Integrering av mobila enheter i verksamhetssystem

Joakim Antonsson och Johan Hasselström
Civilingenjörsutbildningen i Lantmäteri, Lunds Tekniska högskola
Vår och sommar 2007

Avdelningen för Fastighetsvetenskap, Lunds Tekniska Högskola
Lunds Universitet
Box 118
221 00 LUND
Sverige

Department of Real Estate Science. Lund Institute of Technology
Lund University
PO Box 118
SE – 221 00 LUND
Sweden

ISRN LUTVDG/TVLM 07/5152 SE

Titel:
Integrering av mobila enheter i verksamhetssystem

Författare:
Joakim Antonsson Lunds tekniska högskola
Johan Hasselström Lunds tekniska högskola

Handledare:
Lars Harrie Lunds tekniska högskola
Fredrik Ekelund SWECO Position
Ulf Månsson SWECO Position

Examinator:
Klas Ernald-Borges Lunds tekniska högskola

Opponent:
Linda Adler Lunds tekniska högskola
Cecilia Pihl Lunds tekniska högskola

Sökord: mobila system, GIS, GPS, BOSH, tracking

Title:
Integration of mobile units in corporate environments

Authors:
Joakim Antonsson Lunds Institute of Technology
Johan Hasselström Lunds Institute of Technology

Supervisors:
Lars Harrie Lunds Institute of Technology
Fredrik Ekelund SWECO Position
Ulf Månsson SWECO Position

Examiner:
Klas Ernald-Borges Lunds Institute of Technology

Opponent:
Linda Adler Lunds Institute of Technology
Cecilia Pihl Lunds Institute of Technology

Keywords: mobile systems, GIS, GPS, BOSH, tracking

Avdelningen för Fastighetsvetenskap
Lunds Tekniska Högskola
Lunds Universitet
Box 118
221 00 Lund

Förord

Examensarbetet har genomförts som avslutning på civilingenjörsutbildningen till lantmätare med inriktning mot teknisk geomatik. Arbetet har utförts under vår och sommar 2007 på *SWECO Position* i Malmö och har resulterat i denna rapport på Avdelningen för Fastighetsvetenskap vid *Lunds tekniska högskola.*

Vi vill tacka våra handledare, Lars Harrie, tekn. dr vid *GIS- centrum* på *Lunds Universitet*; Fredrik Ekelund, regionchef vid *SWECO Position* i Malmö och Ulf Månsson, tekn. konsult vid *SWECO Position* i Malmö för god handledning och support. Vi vill även tacka Mårten Olsson och Peter Kasslid vid *Landskrona kommun* för att de har ställt upp med sin tid vid diskussioner, tester och utvärdering.

Joakim Antonsson Johan Hasselström

Sammanfattning

Många organisationer har idag ett väl fungerande verksamhetssystem vilket används för utbyte och uppdatering av information. Tanken är att informationen alltid ska vara uppdaterad och därför ges åtkomst till systemet genom organisationens intranät. Med en fortsatt snabb utveckling av mobila enheter finns stora möjligheter att ansluta även dessa till verksamhetssystemet. Detta tillsammans med att många arbetsuppgifter är kopplade till ett geografiskt läge har skapat intresse att integrera mobila enheter som har tillgång till *GPS* –mottagare med verksamhetssystem. Med detta som utgångspunkt syftar examensarbetet till att ta fram en utvecklingsbar grundlösning vilken ska lösa de problem som kan uppstå.

För att identifiera möjliga tillämpningar och skapa en bild av vilka problem som kan uppstå och vad som krävs av den tilltänkta systemdesignen har samråd med konsultföretaget *SWECO Position* skett. De största hindren för organisationer när de ska installera och använda nya system upplevs vara att det är omständligt och medför stora investeringskostnader att över huvud taget komma igång med tester. Detta dels på grund av att investeringar ofta måste göras i hård- och mjukvara, men även på grund av att ny programvara ska testas och godkännas av berörd IT -enhet.

Genom litteraturstudier, diskussioner och tekniktester har en systemdesign för en grundlösning arbetats fram. Grundlösningen löser problemet med en lång uppstartsfas genom att använda ett nav som de mobila klienterna ansluter till istället för att ansluta direkt till måldatorn. Navet kan då placeras utanför organisationens intranät vilket innebär att ett minimalt antal programinstallationer och konfigurationer behöver göras. Vidare bygger grundlösningen på att varje del ska kunna bytas ut för att enkelt kunna anpassa lösningen till olika tillämpningar.

Grundlösningen har därefter implementerats genom en prototyp för ett användarfall som utarbetats tillsammans med *Landskrona kommun*. Prototypen *trackar* mobila enheter för att visa deras positioner i en kartmodul på en stationär dator och har testats i *Landskrona kommuns* IT -miljö. Testerna av prototypen indikerar att grundlösningen fungerar tillfredställande även om vissa kompabilitetsproblem uppdagades.

Rapporten beskriver hur en enkel grundlösning kan utformas, vilken kan fungera som bas för vidare utveckling av systemintegrerande applikationer. Den initiala utvecklingskostnaden kan på detta sätt fördelas på olika applikationer och projekt. Detta tillsammans med att en stor del av applikationerna redan är utvecklad genom grundlösningen gör att startsträckan för att testa ett nytt system förkortas och en kostnadseffektiv lösning åstadkoms.

Abstract

Today many organisations have a well –functioning information system to make theme more efficient and competitive. The evolution of mobile phones and mobile broadband together with the possibility to use the organisations information system more efficiently makes it possible and interesting to incorporate mobile units in the systems. The purpose of this master thesis is therefore to find a solution that incorporates mobile units in an information system.

To identify possible applications and what to expect from the developed solution, consultation with *SWECO Position* has been made. The major problem for new mobile systems is the cost for setting up a test environment. Before an installation of a new application the organisations IT-group have to approve it, they scan the application for potential security leaks. If the new application needs to open a port in the firewall you can be sure its going to be a long process before the tests can start.

A system design for a base solution has been developed. The solution has been implemented as a prototype and tested in *Landskrona kommuns* environment. The prototype application tracks mobile units and displays their position in a digital map. The most essential part of the base solution is that the mobile units connects to a server outside of the organisation's firewall, the target computer inside the firewall then uses the standard port for web traffic to get information from the server. With this dataflow no ports has to be opened in the firewall to incoming traffic. Further, the solution is developed with interfaces and therefore any part of the solution can be changed without changing the entire solution.

The report describes how to develop a simple base solution. With this solution many different applications can be developed. The initial cost of the development and testing of a base solution can be split and paid by many different application and projects. This means the time for starting projects can be shortened and a cost effective solution is achieved.

Ordlista

.NET	En plattform för utveckling av *Windows* applikationer framtagen av *Microsoft*, består av en stor samling av färdiga klasser och hjälpmedel.
2G	Den andra generationens mobilnät.
3G	Den tredje generationens mobilnät. Tillåter uppkoppling mot Internet med högre överföringshastighet än t.ex. 2G.
ActiveSync	*ActiveSync* är ett synkroniseringsverktyg utvecklad av *Microsoft* och används för att synkronisera mobila enheter med en stationär dator. *ActiveSync* kan även användas för andra syften, som att styra den mobila enheten från datorn eller att utföra *debuggning*.
API	*Application Programming Interface* är de klassbibliotek och funktioner i en programvara som kan utnyttjas vid t.ex. *script* -programmering.
ASCII	*American Standard Code for Information Interchange*, teckenkodning som baseras på det engelska alfabetet
Bluetooth	*Bluetooth* är en industriell specifikation för privata nätverk mellan t.ex. mobiltelefoner, handdatorer och datorer.
BOSH	*Bidirectional Streams over Synchronus HTTP*, en teknik för att simulera en *TCP* -liknande kommunikation över *HTTP*.
C#	Uttalas [C Sharp], ett programmeringsspråk utvecklat av *Microsoft* vilket har många likheter med *Java*.
CBSE	*Component-based Software Enginering*, en metodik för hur systemutveckling skall ske.
CDC	*Connected Device Configuration*, ett ramverk för utveckling av applikationer i *Java Micro Edition*.
CGI	*Common Gateway Interface*, en serverteknik för att vidarebefordra parametrar från en klient till ett program på servern
CLDC	*Connected Limited Device Configuration*, precis som *CDC* ett ramverk för utveckling av applikationer i *Java Micro Edition* men är specialanpassat för enheter med begränsade resurser.
CompactFramework	En begränsad version av *.NET* avsedd att användas för mobila enheter.
Compass	Kinas planerade satellitbaserade navigeringssystem.
Cookie	Avser en liten mängd text som kan lagras på klientsidan vid användning av t.ex. webbläsare.
Debugger	En programvara för att testa och felsöka andra program. Ofta tillåts utvecklaren följa anropen i

	koden och se variablers värde för att enklare kunna felsöka.
DGPS	Differentiell *GPS*, *GPS* -teknik som utnyttjar fasta eller flyttbara basstationer på redan kända punkter för att öka nogrannheten i mätningarna.
DMZ	*DeMilitarised Zone*, avser en avgränsad del i ett företags nätverk som ligger mellan det verkliga intranätet och Internet. Används för t.ex. webbservrar.
DOP	*Dilution Of Precision*, ett värde som vid användning av *GPS* anger hur väl utspridda satelliterna är.
DTD	*Document Type Definition*, det dokument som anger vilka element och attribut som får eller måste finnas i ett *XML* -dokument (numera ersatt av *XML-schema*).
EBCDIC	*Extended Binary Coded Decimal Interchange Code*, teckenkodning som användes av *IBM*.
EDGE	*Enhanced Data rates for GSM Evolution*, teknik för att öka överföringshastigheterna vid datatransport i *GSM* -nätet. Ligger hastighetsmässigt mellan *2G* och *3G*.
Ethernet	En metod för datakommunikation i nätverk konstruerad av företaget *Xerox*.
GALILEO	EU -s planerade satellitbaserade navigeringssystem.
GIS	Geografiskt InformationsSystem, ett system för lagring, analysering och hantering av data med geografisk koppling.
GLONASS	*GLObal Navigation Satellite System*, Rysslands satellitbaserade navigeringssystem
GML	*Geography Markup Language*, *XML* -dialekt för hantering av geografiska data
GNSS	*Global Navigation Satellite System*, samlingsnamn för satellitbaserade navigeringssystem.
GoogleMaps	*API* för utveckling av webbaserade kartor.
GPRS	*General Radio Packet Services*, teknik för dataöverföring i mobilnätet.
GPS	*Global Positioning System*, ett system för att positionera användare som innehar en *GPS* -mottagare.
GUI	*Graphical User Interface*, en metod för att underlätta interaktion mellan människa och dator. Interaktionen sker ofta genom bilder och andra grafiska element.
GZIP	*GNU Zip*, programvara för *UNIX* som används för komprimering av filer.
HTML	*HyperText Markup Language*, det märkspråk som används för skapande av hemsidor.

HTTP	*HyperText Transfer Protocol*, ett protokoll som används för dataöverföring över Internet.
HTTPS	*HyperText Transfer Protocl Secure,* en krypterad *HTTP* –anslutning.
IMAP	*Internet Message Access Protocol*, ett protokoll som låter en klient hämta e-post hos en server.
IP	*Internet Protocol*, det protokoll som används för att adressera användare av Internet.
J2ME	Äldre benämning av *Java Micro Edition*, se *JME*
Java	Objektorienterat programspråk som är plattformsoberoende. Utvecklat av *Sun Microsystems*.
Java Micro Edition	Se *JME*.
JME	*Java Micro Edition*, en samling *API* -er att användas för utveckling på mobila enheter eller andra enheter med begränsade resurser.
JPEG	*Joint Photographic Experts Group*, avser bildformatet *jpg* eller *jpeg* som använder sig av cosinuskurvor för att komprimera bilderna.
Location API	Specifikation för *JME* som används för att kommunicera med *GPS* –mottagare.
MIDlet	*Mobile Information Device -let*, benämning på de program som är skrivna för mobiltelefoner med stöd för *Java Micro Edition*
MIDP	*Mobile Information Device Profile*, specifikation för program som skrivs för mobiltelefoner med stöd för *Java Micro Edition*.
MIME	*Multipurpose Internet Mail Extensions*, anger de filtyper som t.ex. en webbläsare kan ta emot och läsa.
Mp3	*MPEG-1 Audio Layer 3*, en förstörande ljudkodning som komprimerar innehållet i en ljudfil.
NMEA	*National Marine Electronics Association*, den organisation som utvecklat protokollet *NMEA 0183* vilket används för kommunikation mellan fartyg. Även de flesta *GPS* -mottagare använder protokollet.
OGC	*Open Geospatial Consortium*, internationell organisation vilken arbetar med att utveckla och införa standarder för geografisk information.
OSI	*Open Systems Interconnection*, ett försök att standardisera nätverkskommunikation som resulterade i *OSI* -modellen.
PDA	*Personal Digital Assistance*, avser en handdator.
PNG	*Portable Network Graphics*, bildformat med icke förstörande komprimering som har stöd för transparens.

PP	*Personal Profile*, ett ramverk för *JME* -applikationer
PPS	Praktiskt ProjektStyrning, metoder för att styra och genomföra projekt utvecklade av TietoEnator.
Shape	Filformat för geografisk vektordata utvecklad av företaget *ESRI*.
SSL	*Secure Sockets Layer*, kryptografiska protokoll som möjliggöra en säker nätverksanslutning för t.ex. e-post och webbsurfning.
STB	*Set-Top Box*, extern enhet för mottagning av ljud och bild som kopplas till t.ex. en TV.
Tablet PC	En variant av bärbar dator som normalt sätt inte har ett tangentbord. Datorn styrs istället med hjälp av *touchscreen* -funktionalitet.
TCP	*Transmission Control Protocol*, protokoll för dataöverföring som garanterar att datapaketen kommer fram och att de gör det i samma ordning som de skickades.
TCP/IP	Avser kombinationen av protokollen *TCP* och *IP* som anses vara de två viktigaste protokollen för att möjliggöra dataöverföring över Internet.
Touchscreen	Avser en skärm där programfunktionalitet kan styras med hjälp av att peka på skärmen.
UDP	*User Datagram Protocol*, protokoll för dataöverföring med hög prestanda som dock inte garanterar datapaketens ordning eller ens att de kommer fram. Lämpligt för t.ex. strömning av ljud och bild.
UML	*Unified Modeling Language*, objektorienterat generellt språk för modellering av olika typer av system.
URI	*Uniform Resource Identifier,* adress som används för att lokalisera t.ex. *bluetooth* –enheter.
URL	*Uniform Resource Locator*, den adress som används för att lokalisera t.ex. hemsidor.
W3C	*World Wide Web Consortium*, sammanslutning av organisationer och arbetar med utveckling och införande av Internetstandarder.
WFS	*Web Feature Service*, gränssnitt för att begära geografisk information från en server.
Windows Mobile	Företaget *Microsofts* plattform för mobiltelefoner och handdatorer.
Visual Studio	Utvecklingsverktyg från *Microsoft* med stöd för en mängd olika programmeringsspråk.
WLAN	*Wireless Local Area Network*, trådlöst nätverk för att koppla samman datorer.
WMS	*Web Map Service*, gränssnitt för att begära kartbilder från en server.

XML	*Extensible Markup Language*, märkspråk där användaren själv sätter regler och riktlinjer för vad dokumenten ska och kan innehålla.
XML-Schema	Dokument som anger vilka element och attribut som får eller måste finnas i ett *XML* -dokument (*XML - schema* har ersatt det äldre *DTD*).

Innehållsförteckning

1 Inledning

I detta kapitel ges en bakgrund till projektet. Syftet presenteras med de avgränsningar som gjorts samt de metoder som använts för att genomföra projektet.

