRecord[DATA_OFFSET_2]);
                APDU.waitExtension();

                //setting itemID
                itemText[0] = (byte)(recordNo + recordsChecked + (byte)1);
                APDU.waitExtension();

                FileSystem.selectFile(BINARY_FILE_ID);
                APDU.waitExtension();

                CyberflexOS.readBinaryFile(itemText, (short)1, itemTextOffset,
                                           (short)tempRecord[DATA_LENGTH]);
                APDU.waitExtension();
                i = 1;
                while(itemText[i] != CH_STAR && itemText[i] != CH_SQUARE &&
                      i <= tempRecord[DATA_LENGTH])
                {
                    APDU.waitExtension();
                    i++;
                }

                itemList[noOfItems].setLength(i);
                APDU.waitExtension();

                itemList[noOfItems].setValue(itemText, NO_OFFSET, i);
                APDU.waitExtension();

                noOfItems++;
            }
            APDU.waitExtension();
            recordsChecked++;
            APDU.waitExtension();
        }while((tempRecord[TREE_REF] == recordNo || noOfItems == 0) &&
               (recordNo+recordsChecked <
                ServiceInterpreter.controlArray[CLF_NO_OF_NODES]));

        APDU.waitExtension();
        for(i = noOfItems;i < MAX_NO_OF_ITEMS;i++)
        {
            itemList[i] = null;
        }

        APDU.waitExtension();
        ProactiveCommand.selectItem(GENERIC_COMMAND_NO, NO_HELP_INFO, head,
                                    itemList);
        APDU.waitExtension();
        return ST_SUCCESS;
    }

    /** Checks which child that has been selected, checks if it´s a submenu-node or
        service-node returns "this" for submenu-nodes and
        "ServiceIterpreter.prepareSMSPage" for service-nodes.*/
    public Page processResponse()
    {
        if(ProactiveCommand.manageGeneralResult() == ST_SUCCESS)
        {
```

```
            ServiceInterpreter.selectedItem =
                                        (byte)(ProactiveCommand.getItemIdentifier());
            APDU.waitExtension();

            FileSystem.selectFile(DIRECTORY_FILE_ID);
            APDU.waitExtension();
            FileSystem.selectFile(RECORD_FILE_ID);
            APDU.waitExtension();
            CyberflexOS.readRecord(newRecord, NO_OFFSET,
                                   ServiceInterpreter.selectedItem, ABSOLUTE,
                                   RECORD_SIZE);
            APDU.waitExtension();
            switch(newRecord[INFOTYPE])
            {
                case IT_SELECT_ITEM:
                    returnPage = this;
                    break;
                case IT_SEND_SMS:
                    returnPage = ServiceInterpreter.prepareSMSPage;
                    break;
            }
        }
        if(ProactiveCommand.manageGeneralResult() == ST_BACK)
        {
            ServiceInterpreter.selectedItem = currentRecord[TREE_REF];
            APDU.waitExtension();

            if(ServiceInterpreter.selectedItem == (byte)0)
            {
                returnPage = null;
            }
            else
            {
                returnPage = this;
            }
        }
        return returnPage;
    }
}
```

A.1.3 PrepareSMS.java

```
/*********************************************
*  (c) Copyright Telia Mobile AB, created 2000 *
*      All rights reserved                     *
*********************************************/

import javacard.framework.*;
import javacardx.framework.*;
import sim.toolkit.*;

/** The class that handles service-nodes.*/
public class PrepareSMS extends Page implements GSMCharacter, ST, MenuConstants
{
    /** Used to store the current node´s information.*/
    byte[] currentRecord = new byte[RECORD_SIZE];

    /** Used to keep track of the current node´s number.*/
    byte recordNo;

    /** Used to hold the responselength limits, the first byte is for the lower
        limit and the second is for the higher limit.*/
    byte[] responseLengthLimits = new byte[2];

    /** Used to hold the question-text.*/
    byte[] questionText;

    /** Used to hold the indexes to every question in the allData-array.*/
    byte[] indexToQuestionText = new byte[MAX_NO_OF_QUESTIONS];

    /** Used to keep track of the total number of questions for a service-node.*/
    short noOfQuestions = 0;

    /** Used to keep track on how many questions that have been answered.*/
    short noOfQuestionsAnswered = 0;

    /** Used to keep track of the length of the SMS to send.*/
    short smsTextLength = 0;

    /** Used to hold a node´s whole data-field.*/
    byte[] allData = new byte[MAX_NODEDATA_LENGTH];

    /** Used to hold the text to send.*/
    byte smsText[] = new byte[MAX_SMS_LENGTH];


    /** Used to hold the answers to the questions.*/
    byte[] answer_text = new byte[MAX_SMS_LENGTH];

    /** The proactive element to send.*/
    ProactiveElement smsTPDU;

    /** The data of the smsTPDU.*/
    byte[] smsTPDUvalue;

    /** The number to send SMS:s to, currently US-server is on 12173.*/
    byte[] nrToSendTo = {(byte)0x21, (byte)0x71, (byte)0xF3};

    /** Used to keep track of a questions length.*/
    byte questionTextLength;

    /** An ordinary cunter-variable.*/
    short i;

    /** Used to keep track of the current offset in the allData-array.*/
    byte dataOffset;
```

```
/** The page to be returned when a response is processed.*/
Page returnPage;

/** The proactive element that is used for questions.*/
ProactiveElement question;

/** The proactive element that is used responselengthlimits.*/
ProactiveElement responseLength;

/** Just initalizes all variables.*/
public PrepareSMS()
{
    smsTPDUvalue = new byte[MAX_SMSTPDU_LENGTH];
    APDU.waitExtension();

    questionText = new byte[MAX_QUESTION_LENGTH];
    smsTPDU = new ProactiveElement(SMS_TPDU_TAG,(short)smsTPDUvalue.length);
    APDU.waitExtension();

