Kommunikation mellan mobiltelefoner och noder i IP-baserade nätverk

IBRAHIM AYATA

KTH Numerisk analys
och datalogi

Examensarbete
Stockholm, Sverige 2005

TRITA-NA-E05162

Numerisk analys och datalogi
KTH
100 44 Stockholm

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

Kommunikation mellan mobiltelefoner och noder i IP-baserade nätverk

IBRAHIM AYATA

TRITA-NA-E05162

Examensarbete i datalogi om 20 poäng
vid Programmet för datateknik,
Kungliga Tekniska Högskolan år 2005
Handledare på Nada var Sinna Lindquist
Examinator var Yngve Sundblad

Kommunikation mellan mobiltelefoner och noder i IP-baserade nätverk

Sammanfattning

Detta examensarbete har genomförts på CID, Center for User Oriented IT-Design, vid Kungliga Tekniska Högskolan i Stockholm. Arbetet påbörjades januari 2003 inom EU-projektet InterLiving, som pågick åren 2001-2003. InterLiving-projektets generella mål var att tillsammans med familjer utveckla teknik som underlättar kommunikation mellan generationer.

Examensarbetets mål var att undersöka möjligheterna att utveckla ett plattformsoberoende system där en mobiltelefon kopplas upp mot en Internetuppkopplad dator. Arbetet resulterade i en prototyp bestående av en klient- och en serverapplikation. Klientapplikationen, som går att exekvera från en mobiltelefon av märket Motorola Accompli A008, kopplar via GPRS upp sig mot serverapplikationen, som finns publicerad på en webapplikationsserver på en av Nadas datorer.

Under examensarbetets genomförande lanserades den tredje generationens mobiltelefoni 3G. Med anledning av det har även en kort analys genomförts för att få reda på vilka möjligheter detta medför för en eventuell utvidgning av den framtagna prototypen.

Prototypen i dess nuvarande form inte är så användbar då GPRS baserad kommunikation i praktiken medför textbaserade meddelanden. Dock kan en vidareutveckling inom ramarna för 3G resultera i att systemet kan tillhandahålla mer avancerade tjänster baserade på såväl ljud- som bildkommunikation.

Communication between cellular phones and nodes in IP-based networks

Abstract

This Master's project has been performed at CID, Center for User Oriented IT-Design, at the Royal Institute of Technology (KTH) in Stockholm. The Master's project started in January 2003 within the EU-project InterLiving, which went on during 2001-2003. The general objective of the InterLiving project was to develop technology in cooperation with families in order to facilitate the communication between generations.

The objective of the Master's project was to investigate the possibilities to develop a platform independent system where a cellular phone should be able to connect to an Internet connected computer. The work resulted in a prototype consisting of a client application and a server application. The client application, which is executable from a Motorola Accompli A008 cell phone, connects to the server application through GPRS. The server application is published on a web application server which is installed on one of the computers at NADA, KTH.

During the implementation of the Master's project, the third-generation wireless, 3G, was introduced on the market. With respect to this a short analysis has been performed in order to be aware of the possibilities it convey for a potential extension of the prototype.

The present edition of the prototype is not so useful since GPRS based communication results in text based messages. However, an extension within the third-generation wireless technology may result in that the system can provide more sophisticated multimedia based services.

Innehållsförteckning

Inledning

Bakgrund

Detta examensarbete har genomförts på CID, Center for User Oriented IT-Design, vid Kungliga Tekniska Högskolan i Stockholm. Arbetet har i huvudsak utförts under våren 2003 inom EU-projektet InterLiving[1], som pågick åren 2001-2003. Projektets generella mål är att tillsammans med familjer utveckla teknik som underlättar kommunikation mellan generationer.

Projektet samarbetar med INRIA, Institut Nationale de Recherche en Informatique (Univertsité de Paris-Sud) i Frankrike samt The Human-Computer Interaction Lab at the University of Maryland USA.

Den svenska delen av projektet har samarbetat med tre olika familjer. Varje familj består av två eller fler hushåll, samtliga boende inom Storstockholm. Respektive familj har uttryckt önskemål om att kunna kommunicera via ett system mellan de olika hushållen. Man ska även kunna skicka bilder, ljud, text etc. till sitt eget hem.

En av familjerna har uttryckt önskemål om att kunna nå hemmet via en mobil enhet då de ofta befinner sig hemifrån. Exempelvis har en mamma nämnt att hon gärna skulle vilja kunna skicka meddelanden till hemmets dörr för att påminna om saker såsom barnens tandläkartider, tala om att hon blir sen hem m.m.

Syfte och mål

Syftet är att, med befintlig teknik, försöka utveckla en prototyp som stöder hushållens behov av kommunikation. Området för kommunikationssystem är stort, varför arbetet avgränsas till att försöka utveckla en prototyp för ett mobilt kommunikationssystem där de mobila enheterna utgörs av mobiltelefoner. Examensarbetet har blivit tilldelad en mobiltelefon, modell Motorola Accompli A008. Telefonen har stöd för Java och är utrustad med ett Comviq kort.

Ett av målen är att utveckla en prototyp för ett system som sammankopplar Motorola-telefonen med en dator som är uppkopplad mot Internet. Ytterligare ett mål är att få prototypen så plattformsoberoende som möjligt. Anledningen till detta är att prototypen bör kunna köras på en mobiltelefon oavsett tillverkare. Vidare är en anledning att hushållen kan ha datorer med varierande plattformar och operativsystem.

Med dessa mål i sikte, har följande frågor identifierats som intressanta att söka svar på:

- Hur sker överföring av data mellan en mobiltelefon och en Internetuppkopplad dator?
- Då Motorola-telefonen har stöd för Java, går det att utveckla en prototyp i Java som passar projektet?
- Går det att skicka ljud- eller bildfiler mellan mobiltelefonen och Internetuppkopplade datorer utan att det tar för lång tid?

[1] För närmare information, se http://interliving.kth.se

Rapportens upplägg

Rapporten inleds med en introduktion i den teoretiska bakgrund som rör examensarbetet. Den teoretiska bakgrunden omfattar kommunikation inom datornät samt systemutveckling i Java. Därefter följer ett avsnitt som redogör för de metoder som använts under arbetets gång. Nästföljande avsnitt beskriver den prototyp som utvecklats, varefter rapporten avslutas med slutsatser dragna av det genomförda arbetet.

Teori

Datorkommunikation

Prototypen som examensarbetet avser utveckla är ett kommunikationssystem bestående av dels en mobiltelefon och dels en dator kopplad till ett globalt datornätverk. Med anledning av detta ges inledningsvis en beskrivning av de vanligast förekommande typerna av datornät. Vidare ges generella beskrivningar av de tekniska standarder som förknippas med mobiltelefoner.

Datornät

Kretskopplade nät är en typ av nätverk där fysiska resurser allokeras utmed sträckan mellan två ändpunkter som kommunicerar med varandra. Ett exempel på denna typ av kommunikation är sedvanlig kommunikation via telefon. Då ett samtal kopplas reserverar telefonbolaget utrymme för det samtalet, vilket i praktiken innebär att andra samtal inte kan använda samma resurser under tiden samtalet pågår.

En annan typ av nätverk är paketförmedlade nät. Ett meddelande som ska skickas över ett paketförmedlat nät bryts ner i flera små dataenheter som kallas paket. Respektive paket bär information om mottagarens adress, och kan därför behandlas separat. Det medför att varje paket kan färdas längs olika vägar genom nätverket, från avsändare till mottagare. Detta i sin tur medför att paketen kan anlända mottagaren i en annan ordning än de skickades. Tekniken att dela upp och skicka data i en mängd mindre paket möjliggör att samma fysiska resurser kan delas mellan flera användare i nätverket. Denna typ av kommunikation kallas *förbindelselös* kommunikation.

I likhet med kretskopplade nät förekommer även allokering av fysiska resurser i paketförmedlade nät. Exempelvis finns s.k. X.25 nätverk som stöder virtuell kretskoppling. En virtuell kretskoppling innebär att fysiska resurser allokeras mellan två ändpunkter. Andra virtuella kretskopplingar kan dock, till skillnad från rent kretskopplade nät, använda samma resurser samtidigt. Den virtuellt kretskopplade tekniken refereras till som *förbindelseorienterad* kommunikation. [STALLINGS 2000, CSNW, PSNW]

Mobila nätverk

GSM – Global System for Mobile Communication

En vanligt förekommande term i samband med mobiltelefoner är GSM. Det är ett digitalt mobiltelefonsystem som används i såväl Europa som andra delar av världen. Mobiltelefoner fungerar som kombinerade radiosändare och radiomottagare. Samtalen till och från mobiltelefoner skickas digitalt som radiovågor mellan mobiltelefonen och en basstation. I Sverige verkar GSM i frekvensbanden 890-960 MHz[2] och 1710-1880 MHz. Frekvenserna delas upp i block om 200 kHz[3] där varje block kallas för en kanal. Varje kanal kan hantera sju samtal, vilket betyder att flera mobiltelefoner kan dela på samma kanal. Respektive mobiltelefon skickar och tar emot sina datapaket i små stötar,

[2] Megahertz

[3] Kilohertz

s.k. *tidsluckor*. Överföringskapaciteten för varje tidslucka är 9600 bps[4] men kan beroende på operatör komma upp i 14400 bps. Totalt delas mediet, som i detta fall är luften, in i N stycken tidsluckor. För GSM är N=8, varav sju används för samtal och den åttonde för styrsignaler.