1.1 Bakgrund

Människor har ett behov av att kartlägga sin omgivning, metoderna för att göra det har förändrats snabbt de senaste decennierna. De tekniska framstegen ger oss möjligheten att kartera, lagra, analysera och presentera kartor digitalt. Dessutom kan stora mängder data snabbt förflyttas mellan olika datorer över Internet vilket gör att informationen inte längre är bunden till en specifik plats, organisationernas informationssystem har blivit effektivare.

Informationssystem i allmänhet har blivit ett viktigt sätt för företag att konkurrera, för att vinna kampen om kunderna krävs låga priser och att få ut nya produkter snabbt (Johnson m.fl., 2005). Ett informationssystem kan vara ett sätt att effektivisera en organisations försörjningskedja, förkorta ledtiderna och öka konkurrenskraften (Jonsson och Matsson, 2005). Därför gäller det att ständigt ligga i framkant av utvecklingen, även gällande informationsflöden.

Den senaste tidens utveckling går mot att organisationer arbetar med lösningar som bygger på att samla all information i en och samma databas och sedan anger rättigheter för vad varje medarbetare ska få tillgång till (Jonsson och Matsson, 2005). Det innebär att databasen alltid innehåller den senaste information och därför kan hela organisationen vara uppdaterad gällande viktiga uppgifter. Enda kravet är att den som skall komma åt informationen är ansluten mot företagets databas, antingen via ett lokalt nätverk eller via Internet.

Tack vare Internet är det även möjligt att involvera de som arbetar utanför kontoret och därmed normalt inte är uppkopplade. Detta åstadkoms vanligen genom att använda bärbara datorer, handdatorer eller avancerade mobiltelefoner som synkroniseras mot databasen. I dagsläget sker en stor utveckling av telefonerna, de blir snabbare, får större minne och fler funktioner. Tillsammans med en större mobil bandbredd är det enklare att utveckla kostnadseffektiva lösningar för att hela tiden vara uppkopplad mot sin organisation. Det innebär att ännu fler kan vara en del av ett informationssystem i realtid.

Vid utomhusarbeten finns det många tillämpningar där kommunikation av geografisk information är en förutsättning. Om man håller reda på var olika medarbetare är blir det möjligt att ta reda på vem som är närmast ett jobb och sedan skicka ut instruktioner till denne om vad som skall göras och var det skall utföras. Denna och liknande tillämpningar kan rationalisera arbetsflödet och minska transportsträckorna. Därför behövs en effektiv och driftsäker metod för att överföra både geografisk och annan information mellan företagets interna system och de mobila enheterna.

1.2 Problemformulering

SWECO Position verkar som konsulter inom *GIS* (Geografiska InformationsSystem). De arbetar bland annat med att integrera olika system och att utveckla skräddarsydda lösningar för organisationer. De upplever att flera av deras kunder har ett behov av att även integrera mobila enheter i sina informationssystem. För att göra detta möjligt gäller det att kunna erbjuda kunderna lösningar som är kostnadseffektiva och snabba att komma igång med.

För att kunna erbjuda en lösning som är enkel och framförallt snabb att komma igång med vill *SWECO* utveckla en grundlösning, vilken bör vara lätt att justera för en specifik kunds behov och kunna integreras i den befintliga miljön.

För att utveckla en sådan grundlösning för ett mobilt *GIS* kommer följande frågeställningar att belysas:

- Vilken typ av hårdvara är möjlig att använda? Måste det vara en handdator eller räcker det med en nyare telefon? Ska positioneringsinformationen hämtas från en inbyggd *GPS* eller från en *bluetooth* –ansluten *GPS*?
- Hur skapas en lösning som är kostnadseffektiv för både leverantör och kund?
- Hur ska den data som skickas enbart vara åtkomlig för behöriga parter?
- Hur hanteras data som ska överföras under tiden som den mobila enheten befinner sig i radioskugga?
- Hur skall handenheterna kommunicera med organisationens interna nätverk när handenheterna befinner sig utanför företagets brandvägg?

Svaret till dessa frågor kan variera efter varje specifikt problem, därför måste grundlösningen vara öppen, väldokumenterad och anpassningsbar.

1.3 Syfte

Baserat på ovanstående problemformulering är syftet:

Ta fram ett förslag till systemdesign för hur man med standardiserade öppna gränssnitt kan skapa kommunikation av geografiska data och annan information mellan mobil enhet och server.

Förslaget skall verifieras genom en prototyp utvecklad för ett användarfall med en klient och en server. Detta för att kunna dra slutsatser inför en eventuell skarp version.

1.4 Metod

Det här projektet skall resultera i två slutprodukter, en rapport och ett förslag till systemdesign för en grundlösning. För att säkerställa en hög kvalitet på dessa används en projektstyrningsmodell samt en modell för att utforma en systemdesign.

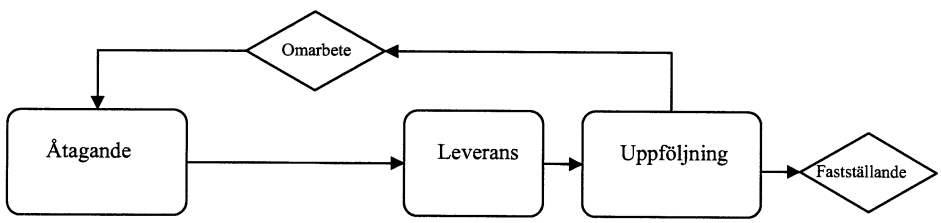
1.4.1 Projektstyrning

Projektstyrning används för att driva projekt under kontrollerade former, för att på ett tidigt stadium upptäcka och avhjälpa problem och för att inom utsatt tid uppnå resultat. (TietoEntator, 2001)

SWECO tillämpar projektstyrningsmodellen *PPS* (Praktisk Projekt-Styrning) framtagen av *TietoEnator*. *PPS* syftar till att strukturera arbetet från idé till resultat med målet att samtliga deltagande parter ska vara och förbli nöjda. Modellen bygger på att parterna i projektet kommer överens om tydliga och mätbara åtaganden. Resultatet av ett åtagande följs upp vid en bestämd tidpunkt för att fastställas eller revideras (se Figur 1.1) (TietoEnator, 2001).

I detta projekt används *PPS* för att dels upprätta en projektplan vilken innefattar projektets omfattning och därmed de översiktliga åtaganden som görs, dels en riskanalys inklusive förslag till åtgärder skapas. Vidare skapas en tidplan med ungefärlig fördelning av olika aktiviteter över projekttiden och de mer specifika åtaganden vilka kommer att utföras vecka för vecka, så kallade milstolpar. Projektplanen återfinns i Bilaga Uppdragsplan.

Genom att använda en projektstyrningsmodell under projektet, ökar möjligheten till att slutprodukten får önskat resultat, då delmål granskas efterhand. Eventuella oenigheter i utförandet upptäcks och avhjälps tidigt.

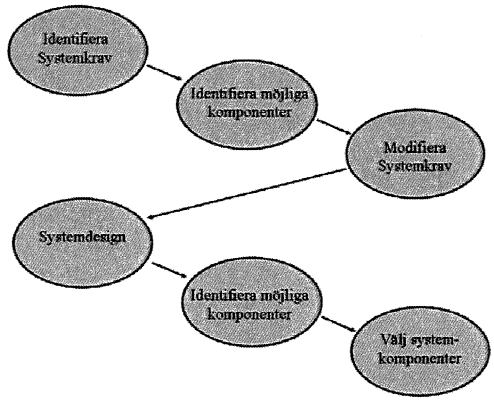


Figur 1.1. Figuren beskriver schematiskt arbetsgången för ett delmål. (Utarbetad från PPS, TietoEnator, 2001)

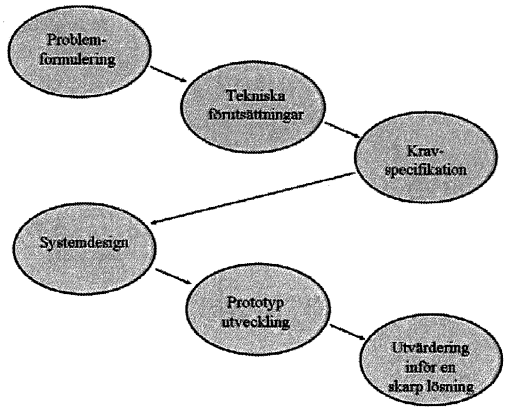
1.4.2 Systemdesign

Att använda en strukturerad modell för att skapa en systemdesign ger bättre kontroll av arbetsflödet och ger större möjlighet att den framtagna designen är tillräckligt bra för att klara de krav som ställs. (Sommerville, 2001)

Arbetet kommer att följa en anpassad *CBSE* – mall (*Component-based Software Enginering*) (se Figur 1.2). *CBSE* bygger på att arbetet delas upp i ett antal olika steg för att identifiera krav, övergripande design, val av komponenter och slutligen skapa ett eller flera förslag (Sommerville, 2001). Projektets fokus ligger på att identifiera och välja lämpliga tekniker och standardkomponenter för att uppfylla de identifierade kraven. Den anpassade *CBSE* –mallen följer i Figur 1.3.



Figur1.2. Figuren beskriver de steg som ingår i CBSE. (Översatt från Lundmark och Tisén, 2006, s. 22)



Figur1.3. Figuren beskriver den modifierade CBSE –modellen

1.5 Avgränsningar

Grundlösningen som utvecklas ska vara leverantörsoberoende och därför tas ingen hänsyn till existerande kommersiella produkter.

1.6 Målgrupp

Rapporten riktar sig till personer och organisationer vilka är intresserade av att integrera mobila enheter i befintliga verksamhetssystem. Delar av rapporten är i hög grad teknisk och kräver goda förkunskaper inom programmering.

1.7 Rapportutformning

I det inledande kapitlet ges en bakgrund som leder fram till projektets syfte och målgrupp. Vidare redogörs för de metoder som används för att nå ett resultat och de avgränsningar som har gjorts.

Den teori som behandlar befintliga tekniker för mobil kommunikation återfinns i det andra kapitlet och innefattar flera olika aspekter så som överföringsteknik, säkerhet och prestanda. Därefter, i kapitel tre, analyseras problemformuleringen och ett förslag till systemdesign för en grundlösning presenteras.

I kapitel fyra behandlas möjliga tillämpningar av grundlösningen för att visa hur grundlösningen kan användas. Därpå, i kapitel fem, implementeras grundlösningen till en prototyp för att verifiera att grundlösningen fungerar i ett användarfall. Utvärderingen av prototypen sker i kapitel sex.

Diskussion kring arbetet förs i kapitel sju och de slutsatser som dragits från projektet presenteras i kapitel åtta.

2 Tekniska förutsättningar

För att identifiera möjliga tekniker för grundlösningen görs i detta kapitel en litteraturstudie. Kapitlet ger en bakgrund till de tekniska förutsättningar som finns för att utveckla kommunikation mellan mobila enheter och verksamhetssystem.

2.1 Programmeringstekniker

Inom programutveckling finns olika tekniker för att dels beskriva uppbyggnaden av ett program och dels för att få programmet att prestera så bra som möjligt samtidigt som det är användarvänligt. Avsnittet tar därför upp ett antal olika tekniker för att kunna skapa en väl fungerande grundlösning.

2.1.1 Unified Modelling Language (UML)

UML är ett standardiserat sätt att beskriva relationer, hierarkier och flöden för alla typer av system. Inom datavetenskapen används det för att erhålla en grafisk presentation av det som har utvecklats eller ska utvecklas och beskriver klasser, objekt eller komponenter och dess relationer. Genom att kunna skapa en modell av systemet blir det lättare att förstå och konstruera detta. *UML* är dock inte begränsat till programutveckling utan används även för att beskriva t.ex. processmodeller och organisationsstrukturer.

I *UML* används en stor mängd diagram för att beskriva ett system. För datasystem finns följande typer:

Användningsfallsdiagram

Diagrammet visar systemets aktörer i typiska uppgifter.

Klassdiagram

Ett klassdiagram presenterar systemets klasser och deras relationer till varandra. Klasserna innehåller attribut och metoder och kan även arrangeras i klasspaket för förbättrad överskådlighet.

Objektdiagram

Objektdiagram används för att visa hur objekten i systemet samverkar. Objekten är skapade från klasser och samverkan mellan dessa visas genom sekvensdiagram.

Sekvensdiagram

Ett sekvensdiagram beskriver kedjor av händelser i ett system och syftar till att illustrera förloppet för en eller flera tillståndsförändringar och deras livscykler.

Collaboration -diagram

Diagrammet beskriver hur olika objekt samverkar med varandra och i vilken ordning detta sker. Ett *collaboration* –diagram är innehållsmässigt likt ett sekvensdiagram. Skillnaden är att *collaboration* –diagrammet är en mer statisk bild medan sekvensdiagrammet även illustrerar objektens livslängd.

Tillståndsdiagram

Diagrammet visar vilka tillstånd ett system kan befinna sig i. Ett tillståndsdiagram visar även vilka tillståndsövergångar som är tillåtna. Ett enkelt exempel på en tillståndsmaskin är en läskautomat som kan befinna sig i tillstånden *väntar på betalning* och *väntar på val*. Beroende på hur kunden handlar (lägger i mer pengar, trycker på en knapp etc.) kommer maskinens tillstånd att förändras.

Aktivitetsdiagram

Aktivitetsdiagram kompletterar ett användningsfallsdiagram genom att aktivitetsflöden för ett företags processer redovisas.

Komponentdiagram

Diagrammet visar komponentgrupperingar och deras relationer med varandra. Komponentdiagram kan användas för att illustrera en statisk bild av systemet.

Genomförandediagram

Diagrammet visar hur hårdvaran i ett system ska modelleras. Det visar även vilka komponenter som hör till vilken del i hårdvaran och hur dessa ska kommunicera med varandra.

(Strand, 2001)

2.1.2 Trådar

En tråd inom datavetenskap är en exekveringsenhet i en process. Det är en teknik för att dela ett program i två delar vilka exekveras samtidigt, eller att samtidig exekvering simuleras beroende på operativsystem och processor. Då samtidig exekvering simuleras sker detta genom att processorn byter tråd och process enligt ett tidsschema. Detta är den teknik som används på datorer med en processorkärna och bytet av tråd går så snabbt att en användare inte märker det, t.ex. om man skriver i ett dokument samtidigt som man lyssnar på musik uppfattar man det som skilda processer. (Oaks, Wong, 1999)

Utvecklingen av processorer går snabbt och flerkärnade processorer börjar bli vanliga på marknaden, på dessa kan processer verkligen köras parallellt. I de nyare operativsystemen stöds både flera processorkärnor och en kärna med tidsschema. Det finns även stöd för att implementera program med trådar i flera olika programmeringsspråk.

2.1.3 Graphical User Interface (GUI)

Ett *GUI* är det grafiska gränssnitt som finns mot användaren för att denne skall kunna kommunicera med programmet. Det klassiska sättet att uppnå kommunikation är genom att programmet skriver ut en textrad där viss information efterfrågas, t.ex. ett filnamn. Användaren får därefter fylla i önskat filnamn och programmet bearbetar informationen. Om ytterligare uppgifter krävs från användaren skrivs en ny textrad ut. Resultatet kan därefter sparas i en fil eller skrivas ut direkt på skärmen. Arbetsflödet i en sådan lösning är enkelt att följa, men det kan lätt bli omständligt om t.ex. fel

information har angivits eller om flera filer ska öppnas samtidigt. Med ett *GUI* förenklas den här typen av uppgifter genom att istället skapa ett fönster med knappar, textfält och menyer. Dessa kan väljas genom att använda datormusen eller tangentbordet. Det är även möjligt att t.ex. markera flera olika filer i en lista för att göra något med alla dessa. När uppgifterna är ifyllda trycker användaren på en knapp och data bearbetas sedan. Resultatet kan presenteras i listor, textfält eller genom att ritas upp grafiskt. En stor fördel med ett *GUI* är att det ofta går mycket snabbare att välja uppgifter från listor och menyer än genom att ange all information själv. (Weiss, 2006)

2.1.4 Händelsestyrd programmering

Händelsestyrd programmering innebär att en applikation styrs av en användares handlingar (t.ex. ett musklick) eller meddelanden från andra applikationer. Tekniken används bland annat för att skapa applikationer där användaren själv bestämmer när något ska hända. Antag att ett enkelt program utan händelser utför två uppgifter. Den första är att användaren får ange en färg och den andra är att en text skrivs ut på skärmen i den av användaren angivna färgen. Ett händelsestyrt program kan istället ha en knapp, en lista med färger och en ruta där texten skrivs ut. När användaren klickar på knappen skrivs texten ut och när en färg väljs i listan byter texten färg. Fördelen är att användaren själv kan välja när och vad som ska hända istället för att vara styrd i ett av programmeraren bestämt flöde. Dessutom låses inte programmet i väntan på att användaren skall utföra något.