    //Shortest answer OK is 1 byte
    responseLengthLimits[LOWER_LIMIT_BYTE] = (byte)0x01;

    //Longest anser OK is 32 bytes
    responseLengthLimits[UPPER_LIMIT_BYTE] = (byte)0x20;

    APDU.waitExtension();

    responseLength = new ProactiveElement(RESPONSE_LENGTH_TAG,
                                          (short)responseLengthLimits.length);
    APDU.waitExtension();

    responseLength.setValue(responseLengthLimits, NO_OFFSET,
                            (short)responseLengthLimits.length);
    APDU.waitExtension();

    question = new ProactiveElement(TEXT_STRING_TAG,
                                    (short)questionText.length);
    APDU.waitExtension();
}

/** Checks if all questions are answered, if they are it builds the SMS and
    builds the send SMS command. If there are unanswered question the next
    question is presented to the user by building the get input command.*/
public byte buildCommand()
{
    if(noOfQuestionsAnswered == 0)
    {
        recordNo = ServiceInterpreter.selectedItem;
        APDU.waitExtension();

        FileSystem.selectFile(DIRECTORY_FILE_ID);
        FileSystem.selectFile(RECORD_FILE_ID);
        APDU.waitExtension();

        CyberflexOS.readRecord(currentRecord, NO_OFFSET, recordNo, ABSOLUTE,
                               RECORD_SIZE);
        APDU.waitExtension();

        FileSystem.selectFile(BINARY_FILE_ID);
        APDU.waitExtension();

        CyberflexOS.readBinaryFile(allData, NO_OFFSET,
                                   (short)(currentRecord[DATA_OFFSET_1]*256 +
                                           currentRecord[DATA_OFFSET_2]),
                                   currentRecord[DATA_LENGTH]);

        //Storing the index in the allData-array for every question
        for(dataOffset=0; dataOffset<currentRecord[DATA_LENGTH]; dataOffset++)
```

```
    {
        APDU.waitExtension();

        //initialize to zero in case there are no questions
        indexToQuestionText[noOfQuestions] = (byte)0;
        if((allData[dataOffset] == CH_STAR) ||
           (allData[dataOffset] == CH_SQUARE))
        {
            APDU.waitExtension();
            indexToQuestionText[noOfQuestions] = (byte)(dataOffset+1);
            noOfQuestions++;
        }
    }

    //Putting the Service name in the sms-bytearray
    if(noOfQuestions == 0)
    {
        for(i = 0; i < (short)currentRecord[DATA_LENGTH]; i++)
        {
            smsText[i] = allData[i];
        }
        smsTextLength = (byte)currentRecord[DATA_LENGTH];
    }
    else
    {
        for(i = 0; i < (short)(indexToQuestionText[0]-1); i++)
        {
            smsText[i] = allData[i];
        }
        smsTextLength = (short)(indexToQuestionText[0]-1);
    }
}

byte commandQualifier;

if(noOfQuestions == noOfQuestionsAnswered)
{
    short pos;

    APDU.waitExtension();
    smsTPDUvalue[0] = (byte)0x11;       //first byte set to SMS-submit,
                                        //TP-VP field integer
    smsTPDUvalue[1] = (byte)TP_MR;      //TP-MR set to 0x00
    smsTPDUvalue[2] = (byte)0x05;       //Number of Digits in adress
    APDU.waitExtension();
    smsTPDUvalue[3] = (byte)0xA1;       //TON-NPI
    smsTPDUvalue[4] = nrToSendTo[0];    //first two digits (in wrong order)
    smsTPDUvalue[5] = nrToSendTo[1];    //next two digits (in wrong order)
    APDU.waitExtension();
    smsTPDUvalue[6] = nrToSendTo[2];    //next two digits (in wrong order)
    APDU.waitExtension();
    smsTPDUvalue[7] = (byte)0x00;           //Protocol identifer
    smsTPDUvalue[8] = (byte)0xF0;           //TP-data coding scheme
    smsTPDUvalue[9] = (byte)0xA7;           //TP-VP (Time to live)
    smsTPDUvalue[10] = (byte)smsTextLength; //User data length

    for(pos = 0; pos < smsTextLength; pos++)
    {
        APDU.waitExtension();
        smsTPDUvalue[pos+11] = smsText[pos];
    }

    APDU.waitExtension();
    smsTPDU.setLength((short)(smsTextLength + nrToSendTo.length + 8));
    APDU.waitExtension();
    smsTPDU.setValue(smsTPDUvalue,(short)0, (short)(smsTextLength +
                     nrToSendTo.length + 8));
    APDU.waitExtension();
```

```
            ProactiveCommand.sendShortMessage(GENERIC_COMMAND_NO, SMS_PACKED, null,
                                              null, smsTPDU);

        }
        else
        {
            if(noOfQuestionsAnswered < (noOfQuestions-1))
            {
                questionTextLength =
                              (byte)(indexToQuestionText[noOfQuestionsAnswered+1] –
                                     indexToQuestionText[noOfQuestionsAnswered]);
                //Making it 1 byte to large to make room for coding scheme ID
            }
            else
            {
                questionTextLength = (byte)(currentRecord[DATA_LENGTH] –
                                    indexToQuestionText[noOfQuestionsAnswered] + 1);
                //Making it 1 byte to large to make room for coding scheme ID
            }
            APDU.waitExtension();
            questionText[0] = CODEBASE;  //Coding scheme identifier

            //Copying the questions into the qustionText-array
            for(byte counter = 0; counter < questionTextLength-1; counter++)
            {
                questionText[counter+1] =
                               allData[indexToQuestionText[noOfQuestionsAnswered] +
                                       (counter)];
            }