Tekniken bakom sändning och mottagning av data via tidsluckor kallas *multiplexing*. Det innebär kort att flera dataströmmar sätts ihop och skickas iväg som en signal. Multiplexingtekniken där data skickas parallellt i olika tidsluckor förkortas *TDM*, vilket står för *Time Division Multiplexing*. GSM använder en variant av en liknande tidsbaserad multiplexingteknik som kallas *TDMA*, vilket är en förkortning för *Time Division Multiple Access*. [GSM1, GSM2, GSM3, MPLEX, TDMA]

GPRS – General Packet Radio Service

GPRS är en paketbaserad trådlös tjänst för datakommunikation. Tekniken är utformad som ett komplement till de befintliga kretskopplade nätverken som vanligtvis används inom GSM och TDMA nätverk. Via GPRS kan datapaket från flera enheter i systemet distribueras. Detta medför en effektivare användning av bandbredden. I teorin kan en GPRS-uppkoppling nå överföringshastigheter på upp till 114 kbps om alla åtta tidsluckorna utnyttjas på en gång. Vidare är det i teorin billigare med GPRS-tjänster än med kretskopplade tjänster eftersom det nätverk som data ska slussas genom endast belastas då data överförs. I praktiken är dock överföringshastigheterna lägre än det teoretiskt möjliga. Detta beror dels på trafikmängden i nätverket och dels på att mobiltelefoner använder olika antal tidsluckor för att skicka och ta emot data.

Med anledning av att GPRS använder samma protokoll som Internet kan GPRS-nätverk ses som en delmängd av Internet. GPRS-enheterna kan sägas utgöra noder med potentiella egna IP-adresser. I det fall ett GPRS-nät ska läggas till ett GSM-nät eller ett TDMA-nät så måste modulerna GGSN (Gateway GPRS Service Node), SGSN (Serving GPRS Service Node) och PCU (Packet Control Unit) läggas till infrastrukturen.

GGSN-noden agerar som en port mellan GPRS-nätet och publika datornät som exempelvis Internet. De kopplar även upp mot andra GPRS-nät för att möjliggöra s.k. *roaming*, vilket innebär att en användare kan få uppkoppling till nätet oavsett geografiskt läge. SGSN-noden tillhandahåller utdelning av datapaket enligt ett bestämt schema till de användare som geografiskt befinner sig inom dess verksamhetsområde. PCU-enheten ansvarar för de radiorelaterade aspekterna då GPRS-nätet är kopplat mot ett GSM-nät. En PCU-enhet kan i vissa fall även refereras till som en BSC (Base Station Controller). Vanligt är då att den sitter kopplad mellan en SGSN-nod och en BTS (Bast Transceiver Station). En BTS utgörs vanligen av en basstation. Dess huvuduppgift är att ha kommunikation med mobila enheter via radio. Om basstationen är den enhet som tar hand om den aktuella kommunikationen, så är BSC den enhet som ansvarar för det som sker. Med det menas exempelvis att BSC bestämmer när basstationen ska sända. [GPRS, PCU, HELLSTRÖM] Bild 1 återger en principiell skiss över GPRS-baserad kommunikation mellan mobiltelefoner och noder i IP-baserade nätverk såsom Internet.

[4] Bits per second. Mängden data mäts i Byte där 1 Byte är 8 bits. Sålunda motsvarar 9600 bps 1200 Bytes per sekund.

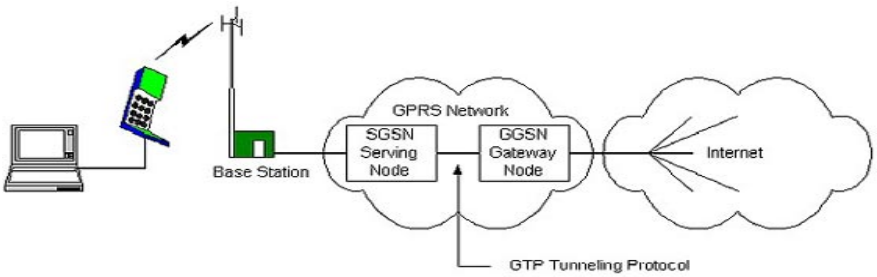

Bild 1: Principiell modell över GPRS-baserad kommunikation.[5]

UMTS – Universal Mobile Telecommunications System

Teknikstandarden UMTS benämns vanligen som Europas variant av den tredje generationens kommunikationssystem. Den använder ett större frekvensspektrum än föregående mobilsystem. Med UMTS sänds alla samtal samtidigt och de skiljs åt från varandra med en kod som är unik för respektive samtal. Det möjliggörs med hjälp av en moduleringsteknik som kallas WCDMA (Wideband Code Division Multiple Access). Principen är att ett samtal söker signaler motsvarandes dess egen kod, varpå all övrig information kan sorteras bort som brus. [UMTS]

UMTS-meddelanden är paketbaserade. Med tekniken kan text, digitaliserat ljud, video och multimedia skickas i hastigheter på upp till 2 Mbps[6]. UMTS möjliggör sålunda en bredbandsteknik för mobiltelefoni, vilket innebär att stora mängder data kan skickas utan att det upplevs ta för lång tid.

[5] Bilden hämtad den 24 april 2003 från
http://www.mobilecommstechnology.com/projects/gprs/gprs2.html

[6] Megabit per sekund

Systemutveckling

Mobiltelefonen som examensarbetet blivit tilldelad har stöd för Java. Detta i kombination med att system utvecklade i Java kan köras på olika plattformar har lett fram till beslutet att försöka utveckla programvaran till prototypen i Java. I de två inledande avsnitten ges en översiktsbild av Java. Därefter ges en mer ingående beskrivning av J2ME, som är Java för mindre enheter såsom mobiltelefoner. De två avslutande avsnitten beskriver Javas servletteknik respektive JDBC (Java Database Connectivity) vilket är tekniken bakom uppkopplingar mot databaser.

Java – en kort inledning

En grundläggande idé inom Javautveckling är att samma programvara ska kunna exekvera på olika typer av enheter. Sedan den kommersiella lanseringen 1995 är Java en teknik som vuxit i popularitet just på grund av detta. Program baserade på Java brukar benämnas som plattformsoberoende. Uttrycket syftar på att det inte spelar någon roll ifall programmet körs på en dator, telefon eller en TV. Det spelar heller ingen roll vilket operativsystem som körs på den aktuella enheten. Det som krävs är att enheten som ska köra programmet har Java-stöd installerat.

Med stöd för Java avses att ett JVM (Java Virtual Machine) finns installerat. JVM brukar benämnas som den virtuella javamaskinen, och kan betraktas som en tolk. Det är den komponent som möjliggör plattformsoberoendet då den översätter instruktioner skrivna i Java till skräddarsydda instruktioner för den maskin programmet körs på [JAVA]

I praktiken går det till så att utvecklaren skriver javakod i ett godtyckligt texthanteringsprogram. Den skrivna koden sparas som en fil med ändelsen *java*. Därefter kompileras koden med en kompilator såsom exempelvis *javac*[7]. När kompilatorn arbetat klart har den genererat en fil med ändelsen *class*, vilket är en fil som nästan är körbar. Denna fil kan flyttas mellan olika plattformar. Anledningen till att den endast är nästan körbar är att kompilatorn kompilerar de delar som är plattformsoberoende. De resterande delar som är plattformsspecifika kompileras under körning av den virtuella javamaskin som finns installerad på den enhet som programmet ska köras på.

Mer tekniskt uttryckt översätter JVM bytekod till maskinspecifik kod under körning. Förloppet beskrivet ovan kan kort sammanfattas i följande steg:

- Skriv kod och spara som fil med ändelsen *java*
- Kompilera *java*-filen med en kompilator, exempelvis *javac*
- Flytta den av kompilatorn genererade *class*-filen till den enhet som programmet ska köras på
- Kör *class*-filen på enheten. Den virtuella javamaskin som finns på enheten kompilerar under körning de plattformsspecifika instruktionerna.

[7] Javac kompilatorn följer med utvecklingspaketet som finns att ladda ner gratis från bl.a. http://java.sun.com

Javaplattformen

Javaplattformen är indelad i tre versioner, J2EE (Java 2 Enterprise Edition), J2SE (Java 2 Standard Edition) och J2ME (Java 2 Micro Edition). J2EE är anpassad för servrar och affärsföretag, medan J2SE är anpassad för arbetsstationer och PC: s. J2ME är det nyaste tillägget i Javas utvecklingsplattform. Målgruppen är enheter med små resurser som exempelvis mobiltelefoner. Bild 2, som är hämtad från RIGGS 2001 SID 8, återger en översikt av Javaplattformen.

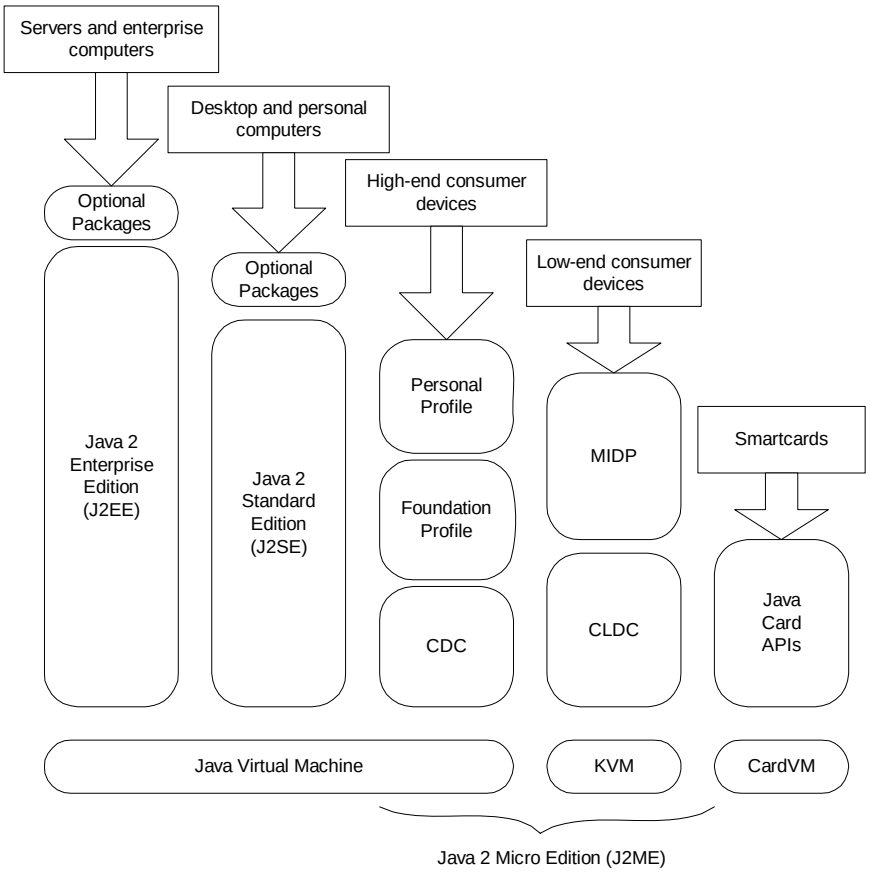

Bild 2: En översikt av Javaplattformen.[8]