Händelser (*events*)

En händelse är ett meddelande som avser en användares handling eller en förändring i applikationen som ska hanteras. Händelsen innehåller information om dess typ, ursprung och eventuell extra data.

Ofta finns en mängd fördefinierade händelser i klassbibliotek för *GUI*. Dessa används av komponenterna när en användare klickar med musen eller använder tangentbordet för att skriva något. I de flesta programmeringsspråk är det dock möjligt att definiera egna händelser som kan framkallas (*raise*). Framkallningen kan ske när som helst i programflödet men görs lämpligen när en operation är klar så att övriga komponenter kan meddelas om detta. Händelser hanteras sedan av det underliggande systemet i form av en händelsekö så att dessa utförs i rätt ordning. (Faison, 2006)

Händelsehanterare (*event handlers*)

Händelsehanterare tar hand om framkallade händelser och ser till att rätt åtgärd utförs. Varje definierad händelse har en händelsehanterare kopplad till sig. Denna anropas av systemet när det finns händelser i händelsekön. Händelsehanteraren ser till att rätt åtgärd utförs och återlämnar sedan kontrollen till huvudprogrammet. En åtgärd är exempelvis att rita ut en figur i ett fönster när användaren klickar på en specifik knapp. (Faison, 2006)

2.1.5 Strömmar

En dataström är en sekvens av okänd längd som används för att kunna processa en liten del data åt gången (Nagel m.fl, 2004). En ström kan

jämföras med en kö till en åkattraktion på ett nöjesfält. Attraktionen kan endast ta hand om ett visst antal gäster per åktur och kön kan antingen växa eller minska beroende på hur många gäster nöjesparken har vid just den tidpunkten. En ström fungerar på liknande sätt:

Antag att en serverapplikation är uppkopplad på ett nätverk och har en nätverksström kopplad till sig. Applikationen kan endast processa en viss mängd data per tidsenhet på grund av datorns prestanda. Om den inkommande datamängden per tidsenhet är större än applikationens processförmåga köas inkommande data i strömmen. Det garanterar att inga data avvisas p.g.a. för hög belastning samtidigt som den ankommer i rätt ordningsföljd. (Nagel m.fl., 2004)

2.2 Kommunikationsmodeller

I grund och botten förstår datorer bara ettor och nollor, för att de skall bli användbara lär vi dem att presentera dessa nollor och ettor på ett bra sätt. Detsamma gäller när datorer skall kommunicera med varandra, de pratar med varandra på ett binärt språk, sen hjälper vi dem att tolka det (Forouzan, 2007). I detta avsnitt beskrivs hur man bryter ner kommunikationen till olika nivåer med kommunikationsmodeller. Dessa modeller är utvecklade för att göra det möjligt att byta teknik för en nivå och ändå behålla egenskaperna på en högre nivå.

2.2.1 OSI -modellen

OSI -modellen är en internationell standard för att beskriva nätverkskommunikation. Att överföra information mellan datorer är en komplex uppgift och *OSI* -modellen delar upp denna i sju mindre, mer hanterbara, underuppgifter. Varje underuppgift ansvarar för en specifik del av informationsöverföringen och kommunicerar endast med intilliggande lager (se Figur 2.1). (Forouzan, 2007)

7	Applikationslager
6	Presentationslager
5	Sessionslager
4	Transportlager
3	Nätverkslager
2	Datalänklager
1	Fysiskt Lager

Figur 2.1. OSI-modellen delar upp kommunikationen mellan datorer i sju lager .(Översatt från Peng och Tsou, 2007, s. 45)

Applikationslager

Det översta lagret i *OSI* -modellen, applikationslagret, är den del av modellen som är närmast användaren. De program som nyttjar sig av kommunikation implementerar något av de protokoll som utgör lagret. Till exempel använder en webbläsare protokollet *HTTP* (*HyperText Transfer Protocol*) och ett e-postprogram kan implementera *IMAP* (*Internet Message Access Protocol*). (Forouzan, 2007)

Presentationslager

Presentationslagret syftar till att lösa de representationsproblem som kan uppstå när olika datorsystem används. Teckenkodningen för ett system kan skilja sig i jämförelse med ett annat när det gäller att representera t.ex. bokstäver och siffror. Ett exempel är att bokstaven "a" representeras av heltalet 97 i *ASCII* (*American Standard Code for Information Interchange*) medan samma tecken i *EBCDIC* (*Extended Binary Coded Decimal Interchange Code*), som används av företaget *IBM,* kodas som heltalet 129. (Peng och Tsou, 2003)

Lagret arbetar genom att översätta informationen till ett väl specificerat överföringsformat. Därefter skickas den omarbetade informationen vidare ner i lagerhierarkin. När informationen når den mottagande datorn översätts denna återigen, men nu till den lokala, eventuellt nya, teckenkodningen som används. Detta innebär att tack vare presentationslagret kan två helt skilda datorsystem ändå kommunicera med varandra. (Peng och Tsou, 2003) Det är också här som t.ex. kryptering och komprimering vanligtvis sker (Nagel m.fl., 2004).

Sessionslager

Med en session menas en virtuell anslutning mellan två datorer som är oberoende av underliggande lager. Sessionslagret används för att öppna, upprätthålla och slutligen avsluta en anslutning. Under upprätthållandet av sessionen ligger det på lagrets ansvar att hålla reda på vilken av datorerna som exempelvis sänder data och under hur lång tid detta pågår. (Peng och Tsou, 2003)

Transportlager

Transportlagrets huvuduppgift i den sändande datorn är att förpacka information i paket av lämplig storlek för att i den mottagande datorn packa upp paketen och sätta ihop informationen till ursprungsdata. Lagret kan även, beroende av anslutningstyp, ansvara för att paketen kommer fram. Vid användandet av en pålitlig (*reliable*) anslutning, t.ex. *TCP* (*Transmission Control Protocol*), så skickar lagret en bekräftelse på att informationen når mottagaren. Om den inte har gjort det inom en viss tidsrymd skickas berörda paket om. Om däremot en icke pålitlig anslutningstyp används, t.ex. *UDP* (*User Datagram Protocol*) skickas paketen iväg och om den avsedda mottagaren får dessa byggs de ihop men utan att sända tillbaka en bekräftelse. (Nagel m.fl. 2004; Peng och Tsou, 2003)

Nätverkslager

Nätverkslagret agerar som en vägvisare för paketen så att dessa hittar fram till den avsedda mottagaren. Det är lagrets uppgift att ge möjlighet att skicka data till och från vilka datorer som helst i nätverket och översätta logiska adresser till fysiska adresser. Det åligger dessutom lagret att avgöra den optimala vägen för paketet att ta sig till mottagaren och undvika eventuella överbelastningar. (Forouzan, 2007)

Datalänklager

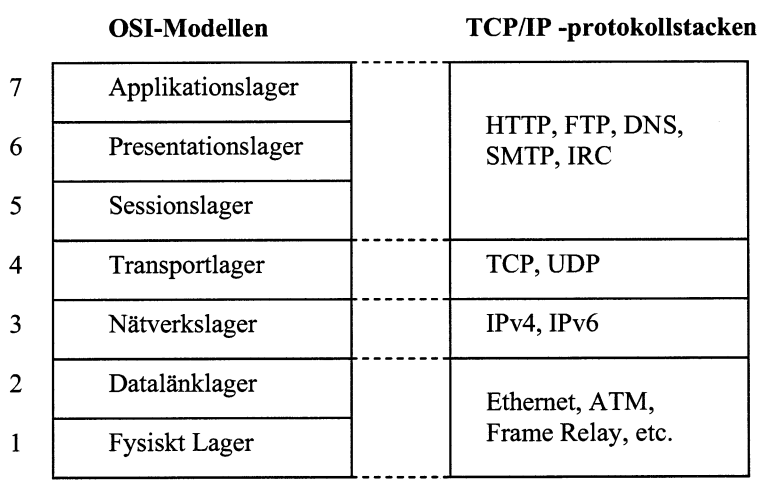
Lagret ser till att den sända informationen tar sig från en nod i ett nätverk till en annan. Det innebär att datalänklagret ansvarar för delsträckorna så att data ska kunna ta sig från avsändare till mottagare. (Forouzan, 2007)

Fysiskt lager

Det fysiska lagret tar hand om hur dataströmmen ska konverteras för att kunna skickas via elektriska eller optiska nätverkskablar. Det är länken mellan datorn och det fysiska nätverket. (Forouzan, 2007)

2.2.2 TCP/IP -modellen

TCP/IP -protokollstacken är en förenklad implementering av *OSI* –modellen (se Figur 2.2 för en jämförelse), den är standard för Internetkommunikation och kallas vanligtvis för Internetstacken. Implementeringen är uppdelad i fyra lager, där *IP* -protokollet motsvarar nätverkslagret och *TCP* eller *UDP* motsvarar transportlagret. De tre översta lagren i *OSI* –modellen, applikations-, presentations- och sessionslagren har slagits ihop till ett applikationslager i *TCP/IP* och finns implementerade i många versioner, t.ex. *HTTP*. De två understa lagren i *OSI* -modellen har även de slagits ihop till ett i *TCP/IP* -modellen där de kallas *nätverksaccess* och finns implementerade som t.ex. *Ethernet*. (Forouzan, 2007)



Figur 2.2. Jämförelse mellan *OSI* –modellen och *TCP/IP* –stacken. (Omarbetad från Peng och Tsou, 2007, s. 53)

2.3 Dataöverföring

För att överföra data via Internet används olika tekniker för olika syften. I avsnittet följer en beskrivning av viktiga standarder. Dels sådana som bygger på att användas som *TCP/IP* –modellens applikationslager och dels de som utvecklas ovanpå modellens transportlager. Dessutom behandlas hur data kan struktureras inför överföring.

2.3.1 Transmission Control Protocol (TCP)

TCP är ett protokoll som tillåter kommunikation mellan datorer över ett nätverk. *TCP* baseras på att en anslutning mellan två datorer skapas. När anslutningen är skapad tillåts datorerna att sända data till varandra. En viktig egenskap hos *TCP* är att all data som sänds över en anslutning garanteras komma fram till sitt mål och att data ankommer i samma ordning som den sändes. Vidare stöds *TCP* av i stort sett alla aktörer på marknaden vilket gör att applikationer på olika operativsystem kan kommunicera med varandra. (Nagel m.fl, 2004)

2.3.2 HyperText Transfer Protocol (HTTP)

HTTP är det protokoll som främst används för överföring av webbsidor. Det är ett så kallat anrops-/svars protokoll vilket innebär att en klient kopplas upp mot en server och efterfrågar viss information. Servern skapar eller hämtar informationen och skickar tillbaka den till klienten varefter uppkopplingen avslutas. (Forouzan, 2007)

Det finns tekniker för att skapa en mer interaktiv kommunikation genom att använda t.ex. *CGI* -parametrar och *cookies*. Med hjälp av *CGI* kan klienten bifoga information i anropet till servern så att denna hämtar eller skapar en sida specifikt för användaren. T.ex. kan en användares personliga sida på en *community* hämtas på detta sätt med användarnamnet som parameter. *Cookies* sparar små mängder text i en lokal fil på användarens dator som webbläsaren kan hämta information från. *Cookies* kan användas för att spara t.ex. användarnamn så att detta endast behöver anges första gången som sidan behövs. (Forouzan, 2007)

HTTP är ett generiskt utvecklat protokoll vilket gör att det kan användas för andra ändamål än att överföra webbsidor. När en webbläsare anropar en server sänder den även med information (så kallade *MIME* -typer) om vilka filer den klarar av att hantera. Om en sådan fil begärs kan webbläsaren direkt öppna filen i webbläsaren och t.ex. spela upp musik eller film. De filer som inte hanteras kan ofta sparas till den lokala hårddisken för att kunna använda den med en annan programvara. Detta innebär att det är fullt möjligt att sända vilken information som helst över *HTTP* så länge både servern och klienten vet hur anropet respektive svaret ska behandlas. En begränsning är dock att det alltid är klienten som initierar anslutningen med ett anrop vilket gör det svårt för servern att meddela klienten om förändringar. (Foruzan, 2007)

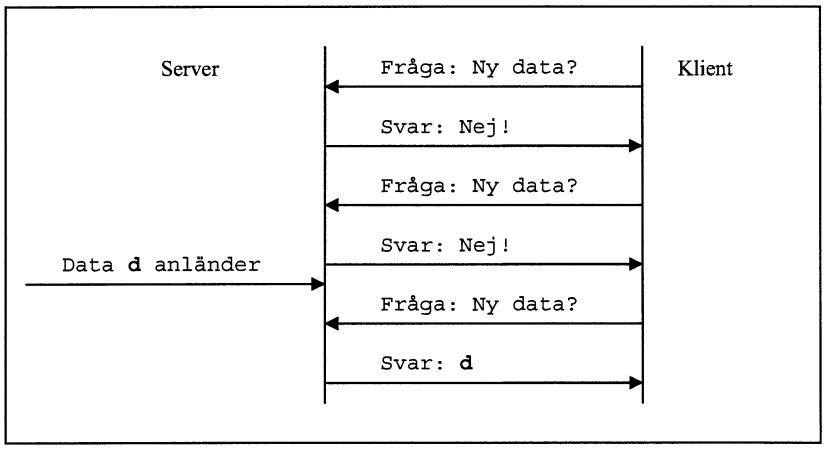
Det finns två typer av anslutningar vid användning av *HTTP*, dels varaktiga (*persistent*) och dels icke varaktiga (*non persistent*). Skillnaden ligger i att en varaktig anslutning hålls öppen en viss tid efter att svaret från servern har skickats, medan anslutningen direkt stängs efter varje svar vid en icke

varaktig. Fördelen med en varaktig anslutning är att om en klient gör många anrop under en kortare tid, kommer serverns resurser inte behöva användas för att koppla upp och ner anslutningar. (Forouzan, 2007)

2.3.3 Bidirectional streams Over Synchronous HTTP (BOSH)

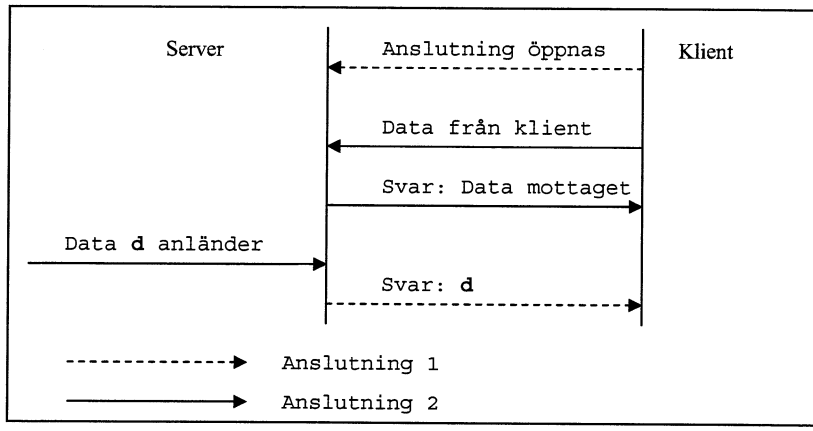
BOSH är en teknik för att upprätta tvåvägskommunikation mellan datorer där ena parten befinner sig bakom en brandvägg och inte kan använda en vanlig *TCP* -anslutning. *BOSH* löser detta genom att utnyttja *HTTP* eftersom detta är det protokoll som oftast tillåts genom brandväggar för att möjliggöra surfning på Internet. (XMPP Standards Foundation, 2007)

En vanlig lösning för att simulera tvåvägskommunikation är så kallad *polling,* vilket innebär att en klient kontinuerligt skickar frågor och kontrollerar om det finns någon data att hämta (se Figur 2.3). Med dessa frågor kan även data som skall skickas till servern följa med. Nackdelen med detta är att belastningen på nätverket blir onödigt stor om det inte finns någon ny data. (XMPP Standards Foundation, 2007)



Figur 2.3. Simulering av tvåvägskommunikation genom *polling*.