            //Determinig type of answer expected
            if(allData[indexToQuestionText[noOfQuestionsAnswered]-1] == CH_STAR)
            {
                commandQualifier = DIGITS_ONLY_SHOW_UNPACKED;
            }
            else
            {
                commandQualifier = ALL_CHARS_SHOW_UNPACKED;
            }

            APDU.waitExtension();

            question.setLength(questionTextLength);
            APDU.waitExtension();
            question.setValue(questionText, NO_OFFSET, (short)questionTextLength);
            APDU.waitExtension();

            ProactiveCommand.getInput(GENERIC_COMMAND_NO, commandQualifier,
                                      question, responseLength);
            APDU.waitExtension();
        }
        return ST_SUCCESS;
    }

    /** Checks if an SMS has been cunstructed yet (all questions anxsered), if so
        it resets all variables and returns the parent-page. If there are questions
        unanswered it returns "this" for processing the next question.*/
    public Page processResponse()
    {
        if(ProactiveCommand.manageGeneralResult() == ST_SUCCESS)
        {
            if(noOfQuestions == noOfQuestionsAnswered)
            {
                noOfQuestionsAnswered = 0;
                if(noOfQuestions == 0)
                {
                    smsTextLength = (short)currentRecord[DATA_LENGTH];
```

```
                }
                else
                {
                    smsTextLength = (short)(indexToQuestionText[0]-1);
                }
                noOfQuestions = 0;
                ServiceInterpreter.selectedItem = currentRecord[TREE_REF];
                APDU.waitExtension();
                returnPage = ServiceInterpreter.selectItemPage;
            }
            else
            {
                answer_text[0] = CH_STAR;
                //getting the length of the getinput-answer.
                short answer_length = (short)ProactiveCommand.getText(answer_text,
                                                                      (short)1);

                answer_length++; //due to answertype-byte
                for(short i = 0; i < answer_length; i++)
                {
                    smsText[smsTextLength+i] = answer_text[i];
                }
                APDU.waitExtension();
                smsTextLength = (short)(smsTextLength  + answer_length);
                noOfQuestionsAnswered++;

                returnPage = this;
            }
        }

        if(ProactiveCommand.manageGeneralResult() == ST_BACK)
        {
            noOfQuestionsAnswered = 0;
            noOfQuestions = 0;

            APDU.waitExtension();
            ServiceInterpreter.selectedItem = currentRecord[TREE_REF];

            APDU.waitExtension();
            returnPage = ServiceInterpreter.selectItemPage;
        }
        return returnPage;
    }
}
```

A.1.4 DisplayText.java

```
/*****************************************************
*  (c) Copyright Telia Mobile AB, created 2000        *
*      All rights reserved                            *
*****************************************************/

import javacardx.framework.*;
import sim.toolkit.*;

/** The class used to display a textmessage to the user.*/
public class DisplayText extends Page implements GSMCharacter, ST
{
    /** The proactive element to hold the text-TLV.*/
    ProactiveElement text;

    /** The byta-array to hold the textstring*/
    byte codedText[];

    /** Takes a textstring, adds coding scheme and creates the corresponding
        proactive element.*/
    public DisplayText(byte textString[])
    {
        codedText = new byte[textString.length+1];
        codedText[0] = CODEBASE;  //Coding scheme identifer
        for(short i=0;i<textString.length;i++)
        {
            codedText[i+1] = textString[i];
        }
        text = new ProactiveElement(TEXT_STRING_TAG,(short)codedText.length);
        text.setValue(codedText,(short)0, (short)codedText.length);
    }

    /** Builds the command display Text and puts it on the commandbuffer.*/
    public byte buildCommand()
    {
        ProactiveCommand.displayText(GENERIC_COMMAND_NO,
                                     (byte)(CLR_BY_USER+HIGH_PRIORITY), text);
        return ST_SUCCESS;
    }

    /** Just calls the processResponse-method of the Page-class and returns it´s
        answer.*/
    public Page processResponse()
    {
        Page returnPage;
        returnPage = super.processResponse();
        return returnPage;
    }
}
```

A.1.5 MenuConstants.java

```
/*******************************************
*  (c) Copyright Telia Mobile AB, created 2000 *
*      All rights reserved                     *
*******************************************/

/** Lots of constants that we use.*/
public interface MenuConstants
{
    //File identifers
    /** File ID for the card´s Master File.*/
    public static final short MASTER_FILE_ID    = (short)0x3F00;

    /** File ID for the directory.*/
    public static final short DIRECTORY_FILE_ID = (short)0x7F41;

    /** File ID for the binary-file.*/
    public static final short BINARY_FILE_ID    = (short)0x7F42;

    /** File ID for the record-file.*/
    public static final short RECORD_FILE_ID    = (short)0x7F43;

    /** File ID for the control-file.*/
    public static final short CONTROL_FILE_ID   = (short)0x7F44;


    //File handling
    /** The size of a record.*/
    public static final short RECORD_SIZE     = (short)5;

    /** ByteCode used to be able to address a record absolutely.*/
    public static final byte  ABSOLUTE        = (byte)4;

    /** Position in the control-file for the byte holding the number of nodes.*/
    public static final byte  CLF_NO_OF_NODES = (byte)0;

    /** Postition in the control-file for the the first byte holding the
        binary-file size.*/
    public static final byte  CLF_BINARY_FILE_SIZE_1 = (byte)1;

    /** Postition in the control-file for the the second byte holding the
        binary-file size.*/
    public static final byte  CLF_BINARY_FILE_SIZE_2 = (byte)2;

    /** The size of the control-file.*/
    public static final short CONTROL_FILE_SIZE = (short)3;


    //Record positions
    /** Position for the Infotype value in the record.*/
    public static final short INFOTYPE      = (short)0;