J2ME – Java 2 Micro Edition

Den huvudsakliga målgruppen J2ME riktas mot är enheter med små resurser som exempelvis mobiltelefoner. Tidigare har det inte varit möjligt att köra externa program på en mobiltelefon. Användaren var begränsad till att använda telefonen inom ramarna för telefonens ursprungliga fabrikskonfiguration. Numera är det möjligt att ladda ner, installera och köra program utvecklade i Java på sin mobiltelefon.

Konfigurationer

Som utvecklare måste man tänka på vilken typ av enhet ett program utvecklas för. Parametrar såsom grafisk upplösning, minnesresurser och processorkraft varierar mellan olika typer av enheter. Med anledning av detta har begreppet *konfiguration* införts. Under våren 2003, då examensarbetet huvudsakligen genomfördes, fanns två konfiguratio-

[8] Bilden tagen ur Riggs et al. Sidan 8

ner definierade: CDC (Connected Device Configuration) och CLDC (Connected Limited Device Configuration). Tabell 1 listar karakteristika för enheter inom respektive konfiguration.

Tabell 1: Karakteristika för CDC respektive CLDC

CDC	CLDC
Minst 512 k minne för att köra Java	128 k minne för att köra Java
Minst 256 k minne för minnesallokering under körning	32 k minne för minnesallokering under körning
Nätverksuppkoppling – fast uppkoppling med hög bandbredd	Nätverksuppkoppling – trådlös med låg bandbredd och periodiskt avbruten
	Begränsat användargränssnitt
	Strömförsörjning via batteri

En konfiguration definierar en javaplattform för en generell mängd enheter. Vidare är den nära förknippad med en virtuell javamaskin. Förutom översättning av bytekod till maskinspecifik kod ansvarar även den virtuella javamaskinen för tillhandahållande av säkerhet, minneshantering, exekvering etc.

För CDC har den virtuella javamaskinen samma specifikation som J2SE. För CLDC finns en speciell virtuell maskin utvecklad som kallas KVM (K Virtual Machine). Bokstaven K i namnet står för kilo och syftar på den lilla minnesbudget som enheterna har och som mäts i tiotals kilobyte. KVM formgavs för att klara av de speciella krav som resursbegränsade enheter ställer. Det har resulterat i att KVM endast kräver 40-80 kilobyte minne och 20-40 kilobyte heap. Vidare klarar den av att köra på en 16-bits processor som har en klockfrekvens på 25 MHz.

MIDP – Mobile Information Device Profile

Såväl handdatorer som mobiltelefoner uppfyller kriterierna för CLDC konfigurationen. Dock finns skillnader mellan dessa två typer av enheter. Ett exempel är att de kan ha olika upplösningar på skärmen. För att bibehålla flexibiliteten även då de tekniska förutsättningarna ändras, finns konceptet *profil* framtaget. En profil tillhandahåller verktyg för utvecklare som skriver program avsedda för specifika enheter.

Den profil som är aktuell gällande utveckling av program till mobiltelefoner kallas MIDP. En profil definierar API (Application Program Interface) för komponenter i användargränssnittet, inmatningshanteringen, händelsehanteringen etc. Definitionen beaktar parametrar som skärmupplösning och begränsat minne.

För enheter som implementerar MIDP gäller att följande krav på hårdvara uppfylls:

- Skärmupplösning på minst 96*54 pixlar
- Inmatning från användare måste ske från ett av följande alternativ:
    - Tangentbord avsedda för inmatning med en hand, typiska för mobiltelefoner
    - Tangentbord avsedda för inmatning med två händer, s.k. QWERTY tangentbord
    - En tryckkänslig skärm

- 128 k icke-flyktigt minne[9] för körning av MID-komponenter
- Minst 8 k icke-flyktigt minne för program att spara ihållande information såsom programinställningar och data
- 32 k icke-flyktigt minne för körning av Java
- Trådlös nätverksuppkoppling

Generellt ska så många enheter som möjligt kunna implementera MID-profilen. Dock kan operativsystemen för olika enheter variera kraftigt. Med anledning av detta har kraven på mjukvara satts så minimalt som möjligt. De krav som är satta är som följer:

- Enhetens operativsystem måste tillhandahålla minimal schemaläggning, undantagshantering samt behandla avbrott.
- Det måste finnas tillräckliga möjligheter för att köra en virtuell javamaskin.
- Givet att en användare matar in data från ett godkänt inmatningsalternativ enligt ovan, måste mjukvaran acceptera inmatningen och vidarebefordra informationen till den virtuella javamaskinen.
- Mjukvaran måste stöda skrivning av bitmappad grafik till skärmen.
- För stöd av lagring av ihållande information måste det finnas möjlighet att läsa från och skriva till det icke-flyktiga minnet. Observera dock att det inte finns några krav definierade för eventuella filsystem.
- Det måste finnas åtkomst till enhetens nätverksfinesser. Speciellt gäller det möjligheten att skriva och läsa information via ett trådlöst nätverk.

Program skrivna för mobila enheter som implementerar CLDC/MIDP kallas för MIDlets. En eller flera MIDlets som packas ihop i en JAR-fil (Java Archive File) kallas för MIDlet-serie[10].

En JAR-fil är en fil som innehåller all nödvändig information som behövs för att ett program ska kunna köras. Exempelvis kan ett program skrivet i Java bestå av flera class-filer. Vanligt förekommande är då att class-filerna packas ihop till en JAR-fil. Ytterligare en fil som är relevant i samband med MIDlets som packas ihop till en JAR-fil är den så kallade JAD-filen (Java Application Descriptor file). Syftet med en JAD-fil är att den tillhandahåller information om de MIDlets som finns i JAR-filen [MUCHOW 2002]

GCF – Generic Connection Framework

Många av klasserna och gränssnitten som återfinns i biblioteken `java.io` och `java.net` kräver mer prestanda än vad exempelvis en mobiltelefon har. För dessa ärenden finns GCF utvecklat, vilket är ett ramverk för I/O och nätverksärenden. Målet med GCF är att tillhandahålla en delmängd av J2SE som överensstämmer med de begränsningar som finns på enheter som implementerar MIDP. [MUCHOW 2002]

Det grundläggande är att all uppkoppling sker via klassen `Connector`. Klassen kan skapa godtycklig typ av anslutning såsom exempelvis fil, http, datagram etc. Klassen tillhandahåller en metod som heter `Open`, vars struktur är som följer:

```
Connector.Open(”file://godtyckligFil.txt”);

Connector.Open(“http://www.godtyckligWebbadress.com”);
```

[9] Icke-flyktigt minne kallas på engelska för non-volatile memory. Det är ett slags RAM-minne vars innehåll sparas då enheten stängs av. [NVMEM]

[10] Den engelska termen är MIDlet-suite

```
Connector.Open("socket://godtyckligIPadress:godtyckligPort");
```

Vid körning söker Connector efter lämplig klass som implementerar det efterfrågade protokollet. Detta sker via ett anrop till Class.ForName(). I det fall en lämplig klass hittas, returneras ett objekt som implementerar ett Connection-gränssnitt. Såväl Connector som Connection är definierade i CLDC.

Säkerhet

Då J2ME var nytt och MIDP 1.0 var den profil som fanns tillgänglig för implementation av MID-applikationer fanns inte några större möjligheter att implementera säkerhet. I och med lanseringen av MIDP 2.0 har möjligheterna dock ökat.

Version 2.0 av MIDP har en säkerhetsmodell från ände till ände som bygger på öppna standarder. Exempelvis stöds HTTPS, som är http över SSL (Secure Socket Layer). SSL är ett socketprotokoll som krypterar data som skickas över nätverket, samt tillhandahåller verifiering för en sockets ändpunkter. Vidare skyddar så kallade säkra domäner i MIDP 2.0 mot obehörigt tillträde för program via MIDlet-serier på enheten. Det förvalda värdet för MIDlet-serier är att de inte är betrodda. Därav anvisas de till ej betrodda domäner som förhindrar uppkoppling till privilegierad funktionalitet. För att kunna få privilegierat tillträde måste en MIDlet-serie anvisas till specifika domäner som finns definierade på mobiltelefonen. Vidare måste MIDlet-serien vara korrekt signerad med avseende på X.509 certifikat, som kort kan beskrivas som ett elektroniskt identitetskort. [MDIP20, X509]

Media i MID-applikationer

MMAPI (Mobile Media API) är ett gränssnitt för uppspelning och inspelning av ljud- och bilddata. I MIDP 2.0 ingår ABB (Audio Building Block) som är en del av MMAPI. Med ABB är det möjligt för programmerare att lägga till toner, tonsekvenser och ljudfiler i WAV-format till MID-applikationen. Det krävs dock att den mobila enheten har stöd för tongenerering och uppspelning av WAV-filer. På enheter som har en MMAPI implementation kan programmerare även lägga till multimedia innehåll såsom strömning av bilddata, även kallat *video streaming*.