BOSH använder sig istället av två samtida anslutningar där en används för att skicka in data till servern och den andra håller en anslutning öppen för att kunna ta mot data (se Figur 2.4). På det sättet minimeras nätverksanvändningen och datorernas resurser. Det är speciellt värdefullt om ett stort antal klienter är anslutna till samma server.



Figur 2.4. Tvåvägskommunikation med *BOSH* vilket innebär att två samtida anslutningar används.

2.3.4 eXtensible Markup Language (XML)

Syftet med märkspråket *XML* är att det ska vara ett standardiserat format för lagring och överföring av data. Formatet ska fungera oberoende av vad det beskriver och det ska inte spela någon roll vilken hård- eller mjukvara som används. (W3C, 2006) *XML* är en standard som har utvecklats av *W3C* och stöds av flera stora mjukvaruföretag.

XML består av vanlig text och är därför lätt för människor att förstå. Vidare är *XML* inte ett språk i den mening att det innehåller en mängd olika fördefinierade element utan är istället en uppsättning regler vilka beskriver hur ett *XML* -dokument är strukturerat. (W3 Schools, 2006) Ett enkelt *XML* -dokument visas i Figur 2.5.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<meddelande ämne="Påminnelse">
            <till>Kalle</till>
            <kopia/>
            <från>Mia</från>
            <text>Glöm inte mötet klockan 15:00</text>
</meddelande>
```

Figur 2.5. Figuren visar ett enkelt *XML* -dokument som beskriver ett meddelande.

Om ett *XML* -dokument följer de specificerade reglerna är det välformulerat (*well formed*) (W3C, 2006). Exemplet i Figur 2.5 beskriver ett meddelande från Kalle till Mia och är välformulerat eftersom det uppfyller följande krav:

- Tomma element beskrivs av både en starttagg och en sluttagg

  Meddelandet ska endast skickas till Kalle så elementet **kopia** är tomt och både börjar och slutar i samma tagg. Skrivsättet <kopia/> är ekvivalent med att skriva <kopia></kopia>.

- Alla attribut är omgivna av antingen apostrofer eller citationstecken

  I elementet **meddelande** är attributet **ämne** omgivet av citationstecken.

- Elementen får inte vara överlappande

  De element som har sin starttagg efter starttaggen för **meddelande** avslutas före sluttaggen för **meddelande**. Detsamma ska gälla för samtliga element i ett *XML* -dokument.

- Den teckenkodning som anges följs

  Första raden i *XML* -dokumentet anger vilken *XML* -version och teckenkodning som används. *ISO-8859-1* kodar tecken ur det latinska alfabetet och här finns även svenska tecken så som å, ä och ö definierade.

- Dokumentet måste innehålla ett, och endast ett, rotelement

  Ett rotelement innehåller alla andra element i dokumentet. I exemplet är **meddelande** rotelement.

Detta är de allmänna regler som gäller för samtliga *XML* -dokument. Dessa beskriver dock endast strukturen för dokumentet. För att definiera innehållet i ett dokument används istället en dokumentmall. I mallen beskrivs vilka element och attribut som ska eller får finnas och fungerar som en säkerhet för att olika program ska kunna läsa ett visst *XML* -dokument om det följer en specifik dokumentmall. Den tidigaste mallen kallas *DTD* (*Document Type Definition*) men är ersatt av det mer komplexa *XML-Schema*. Ett *XML* –dokument som följer en dokumentmall sägs vara giltigt (*valid*) (W3 Schools, 2007).

2.4 Mobila Plattformar

Marknaden för mobiltelefoner är väldigt stor och antalet modeller är näst intill oändligt. För att göra det enkelt att skapa applikationer som går att köra på många olika modeller oavsett operativsystem, bygger mobiltillverkarna in stöd för så kallade *MIDlets* i sina telefoner. Dessa använder *JME* (*Java Micro Edition*) som plattform. En annan vanlig plattform för mobila enheter är *Windows Mobile*. Vid utveckling av applikationer till dessa kan *.NET Compact Framework* användas.

2.4.1 Hårdvara

Det finns en rad olika möjliga alternativ av hårdvara att välja mellan för en mobil enhet. Den hårdvara som används måste ha en anslutning till Internet samt möjlighet att kommunicera med en *GPS*. I övrigt har de olika typerna av möjlig hårdvara varierande egenskaper vilket gör dem till tänkbara alternativ i olika tillämpningar. Nedan listas olika grupper av hårdvara med en förklaring och kort översikt av deras egenskaper.

- *Mobiltelefon*. I första hand en enhet för kommunikation via tal men som ofta även klarar av datakommunikation via Internet. Många gånger finns också möjlighet att installera och använda enklare program och spel. Kommunicerar med andra enheter via *bluetooth*, *IR* och i vissa fall *WLAN*.

- *Smartphone.* Mobiltelefon med utökad funktionalitet och prestanda där mer avancerade program kan användas. Klarar av att läsa och redigera vanliga filformat från stationära datorer som textdokument och bilder. En *smartphone* brukar vara något större än en mobiltelefon och kan ha t.ex. *touchscreen* och tangentbord för att underlätta inmatning. Kommunicerar med andra enheter på samma sätt som en mobiltelefon.

- *Handdator* (*PDA - Personal Digital Assitant*). En digital planeringskalender som även kan användas för att läsa och redigera vanliga filformat. En handdator har vanligen en *touchscreen* och används för att skriva in möten, göra noteringar och för att visa t.ex. textfiler. Jämfört med en *smartphone* är handdatorn mer inriktad på kalender- och filhanteringsfunktioner även om det finns produkter med inbyggd telefon. Kommunikation med andra enheter sker via *bluetooth, IR* eller *WLAN*. Åtkomst till Internet kan fås genom *WLAN* eller att ansluta till en annan enhet, t.ex. mobiltelefon, som i så fall fungerar som modem.

- *Tablet PC.* En bärbar dator som endast består av en skärm med *touchscreen* –funktionalitet. En *Tablet PC* är i samma storlek som en hopfälld bärbar dator om än tunnare och lättare på grund av att tangentbord i de flesta fall saknas. Inmatning sker istället genom att peka och skriva på skärmen med en penna. Prestanda är i klass med bärbara datorer och samma operativsystem och program kan köras. Den här typen av dator kan användas för att t.ex. kartera eller göra skisser på ett smidigt sätt eftersom man kan rita direkt på skärmen. Kommunikation med andra enheter sker via *bluetooth, IR, WLAN* eller *USB*. Åtkomst till Internet fås genom *WLAN* eller anslutning till annan enhet som fungerar som modem.

- *Laptop.* En fullt utrustad bärbar dator. Har tangentbord för inmatning tillsammans med en tryckkänslig styrplatta som ersätter datormusen. Kommunicerar med andra enheter via *bluetooth, IR, WLAN* eller *USB*. Åtkomst till Internet fås genom *WLAN* eller anslutning till annan enhet som fungerar som modem. Det finns även expanderingskort att köpa från mobiltelefonioperatörerna för att få tillgång till mobilt Internet utan att ansluta till annan enhet.

2.4.2 Java Micro Edition (JME)

JME är en begränsad variant av programmeringsspråket *Java* -s standardklasser och används för utveckling på vissa handdatorer, mobiltelefoner och andra mindre enheter. Mobila enheter har i regel sämre prestanda och mindre minne än stationära datorer och därför kan inte alla delar ur ett programmeringsspråk användas. *JME* är utvecklat av *Sun Microsystems* och stöds av de flesta nyare mobiltelefoner.

Alla enheter som stöder *JME* implementerar en profil. De två vanligaste profilerna är *MIDP* (*Mobile Information Device Profile*) och *PP* (*Personal Profile*). *MIDP* är avsett för mobila enheter som mobiltelefoner medan *PP* är inriktat mot handdatorer och så kallade *STB* (*Set-Top-Box*) vilket t.ex. är externa enheter för mottagning av bild och ljud som kan kopplas till en TV

(Sun Microsystems, 2007). Examensarbetet behandlar endast så kallade *MIDlets* vilket är program utvecklade för *MIDP*.

MIDP innefattar två undergrupper som begränsar användningen i olika grad, *CDC* (*Connected Device Configuration*) och *CLDC* (*Connected Limited Device Configuration*). *CDC* är den variant som är minst begränsad av de två och kan använda nästan alla *Java* -bibliotek som inte är relaterade till *GUI* (*Graphical User Interface*) (Sun Microsystems, 2007). De flesta mobila enheter implementerar dock inte detta utan det mer begränsade *CLDC* där en av de viktigaste komponenterna är ett *GUI* anpassat för enheter där inmatningen och programstyrning sker med några få knappar. Förutom detta finns en mängd valfria tilläggskomponenter så som *API* -er för såväl *bluetooth*- och *TCP/IP*- kommunikation.

2.4.3 .NET Compact Framework

.NET är *Microsofts* utvecklingsplattform vilken innehåller stöd för en mängd olika programmeringsspråk. *.NET* innehåller en stor samling klasser som är färdiga att användas som komponenter i applikationer. Utvecklingen sker i *Microsoft Visual Studio* som ger utvecklaren möjlighet att enkelt skapa *GUI* -s och dessutom dra nytta av det stora klassbiblioteket.

Compact Framework är en avskalad version av *.NET* speciellt utvecklad för mobila enheter. Även för *Compact Framework* sker utvecklingen i *Microsoft Visual Studio* och utvecklingen av *GUI* -s är specialanpassad för mobila enheter med *touchscreen*- funktionalitet och de mindre skärmarna. Skillnaden mot *.NET* är att *Compact Framework* har ett mindre klassbibliotek vilket kan innebära att applikationerna blir begränsade till viss del. Vid utveckling i *Visual Studio* kan man använda den så kallade *debuggern* tillsammans med *ActiveSync* om den mobila enheten har stöd för detta. Fördelen med detta är att all felhantering och kontroll av kod kan göras direkt på enheten vilket gör det lätt att upptäcka och lösa eventuella tillverkar- eller enhetsspecifika problem. En nackdel med *ActiveSync* är dock att programvaran under *debuggningen* av säkerhetsskäl deaktiverar alla nätverksenheter när detta används. Det gör det svårt att utveckla och testa nätverksapplikationer med hjälp av *ActiveSync*.

2.5 Säkerhet

Säkerhetsaspekten är i dagsläget mycket viktigt vid utveckling av olika nätverkslösningar för att förhindra att obehöriga får åtkomst till potentiellt känslig information. De verksamhetssystem som används är oftast skyddade av en så kallad brandvägg för att undvika att systemet utsätts för onödiga risker. Därför ges här en teoretisk bakgrund till hur en brandvägg fungerar. En annan viktig säkerhetsaspekt är att man vid överföring av information på Internet inte har någon kontroll över vilken rutt informationen som skickas mellan två datorer tar. Detta gör att det är omöjligt att veta hur många och vilka som har möjlighet att inspektera innehållet i datapaketen. Genom kryptering kan dock obehöriga förhindras från att återskapa informationen (se vidare avsnitt 2.5.2).

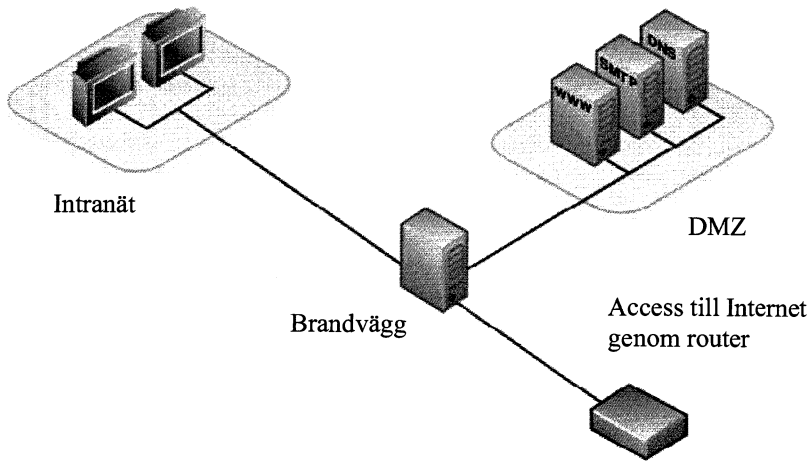
2.5.1 Brandväggar

Brandväggar används för att skydda en dator eller ett nätverk från åtkomst av obehöriga. För hemanvändare installeras ofta brandväggen som en mjukvara medan en brandvägg hos ett företag många gånger är en extern enhet vilken skyddar hela företagets intranät och även kan fungera som nätverksdelare.

En brandvägg tillåter eller nekar trafik på vissa portar med bestämda protokoll och *IP* -nummer. Administratören konfigurerar brandväggen genom att ange vilka portar som tillåts för utgående respektive inkommande anslutningar. Utgående anslutningar är sannolikt de som användaren av en dator tar initiativ till. T.ex. genom att surfa till en hemsida eller koppla upp sig mot en e-postserver. Därför tillåts dessa ofta i långt större utsträckning än inkommande anslutningar som ofta nekas helt. Detta beror på att det sällan är önskvärt att en användare kör en server på sin dator eftersom detta potentiellt kan utnyttjas av hackare. Många företag har ändå behov av att tillhandahålla en webb- eller filserver och den vanligaste lösningen för detta är att placera dessa på en del av företagets nätverk som är extra skyddat. (McClure m.fl, 2003)

En *DMZ* (*DeMilitarised Zone*) är en avskärmad del av ett lokalt nätverk där aktivitet som normalt sett inte bör tillåtas kan ske (se Figur 2.6). Rent fysiskt är den avskärmade delen inget mer än en extra port i brandväggen där en eller flera datorer kan kopplas in. Dessa datorer fungerar sedan som de servrar man vill ska vara åtkomliga för vem som helst. Samtidigt så nekar man alla inkommande anslutningar till företagets intranät. Fördelen är att även om en hackare skulle få kontroll över en server kommer han eller hon inte att kunna komma åt datorerna på intranätet. (McClure m.fl, 2003)

En annan metod för att hindra obehöriga att komma igenom brandväggen är att använda en så kallad *RADIUS* –server (*Remote Authentication Dial In User Service*). En klient ansluter då med användarnamn och lösenord till servern som i sin tur öppnar en fördefinierad port för det *IP* –nummer klienten använder. (Internet Engineering Task Force, 2000).



Figur 2.6. Schematisk figur över hur ett intranät med en DMZ och brandvägg kan vara kopplat mot Internet. (Omarbetad från Wikipedia, 2007)

2.5.2 Kryptering

Det är i många fall väldigt viktigt att ingen får reda på vilken information som någon skickar. Det kan handla om kontokortsuppgifter, hemliga dokument eller bara en vilja av att känna sig säker.

Avlyssning kan ske var som helst på vägen när data transporteras. Antingen i någon av de punkter som skickar vidare informationen eller genom att rent fysiskt gå in och koppla in en dator i nätverket och sedan analysera alla paket som skickas där igenom. Paketen innehåller all information som behövs för att bygga ihop ursprungsmeddelandet (det är ju det måldatorn ska göra) och därför är det ytterst osäkert att skicka t.ex. kontokortsuppgifter, eller inloggningsuppgifter över Internet.

För att komma runt detta har man utvecklat tekniker som omöjliggör att informationen i paketen kan tolkas. Det handlar då om att förvränga data på ett visst sätt som gör att endast den avsedda mottagaren åter kan tolka denna. Detta kallas kryptering och har använts länge, redan Julius Caesar använde en enklare form och under andra världskriget använde sig Tyskland av en krypteringsmaskin kallad Enigma.

En enkel form av kryptering är att t.ex. förskjuta alla tecken i ett meddelande ett steg, det vill säga att ”a” blir ett ”b”, ”b” blir ett ”c” och så vidare. Om mottagaren vet hur många steg varje tecken är förskjutet blir det enkelt att få tillbaka ursprungsmeddelandet. Det antal tecken som meddelandet är förskjutet kan ses som meddelandets nyckel. I exemplet ”låses” meddelandet med nyckeln ”1” och låses upp med samma nyckel. Mer avancerade metoder förskjuter varje tecken olika för att försvåra analys av mönster i meddelandet för att på så vis kunna ta fram ursprungsmeddelandet. (Ford, 1994)

En krypteringslösning där samma nyckel används för att både kryptera och dekryptera ett meddelande kallas symmetrisk och fördelen är att dessa är snabba och enkla att använda (Ford, 1994). Det finns idag en mängd olika algoritmer för att kryptera data som är omöjliga att knäcka annat än genom att prova alla möjliga nycklar. En stor nackdel är dock följande:

Antag att en person vill köpa något genom en Internetaffär och betala med sitt kontokort. Då måste kontokortsnumret, namn, adress m.m. anges för att detta ska kunna genomföras. Om uppgifterna inte är krypterade kan vem som helst få tillgång till dem genom att analysera de paket som skickas. Lösningen borde vara att kryptera all information. Frågan är bara hur det ska göras. Hemsidan måste kunna se alla uppgifter och behöver således tillgång till rätt nyckel. Likaså behöver personen som genomför köpet tillgång till samma nyckel. Men för att distribuera nyckeln kommer Internet att användas och trafiken kan då avlyssnas och nyckeln bli tillgänglig för andra. Det är med andra ord osäkert. Därför finns asymmetriska nycklar.

Med asymmetriska nycklar bildar man ett nyckelpar där den ena nyckeln, kallad publik, används för att kryptera informationen. Den andra, den privata nyckeln, används för dekryptering. Den stora fördelen är att den publika nyckeln är oanvändbar annat än för att kryptera information (Ford, 1994). I exemplet ovan kan alltså personen som vill genomföra köpet begära hemsidans publika nyckel, kryptera sin information och sedan skicka in den.

Hemsidan använder sedan sin privata nyckel för att dekryptera samma information och köpet kan genomföras. Tekniken används bland annat i *SSL* (*Secure Sockets Layer*).

2.6 Prestanda

För att mobila system ska hålla en hög prestanda finns ett antal aspekter att tänka på. En begränsande faktor är att mobila enheter vanligtvis kommunicerar med Internet genom en *GPRS* - eller *3G* –uppkoppling. Dessa uppkopplingar har i jämförelse med fasta uppkopplingar en låg överföringshastighet, därför är det intressant att t.ex. komprimera den data som skall överföras.

2.6.1 Komprimering

Komprimering används för att begränsa mängden data som lagras eller överförs. Därmed kan t.ex. större mängd information överföras på en given tid. Att komprimera data kan göras genom att antingen radera överflödig information eller genom att strukturera om informationen så att denna tar så lite plats som möjligt.

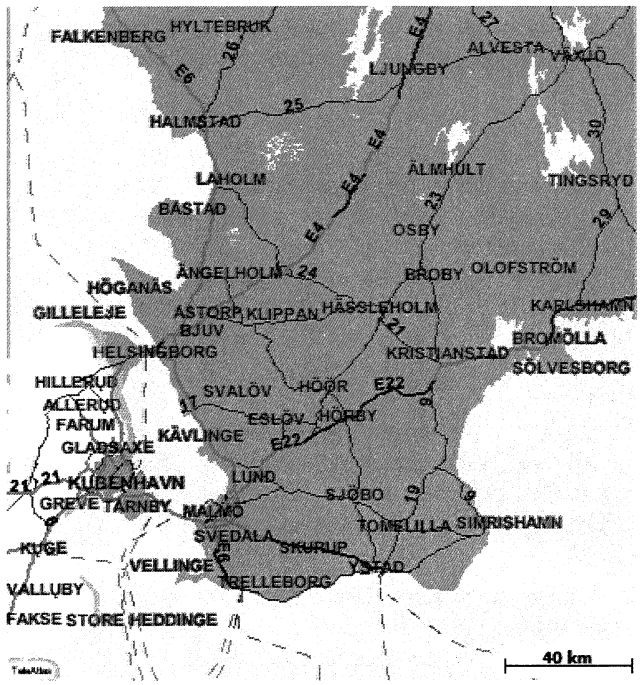
Destruktiv komprimering innebär att vissa delar av informationen tas bort och därmed minskar datamängden. Den stora nackdelen är att den raderade informationen inte kan återskapas och därför lämpar sig tekniken endast för syften där en viss skillnad mellan ursprungsdata och komprimerad data spelar liten roll. Vanliga tillämpningar för destruktiv komprimering är ljud- och bildfiler med filformat som *mp3* och *JPEG*. Exempelvis förlorar en ljudfil en stor del av den ljudinformation som ligger utanför örats uppfattningsförmåga när den konverteras till *mp3*-formatet. (Khalid, 2000)

Vid bevarande komprimering kan all information återskapas till sitt ursprungliga format. Detta åstadkoms genom att man genom algoritmer strukturerar om den data som skall komprimeras så att den tar så lite plats som möjligt. (Khalid, 2000) Antag t.ex. att en textfil innehåller ordet "komprimering" ett stort antal gånger. Principen är sedan att ersätta vanliga teckenkombinationer med ett eller några få tecken. I den komprimerade filen som skapas skrivs även information om hur de tecken eller teckenkombinationer ska tolkas när filen dekomprimeras.

2.6.2 Anslutning till Internet

En förutsättning för mobil datakommunikation är att en anslutning till Internet finns tillgänglig. Anslutningen kan ske på olika sätt beroende på den mobila enhetens egenskaper och täckningen i området den befinner sig.

I dagsläget dominerar två tekniker marknaden för mobil uppkoppling mot Internet, *GPRS* och *3G*. Båda använder sig av mobilnätet för att skapa en uppkoppling och för användaren skiljer sig endast hastigheten, där *3G* är snabbast men har sämre täckning. En telefon som kan utnyttja *3G* har dock i regel även möjlighet att använda *GPRS* och kombinationen av dessa ger en mycket god täckning (se Figur 2.7 för täckningskarta över sydvästra Sverige). Mobiloperatörerna tar ofta betalt för mängden data som överförs även om det även finns abonnemang där en fast avgift betalas för obegränsad tillgång.



Figur 2.7. Täckningskarta för *3G*, *GPRS* och *EDGE* över sydvästra Sverige. Mörkgrå färg på landytor anger att täckning finns. Vita prickar och ytor innebär obefintlig täckning. (Telia, 2007)

Det finns ibland även möjlighet att använda *WLAN* (*Wireless Local Area Network*), vilket är ett trådlöst nätverk som kan nås om enheten har ett nätverkskort som stödjer detta. Ett antal kostnadsfria *WLAN* har byggts ut i städer och då framförallt på publika platser som tågstationer, bibliotek eller shoppingcenter. Dessutom täcks ännu större områden in av leverantörer som tar betalt för denna tjänst, ofta per tidsenhet. *WLAN* skiljer sig mot *3G* och *GPRS* genom att kunna erbjuda en betydligt högre överföringshastighet men har å andra sidan väldigt begränsad täckning jämfört med dessa.

Utvecklingsmässigt spelar det ingen roll vilken av teknikerna som används för att skapa en uppkoppling eftersom alla bygger på att enheten tilldelas en *IP-adress* för att kunna kommunicera. De problem som kan uppstå och som måste hanteras av utvecklaren är likvärdiga för alternativen. T.ex. kan samtliga tappa uppkopplingen och det finns risker för långsam överföringshastighet beroende på tappade datapaket på grund av dålig täckning.

2.7 Positionering

Global Navigation Satellite System (*GNSS*) kallas de navigationssystem där man med hjälp av satelliter kan positionera en enhets läge. I skrivande stund finns det endast ett fullt fungerande *GNSS* -system vilket är det av USA utvecklade *GPS* (*Global Positioning System*) som felaktigt blivit ett vedertaget begrepp för samtliga *GNSS*. Vidare försöker Ryssland åter få sitt system, *GLONASS*, operationellt samtidigt som det pågår projekt i både Europa (*Gallileo*) och Kina (*Compass*).

Systemen skiljer sig något åt men bygger på samma grundtanke. *GNSS* - mottagaren tar emot signaler från de satelliter den har programmerats med och har fri sikt till. En mjukvara i mottagaren tolkar informationen som signalen bär med sig och kan sedan beräkna positionen om kontakt finns med minst fyra satelliter. Vid positionsbestämningen beräknas även ett antal noggrannhetsmått och därefter skickas resultatet vidare tillsammans med eventuell annan information via ett överföringsprotokoll. (Hofmann – Wellenhof, 2001)

2.7.1 Överföringsprotokoll

Inom *GPS* används överföringsprotokollet *NMEA 0183* för att på ett standardiserat sätt kunna överföra positioneringsinformationen från *GPS* - mottagaren till t.ex. ett kartprogram. Detta möjliggör för mjukvarutillverkare att stödja flera olika *GPS*- mottagare trots att dessa kan vara tillverkade av olika företag. *NMEA 0183* är utvecklat av det USA-baserade *National Marine Electronics Association*. Protokollet är från början utvecklat för kommunikation mellan elektroniska marina enheter så som radar och sonar men har kommit att användas för samtliga *GPS* - mottagare, även de som är tillverkade för att användas på land. Specifikationen är avgiftsbelagd men stora delar av den finns ändå tillgänglig via privata hemsidor.

NMEA 0183 använder ett enkelt *ASCII*-protokoll vilket specificerar hur data överförs mellan enheter och börjar alltid med ett dollartecken. Därefter följer vilken typ av meddelande som skickas och ett antal kommaseparerade fält med data. Meddelandet avslutas med en stjärna och en kontrollsiffra. (Baddeley, 2007)

De mest grundläggande meddelandena som används vid överföring av *GPS*-data är *GPGGA* och *GPGLL*. *GPGGA* beskriver en *GPS* -fix och innehåller förutom positionen även fält för noggrannhet, antal använda satelliter och annan data från *GPS*-systemet. Ett *GPGLL* -meddelande innehåller endast fält för positionen (se Figur 2.8). (Baddeley, 2007)