    /** Position for the Tree ref value in the record.*/
    public static final short TREE_REF      = (short)1;

    /** Position for the Data Length value in the record.*/
    public static final short DATA_LENGTH   = (short)2;

    /** Position for the first byte holding the offset for the data
        in the binary file.*/
    public static final short DATA_OFFSET_1 = (short)3;
```

```
/** Position for the first second holding the offset for the data
    in the binary file.*/
public static final short DATA_OFFSET_2 = (short)4;


//Infotypes
/** Constant for Select Item node.*/
public static final byte IT_SELECT_ITEM   = (byte)0x01;

/** Constant for Prepare SMS node.*/
public static final byte IT_SEND_SMS      = (byte)0x02;


//Characters
/** Bytecode for a '*' character.*/
public static final byte CH_STAR = (byte)0x2A;

/** Bytecode for a '#' character.*/
public static final byte CH_SQUARE = (byte)0x23;

/** Bytecode for a 'å' character.*/
public static final byte å = (byte)0x0F;

/** Bytecode for a 'ä' character.*/
public static final byte ä = (byte)0x7B;

/** Bytecode for a 'ö' character.*/
public static final byte ö = (byte)0x7C;


//miscellaneous
/** A generic commandnumber that is used for every command.*/
public static final byte  GENERIC_COMMAND_NO = (byte)0x01;

/** The coding scheme identifier for 8-bit data unpacked format.*/
public static final byte  CODEBASE   = (byte)0xF4;

/** The offset 0.*/
public static final short NO_OFFSET  = (short)0;

/** The position of the byte stating the lower lengthlimit of a response.*/
public static final byte  LOWER_LIMIT_BYTE = (byte)0x00;

/** The position of the byte stating the higher lengthlimit of a response.*/
public static final byte  UPPER_LIMIT_BYTE = (byte)0x01;

/** The identifier for the type of response that takes all GSM-characters
    and displays echo.*/
public static final byte  ALL_CHARS_SHOW_UNPACKED = (byte)0x01;

/** The identifier for the type of response that takes only numbers
    and displays echo.*/
public static final byte  DIGITS_ONLY_SHOW_UNPACKED = (byte)0x00;

/** Constant defining maximum number of questions in a Prepare SMS node.*/
public static final short MAX_NO_OF_QUESTIONS = (short)10;

/** Constant defining maximum length of the text for one question.*/
public static final short MAX_QUESTION_LENGTH = (short)30;

/** Constant defining maximum length of the data for one node.*/
public static final short MAX_NODEDATA_LENGTH = (short)50;

/** Constant defining maximum length of a SMS.*/
public static final short MAX_SMS_LENGTH = (short)140;

/** Constant defining maximum length for a nodes name.*/
public static final short MAX_NODENAME_LENGTH = (short)30;
```

```
    /** Constant defining maximum number of childs a selectItem node can have.*/
    public static final short MAX_NO_OF_ITEMS = (short)10;

    /** Constant defining maximum length of the SMSTpdu.*/
    public static final short MAX_SMSTPDU_LENGTH = (short)164;

    /** Constant defining maximum length of the datadownload*/
    public static final short MAX_DATADOWNLOAD_LENGTH = (short)114;

    /** The position for the byte stating the selected itemID in a
        Menu-selcetion envelope*/
    public static final short ITEM_ID_POSITION = (short)13;
}
```

A.2 Programkod för det grafiska gränsnittet

A.2.1 MenuBuilder.java

```
/*******************************************
*  (c) Copyright Telia Mobile AB, created 2000 *
*      All rights reserved                     *
*******************************************/

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.tree.*;
import java.util.*;
import java.net.*;

/** Main class for the GUI.*/
public class MenuBuilder extends JApplet implements ActionListener,
                                                     TreeSelectionListener,
                                                     ItemListener, KeyListener
{
    /** Panel for the buttons.*/
    private JPanel buttonHolder = new JPanel(new GridLayout(1,4));

    /** The sendbutton.*/
    private Button sendButton = new Button("Sänd");

    /** The button for adding a new submenu.*/
    private Button addLevelButton = new Button("Lägg till undermeny");

    /** The button for adding a new service.*/
    private Button addServiceButton = new Button("Lägg till tjänst");

    /** The button for removing a node.*/
    private Button removeNodeButton = new Button("Ta bort nod");

    /** Information text area.*/
    private JTextArea textarea = new JTextArea();

    /** The panel holding all components used when choosing a service.*/
    private JPanel serviceNodeNamingPanel = new JPanel();

    /** A information textlabel.*/
    private JLabel serviceLabel = new JLabel("    Välj tjänst till noden      ");

    /** The combobox to choose services from.*/
    private JComboBox serviceBox = new JComboBox();

    /** The panel holding all components used when naming a submenu.*/
    private JPanel subMenuNodeNamingPanel = new JPanel();

    /** A information textlabel.*/
    private JLabel subMenuNodeNamingLabel = new JLabel(
                                            "Skriv önskat nodnamn. Max 15 tkn");

    /** The button for confirming the renaming a submenu.*/
    private JButton subMenuNodeNamingOkButton = new JButton("Ok");

    /** The textfield used to enter a new submenu-name.*/
    private JTextField subMenuNodeNamingTextField = new JTextField(13);

    /** A simple counter that holds the total number of nodes.*/
```

```
private int nodeCounter;

/** Tells whether the tree is valid or not.*/
private boolean isTreeValid;

/** A counter that holds the number of deprecated services.*/
private int servicesNotExisting = 0;

/** Holds the current users GSM-phonenumber.*/
private String userphone;

/** Holds the old datastructure.*/
private String datastruct;

/** Holds the current user´s active services.*/
private String servicestruct;