Bifogat rapporten finns exempelkod för ett program, en MIDlet, som spelar upp ljud. Exemplet återfinns i *Bilaga 5 – Källkod för AudioExample*.

Servlets

Javas servletteknik tillhandahåller mekanismer för webutvecklare att såväl utvidga funktionaliteten på webservrar som åtkomst av befintliga system. En servlet kan liknas vid en applet som exekverar på serversidan. Det är ett sedvanligt javaprogram som kompilerats med en javakompilator. Den körbara filen, class-filen, läggs upp på serversidan varpå den finns tillgänglig för exekvering närhelst den anropas. Anrop kan ske från en fil som kan utgöras av exempelvis html-, jsp-, eller wml-filer. Servleten exekverar via den virtuella javamaskin som laddas på servern. Med servlets kan programmerare bland annat bygga dynamiska websidor och koppla mobila program till databaser. En fördel med servlets är att de tillhandahåller en komponentbaserad och plattformsoberoende metod för utveckling av webbaserade program.

Dynamiska webbaserade program som kopplas upp mot databaser kan även skrivas i andra språk såsom exempelvis CGI/Perl. En nackdel med program skrivna i Perl är

dock att en ny process startar varje gång programmet anropas. Givet att ett perlprogram anropas X gånger, så laddas koden för programmet till minnet X gånger.

För servlets gäller att den virtuella javamaskinen är den enda process som exekverar oavsett hur många gånger en servlet anropas. Respektive anrop av servleten hanteras som en variant av en Java-tråd. Detta medför att en och samma servletklass ligger laddad i minnet oavsett antal gånger den anropas. Med andra ord är servlets effektivare vid hög belastning. [HELLSTRÖM 2003]

JDBC – Java Database Connectivity

JDBC är en API-specifikation för uppkoppling av javaprogram mot databaser. Det består av två gränssnittsnivåer. Dels finns en mängd objektorienterade klasser definierade som programmerare kan använda vid konstruktion av exempelvis förfrågningar mot databasen, och dels finns ett API för en JDBC-hanterare.

Förfrågningar mot databaser kan skrivas i SQL (Structured Query Language). JDBC liknar motsvarigheten från Microsoft, ODBC (Open Database Connectivity). Med ett s.k. bryggprogram kan JDBC-gränssnittet användas för att få kontakt med databaser vida ODBC-gränssnittet. Detta är dock ingenting som en programmerare behöver kunna i detalj då JDBC-hanteraren sköter kommunikationen med produktspecifika drivrutiner såsom JDBC/ODBC-bryggan samt nätverksdrivrutinen i det fall javaprogrammet körs i nätverksmiljö. Kod som identifierar en databas kan se ut enligt följande:

```
jdbc:odbc//www.adressen.com:1234/databasfil
```

JDBC tillhandahåller ett enkelt sätt för programmerare att koppla javaprogram mot olika typer av databaser.

Metod

Här redogörs för hur examensarbetet genomförts. Inledningsvis presenteras en modell som beskriver arbetsgången. Därefter ges en mer ingående beskrivning av respektive fas.

Modell

Den inledning som bäst beskriver hur examensarbetet genomförts är den så kallade vattenfallsmodellen. Liknelsen med vattenfall kommer från den teoretiska modellen enligt bild 3. Modellen kännetecknas av olika faser som fortlöpande avlöser varandra. Den inledande fasen ritas högst upp och de efterföljande faserna längre och längre ner. När en fas är avslutad påbörjas nästa, och så fortgår arbetet till dess att sista fasen genomförts. De olika faserna i detta arbete utgörs av *Inledning*, *Informationssökning*, *Inläsning*, *Utveckling* och *Dokumentation*.

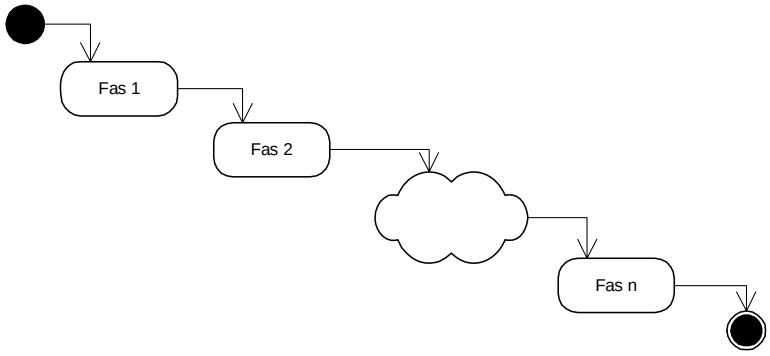

Bild 3: Vattenfallsmodellen

Inledning

Inledningsfasen bestod dels i att bli tilldelad en dator på vilken examensarbetet skulle genomföras, och dels i att bli tilldelad en arbetsplats på institutionen. När dessa två kriterier efter en dryg vecka blivit uppfyllda kunde själva arbetet påbörjas.

Informationssökning

Den första bok jag började läsa var en bok rörande programmering av mobila enheter[11] som jag fick låna av min handledare Björn Eiderbäck. Boken ifråga visade sig vara från 2001, varför den i sammanhanget betraktas som gammal då det händer mycket inom området mobila nätverk. Dock ansåg såväl min handledare som jag att den kunde fylla syftet att komma igång med inläsningsfasen samtidigt som sökandet av annan kompletterande litteratur kom igång.

[11] Riggs Roger et al.

Informationssökningsfasen fortsatte med sökande på Internet. Till en början sökte jag på nyckelord såsom mobila nätverk, applikationer för mobiltelefoner etc. på sökmotorn Google[12]. Sökresultaten ledde mig till Jukka Manners hemsida på Helsingfors Universitet. Inledningsvis såg hans hemsida intressant ut då den innehöll en hel del dokumentation kring hans arbeten rörande trådlösa och mobila nätverk. Dessvärre angränsade inte dokumenten mitt arbete i den omfattning jag hoppats. Sökandet gick därefter vidare till att innefatta sökning på specifika termer såsom GSM, WAP, GPRS, servlets, JDBC etc. Resultatet av detta blev en hel del surfande från vilket en del av informationen togs tillvara till den bakomliggande teorin för examensarbetet.

Under sökfasens gång fick jag även tips av Björn att delta på en exjobbspresentation som hölls av Mattias Hellström på KTH. Presentationen omfattade bl.a. programmering av javaapplikationer för mobiltelefoner. Hellströms rapport visade sig innehålla en hel del intressant och relevant information för mitt examensarbete. I referenslistan på Hellströms rapport återfinns en bok rörande utveckling av javaprogram för mobila enheter. Då boken verkade intressant införskaffade Björn den, varpå den utgjorde en del av den teori jag hade att läsa in.

Då jag under arbetets gång fastnat vid exempelvis terminologi, okända ord och ny teknik har information som löst problemen i huvudsak återfunnits på Internet. Exempel på detta är den information jag i arbetets slutskede sökte och fann svar på rörande tekniken UMTS som relaterar till 3G.

Inläsning

Inläsningsfasen var i huvudsak en inledande fas i arbetet. Så fort informationssökning resulterade i potentiell litteratur eller information som ansågs relevant så lästes det in. Den största delen av fasen avklarades innan utvecklingsfasen påbörjades, samt innan examination av den inlästa teorin genomfördes. Dock har inläsning även förekommit efter såväl utveckling som examinering av inläst teori. Främst har det då handlat om inläsning av teori som inte är ny kunskap, utan snarare kunskap som jag behövt aktualisera. Exempelvis har det varit kunskap rörande servlets och JDBC som jag kommit i kontakt med i andra föregående kurser under mina studieår på KTH.

Utveckling: Installation, konfiguration och test

Utvecklingsfasen kännetecknas främst av en arbetsmetod som i vardagligt tal benämns *trial and error*. Den går ut på att man via en avancerad gissning gör en ansats till att lösa ett aktuellt problem. Denna första ansats löser ofta inte hela problemet, utan snarare kommer man en bit på vägen. I nästföljande steg tar man reda på relevant information med utgång från det man kört fast vid. På så vis kan man göra nya avancerade gissningar och arbetet fortskrider framåt.

De mest tidskrävande momenten i utvecklingen av prototypen utgjordes av installations-, konfigurations- och testmomenten. Innan någon mjukvaruutveckling påbörjades lästes relevant teori in. Med relevant teori avses främst de två böcker skrivna av RIGGS respektive MUCHOW som jag förskaffat mig rörande programutveckling i Java för mobila enheter. Utgående från boken skriven av Muchow laddade jag ner de delar som

[12] http://www.google.se

behövdes varpå grunden för utveckling av klientdelen var lagd. Vidare behövdes en miljö på vilken serverdelen skulle kunna köras. Då jag sedan tidigare kände till att Tomcatservrar lämpade sig för publicering av Java-servletar, laddade jag ner en dylik distribution med avsikt att initialt installera den på den lokala datorn på vilken jag genomförde examensarbetet. Det är dock inte trivialt att konfigurera en Tomcatserver för första gången, varför det vid detta moment uppstod ett behov av informationssökning. Efter en hel del sökande på Internet fick jag en grundläggande förståelse för Tomcatserverns struktur, varpå jag kunde börja konfigurera servern. En korrekt konfiguration medför att servern klarar av att mappa ett anrop till servleten korrekt. Tomcatservern konfigureras genom att man kompletterar befintliga xml-filer i tomcatserverns struktur med information som rör den servlet som ska publiceras.

En enkel test som verifierade att konfigurationen var korrekt genomförd bestod i att jag först startade Tomcatservern varpå ett anrop till servleten gjordes via en webbläsare. Då svaret från servleten visades i webbläsaren visste jag att servleten fungerade som det var tänkt.