```
$GPGLL,57.252612,N,12.304523,E*25
```

Figur 2.8. Exempel på ett NMEA -meddelande

2.7.2 Noggrannhet

Noggrannheten vid en positionsbestämning med *GPS* avgörs dels beroende på vilken typ av mottagare som används, och dels på hur bra mätningsförhållanden som mätningen utförs i.

Olika typer av *GPS*-mottagare på marknaden använder olika tekniker för att beräkna sin position och därmed påverkas också den noggrannhet som kan uppnås. De enklare varianterna använder kodmätning och kan göras både små och billiga. Det brukar sägas att de har en noggrannhet på ungefär tio meter. För att öka noggrannheten kan vissa *GPS*-mottagare även använda satelliternas bärvåg för att beräkna positionen. Noggrannheten förbättras till ca en centimeter samtidigt som mottagaren blir större och dyrare.

Mätningsförhållanden kan ofta inte påverkas annat än genom att invänta molnfritt väder eller att välja en tidpunkt där så många satelliter som möjligt kan användas. Däremot är det önskvärt att ge användaren av både enheten och data ett mått på hur bra noggrannheten är vid mätningstillfället.

Ett noggrannhetsmått som ofta beräknas är *DOP* (*Dilution Of Precision*). *DOP* -värdet ger ett mått på hur bra satellitkonfigurationen är vid mätningstillfället. Ett högt värde innebär att satelliterna ligger nära varandra, vilket ger sämre noggrannhet än om de har en god spridning. Genom att ta hänsyn till *DOP* -värdet kan användaren ta ställning till om denne bör vänta eller utföra mätningen direkt. (Hofmann- Wellenhof, 2001)

Genom att använda olika tekniker kan dock vissa negativa inverkningar från mätningsförhållanden undvikas. Genom att använda korrektionsdata från fasta eller mobila referensstationer kan noggrannheten ökas betydligt. Korrektionsberäkningarna kan göras som efterberäkningar där hänsyn tas till värden från ett antal referensstationer. Beräkningarna kan även göras i realtid med tekniken *DGPS* (*Differentiell GPS*). Mottagaren är då uppkopplad mot en central för att kontinuerligt kunna få korrektionsdata som gäller för det område mottagaren befinner sig i. (Hofmann –Wellenhof, 2001)

2.7.3 Kommunikation med *GPS* –mottagare

Informationen i följande avsnitt är baserat på erfarenheter från löpande tester gjorda av rapportförfattarna.

För att överföra positions information från en *GPS* –mottagare till en annan enhet måste de kunna kommunicera med varandra. Denna kommunikation kan åstadkommas på olika sätt.

Ett alternativ är att upprätta kommunikationen till en *GPS* –mottagare med *bluetooth* via en virtuell kommunikationsport. Den virtuella porten skapas genom att para ihop *GPS* –mottagaren med en för operativsystemet fördefinierad port. Den programvara som ska använda informationen från *GPS* –mottagaren lyssnar därefter på porten. Ett annat alternativ är att programvaran själv söker av närområdet efter *bluetooth* –enheter. Därefter kan en anslutning skapas till en specifik *bluetooth* –enhet via dess unika *URI (Uniform Resource Identifier)*.

För de mobila enheter som har en inbyggd mottagare finns två andra tekniker att komma åt *GPS* –data. Programvaran ansluter antingen till enhetens fysiska kommunikationsport eller så görs data tillgängligt via ett gränssnitt kallat *Location API*.

2.8 Standarder för geografiska data

Då intresset för geografisk information stadigt har ökat har också en mängd olika standarder utvecklats inom området. Standarderna behandlar hur geografiska data representeras vid lagring och överföring. *Open Geospatial Consortium* (*OGC*) är en sammanslutning av över 300 företag, universitet och statliga förvaltningar som verkar för att utveckla sådana standarder. Specifikationerna för dessa är tillgängliga för alla och syftar till att förenkla utbyte av data mellan olika källor och även säkerställa såväl bakåt- som

framåtkompabilitet. *OGC* benämner sina standarder med "*Open GIS*" som är ett registrerat varumärke i en mängd olika länder. (OGC, 2007)

2.8.1 Geography Markup Language (GML)

GML är en standard utvecklad av *OGC* för överföring och lagring av geografisk information. Standarden är öppen för alla och en av grundtankarna med formatet är att informationen skall vara tillgänglig oberoende av vilken hård- och mjukvara som används. Vidare ska formatet användas för just den geografiska informationen, det vill säga koordinaterna för objekten och eventuell tilläggsinformation. Därefter kan kartor genereras utifrån informationen i *GML* -filen och då görs också eventuella förenklingar och urskiljning i vad som skall visas. *GML* är en *XML* –dialekt och ärver därför de grundregler som gäller för *XML* –dokument (se avsnitt 2.3.4).

I *GML* finns ett antal olika objekt definierade som används för att beskriva de geografiska objekten med t.ex. geometrier, topologi, koordinatsystem och tid. Ett objekt kan ha flera olika attribut som kan innehålla ytterligare information så som antalet våningar i en byggnad. Det finns även någon som kallas samlingar (*collections*) som beskriver flera olika objekt som hör ihop. (OGC, 2004)

Dokumentet i Figur 2.9 beskriver en punkt vilken anger platsen för en orienteringstävling. Alla *property* -taggar beskriver information som är kopplad till tävlingen och slutligen anges geometrin, i detta fall en punkt, med sina koordinater och vilket koordinatsystem dessa anges i.