/** Holds the current user´s active services.*/
private Vector serviceVector = new Vector();

/** Holds the new datastructure.*/
private String structToSend = new String();

/** A Vector for storing the tree´s nodes.*/
private Vector nodeVector = new Vector();

/** The tree.*/
private JTree tree;

/** The tree´s treemodel.*/
private DefaultTreeModel treeModel;

/** The rootnode with default text.*/
private Node rootNode = new Node("US-Meny", true);

/** Initatilizes all graphic components, parse inparamaters
    and builds the old tree if there is one.*/
public void init()
{
    int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
    int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;

    userphone = getParameter("gsm");
    datastruct = getParameter("datastruct");
    servicestruct = getParameter("servicestruct");

    parseServicestruct();

    /*Setting up JPanel for service-node naming*/
    serviceLabel.setForeground(Color.white);
    serviceNodeNamingPanel.add(serviceLabel);
    serviceBox.setSize(100,25);
    serviceBox.addItemListener(this);
    serviceNodeNamingPanel.add(serviceBox);

    serviceNodeNamingPanel.setLayout(new FlowLayout());
    serviceNodeNamingPanel.setLocation(300,0);
    serviceNodeNamingPanel.setSize(210,230);
    serviceNodeNamingPanel.setBackground(new Color(110,30,110));
    serviceNodeNamingPanel.setVisible(false);
    getContentPane().add(serviceNodeNamingPanel);

    /*Setting up JPanel for level-node naming*/
    subMenuNodeNamingLabel.setForeground(Color.white);
    subMenuNodeNamingPanel.add(subMenuNodeNamingLabel);

    subMenuNodeNamingTextField.setBackground(Color.white);
    subMenuNodeNamingTextField.addKeyListener(this);
```

```
    subMenuNodeNamingPanel.add(subMenuNodeNamingTextField);

    subMenuNodeNamingOkButton.addActionListener(this);
    subMenuNodeNamingPanel.add(subMenuNodeNamingOkButton);

    subMenuNodeNamingPanel.setLayout(new FlowLayout());
    subMenuNodeNamingPanel.setLocation(300,0);
    subMenuNodeNamingPanel.setSize(210,60);
    subMenuNodeNamingPanel.setBackground(new Color(110,30,110));
    subMenuNodeNamingPanel.setVisible(false);
    getContentPane().add(subMenuNodeNamingPanel);

    getContentPane().setLayout(null);
    getContentPane().setBackground(new Color(110,30,110));

    buttonHolder.setSize(508,30);
    buttonHolder.setLocation(0,300);
    getContentPane().add(buttonHolder);

    addLevelButton.addActionListener(this);
    buttonHolder.add(addLevelButton);

    addServiceButton.addActionListener(this);
    buttonHolder.add(addServiceButton);

    removeNodeButton.addActionListener(this);
    buttonHolder.add(removeNodeButton);

    sendButton.addActionListener(this);
    buttonHolder.add(sendButton);

    textarea.setEditable(false);
    JScrollPane isp = new JScrollPane(textarea, v, h);
    isp.setSize(207,80);
    isp.setLocation(300,230);
    getContentPane().add(isp);

    rootNode.setInfoType(0);  //infotype 0, to separate it from other nodes
    rootNode.setNodeNumber(1);
    treeModel = new DefaultTreeModel(rootNode);
    tree = new JTree(treeModel);
    tree.addTreeSelectionListener(this);
    tree.getSelectionModel().setSelectionMode(
                                    TreeSelectionModel.SINGLE_TREE_SELECTION);
    tree.setShowsRootHandles(true);

    JScrollPane tsp = new JScrollPane(tree, v, h);
    tsp.setSize(300,300);
    tsp.setLocation(0,0);
    getContentPane().add(tsp);

    parseDatastruct();
    addChildren(rootNode);
    tree.setSelectionPath(new TreePath(rootNode.getPath()));
    if(servicesNotExisting > 0)
    {
        textarea.append(servicesNotExisting+
                        " tjänst(er) i din gamla meny finns");
        textarea.append("\ninte längre i din Tjänsteportfölj.");
        textarea.append("\nDen/Dessa tjänster har tagits bort ur");
        textarea.append("\nmenyn.");
    }

}

/** Parses the string that represents the users active services and puts all
    of them in the serviceVector.*/
public void parseServicestruct()
```

```
{
    StringTokenizer st = new StringTokenizer(servicestruct, ";");
    String token;
    try
    {
        while(true)
        {
            token = st.nextToken();
            Service service = new Service(token);
            serviceBox.addItem(service.getServiceName());
            serviceVector.addElement(service);
        }
    }catch (NoSuchElementException e){}
}

/** Parses the datastructure and puts each node in the nodeVector.*/
public void parseDatastruct()
{
    if(datastruct != null)
    {
        int dSOffset = 0;
        int dataLength;
        Node tempNode;
        Service tempService;
        int nodeType;
        int nodeNumber = 1;
        String nodeString;

        while(dSOffset < datastruct.length())
        {
            dataLength = Integer.parseInt(datastruct.substring(dSOffset+4,
                                          dSOffset+6),16);
            nodeString = new String(convertToAscii(datastruct.substring(
                                    dSOffset+6, dSOffset+6+2*dataLength)));
            tempService = new Service(nodeString);
            nodeType = Integer.parseInt(datastruct.substring(dSOffset,
                                        dSOffset+2),16);
            if(nodeType == 1)
            {
                tempNode = new Node(tempService.getServiceName(), true);
                tempNode.setNodeNumber(nodeNumber++);
                tempNode.setTreeRef(Integer.parseInt(
                                            datastruct.substring(dSOffset+2,
                                            dSOffset+4),16));
            }
            else
            {
                tempNode = new Node(tempService.getServiceName(), false);
                tempNode.setNodeNumber(nodeNumber++);
                tempNode.setTreeRef(Integer.parseInt(
                                            datastruct.substring(dSOffset+2,
                                            dSOffset+4),16));
            }
            nodeVector.addElement(tempNode);
            dSOffset = dSOffset + 6 + 2*dataLength;
        }
    }
}