Nästa steg blev att testa klientapplikationen. Då koden för klienten var färdigskriven och jag försökte exekvera programmet uppstod ett problem. Det gick inte att köra koden då jag misslyckades med att skapa korrekta JAR- respektive JAD-filer. Med anledning av det kompletterade jag utvecklingsmiljön med J2ME wireless toolkit, kallat WTK, som är en integrerad miljö för utveckling av J2ME-applikationer. Via WTK genererades korrekta JAR- och JAD-filer varefter programmet blev körbart, varpå nästa steg blev att exportera och köra applikationen från Motorola-telefonen.

Vid första försöket att köra programmet från Motorola-telefonen uppstod dock problemet att inställningarna på telefonen inte var korrekta. För att klienten ska kunna koppla upp sig mot Comviqs GPRS portal krävs att man konfigurerar Motorola-telefonen med rätt information rörande IP-adress, portnummer och dylikt, varför jag ringde till Comviqs telefonsuppport. På supporten fick jag hjälp att lägga det sista problemet bakom mig då jag efter samtalet lyckades genomföra en testkörning där Motorola-telefonen kopplades upp till servleten enligt förväntan. De aktuella inställningarna återfanns jag senare även på Internet. De finns bifogade rapporten i bilaga 3, *Konfigurering av mobiltelefon*. Bilder på testkörningarna finns dokumenterade i bilaga 4, *Bilder på körning av prototyp*.

För närmare beskrivning av hur prototypens server- respektive klientdel är utvecklade hänvisas till avsnittet *Prototyp*.

Dokumentation

I huvudsak har dokumentation skett kontinuerligt under arbetets gång, när så har varit möjligt. Med detta menas att jag löpande nedtecknat information om exempelvis besökta sidor, nerladdade applikationer, installationsanvisningar etc. Innan något arbete påbörjades stod även en mall för slutrapporten färdig i form av ett rubrikskelett. Varefter arbetet fortskridit har information relaterad till specifika rubriker skrivits till. Vidare har namnen på rubrikerna fortlöpande justerats då tidigare val inte visat sig lämpliga varpå behovet av nya rubriker uppstått. Rapporten kan därför sägas ha varit levande under arbetets gång, vilket resulterade i att en första version stod klar hösten 2003. Därefter avstannade dessvärre dokumentationsarbetet. Den huvudsakliga anledningen till detta var att jag av ekonomiska skäl sökte mig ut i arbetslivet, vilket i sin tur ledde till

tidsbrist. Arbetet med dokumentationen har dock fortgått. Korrekturläsningar och ändringar har införts i dryga två års tid, och slutresultatet är denna slutrapport.

En detalj som kan nämnas är att den teoretiska inläsningsdelen som utgör examination av inläst litteratur i mångt och mycket är den information som ligger till grund för den teoretiska bakgrunden i denna rapport. Den teoretiska bakgrunden utgörs av avsnittet *Teori*.

Prototyp

I detta avsnitt redogörs för den prototyp som examensarbetet resulterat i. Inledningsvis ges en beskrivning av systemets struktur. Därefter följer en mer ingående beskrivning för systemets respektive delar, som utgörs av en server- och en klientdel. Avslutningsvis ges förslag till vidareutveckling av prototypen.

Beskrivning

Den framtagna prototypen baseras på den så kallade klient- servermodellen. Tanken med ett dylikt system är att serverdelen tillhandahåller tjänster som klienter kan nyttja. I teorin finns inga begränsningar på vare sig antalet klienter eller servrar. Ett system kan utgöras av godtyckligt antal av respektive sort. Den utvecklade prototypen består i skrivandes stund av en server och en klient. Bild 4 återger prototypens principiella struktur.

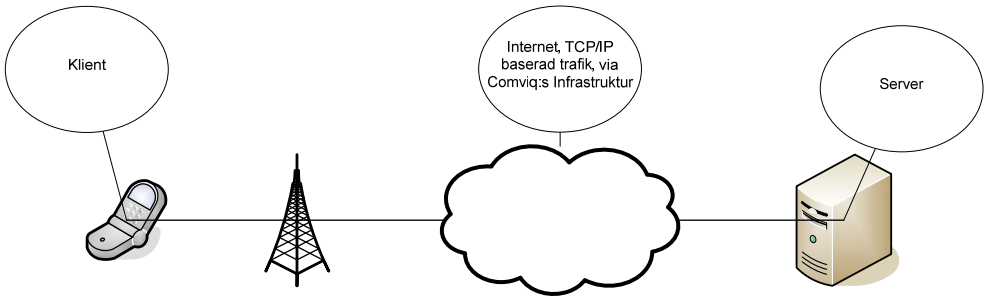

Bild 4: Skiss över prototypen.

Funktionaliteten i prototypen är minimal då syftet är att påvisa etablering av kontakt mellan två enheter snarare än användning av färdigutvecklade tjänster. En klient som kopplar upp sig mot servern får en hälsningsfras som svar vilket är ett bevis på att kontakt har etablerats.

Server

Serverprogramvaran är en Java-servlet, publicerad på en webapplikationsserver som finns installerad på mitt användarkonto på datorn `kentaur.nada.kth.se`. Webapplikationsservern är en Tomcatserver[13] som stöder exekvering av Java-servlets.

Strukturen för en servlet är att den ärver den abstrakta klassen `javax.servlet.http.HttpServlet`. En abstrakt klass kan sägas utgöra en mall för den kod som ska skrivas. Att låta en klass ärva från en annan klass innebär att klassen som ärver får tillgång till alla de egenskaper som ursprungsklassen har. Arv från en abstrakt klass innebär dessutom att den klassen som ärver måste implementera kod för de abstrakta avsnitten, dvs. för de delar som utgör en mall.

Tabell två listar de metoder/funktioner i klassen `HttpServlet` som enligt javadokumentationen [JAVADOC] finns tillhandahållna som mall.

[13] För närmare info, se http://jakarta.apache.org/tomcat/

Tabell 2: Abstrakta metoder i klassen HttpServlet

Metod	Implementation
doGet	Implementeras i de fall servleten ska stöda HTTP GET anrop
doPost	Implementeras i de fall servleten ska stöda HTTP POST anrop
doPut	Implementeras i de fall servleten ska stöda HTTP PUT anrop
doDelete	Implementeras i de fall servleten ska stöda HTTP DELETE anrop
init/destroy	Implementeras i de fall man vill styra de resurser som styr servletens livscykel
getServletInfo	Implementeras i de fall man vill att servleten ska kunna lämna information om sig själv

Prototypens servlet-del implementerar funktionalitet för metoderna `doGet` och `doPost`. Respektive metod tar ett `HttpServletRequest`- och ett `HttpServletResponse`-objekt som argument. Via `HttpServletRequest`-objektet kan information om anropande klient erhållas. Vidare är det tänkbart att den anropande klienten skickar information till servleten. Denna information kommer servleten åt via samma objekt. `HttpServletResponse`-objektet används av servleten för att skicka information tillbaka till den anropande klienten.

Den implementerade funktionaliteten för servern är i skrivandes stund en hälsningsfras, som ett bevis på att kontakt med servern har upprättats. Det är dock fullt tänkbart med ett mer dynamiskt beteende. Exempelvis skulle servern via JDBC kunna kopplas till en databas för att söka och returnera information enligt begäran från klienten. Dylik funktionalitet kan implementeras inom metoderna `doGet` och `doPost`.

För detaljerad kod rörande prototypens serverdel hänvisas till bilaga 2, *Källkod för prototypen*.

Klient

Programvaran för prototypens klientdel är en MIDlet. De essentiella delarna för klientdelen utgörs av användargränssnittet och kopplingsmekanismen för kommunikation med prototypens serverdel. För uppbyggnad av användargränssnitt tillhandahålls en mängd klasser av Javabiblioteket. Tabell 3 listar de klasser som använts.

Tabell 3:Klasser som använts i uppbyggnaden av klientens användargränssnitt

Klass	Funktionalitet
Command	Ett objekt som motsvarar ett kommando från användaren
StringItem	Ett textsträngsobjekt
TextField	Ett inmatningsfält i vilket användaren kan mata in information
Form	Ett formulär

Formuläret utgör själva huvudobjektet, och kan ses som en behållare som innehåller de övriga objekten. Inledningsvis skapas kommandon för att anropa servleten (`Connect`) samt för att avsluta applikationen (`Exit`). Vidare skapas ett textfältsobjekt i vilket adressen till prototypens servlet anges.

Dessa tre objekt läggs till i formulärobjektet som i sin tur visas i mobiltelefonens fönster då programmet körs. Till formuläret läggs även en s.k. lyssnare till. Lyssnaren reagerar på när användaren använder något av kommandoalternativen som implementerats. Om exempelvis användaren väljer att koppla upp sig mot servleten genom att välja alternativet `Connect` så kommer programmet via lyssnaren att exekvera den funktionalitet som finns implementerad för det valet. I det aktuella fallet kommer ett försök till uppkoppling mot den adress som finns specificerad i adressfältet att genomföras. I det fall uppkopplingen lyckas och ett svar erhålls, kommer svaret att presenteras som en textsträng i form av ett `StringItem`.

För detaljerad kod rörande prototypens klientdel hänvisas till Bilaga 2, *Källkod för prototypen*.

Förslag till vidareutveckling

Prototypen är att betrakta som en grund för ett kommunikationssystem. Ytterligare funktionalitet, av mer praktisk karaktär, kan implementeras. Innan eventuell vidareutveckling sker bör dock beslut tas för huruvida prototypen ska baseras på kommunikation via GPRS, eller om den ska modifieras till att stöda kommunikation inom ramarna för 3G. I texten som följer ges förslag på förbättrad arkitektur där exempelvis hänsyn tas till hantering av multipla klienter etc. Därefter ges förslag på hur prototypens funktionalitet kan utökas då kommunikation sker via GPRS. Avslutningsvis ges en kort översikt över de möjligheter som 3G medför för prototypen.