```
<featureMember typeName="point">
   <Feature typeName="pointOfEvent">
     <property typeName="pointID" type="Integer">1</property>
     <property typeName="name" type="string">SM Sprint</property>
     <property typeName="date" type="string">8/4</property>
     <property typeName="ref" type="string">http://www.sm2007.se/</property>
     <property typeName="notes" type="string">Svenska mästerskapen
                                              I sprintorientering</property>
     <geometricProperty typeName="_">
       <Point srsName="EPSG:4326">
         <coordinates>
         57.10567321405914, 12.240486145019531
         </coordinates>
       </Point>
     </geometricProperty>
   </Feature>
</featureMember>
```

Figur 2.9. Ett exempel på del av ett GML-dokument.

2.8.2 Web Feature Service (WFS)

WFS är ett gränssnitt för överföring och manipulation av geografiska data via *HTTP*. Svaret till ett *WFS* -anrop är kopior på efterfrågad data vilket innebär att det är möjligt att förändra och utföra analyser lokalt på *WFS* -klienten. Om data har förändrats är det även möjligt att uppdatera och även skapa ny data hos servern. *WFS* använder normalt sett *GML* som överföringsformat men är inte begränsat till det utan kan även använda t.ex. *shape* -filer.

Nedan följer de funktioner som måste implementeras av samtliga *WFS* -servrar.

- *GetCapabilities.* Anropet används för att ta reda på vilka tjänster servern tillhandahåller. Svaret är ett *XML* -dokument vilket bland annat anger vilka objektstyper som kan efterfrågas, vilket referenssystem som används och vilka geografiska områden som täcks. Dessutom anges om de utökade funktionerna för manipulering och låsning av data kan användas.
- *DescribeFeatureType.* Anropet används för att begära information om ett visst objekt. Svaret kan innehålla t.ex. metadata eller en beskrivning av objektet.
- *GetFeature.* Med anropet *GetFeature* efterfrågas ett specifikt eller en mängd olika geografiska objekt och svaret blir ett *GML* -dokument som beskriver dessa.

(OGC, 2005)

Med en utökad *WFS* kan man även använda metoder som förändrar data hos källan (servern).

- *InsertFeature* används för att skapa ett nytt objekt hos källan.
- *UpdateFeature* används för att uppdatera ett objekt.
- *DeleteFeature* används för att ta bort ett objekt.
- *LockFeature* är ingen metod som manipulerar någonting utan används för att flera klienter ska kunna jobba mot databasen samtidigt och gör det möjligt att låsa en eller flera geometrier för att kunna utföra förändringar på dessa samtidigt som andra klienter då inte kan förändra samma geometri.

(OGC, 2005)

2.8.3 Web Map Service (WMS)

WMS producerar kartor i form av bilder utifrån en förfrågan av en klient. Detta skiljer sig från *WFS* som returnerar faktisk data. En *WMS* -klient kan vara en webbläsare och frågorna ställs med vanliga *CGI* -parametrar. Det finns tre varianter på en *WMS*: *Basic*, *Queryable* och *Cascading*. Med en *Basic* kan man endast hämta kartbilder, om en *WMS* är *Queryable* finns även möjlighet att hämta information om geometrierna i kartbilden. Om en *WMS* är *Cascading* fungerar den som ett nav för flera olika *WMS* -er så att informationen från dem alla kan kombineras. (OGC, 2004)

- *GetCapabilities.* Precis som en *WFS* -server kan *WMS* ge information om vilka anrop servern klarar av. Svaret innehåller information om vilka lager som erbjuds, i vilka geografiska områden dessa finns, vilka koordinatsystem som finns att tillgå och vilka format resultatbilden kan levereras i.
- *GetMap.* I en *GetMap* -fråga anges vilka lager och vilken information bilden skall innehålla. Det är den här frågan som resulterar i att en bild skickas tillbaka.
- *GetFeatureInfo.* Med frågan kan information om ett objekt hämtas. Vanligen klickar användaren i sin kartbild, koordinaterna skickas

sedan till servern som söker fram vilket objekt som det klickades på och informationen returneras. Funktionaliteten behöver inte implementeras.

(OGC, 2004)

3 Generisk lösning

Detta kapitel behandlar en generisk lösning vilken skall fungera som grundlösning för integrering av mobila enheter i ett verksamhetssystem. En analys av problemformuleringen görs för att fastställa krav för grundlösningen. Med kapitel 2 (Tekniska förutsättningar) och kraven som underlag skapas sedan en systemdesign för en generisk lösning.

3.1 Analys av problemformulering

I rapportens problemformulering (se avsnitt 1.2) beskrivs ett antal frågeställningar vilka här skall utredas. Vissa av dessa frågeställningar kan ses som krav, som systemdesignen för grundlösningen måste uppfylla. Under litteraturstudierna och genom diskussioner med handledare har frågeställningarna utökats.

- Först frågades vilken hårdvara som är möjlig att använda. Då syftet med systemdesignen är att skapa en grundlösning för kommunikation är det viktigt att kunna implementera systemdesignen på så många typer av mobila enheter som möjligt. Detta för att skapa möjlighet att till varje specifik tillämpning välja den mobila enhet som passar bäst.
- Den mobila enheten skall kunna hämta positioneringsinformation från en inbyggd eller *bluetooth-* ansluten *GPS*. Alla de tillämpningar som är intressanta att utveckla behöver kunna hålla reda på den mobila enhetens position. Genom att tillåta både inbyggda och *bluetooth-* anslutna *GPS* –er införs inga begränsningar i grundlösningen och den blir på så sätt mer flexibel.
- Att grundlösningen skall vara kostnadseffektiv är ett måste och det finns flera vägar för att uppnå detta. En av dessa är att bygga systemdesignen kring öppna gränssnitt vilket innebär att inga licenser behöver köpas. En annan väg är att grundlösningen används för att utveckla flera olika färdiga system. Detta gör att utvecklingskostnaden för grundlösningen endast behöver betalas en gång. Systemdesignen måste även skapa förutsättning för ett rationaliserat arbetsflöde, nyckeln till detta är att möjliggöra en tvåvägskommunikation mellan de mobila enheterna och verksamhetssystemet.
- Kommunikationen som sker kan innehålla verksamhetskänslig data och måste därför skyddas. För att uppnå kommunikation mellan mobila enheter och ett verksamhetssystem måste data skickas över Internet. Vilken väg denna data tar kan ej kontrolleras och det skapar möjlighet för obehöriga att avläsa den skickade informationen.
- Om en mobil enhet befinner sig i radioskugga måste grundlösningen meddela detta. Olika tillämpningar kräver olika åtgärder för de fall då den mobila enheten tappar sin anslutning. För att grundlösningen skall vara flexibel måste åtgärderna utvecklas för de unika tillämpningarna men grundlösningen skall ge information om att meddelanden kan skickas och om de kommit fram.

- Grundlösningen skall kunna installeras och köras med minimal konfiguration av brandväggar. De flesta organisationer jobbar hårt med att motverka dataintrång och ser det som en mycket prioriterad fråga. En brandvägg skall skydda den verksamhetskänsliga informationen och att öppna en port i denna innebär alltid en risk. Genom att utveckla en grundlösning där inga portar behöver öppnas minskar risken för intrång. Att använda en lösning där inga portar behöver öppnas kan även förkorta ett projekts startsträcka då en utdragen process kan krävas för att öppna en port.
- Grundlösningen skall möjliggöra överföring av såväl strängar som datafiler. Den kommunikation som sker mellan de mobila enheterna och verksamhetssystemet kan bestå av enkla medelanden om vart en enhet befinner sig. För detta krävs att strängar kan överföras. Det kan finnas tillämpningar då man vill skicka mer komplex information, t.ex. en bild.

Med utgångspunkt från denna analys har en kravspecifikation för grundlösningen utformats. Kravspecifikationen återfinns i Appendix A.

3.2 Systemdesign

Detta avsnitt beskriver grundlösningens systemdesign. Denna är utvecklad med utgångspunkt från tekniska tester under litteraturstudier samt diskussioner med handledare. Lösningen innehåller flera delar, därför görs först en beskrivning av den övergripande designen och sedan en mer detaljerad beskrivning av de olika delarna. Målet med systemdesignen är att uppfylla kraven i kravspecifikationen för grundlösningen. De tekniker som används är beskrivna i kapitel 2.

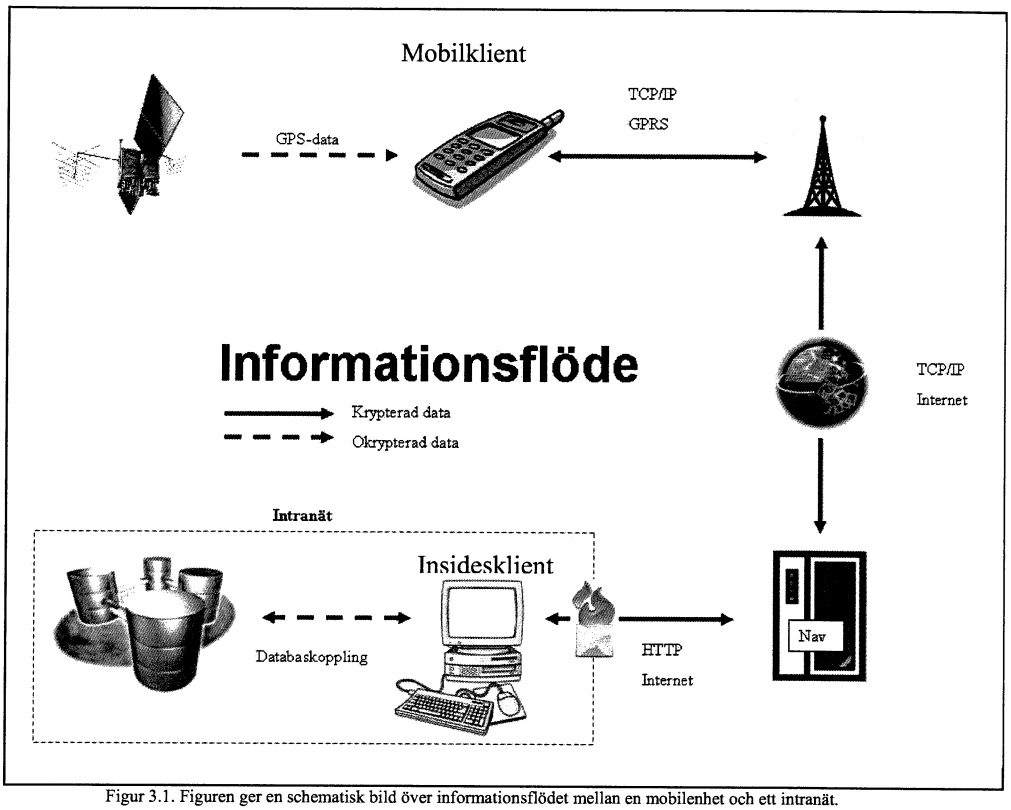
3.2.1 Övergripande systemdesign

Grundlösningen består av tre delar.

- *Mobilklient*. Syftar på den del av lösningen som skall installeras på en mobil enhet. Grundlösningen för denna del innehåller komponenter för att tolka *GPS* –data och ansluta till navet.
- *Insideklient*. Syftar på en del av lösningen vilken skall installeras på en maskin inne på intranätet vilken har tillgång till en organisations verksamhetssystem. Grundlösningen för denna del innehåller komponenter för att kommunicera med navet och i förlängningen mobilklienterna.
- *Nav*. Denna del behövs för att möjliggöra kommunikationen mellan en insideklient och mobilklient, utan att konfigurera brandväggen mellan insideklienten och Internet. Navet fungerar som en server och skall installeras på en dator ansluten till Internet, till denna dator måste även portar för ingående trafik öppnas. De båda klienterna ansluter till navet som sedan skickar vidare data dem emellan.

I Figur 3.1 visas en schematisk bild över det informationsflöde som sker mellan de olika delarna av grundlösningen. Grundlösningen innehåller en krypterad tvåvägskommunikation mellan mobilklienter och en insideklient. Data som överförs dem emellan komprimeras för att öka prestanda.

Kommunikationen mellan mobilklienten och navet sker över *TCP/IP* - protokollet, anledningen till att *TCP/IP* valts är att det är den minsta gemensamma nämnaren för nätverkskommunikation och implementeras i alla plattformar för mobila enheter. Kommunikationen mellan insideklienten och navet sker över *HTTP* –protokollet, detta har valts för att kunna utnyttja den port i brandväggen som är öppen för Internet. Genom att använda *BOSH* –teknik (se avsnitt 2.3.3) kan tvåvägskommunikation simuleras.



Figur 3.1. Figuren ger en schematisk bild över informationsflödet mellan en mobilenhet och ett intranät.

3.2.2 Mobilklient

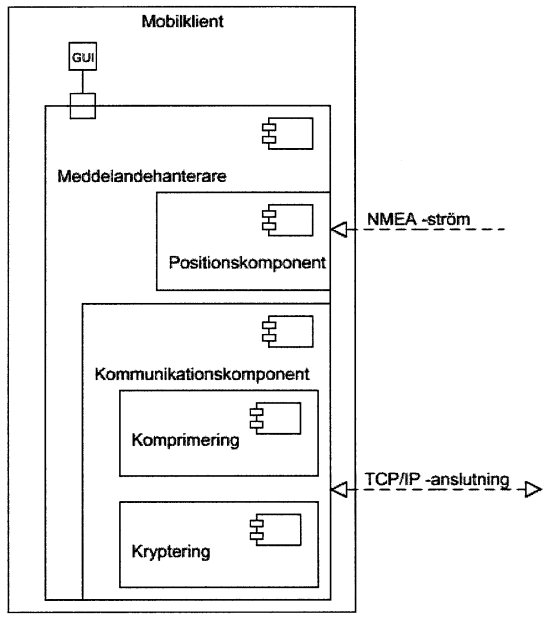
Grundlösningen för mobilklienten består av tre delar.

- *Kommunikationskomponent.* En komponent som ansvarar för kommunikationen till navet. Denna komponent skall kunna upprätta en anslutning till navet, skicka och ta emot data samt upptäcka förändringar i anslutningen. Både strängar och datafiler skall kunna överföras. Komponenten sköter även kryptering och komprimering av data som skall sändas och dekryptering och dekomprimering av data som tas emot.
- *Positionskomponent.* En komponent som ansvarar för att hålla reda på enhetens position. Komponenten skall kunna hitta en aktiv *GPS* – mottagare för att sedan lyssna på den *NMEA* –ström mottagaren

sänder ut. Den skall tolka informationen i strömmen och spara aktuell positioneringsinformation.

- *Meddelandehanterare*. En komponent som ansvarar för att ta hand om de händelser som framkallas då data tas emot eller en förändring i anslutningen sker. I grundlösningen är denna komponent ett skal vilket skall utvecklas vidare för de unika tillämpningarna. Då en tillämpning byggs skall även metoder för att bygga ihop utgående meddelanden implementeras i denna komponent.

För att komma från grundlösningen till en applikation för mobilenheten måste fler komponenter implementeras. Ett *GUI* för att styra applikationen och presentera data för en användare är en komponent som nästan alla tillämpningar kommer att behöva. Figur 3.2 visar en *UML* –representation av mobilklienten.



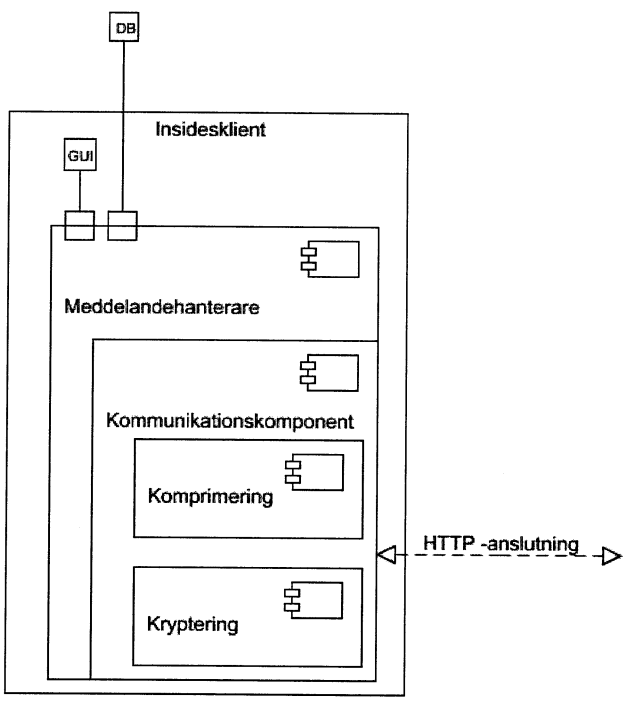
Figur 3.2. *UML* –representation av mobilklienten.

3.2.3 Insideklient

Grundlösningen för insideklienten består av två komponenter.

- *Kommunikationskomponent.* En komponent som ansvarar för kommunikationen till navet. Denna komponent skall kunna upprätta en anslutning till navet, skicka samt ta emot data samt upptäcka förändringar i anslutningen. Både strängar och datafiler skall kunna överföras. Komponenten sköter även kryptering och komprimering av data som skall sändas och dekryptering och dekomprimering av data som tas emot. Komponenten skall även hålla reda på vilka mobilklienter som är anslutna till navet och kunna skicka ut data till en unik mobilklient.
- *Meddelandehanterare.* En komponent som ansvarar för att ta hand om de händelser som framkallas då data tas emot eller en förändring i anslutningen sker. I grundlösningen är denna komponent ett skal vilket skall utvecklas vidare för de unika tillämpningarna. Då en tillämpning byggs skall även metoder för att bygga ihop utgående meddelanden implementeras i denna komponent.

Beroende på tillämpningen kan sedan olika komponenter utvecklas så som databaskoppling för att spara insamlad information eller ett *GUI* där de mobila enheterna kan övervakas och styras. Figur 3.3 visar insideklienten som ett *UML* –diagram.

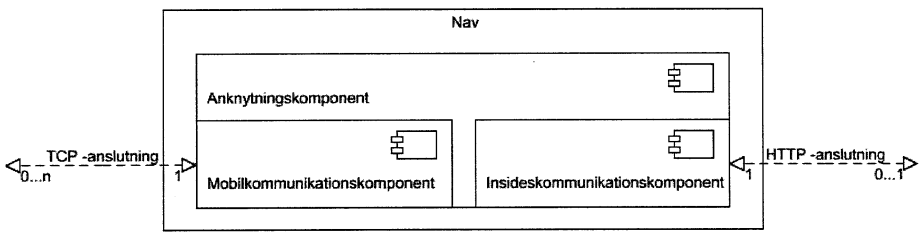


Figur 3.3. Insideklienten representerad som *UML* –diagram.

3.2.4 Nav

Den del av grundlösningen som kallas nav skall inte behöva förändras då olika tillämpningar utvecklas utan skall fungera som en färdig applikation. Navet består av tre delar (se figur 3.4).

- *Mobilkommunikationskomponent.* En komponent som ansvarar för kommunikation med mobilklienter. Den skall kunna ta emot nya anslutningar och hålla reda på de aktiva anslutningarna. Den skall meddela anknytningskomponenten då en ny anslutning upprättas eller stängs och vidarebefordra inkommande data till denna. Komponenten skall även kunna skicka data till en unik mobilklient. Den skall även på begäran kunna skicka ut information om vilka mobilklienter som är anslutna.
- *Insidekommunikationskomponent.* En komponent som ansvarar för kommunikation med en insideklient. Skall kunna upprätta en anslutning till en insideklient som anropar navet från ett specifikt *IP* –nummer. Den skall kunna skicka och ta emot data. Den skall vidarebefordra inkommande data till anknytningskomponenten.
- *Anknytningskomponent.* En komponent som ansvarar för att knyta ihop de komponenter som kommunicerar med insideklienten och de anslutna mobilklienterna. Den skall vidarebefordra data i båda riktningarna. Om anslutningen till insideklienten försvinner skall data från de mobila enheterna sparas och skickas in då insideklienten återansluter.



Figur 3.4. Navet representerat som *UML* –diagram.

3.2.5 Kommunikationskomponent

Kommunikationskomponenterna hos mobilklienten och insideklienten behöver även kunna kryptera och komprimera data. Därför skall de utöver själva kommunikationskomponenten innehålla ytterligare två komponenter:

- *Krypteringskomponent.* En komponent som ansvarar för kryptering och dekryptering av data. Komponenten skall utföra en symetrisk kryptering med en nyckel som går att byta. Det skall inte finnas någon känd kryptoanalyslösning för den krypteringsalgoritm som används.

- *Komprimeringskomponent.* En komponent som ansvarar för att komprimering och dekomprimering av data. Komponenten skall strukturera data så den tar så lite plats som möjligt.

3.2.6 Protokoll

I grundlösningen måste navet och insideklienten kunna meddela varandra eftersom insideklienten skall kunna skicka ut meddelanden till unika mobilklienter samt hålla reda på vilka klienter som är anslutna till navet. Navet kan inte dekryptera de meddelanden som skickas mellan de båda klienterna, därför måste ett enkelt protokoll för kommunikation mellan navet och insideklienten upprättas.

Då navet inte kan läsa de krypterade meddelandena skapar det ett unikt id för varje klient som är ansluten. Detta id- nummer används sedan för att navet och insideklienten skall kunna kommunicera om samma unika mobilklient. Figur 3.5 visar hur ett meddelande är uppbyggt.