/** Adds all nodes in nodeVector to the tree recursively,
    used only for the old datastructure.*/
private void addChildren(Node currentNode)
{
    int treeRef;
    int nodeType;
    Node tempNode;
    boolean serviceExists;
```

```
    for(int i = currentNode.getNodeNumber();i < nodeVector.size(); i++)
    {
        tempNode = (Node)nodeVector.elementAt(i);
        treeRef = tempNode.getTreeRef();
        nodeType = tempNode.getInfoType();
        if(treeRef == currentNode.getNodeNumber())
        {
            if(nodeType == 2)
            {
                serviceExists = false;
                for(int j = 0; j < serviceVector.size(); j++)
                {
                    if(((Service)serviceVector.elementAt(j)).
                       getServiceName().compareTo(tempNode.getNodeName()) == 0)
                    {
                        serviceExists = true;
                    }
                }
                if(serviceExists)
                {
                    String tempName = tempNode.getNodeName();
                    tempNode.setNodeName("aaaaaaaaaaaaaaa");
                    treeModel.insertNodeInto(tempNode, currentNode,
                                             currentNode.getChildCount());
                    tree.setSelectionPath(new TreePath(tempNode.getPath()));
                    tempNode.setNodeName(tempName);
                }
                else
                {
                    servicesNotExisting++;
                }
            }
            else
            {
                treeModel.insertNodeInto(tempNode, currentNode,
                                         currentNode.getChildCount());
                addChildren(tempNode);
            }

        }
    }
}

/** Removes the selected node and all it´s descendants from the tree.*/
public void removeCurrentNode()
{
    TreePath currentSelection = tree.getSelectionPath();
    if(currentSelection != null)
    {
        Node currentNode = (Node)(currentSelection.getLastPathComponent());
        MutableTreeNode parent = (MutableTreeNode)(currentNode.getParent());

        if(parent != null)
        {
            treeModel.removeNodeFromParent((MutableTreeNode)currentNode);
            tree.setSelectionPath(new TreePath(
                                  ((DefaultMutableTreeNode)parent).getPath()));
            serviceNodeNamingPanel.setVisible(false);
            subMenuNodeNamingPanel.setVisible(false);
            textarea.setText("");
            return;
        }
        else
        {
            textarea.setText("");
            textarea.append("Rotnoden kan inte tas bort.");
        }
    }
```

```
    else
    {
        textarea.setText("");
        textarea.append("Ingen nod vald.");
    }
}

/** Adds a new node under the selected one if allowed.*/
public void addNode(String nodeName, boolean allowsChildren)
{
    TreePath parentPath = tree.getSelectionPath();
    if(parentPath != null)
    {
        Node parent = (Node)(parentPath.getLastPathComponent());
        if(parent.isOkWithChildren())
        {
            if(parent.getChildCount() < 10)
            {
                 Node newNode = new Node(nodeName, allowsChildren);
                 treeModel.insertNodeInto(newNode, parent,
                                          parent.getChildCount());
                 tree.scrollPathToVisible(new TreePath(newNode.getPath()));
                 tree.setSelectionPath(new TreePath(newNode.getPath()));
                 if(allowsChildren)
                 {
                    subMenuNodeNamingTextField.grabFocus();
                    subMenuNodeNamingTextField.selectAll();
                 }
                 else
                 {
                    serviceBox.grabFocus();
                 }
             }
             else
             {
                 textarea.setText("");
                 textarea.append("Denna nod kan ej ha fler barn.");
             }
        }
        else
        {
             textarea.setText("");
             textarea.append("Denna nod kan ej ha några barn.");
        }
    }
    else
    {
        textarea.setText("");
        textarea.append("Ingen nod vald.");
    }
}

/** Walks through the tree and sets the treeRef for each node.*/
public void setTreeRef()
{
    Vector nodeVector = new Vector();
    Service service;
    int i = rootNode.getChildCount();
    int x = 0;

    for(int v = 0; v < i; v++)
    {
        nodeVector.add(rootNode.getChildAt(v));
    }

    while(!nodeVector.isEmpty())
    {
        Node tempNode;
```

```
        tempNode = (Node)nodeVector.firstElement();
        tempNode.setNodeNumber(nodeCounter++);
        tempNode.setTreeRef(((Node)tempNode.getParent()).getNodeNumber());

        structToSend = structToSend.concat(tempNode.getDataToSend());
        if(!tempNode.isOkWithChildren())
        {
            x = 0;
            do
            {
                service = (Service)serviceVector.elementAt(x++);
            }while(service.getServiceName().compareTo(tempNode.getNodeName())
                   != 0);

            if(service.getQuestionText() != null)
            {
                structToSend = structToSend.concat(
                           Integer.toString(service.getServiceName().length() +
                           service.getQuestionText().length())+",");
                structToSend = structToSend.concat(service.getServiceName());
                structToSend = structToSend.concat(service.getQuestionText() +
                                                   ",");
            }
            else
            {
                structToSend = structToSend.concat(
                          Integer.toString(service.getServiceName().length()) +
                                           ",");
                structToSend = structToSend.concat(
                                                service.getServiceName()+",");

            }
        }
        else
        {
             String temp = tempNode.getNodeName();
             structToSend = structToSend.concat(
                                        Integer.toString(temp.length())+",");
             structToSend = structToSend.concat(temp+",");
        }

        if(tempNode.isOkWithChildren() && tempNode.getChildCount() == 0)
        {
            isTreeValid = false;
        }
        for(int y = 0; y < tempNode.getChildCount(); y++)
        {
            nodeVector.add(tempNode.getChildAt(y));
        }
        nodeVector.remove(0);
    }
}