Förbättrad arkitektur

Arkitekturen i den framtagna prototypen är av enklast möjliga karaktär. Ingen hänsyn har tagits till parametrar såsom antal användare, säkerhet eller uppkoppling mot databas. Vid en eventuell vidareutveckling föreslås därför att man kompletterar den befintliga arkitekturen enligt följande punkter:

- Implementering av stöd för flera användare. Man bör vidareutveckla servern att parallellt kunna kommunicera med multipla klienter via exempelvis Java-trådar.
- Man bör konstruera ett lager, mellan servleten och databasen, som sköter kommunikationen med databasen. Tydligt definierade standardmetoder för att hämta, uppdatera eller lagra information i databasen tillhandahålls av detta lager. Förutom att minimera antalet rader kod i servleten underlättas även möjligheten att vidareutveckla funktionaliteten samt att det blir lättare att felsöka i det fall något inte går som det är tänkt. Att tänka på är även att de metoder som lagrar eller uppdaterar databasen bör vara synkroniserade för att undvika krockar då två eller fler klienter exempelvis uppdaterar en och samma tabell i databasen.
- Någon form av säkerhet bör implementeras för att undvika att obehöriga användare tar sig in i systemet. Minimalt bör en kombination av inloggning och kryptering av sänt data implementeras.

Utökad funktionalitet med GPRS

Givet att det finns tillgång till en informationsstruktur i form av en databas, kan man på serversidan implementera objekt som kopplar upp sig mot den aktuella databasen. På klientsidan, dvs. på mobiltelefonen, kan man då skapa ett användargränssnitt från vilket användaren kan välja huruvida den exempelvis vill lämna eller hämta ett meddelande. Beroende på vad användaren väljer så kopplar servern via uppkopplingsobjektet upp sig mot aktuell databas och hämtar eller lagrar ett meddelande. Detta kan simulera det förlopp som användarna inom projektet beskrivit som att lämna meddelanden på Post-It lappar på exempelvis en anslagstavla. Anslagstavlan ifråga utgörs av en databas, och Post-It lapparna som sitter på anslagstavlan utgörs av de meddelanden som man lämnar till, eller hämtar från, databasen från sin mobiltelefon. Angående stöd för ljud så har inte alla mobiltelefoner stöd för att spela upp ljudfiler i WAV-format. Tjänsten kan istället realiseras med textmeddelanden.

Vidare kan även säkerhet implementeras i systemet. Inledningsvis kan uppkopplingsfasen modifieras så att systemet kommunicerar via `https`-protokollet istället för `http`. På så vis skickas data krypterat. Det kan även vara intressant att undersöka möjligheten att låta informationen som sänds i systemet autenticeras via certifikat. Stöd för denna typ

av säkerhet tillhandahålls av gränssnitten i paketet javax.microedition.pki i MID-profilen.

Vad medför 3G

3G står för den tredje generationens mobiltelefonnät, och avser främst teknikstandarden UMTS. För prototypens del skulle det kunna medföra att man kan utvidga typen data som man vill kunna komma åt. Bland annat kan man tänka sig att användarna skulle kunna lämna bildmeddelanden i en databas som sedan nästa användare kan hämta. Detta är en tjänst som inte är realiserbar i den framtagna prototypen då såväl dataöverföringshastigheten är för låg som att den anvisade Motorola-telefonen inte stöder uppspelning av bildmeddelanden. Förutom kommunikation med databasen skulle även servern kunna tillhandahålla tjänster för hämtning/lagring av godtyckliga dokument eller filer på en godtycklig dator som den kommunicerar med. I praktiken skulle man som användare i systemet då exempelvis kunna hämta viktiga dokument som finns lagrade på en dator till sin mobiltelefon. Detta oavsett var man befinner sig, förutsatt att man givetvis har täckning på mobiltelefonen.

Diskussion och slutsatser

Prototypen är konstruerad i Java enligt klient/server modellen. Klienten utgörs av en MIDP-applikation som körs på en mobiltelefon som har stöd för Java. Servern utgörs av en javaservlet som körs på en tomcatserver installerad på en dator på NADA. Javas utvecklingsplattform J2ME för mobila enheter är uppdelad i två olika konfigurationer:

- CLDC För enheter med små resurser
- CDC För övriga enheter

Då mobiltelefonen är en enhet med små resurser faller den inom ramarna för CLDC. Detta begränsar utvecklingsmöjligheterna, exempelvis med avseende på grafisk upplösning.

Ytterligare en begränsning är att prototypen inte klarar av att spela upp ljudfiler, vilket beror på det faktum att mobiltelefonen saknar stöd för att spela WAV-filer. Då det inte är en begränsning i J2ME plattformen implicerar det att det bör gå att vidareutveckla prototypen till att spela upp ljudfiler och sedan pröva den på en annan mobiltelefon som såväl har stöd för Java som uppspelning av WAV-filer. I det fall man vill låta prototypen bestå av den mobiltelefon som använts i examensarbetet bör man istället kunna realisera hämtning/lämning av meddelanden i textformat.

En teknikstandard som har betydelse inför valet av eventuell vidareutveckling är UMTS. Standarden brukar benämnas som Europas variant av den tredje generationens mobila kommunikationssystem, och ger möjligheter till överföring av stora datamängder på kort tid. Detta blir möjligt då överföringshastigheten med UMTS är mycket högre jämfört med hastigheten i GPRS. Vidare har telefoner som kommunicerar via UMTS troligen bättre resurser med avseende på såväl grafisk upplösning som processorkraft och minneskapacitet. Totalt sett bör man med andra ord såväl kunna konstruera kraftfullare klientapplikationer som tjänster tillhandahållna av servern.´

Mot bakgrund av denna summering av prototypen är det relevant att fråga sig hur väl den lämpar sig för medlemmarna i InterLiving-projektet. Vidare ställdes i rapportens inledning tre frågor som utgör grunden för det arbete som genomförts:

- Hur sker överföring av data mellan en mobiltelefon och en Internetuppkopplad dator?
- Då Motorola-telefonen har stöd för Java, går det att utveckla en prototyp i Java som passar projektet?
- Går det att skicka ljud- eller bildfiler mellan mobiltelefonen och Internetuppkopplade datorer utan att det tar för lång tid?

I den befintliga prototyp som utvecklats transporteras data från mobiltelefonen via den paketbaserade trådlösa standardtekniken GPRS. Paketen når Comviqs GPRS-portal via GSM-leverantören. Från portalen ompaketeras de och slussas vidare till servern via Internet. Standardprotokollen för överföringen är TCP/IP.

Det finns inget som talar emot användningen av Java som teknisk utvecklingsplattform för ett dylikt system som prototypen. Fördelarna med Java är flera, varav några är som följer:

- Plattformsoberoende medför att prototypen kan utvecklas generellt. Ingen hänsyn behöver tas till vilka plattformar klienten respektive servern ska köras på.
- Utvecklare har tillgång till standardiserade och väldokumenterade klassbibliotek. Detta medför att utvecklaren kan fokusera på design och konstruktion av systemet på högre nivå, och kan bortse från detaljer såsom diverse protokolls design, hur data överförs, vilken modulationsteknik som används etc.
- Då Java är en teknisk plattform som ständigt utvecklas i takt med nya tekniska möjligheter och krav är det rimligt att anta att även J2ME kommer att utvecklas till att exempelvis omfatta enheter som kommunicerar via UMTS. Detta medför att potentiella system enkelt bör kunna uppdateras ifall det bedöms som nödvändigt.

Rörande överföring av text- respektive ljuddata drar examensarbetet slutsatsen att GPRS-baserade system bör begränsas till att kommunicera med textbaserad data, medan UMTS-baserade system klarar såväl text- som ljudkommunikation.

Avslutningsvis kan sägas att InterLiving-projektet kan ha nytta av en utvidgning av den befintliga prototypen. Huruvida den fortsättningsvis ska vara GPRS-baserad eller inte överlåter jag åt projektet att besluta. Dock är min personliga uppfattning att man bör undersöka möjligheten att implementera UMTS-baserad kommunikation då det öppnar upp för kraftfullare och mer avancerade tjänster som systemet då kan tillhandahålla. Om detta blir verklighet kan familjemedlemmarna inom projektet se fram emot att ha ett kraftfullt hjälpmedel som förenklar kommunikationen dem emellan.

Referenser

Litteratur

MUCHOW, JOHN W. 2002. *Core J2ME Technology & MIDP*. The Sun Microsystems Press. ISBN 0-13-066911-3.

I huvudsak är det denna bok som legat till grund för konstruktion av mjukvarudelarna i prototypen.

RIGGS, ROGER ET AL. 2001. *Programming Wireless Devices with the Java2 Platform, Micro Edition.* Addison-Wesley. ISBN 0 201-74627-1.

STALLINGS, WILLIAM. 2000. *Data & Computer Communications.* Prentice Hall. ISBN 0-13-084370-9.

Kurslitteratur som användes i kursen 2G1316 Datorkommunikation och datornät läsåret 2000-2001.

Internet

Samtliga länkars giltighet har verifierats per den 18 oktober 2005.