```
TYP#CLIENT_ID<SYSINFO>ENCRYPTED_TEXT
```

Figur 3.5 Uppbyggnad av grundlösningens meddelande

Meddelandet består av två delar. En krypterad del som bär information mellan en mobilklient och insideklienten och en okrypterad del som bär information mellan navet och insideklienten. Delarna separeras av en tagg, <SYSINFO>, som används då det är högst otroligt att den krypterade strängen innehåller samma kombination. Den okrypterade delen består av ett fält för typ av meddelande och ett för vilken unik mobilklient meddelandet gäller, dessa är separerade med en brädgård.

Det finns tre olika typer av meddelanden i detta protokoll.

- *ClientConnected.* Typ som används då en ny mobilklient ansluter till navet. Det används även då det finns mobilklienter anslutna till navet då insideklienten ansluter.
- *ClientDisconnected.* Typ som används då navet tappar anslutningen till en mobilklient.
- *SendMessage.* Typ som används då insideklienten skickar ett meddelande till en unik mobilklient.

3.3 Objekt, klasser och gränssnitt

I systemdesignen för grundlösningen beskrivs vilka komponenter den innehåller och vilken funktionalitet dessa skall ha. I Appendix B definieras gränssnitt till komponenterna i grundlösningen. Gränssnitten beskriver vilka metoder komponenterna skall innehålla samt vilket syntax som skall användas.

Appendix B kan även läsas som en fördjupning om hur de olika komponenterna samverkar och kan implementeras som klasser.

4 Tillämpningar

Detta kapitel diskuterar hur grundlösningen kan användas för att utveckla olika typer av tillämpningar.

För att göra läsaren uppmärksam på vad grundlösningen kan användas till skall här en rad olika tillämpningsområden beröras. Syftet med kapitlet är att förmedla idéer som har alstrats under arbetet med rapporten. Det skall även ge en bild av hur rapportförfattarna har tänk att funktionaliteten i grundlösningen kan byggas ut till mer avancerade system.

- Ett system där insideklientens meddelandehanterare kan serva flera olika typer av mobilklienter. Om en organisation väljer att använda en lösning liknande den presenterade grundlösningen för att integrera mobila enheter i dess verksamhetssystem kan den användas för att utveckla en rad olika tillämpningar. Dessa kan bestå av mobilklienter utvecklade för olika hårdvara och därmed anpassas efter prestanda och de funktioner som finns tillgängliga. Meddelandehanteraren i grundlösningens insideklient kan i sådana fall utvecklas för att hantera meddelanden från alla mobilklienter. Detta bygger på att det protokoll som skall utvecklas parallellt med tillämpningen stödjer alla de funktioner som behövs för samtliga tillämpningar.

- Ett system för att redigera geografiska objekt i fält direkt mot en spatial databas. Då flera mobila enheter arbetar i samma område kan det uppstå problem om de redigerar samma objekt. Det är därför viktigt att systemet kan hålla reda på de objekt som redigeras. *WFS* – standarden innehåller funktion för att låsa objekt och sedan redigera dem. Genom att använda grundlösningen som en tunnel kan ett *WFS* –anrop skickas från mobilklienten till insideklienten som i sin tur anropar en *WFS* –server. Då insideklienten får svar skickas det ut till mobilklienten. Fördelen med detta är att *WFS* –servern kan finnas på det intranät som insideklienten finns.

- Ett system där mobilklientens *GUI* styrs av dess position. Att bygga ett lättanvänt *GUI* för mobilklienter kan vara svårt, då dess display och inmatningsmöjligheter är begränsade. Om mobilklienten kontinuerligt sänder in sin position till insideklienten kan dock gränssnittet förändras beroende på positionen. T.ex. skulle mobilklienten kunna visa en vägbeskrivning i ett läge och när mobilklientens position är tillräckligt nära målet så hämtas och presenteras information om det aktuella uppdraget eller den aktuella platsen. Användaren behöver på så sätt inte manuellt leta rätt på menyer och komma ihåg kommandon vilket minskar tiden för varje enskild användning av programmet.

- Ett system där insideklienten aktivt styr mobilklienternas arbetsflöde. Då en organisation använder flera mobila enheter för att utföra uppdrag inom ett område kan dessa samköras för att effektivisera deras transporttid. Om ett eller flera uppdrag drabbas av förseningar kan t.ex. en ny rutt- och tidsoptimering göras för att så många uppdrag som möjligt ska bli klara. Detta skulle även skapa en

bra grund för utvärdering och kontroll av de olika uppdragen. Om ett *GUI* byggs till insideklienten kan dessutom organisationens arbetsledning styra verksamheten och få ögonblicksbilder på ett smidigt sätt.

- Ett system där insideklienten efter ett antal givna regler kan förbättra data från mobilklienten. Då data samlas in för lagring kan den ibland bearbetas för att höja kvaliteten av den. Om en enkel variant av *GPS* –mottagare används för att spara rutter från mobila enheter kan resultatet bli att linjen hoppar mellan olika sidor av vägen. Detta beror på en begränsning i mättekniken men med två enkla antaganden kan linjens utseende förbättras: Man vet att fordonet håller sig på vägen och kan därför knyta mätningen till vägnätet. Vidare kan man även se åt vilket håll det färdas och på så sätt vilken sida av vägen den kör på. Detta gör att man kan spara en rutt vilken bör vara mer riktig än den inmätta.

5 Prototyp

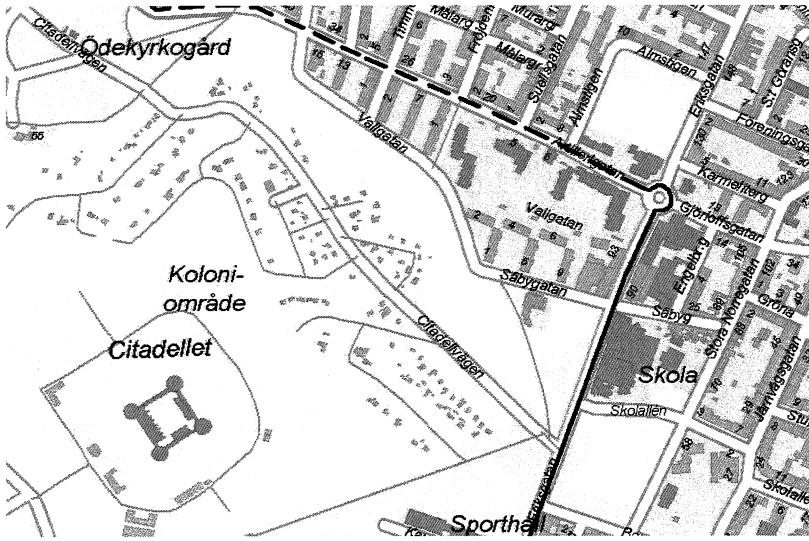
Det systemförslag som presenterades i kapitel 3 skall gälla för ett generiskt fall. Systemförslaget implementeras här för ett verkligt användarfall, vilket beskrivs i detta avsnitt. Avsnittet inleds med att beskriva ett antal tillämpningar som Landskrona kommun kan tänka sig att använda lösningen till. Därefter beskrivs hur prototypen har utvecklats och vilka justeringar av det generiska fallet som har gjorts.

5.1 Tillämpningar i Landskrona kommun

Vid de diskussioner som fördes med *Landskrona kommun* var en av de första frågorna som ställdes hur data skulle överföras till kommunens intranät genom deras brandvägg. Att testa en prototyp i *Landskrona kommuns* miljö är därför extra intressant. Vidare framkom ett antal tillämpningar där grundlösningen kan användas i den dagliga verksamheten.

Tracking av snöröjningsfordon

Presentation av rutten för kommunens snöröjningsfordon underlättar för både allmänheten och kommunens personal. Vid tillfällen med halt väglag eller stora snömängder ringer ibland upprörda medborgare och hävdar att någon snöröjning eller halkbekämpning inte har utförts. Detta är ibland korrekt och ett fordon kan vara på väg eller så har det snöat så pass mycket efter snöröjningen att det inte ser ut som att något har gjorts. Tanken är att utrusta varje förare med en *GPS* –mottagare samt en mobiltelefon och sedan kontinuerligt skicka in koordinaterna till *Landskrona kommuns* databas. Den körda rutten kan sedan presenteras på kommunens hemsida och jämföras med den planerade rutten (se Figur 5.1). På så sätt kan allmänheten få en uppfattning om var snöröjning sker och ungefär när den kommer att ske på deras gata. Vidare kan kommunens personal vid förfrågningar berätta exakt när snöröjning har skett om sådan alls har utförts. Ytterligare funktionalitet kan vara att föraren av ett snöröjningsfordon kan markera platser där snöröjning inte kan utföras, t.ex. då bilar är felaktigt parkerade eller där andra hinder finns. Markeringen skulle kunna ske genom en knapptryckning och behandlas sedan i kommunens intranät.



Figur 5.1. Figuren visar en vision av hur en resulterande bild av trackning av snöröjningsfordon kan se ut.

Avläsning av elskåp

Avläsning av parametrar i kommunens elskåp underlättas. I dagsläget letas korrekt data upp på kontoret för att sedan laddas in i en handdator och tas med ut i fält. Väl där såväl uppdateras som jämförs informationen i medhavd data med parametrarna i elskåpet. Detta kan vålla problem om t.ex. fel data har laddats in i handdatorn eller om några data har glömts. Den som utför arbetet måste då återvända till kontoret för att sedan hämta korrekt data. Lösningen kan vara att begära data när man befinner sig på plats. Detta skulle kunna göras antingen med det id –nummer som hör till elskåpet eller så kan man tänka sig att hämta data för det elskåp som ligger närmast den plats man befinner sig med hjälp av koordinater från en *GPS* –mottagare. När berörd data är ändrad kan de nya uppgifterna sändas tillbaka för att uppdatera databasen.

Inventering av lekparker

Inventering av lekparker förenklas med koder specifika för tillämpningen. Vid inventering av lekparker används ofta förkortningar för de olika föremål som finns. Med hjälp av koordinaterna för den som inventerar kan en specifik lekpark letas upp i databasen och alla objekt som finns hämtas till den mobila enheten. Det blir då enkelt att välja det objekt som inventeras och skriva in förändringar, t.ex. kan en gunga eller rutschkana vara trasig. Förändrad data överförs sedan till databasen som uppdateras tillsammans med datumet. När de trasiga objekten sedan är lagade anges detta på samma sätt och databasen uppdateras återigen. Metoden minskar risken för missförstånd genom att fel objektsförkortning anges och minskar även risken för att samma objekt förs in i databasen flera gånger.

5.2 Förutsättningar

Samtliga tillämpningar i avsnitt 5.1 är intressanta, men är något omfattande att genomföra. Därför är överenskommelsen med *Landskrona kommun* att målet för prototypen är en enklare form av *tracking* där syftet är att testa om grundlösningen fungerar i deras miljö. Tillämpningen består i att visa mobila enheters positioner i en kartmodul.

5.3 Val av komponenter

Prototypen utvecklas som en tillämpning av grundlösningen. Meddelandehanterarna utvecklas för att kunna skicka in positionsdata från mobilklienten till insideklienten samt att kunna skicka textmeddelanden mellan klienterna.

GUI –t för mobil- och insideklienten blir unika för tillämpningen. Mobilklientens *GUI* utvecklas för att innehålla enkla funktioner för att starta och stoppa *tracking*en, samt att sända och visa textmeddelanden. *GUI* –t består av *.NET Compact Framework* –s och *JME* –s standardklasser för *GUI*. Insideklienten byggs upp av *.NET Framework* –s klasser för *GUI* -s kopplade till en *Google Maps* –kartmodul som placeras i ett webbfönster. Webbfönstret integreras sedan med övriga delar av *GUI* -t.

5.4 Prototyputveckling

Utvecklingen av navet, insideklienten och mobilklienten för *.NET Compact Framework* sker i *Microsoft Visual Studio 2005*. Mobilklienten för *JME* utvecklas i *NetBeans 5.5*. De kontinuerliga testerna utförs för insideklienten och navet på standarddatorer med operativsystemet *Microsoft Windows XP* samt tillgång till Internet. För mobilklienten anpassad för *JME* används mobiltelefonerna *Nokia N95* och *SonyEricsson W800i*. Vad gäller mobilklienten för *.NET Compact Framework* används *PDA* -n *ETEN OM600+*. Samtliga enheter för mobilklienter har tillgång till Internetanslutning via *GPRS*.

5.4.1 Grundlösning

I detta avsnitt behandlas, mycket översiktligt, grundlösningens implementering. För mer detaljerad information hänvisas till Appendix A.

Implementeringen av grundlösningen använder programspråkens standardklasser. Detta gör att grundlösningen blir relativt enkel att skapa eftersom det mesta redan finns och det istället handlar om att bygga ihop delarna med varandra. Detta gäller framförallt de delar som kommunicerar med varandra över Internet, men även navets serverfunktioner.

Den kryptering som har valts är *Blowfish* som är en symmetrisk 128- bitars krypteringsalgoritm. Anledningen är att algoritmen finns fritt tillgänglig genom ett kryptografiskt bibliotek kallat *bouncycastle,* vilket är ett projekt med öppen källkod för *Java, JME* och *Compact Framework.*

Komprimeringen utelämnas i prototypen på grund av flera anledningar. En är att den lilla datamängd som skickas per meddelande inte förminskas av komprimering på grund av att en stor *header* läggs till. Dessutom ingår inte

komprimering i *JME* och inga fritt tillgängliga klassbibliotek har hittats som kan användas tillsammans med den komprimering som ingår i *.NET*.

5.4.2 Meddelandehanterare och meddelandeutformning

Meddelandehanteraren på mobilklienten ansvarar för att kunna tolka inkommande meddelanden och se till att rätt åtgärd utförs. Utöver det ska den hämta den senaste positioneringsinformationen, bygga ihop de meddelanden som ska sändas och se till att de sänds. När klassen skapas tar den därför *NMEAparser* och *MobileToServerCommunication* (se Appendix A) som inparametrar.

Med jämna användardefinierade tidsintervaller hämtar meddelandehanteraren den senaste positioneringsinformationen från *NMEAparser* och bygger därefter ihop det meddelande som skall sändas. Om användaren i *GUI* –t väljer att sända ett textmeddelande ansvarar meddelandehanteraren även för att bygga ihop detta meddelande och använder sedan *MobileToServerCommunication* för att sända meddelandet. Vid ett till mobilklienten inkommande meddelande, som i vårt fall endast kan bestå av ett textmeddelande, meddelas *GUI* –klassen genom en händelse och meddelandet visas för användaren.

För att insideklienten ska kunna klara av uppgiften att hålla reda på olika mobilklienter ska ett meddelande i sin enklaste form innehålla en position med latitud och longitud. Dessa separeras med blanksteg för att enkelt kunna ta fram respektive koordinat. Ett koordinatpar räcker dock inte vid en situation där två eller flera mobila enheter samtidigt ska kunna skicka in sina positioner. Problemet är att insideklienten omöjligt kan veta vilken mobil enhet som skickar det meddelande som kommer in. Lösningen är att inkludera ett användarnamn i det meddelande som skickas och separera de olika delarna av meddelandet med brädgård. Detta ger en första version av ett meddelande (se Figur 5.2).