/** Handles the keyboard-events.*/
/*Implements KeyListener*/
public void keyPressed(KeyEvent e)
{
    if(e.getKeyChar() == KeyEvent.VK_ENTER)
    {
        String text = subMenuNodeNamingTextField.getText();
        Node n = (Node)tree.getLastSelectedPathComponent();
        n.setNodeName(text);
        tree.repaint();
        subMenuNodeNamingPanel.setVisible(false);
        subMenuNodeNamingPanel.repaint();
    }
}
```

```
/** Not used.*/
public void keyReleased(KeyEvent e)
{}


/** Not used.*/
public void keyTyped(KeyEvent e)
{}

/** Handles changes in the serviceBox.*/
/*Implements ItemListener*/
public void itemStateChanged(ItemEvent e)
{
    if(e.getSource() == serviceBox)
    {
        String selected = new String((String)serviceBox.getSelectedItem());
        Node n = (Node)tree.getLastSelectedPathComponent();
        n.setNodeName(selected);
        ((DefaultTreeCellRenderer)tree.getCellRenderer()).setPreferredSize(
        ((DefaultTreeCellRenderer)tree.getCellRenderer()).getPreferredSize());
        tree.repaint();
    }
}

/** Handles all events for the buttons.*/
/*Implements ActionListener*/
public void actionPerformed(ActionEvent e)
{
    if(e.getSource() == sendButton)
    {
        nodeCounter = 2;
        isTreeValid = true;
        structToSend = "1,0,7,US-Meny,";
        setTreeRef();
        if(isTreeValid && nodeCounter < 48)
        {
            structToSend = convertToHex(structToSend);
            try
            {
                URL u = new URL(
                          http://us.kar.telia.se/servlet/SendAndStoreMenu?gsm= +
                          userphone + "&datastruct=" + structToSend);
                getAppletContext().showDocument(u);
            }catch(Exception url){}
        }
        else if(!isTreeValid)
        {
            textarea.setText("Ogiltigt träd.\n");
            textarea.append("Undermenyer får ej vara tomma.");
        }
        else
        {
            textarea.setText("Ogiltigt träd.\n");
            textarea.setText("Du har för många noder\ni trädet.(max 47)");
        }
    }
    else if(e.getSource() == addLevelButton)
    {
         addNode("Ny undermeny",true);
    }
    else if(e.getSource() == addServiceButton)
    {
         addNode("Ny tjänst", false);
    }
    else if(e.getSource() == removeNodeButton)
    {
         removeCurrentNode();
    }
```

```
    else if(e.getSource() == subMenuNodeNamingOkButton)
    {
        String text = subMenuNodeNamingTextField.getText();
        Node n = (Node)tree.getLastSelectedPathComponent();
        n.setNodeName(text);
        tree.repaint();
        subMenuNodeNamingPanel.setVisible(false);
        subMenuNodeNamingPanel.repaint();
    }
}

/** Handles Mouse-selections in the tree.*/
/*Implements TreeSelectionListener*/
public void valueChanged(TreeSelectionEvent e)
{
    Node n = (Node)tree.getLastSelectedPathComponent();
    if(n != null)
    {
        if(n.getInfoType() == 1)
        {
            serviceNodeNamingPanel.setVisible(false);
            subMenuNodeNamingTextField.setText(n.getNodeName());
            subMenuNodeNamingPanel.setVisible(true);
            subMenuNodeNamingTextField.grabFocus();
            subMenuNodeNamingTextField.selectAll();
        }
        else if(n.getInfoType() == 2)
        {
            subMenuNodeNamingPanel.setVisible(false);
            serviceBox.setSelectedItem(n.getNodeName());
            serviceNodeNamingPanel.setVisible(true);
        }
        else
        {
            subMenuNodeNamingPanel.setVisible(false);
            ServiceNodeNamingPanel.setVisible(false);
        }

        textarea.setText("");
    }
}

/** Takes a normal string and converts it to GSM-alphabet string.*/
public String convertToHex(String s)
{
    StringTokenizer st = new StringTokenizer(s, ",");
    String converted = new String();
    int counter = 1;

    try
    {
        while(true)
        {
            String token = st.nextToken();
            if(counter%4 == 0)
            {
                for(int i = 0; i < token.length(); i++)
                {
                    if((int)token.charAt(i) == 'å')
                    {
                        converted = converted.concat("0f");
                    }
                    else if((int)token.charAt(i) == 'Å')
                    {
                        converted = converted.concat("0e");
                    }
                    else if((int)token.charAt(i) == 'ä')
                    {
```

```
                        converted = converted.concat("7b");
                    }
                    else if((int)token.charAt(i) == 'Ä')
                    {
                        converted = converted.concat("5b");
                    }
                    else if((int)token.charAt(i) == 'ö')
                    {
                        converted = converted.concat("7c");
                    }
                    else if((int)token.charAt(i) == 'Ö')
                    {
                        converted = converted.concat("5c");
                    }
                    else
                    {
                        converted = converted.concat(
                                    Integer.toHexString((int)token.charAt(i)));
                    }
                }
            }
            else
            {
                if(token.length() > 1)
                {
                    if(Integer.parseInt(token) > 9 &&
                       Integer.parseInt(token) < 16)
                    {
                        converted = converted.concat("0");
                    }
                    converted = converted.concat(
                                Integer.toHexString(Integer.parseInt(token)));
                }
                else
                {
                     converted = converted.concat("0");
                     converted = converted.concat(token);
                }
            }
            counter++;
        }
   }catch(Exception e){}

   return converted;
}

/** Takes a GSM-alphabet string and converts it to a normal one.*/
public String convertToAscii(String s)
{
    String converted = new String();

    for(int i = 0 ; i < s.length(); i = i+2)
    {
        String sub = s.substring(i,i+2);

        if(sub.compareTo("0f")==0)
        {
            converted = converted.concat("å");
        }
        else if(sub.compareTo("0e")==0)
        {
            converted = converted.concat("Å");
        }
        else if(sub.compareTo("7b")==0)
        {
            converted = converted.concat("ä");
        }
        else if(sub.compareTo("5b")==0)
```