HELLSTRÖM, MATTIAS. 2003. Exjobbsrapport. *Java Applications in Mobile Devices – concerning business services at Telia Mobile AB,* http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2003/rapporter03/hellstrom_mattias_03029.pdf

CSNW, Circuit Switched Networks. http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci211787,00.html

PSNW, Packet Switched Networks. http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci212737,00.html

TDMA, Time Division Multiple Access. http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci214175,00.html

GSM1, Information om GSM. http://searchmobilecomputing.techtarget.com/sDefinition/0,,sid40_gci213988,00.html

GSM2, Information om GSM. http://www.susning.nu/susning.fcgi?action=browse&id=GSM&oldid=Gsm

GSM3, Information om GSM. http://www.communica.se/kunskapsbank/hscsd.htm

MPLEX, Multiplexing. http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci212614,00.html

GPRS, General Packet Radio Service. http://www.mobilecomms-technology.com/projects/gprs/

PCU, Packet Control Unit. http://www.mpirical.com/companion/GPRS/PCUUnit.htm

MIDP20, MIDP 2.0. http://java.sun.com/products/midp/whatsnew.html

X509, Datasäkerhet, standard för certifikat. http://www.susning.nu/X.509

NVMEM, Nonvolatile Memory.
http://searchstorage.techtarget.com/sDefinition/0,,sid5_gci212679,00.html

MMAPI, Mobile Media API. http://wireless.java.sun.com/apis/articles/mmapi_overview/

JAVADOC, Dokumentation av klassen HttpServlet.
http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/http/HttpServlet.html

UMTS, Universal Mobile Telecommunications System. http://www.susning.nu/UMTS

JAVA, Information om Javaplattformen. http://java.sun.com/java2/

Länken till den ursprungliga sidan, som senast besöktes den 15 januari 2004, är http://java.sun.com/java2/whatis/. Denna länken fungerar dock ej per den 18 oktober 2005.

Bilagor

Bilaga 1: Förkortningar

Nedan överskådliggörs förkortningar som förekommer i denna rapport.

ABB	Audio Building Block
API	Application Program Interface
BSC	Base Station Controller
BTS	Base Transceiver Station
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
GGSN	Gateway GPRS Node
GPRS	General Packet Radio Service
GSM	Global System for Mobile communication
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAD	Java Application Description file
JAR	Java Archive File
JDBC	Java Database Connectivity
JVM	Java Virtual Machine
KVM	K Virtual Machine
MIDP	Mobile Information Device Profile
MMAPI	Mobile Media API
ODBC	Open Database Connectivity
PCU	Packet Control Unit
SGSN	Serving GPRS Service Node
SQL	Structured Query Language
SSL	Secure Socket Level
TDM	Time Division Multiplexing
TDMA	Time Division Multiple Access
UMTS	Universal Mobile Telecommunications System
WCDMA	Wideband Code Division Multiple Access

Bilaga 2: Källkod för prototypen

Serverapplikationen

```
/**
 * Implemented by d98-iay@nada.kth.se
 */
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res)
         throws IOException, ServletException{
      res.setContentType("text/plain");
      PrintWriter out = res.getWriter();
      out.print("HelloServlet says Hi! (doGet)");
      out.close();
   } // close doGet

   public void doPost(HttpServletRequest req, HttpServletResponse res)
         throws IOException, ServletException{
      res.setContentType("text/plain");
      PrintWriter out = res.getWriter();
      out.print("HelloServlet says Hi! (doPost)");
      out.close();
   }
} // close class HelloServlet
```

Klientapplikationen

```
/**
 * Implemented by d98-iay@nada.kth.se after example in Muchow John,
 * "Core J2ME techonology and midp"
 *
 * Use POST to communicate with the Java Servlet.
 * The servlet will respond with a simple text string.
 * The name of the servlet is HelloServlet.
 */

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;
import java.awt.*;

public class TestClient extends MIDlet implements CommandListener{
    /*
    *Attributes
    */
    // Reference to the Display object
    private Display display;
    // The main form
    private Form fmMain;
    // Command to exit the MIDlet
    private Command cmExit;
    private Command cmConnect;
    // Field for displaying account balance
    private StringItem msgFromServlet;
    // Text field on which user can input url to connect to
    private TextField url;

    /*
    * Constructor
    */
    public TestClient(){
        display = Display.getDisplay(this);

        // create commands
        cmExit = new Command("Exit", Command.EXIT, 1);
        cmConnect = new Command("Connect", Command.OK, 1);
        System.err.println("IAE on cmConnect");

        // Balance string item
        msgFromServlet = new StringItem("", "");
        url = new TextField(
              "Url to servlet",
              "http://kentaur.nada.kth.se:6969/labbar/servlet/HelloServlet",
              150,
              TextField.URL
        );

        System.err.println("IAE on textfield");

        // Create Form, add commands and components, listen for events
        fmMain = new Form("Communicate with Servlet");

        fmMain.addCommand(cmExit);
        fmMain.addCommand(cmConnect);
        fmMain.append(url);
        fmMain.setCommandListener(this);
    } // close constructor
```

```
/*
* Methods
*/
public void startApp(){
   display.setCurrent(fmMain);
}

public void pauseApp(){}
public void destroyApp(boolean unconditional){}

public void commandAction(Command c, Displayable s){
   if(c == cmExit){
      destroyApp(false);
      notifyDestroyed();
   }
   else{
      if(c == cmConnect){
         String tmp = this.url.getString();
         if((tmp == null) || tmp.equals("")){
            fmMain.delete(fmMain.size()-1);
            fmMain.append(msgFromServlet);

            msgFromServlet.setText(
                  "No url was given, cannot connect. Try again"
            );
         }
         else{
            try{
               fmMain.delete(fmMain.size()-1);
               fmMain.append(msgFromServlet);

               msgFromServlet.setText(
                     "\nAttempt to connect to " + tmp + "\n"
               );
               communicate(tmp);
            }
            catch(Exception e){
               System.err.println("Msg (commandAction): " + e.toString());
            }
         }
      }
   }
} // close method commandAction

/*
* Access servlet
*/
private void communicate(String url){// throws IOException{
   HttpConnection c = null;
   InputStream is = null;
   OutputStream os = null;
   int rc;
   try{
      c = (HttpConnection)Connector.open(url);
      // Set the request method and headers
      c.setRequestMethod(HttpConnection.POST);
      c.setRequestProperty(
            "User-Agent", "Profile/MIDP-2.0 Configuration/CLDC-1.0"
      );
      c.setRequestProperty(
            "Content-Language", "en-US"
      );
      // Getting the output stream may flush the headers
```

```
            os = c.openOutputStream();
            os.write("Hi from client\n".getBytes());
            os.flush();          // Optional, getResponseCode will flush

            // Getting the response code will open the connection,
            // send the request, and read the HTTP response headers.
            // The headers are stored until requested.
            rc = c.getResponseCode();
            if (rc != HttpConnection.HTTP_OK){
               throw new IOException("HTTP response code: " + rc);
            }
            is = c.openInputStream();

            // Get the ContentType
            String type = c.getType();
            String str = "";

            // Get the length and process the data
            int len = (int)c.getLength();
            if (len > 0){
               int actual = 0;
               int bytesread = 0 ;
               byte[] data = new byte[len];
               while ((bytesread != len) && (actual != -1)){
                  actual = is.read(data, bytesread, len - bytesread);
                  bytesread += actual;
               }
               str = new String(data);
               msgFromServlet.setText(str);
            }
            else{
               int ch;
               ByteArrayOutputStream bStrm = new ByteArrayOutputStream();
               while ((ch = is.read()) != -1){
                  bStrm.write((byte)ch);
               }
               str = new String(bStrm.toByteArray());
               bStrm.close();
               msgFromServlet.setText(str);
            }
         }
               // Catch Exceptions thrown within method Communicate
         catch (ClassCastException e){
            throw new IllegalArgumentException("Not an HTTP URL");
         }
         catch(Exception e){
            System.err.println("Msg: " + e.toString());
         }
         finally{
            try{
               if (is != null)
                  is.close();
               if (os != null)
                  os.close();
               if (c != null)
                  c.close();
            }
            catch(Exception e){
               System.err.println("Msg: " + e.toString());
            }
         }
      } // close communicate
   } // close class TestClient
```

Bilaga 3: Konfigurering av mobiltelefon

Bild 5 visar de inställningar som gäller för Motorola-telefonen gällande uppkoppling via GPRS på Comviqs nät.

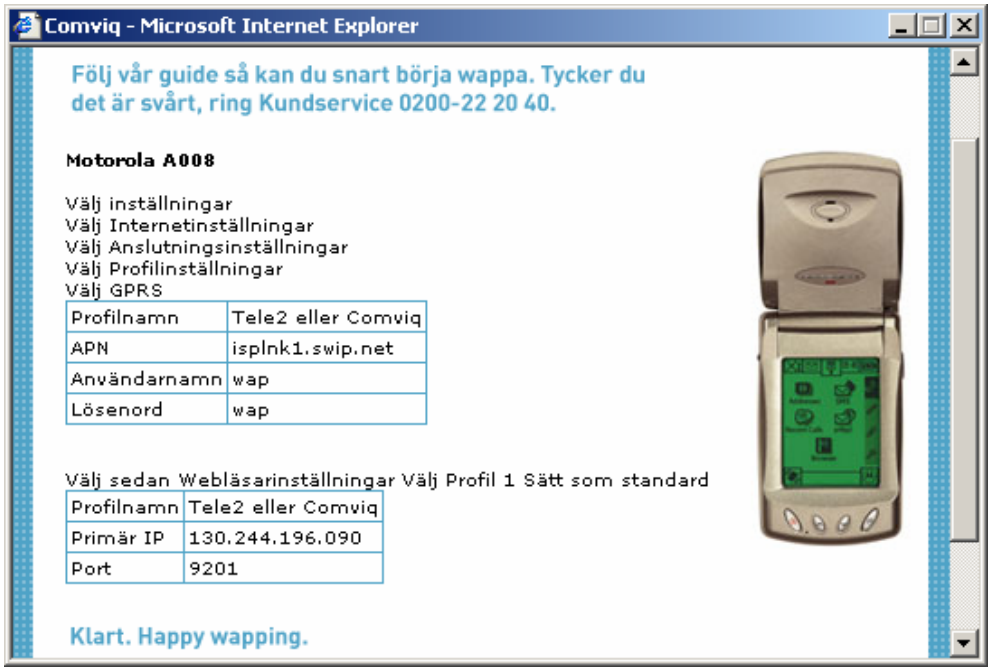

Bild 5: Inställningar för Motorola A008.