```
            {
                converted = converted.concat("Ä");
            }
            else if(sub.compareTo("7c")==0)
            {
                converted = converted.concat("ö");
            }
            else if(sub.compareTo("5c")==0)
            {
                converted = converted.concat("Ö");
            }
            else
            {
                converted = converted.concat(new Character(
                                    (char)Integer.parseInt(sub, 16)).toString());
            }
        }
        return converted;
    }
}
```

A.2.2 Service.java

```
/*********************************************
*  (c) Copyright Telia Mobile AB, created 2000 *
*      All rights reserved                     *
*********************************************/

/** Class representing a service.*/
class Service
{
    /** The name of the service.*/
    private String serviceName = new String();

    /** The questions the service needs answers for.*/
    private String questionText = new String();

    /** The maximum length of a service name.*/
    private static int MAX_NAME_LENGTH = 15;

    /** Takes a string and splits it so the name is put serviceName
        and the questions is put in the questionText.*/
    Service(String s)
    {
        int star = s.indexOf('*');
        int square = s.indexOf('#');

        if(star == -1)
        {
            if(square == -1)
            {
                serviceName = s;
                questionText = null;
            }
            else
            {
                serviceName = s.substring(0, square);
                questionText = s.substring(square, s.length());
            }
        }
        else
        {
            if(square == -1)
            {
                serviceName = s.substring(0, star);
                questionText = s.substring(star, s.length());
            }
            else
            {
                serviceName = s.substring(0, Math.min(star, square));
                questionText = s.substring(Math.min(star, square), s.length());
            }
        }
    }

    /** Returns the service name.*/
    public String getServiceName()
    {
        if(serviceName.length() > MAX_NAME_LENGTH)
        {
            serviceName = serviceName.substring(0, MAX_NAME_LENGTH);
        }
        return serviceName;
    }
```

```
    /** Returns the question text.*/
    public String getQuestionText()
    {
        return questionText;
    }
}
```

A.2.3 Node.java

```
/*********************************************
*  (c) Copyright Telia Mobile AB, created 2000 *
*      All rights reserved                     *
*********************************************/
import javax.swing.*;
import javax.swing.tree.*;
import java.util.*;

/** The class used to represent a node.*/
public class Node extends DefaultMutableTreeNode
{
    /** Represents the node´s infotype, selectItem or prepareSMS.*/
    private int infoType;

    /** The number of this node´s parent.*/
    private int treeRef;

    /** The name of the node.*/
    private String nodeName;

    /** Tells whether the node allows children or not.*/
    private boolean isOkWithChildren;

    /** The nodes number in the tree.*/
    private int nodeNumber;

    /** Maximum length for the node´s name.*/
    private static int MAX_NAME_LENGTH = 15;

    /** Takes a string and a value representing if the node
        accepts children or not, and constructs a node.*/
    Node(String name, boolean value)
    {
        super(name);
        nodeName = new String(name);
        isOkWithChildren = value;
        if(isOkWithChildren)
        {
            infoType = 1;           //SelectItem
        }
        else
        {
            infoType = 2;           //PrepareSMS
        }
    }

    /** Returns whether the node accepts children or not.*/
    public boolean isOkWithChildren()
    {
        return isOkWithChildren;
    }

    /** Sets the node´s number.*/
    public void setNodeNumber(int number)
    {
        nodeNumber = number;
    }

    /** Returns the node´s number.*/
    public int getNodeNumber()
    {
        return nodeNumber;
    }
```

```
    /** Sets the node´s infotype.*/
    public void setInfoType(int it)
    {
        infoType = it;
    }

    /** Returns the node´s infotype.*/
    public int getInfoType()
    {
        return infoType;
    }

    /** Sets the node´s tree ref.*/
    public void setTreeRef(int tR)
    {
        treeRef = tR;
    }

    /** Returns the node´s tree ref.*/
    public int getTreeRef()
    {
        return treeRef;
    }

    /** Sets the node´s name.*/
    public void setNodeName(String dt)
    {
        nodeName = dt;
        setUserObject(dt);
    }

    /** Returns the node´s name.*/
    public String getNodeName()
    {
        if(nodeName.length() > MAX_NAME_LENGTH)
        {
            nodeName = nodeName.substring(0, MAX_NAME_LENGTH);
        }
        return nodeName;
    }

    /** Returns a string containing the node´s infotype and treref
        in string format.*/
    public String getDataToSend()
    {
        String returnString = new String(Integer.toString(infoType)+",");
        returnString = returnString.concat(Integer.toString(treeRef)+",");
        return returnString;
    }
}
```

B Lista över förkortningar

API – Application Programmers Interface

EEPROM – Electrical Eraseable Programmable Read Only Memory

GSM – Global System for Mobile communications

HTML – Hypertext Markup Language

RAM – Random Access Memory

SIM – Subscriber Identity Module

SIMAT – SIM Application Toolkit

SMS – Short Message Service

TLV – Tag, Length, Value

TMUKM – Telia Mobile Utveckling, Kompetensgrupp Mobile Internet lab

US – Unified Services

WAP – Wireless Application Protocol

WML – Wireless Markup Language