Bilaga 4: Bilder på körning av prototyp

Bild 6 visar start av tomcatservern på vilken servleten finns publicerad. På bild 7 verifieras att servleten svarar på anrop via webbläsare. Bild 8 och 9 visar att klientapplikationen fungerar enligt förväntan då det körs från en mobiltelefonssimulator. Bilderna10-12 visar att klientapplikationen på Motorola-telefonen kopplar upp sig mot servleten på tomcatservern.

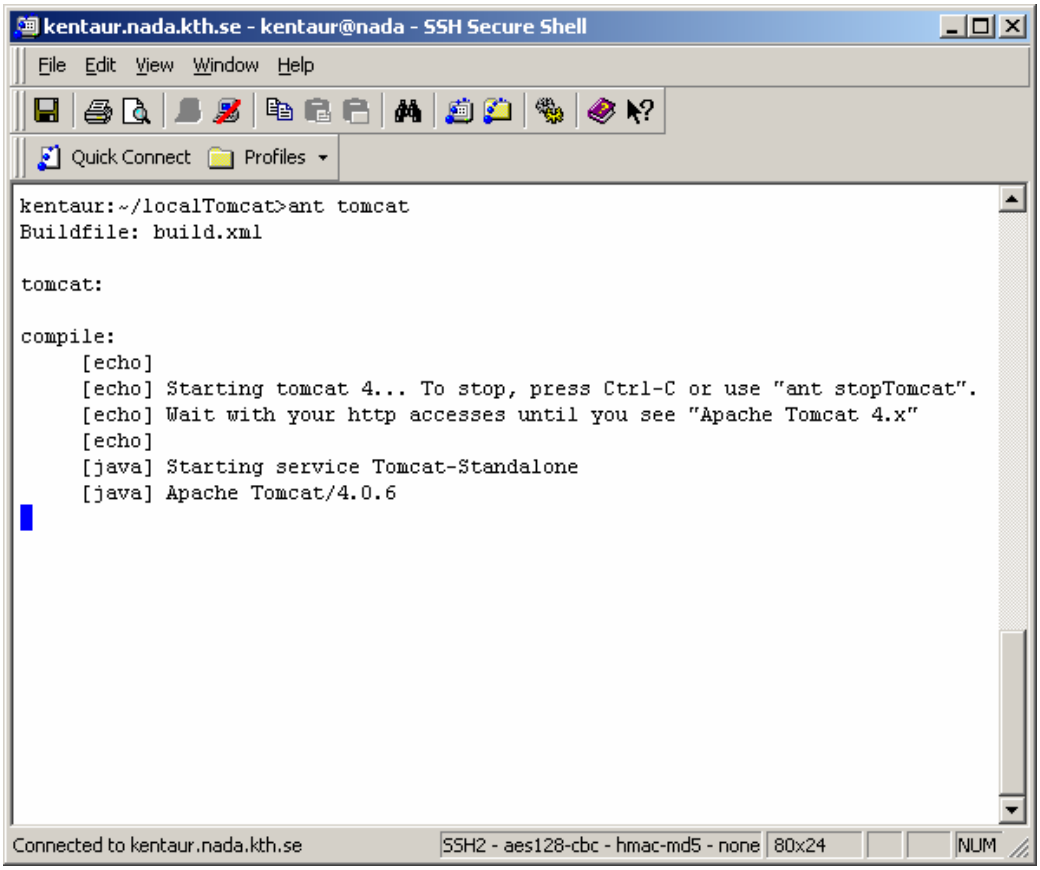

Bild 6: Start av tomcatserver

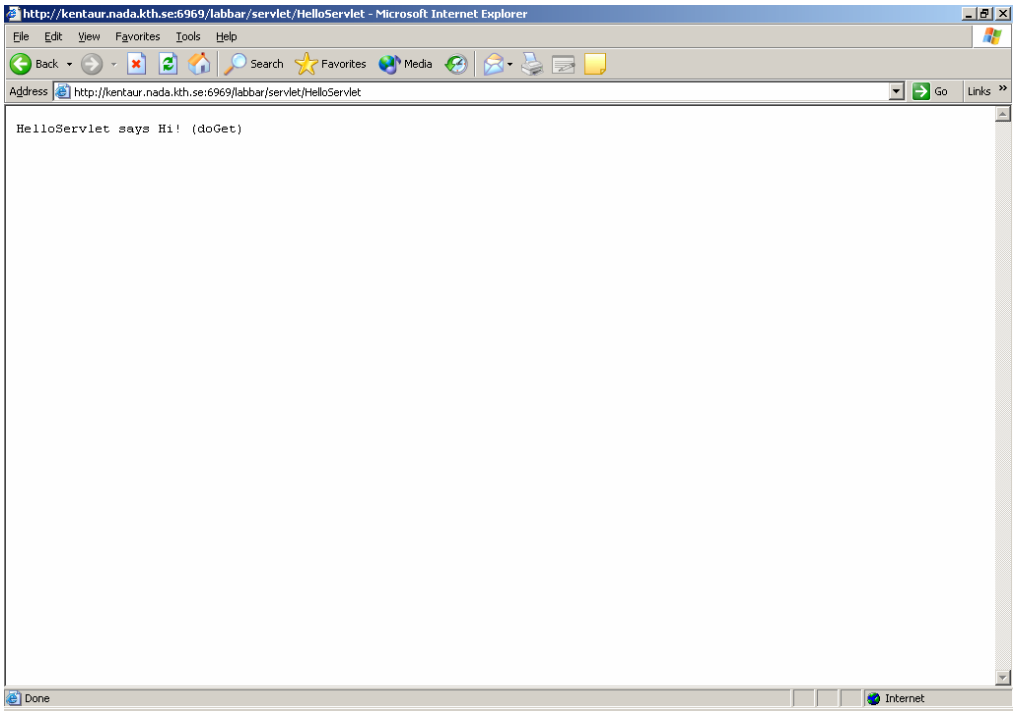

Bild 7: Servleten svarar på anrop från webläsare

Bild 8 och 9 Exekvering av klientapplikation från mobiltelefonsimulator

Bild 10: Klientapplikationen redo att köras från Motorola-telefonen.

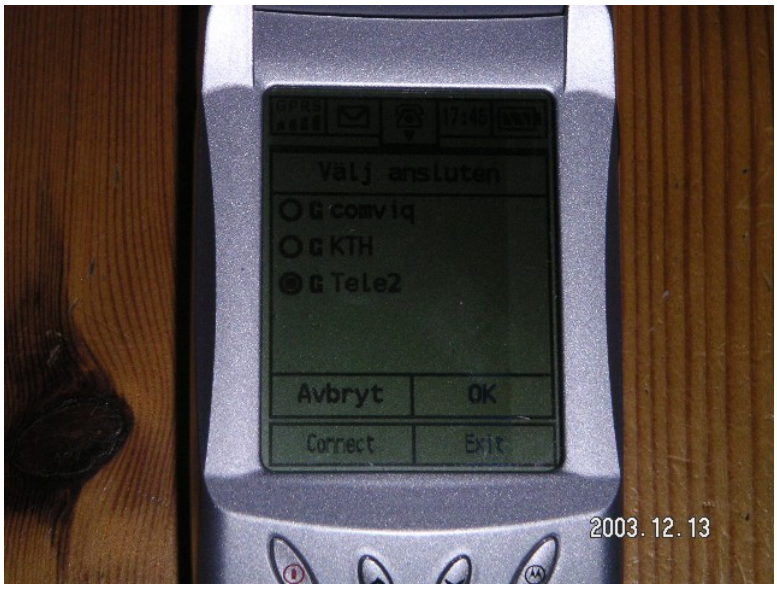

Bild 11: Koppla upp via rätt anslutning.

Bild 12: Servleten har svarat på anrop.

Bilaga 5: Källkod för AudioExample

Följande kod exemplifierar hur ljudfiler kan spelas upp från en MIDlet. Exempelkoden är hämtad från Internet, och den fanns tillgänglig så sent som den 9 oktober 2003 via länken http://wireless.java.sun.com/midp/articles/midp20/src/New20_src.zip

```
import java.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.media.*;
import javax.microedition.midlet.*;

public class AudioExample extends MIDlet
      implements CommandListener{
   private Display mDisplay;
   private Form mMainForm;
   private Command mPlayCommand;

   public AudioExample(){
      mMainForm = new Form("AudioExample");

      mPlayCommand = new Command
            ("Play",Command.SCREEN, 0);
      mMainForm.addCommand(mPlayCommand);
      mMainForm.addCommand(new Command("Exit",
            Command.EXIT, 0));
      mMainForm.setCommandListener(this);

      mDisplay = Display.getDisplay(this);
   }

   public void startApp(){
      mDisplay.setCurrent(mMainForm);
   }

   public void pauseApp() {}

   public void destroyApp(boolean unconditional) {}

// CommandListener method
   public void commandAction(Command c, Displayable s){
      if (c.getCommandType() == Command.EXIT)
         notifyDestroyed();
      else if (c == mPlayCommand){
         try{
            InputStream in = getClass().getResourceAsStream
                  ("/signs_m.wav");
            Player p = Manager.createPlayer
                  (in, "audio/x-wav");
            p.start();
         }
         catch (Throwable t) {
            Alert a = new Alert("Exception", t.toString(),
                  null, AlertType.ERROR);
            mDisplay.setCurrent(a, mMainForm);
         }
      }
   }
}
```