```
Användarnamn#Lat Long
```

Figur 5.2. Första versionen av ett meddelandes uppbyggnad innehåller ett användarnamn och ett koordinatpar.

Meddelandehanteraren kan nu först dela den inkommande textsträngen med avseende på brädgård. Då fås två strängar där den första innehåller användarnamnet och den andra innehåller användarens position. Detta gör att positionen nu är knuten till rätt användare. Vidare delas strängen som innehåller positionen med avseende på blanksteg för att kunna konvertera latitud och longitud från text till decimaltal.

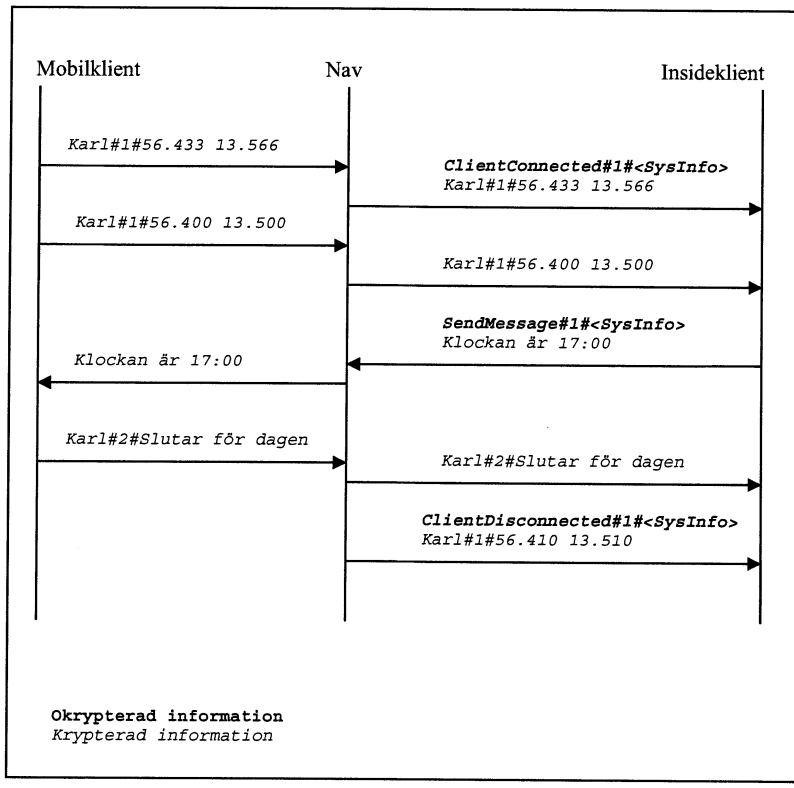
Ett meddelande ska dock kunna innehålla annan information än användarens position. Detta klarar inte den första versionen av meddelandet av och därför krävs vissa tillägg. Om meddelandehanteraren begränsas till att hantera dels position och dels textmeddelande innebär det att två meddelandetyper existerar. Detta resulterar i en andra version av ett meddelande (se Figur 5.3).

```
(1)          Användarnamn#1#Lat Long
(2)          Användarnamn#2#Text

Allmänt:     Användarnamn#Typ#Innehåll
```

Figur 5.3. Andra versionen av ett meddelande hanterar både position- och textmeddelande.

I Figur 5.4 visas flödet av meddelanden mellan mobilklient, nav och insideklient.



Figur 5.4. Exempel på flöde av meddelanden mellan mobilklient, nav och insideklient.

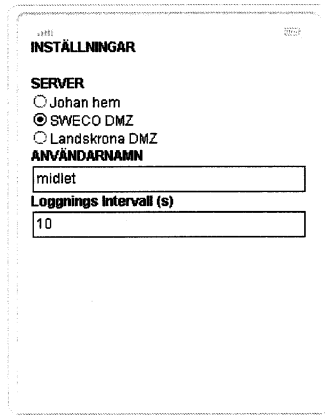
5.4.3 Graphical User Interface (GUI)

Mobilklient

Applikationen på den mobila enheten är utvecklad för att vara flexibel med hänsyn till testning i olika miljöer. Därför finns det i såväl *JME*- som *Windows Mobile* –varianten möjlighet att byta anslutningspunkt (nav). Vidare kan man ändra användarnamn och själv välja när applikationen ska börja eller sluta sända information till navet.

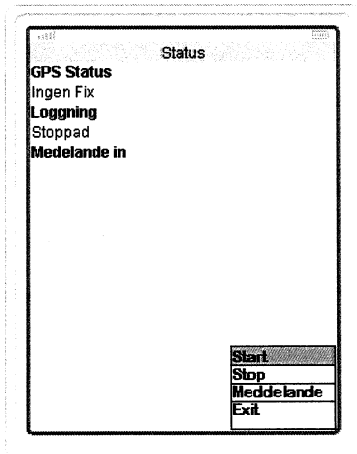
JME

GUI –t för en mobil enhet som stödjer *JME* är en serie skärmbilder med ett antal val och menyer som användaren kan göra. Menyerna navigeras genom enhetens styrspak eller styrknappar. Vid applikationens start anges vilket av tre fördefinierade nav som skall användas. Därefter fylls användarnamn i och även hur ofta enhetens position ska rapporteras till navet. Figur 5.5 visar bildskärmen i startläget.



Figur 5.5. I startläget för mobilklienten fylls ett antal inställningar i av användaren.

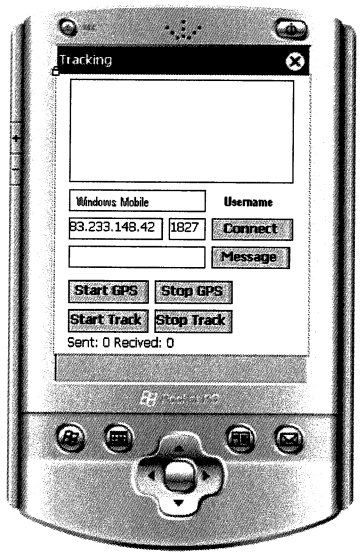
Nästa steg är att söka efter *bluetooth* -enheter och välja den *GPS* –mottagare som skall användas. Mobilklienten kopplar därefter upp sig mot navet och visar information om *GPS* –mottagarens status, om loggningen av positioner är startad och det senast inkomna meddelandet från navet. Via en meny sköts kontrollen av programmet där start och stoppning av loggning sker, meddelande kan skickas till navet och programmet kan avslutas (se Figur 5.6).



Figur 5.6. Mobilklienten i läget där start av loggning sker.

Windows Mobile

För enheten med *Windows Mobile* finns samtliga inställningar och knappar samlade på samma skärmbild. Detta tack vare att skärmen på en *PDA* oftast är större än på en *JME*- enhet. Knapparna och inställningarna aktiveras genom att peka på dessa på skärmen som är av *touchscreen* –typ. Figur 5.7 visar *GUI* -t för *Windows Mobile.*



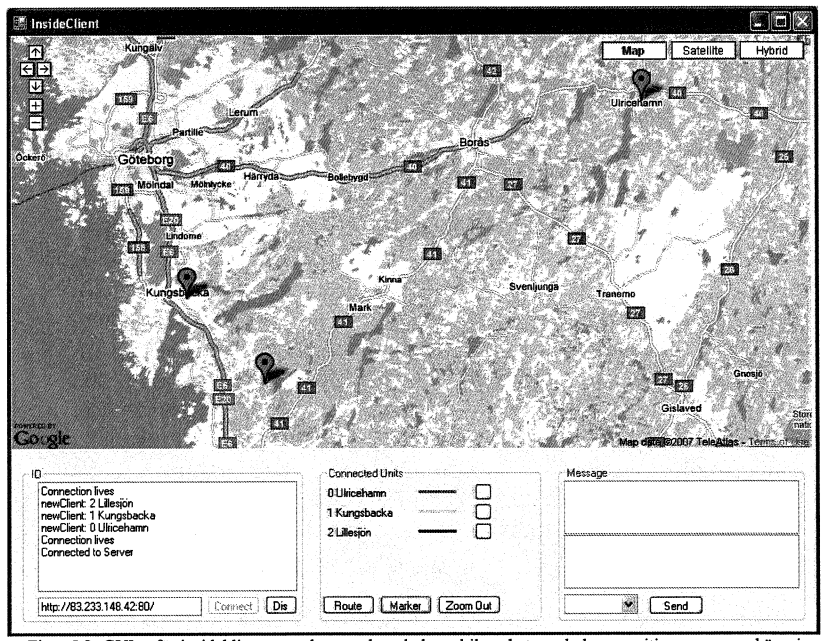
Figur 5.7. *Windows Mobile* –enhetens stora skärm gör det möjligt att samla alla funktioner på samma yta.

Enheten ansluter till navet genom att användaren klickar på *Connect* – knappen efter att ha fyllt i korrekt användarnamn, adress och portnummer. Därefter skall *GPS* –en startas. Denna ska vara knuten till *PDA* -ns virtuella kommunikationsport *COM0* för att applikationen skall få tillgång till positioneringsinformationen. Anknytningen sker inte i applikationen utan måste vara gjord i handdatorns operativsystem innan programmet startas. När användaren klickar på *Start Track* –knappen aktiveras loggningen och data börjar skickas in till navet. Det stora vita fältet överst i gränssnittet visar information om vilken position enheten har och även de eventuella meddelanden som skickas från servern. *Message* –knappen sänder ett textmeddelande till servern. Texten som sänds är det som användaren har angett i det vita fältet till vänster om knappen.

Insideklient

Insideklienten är utvecklad för att säkerställa att korrekt information har skickats in från den mobila klienten via navet. Uppgiften (se avsnitt 5.2) består av att få in informationen till *Landskrona kommuns* intranät. För att kunna verifiera att rätt information har kommit in har vi valt att presentera denna i ett fönster med kartkoppling till *Google Maps*. I en skarp version av programvaran är det mer sannolikt att data ska lagras i en databas och sedan presenteras på annat sätt. Anledningen till lösningen är att den är mindre omständlig då någon databaskoppling ej behöver upprättas.

Applikationen initieras genom att användaren anger adressen till det nav som ska anslutas till. Därefter presenteras varje uppkopplad enhet som en markör eller den loggade sträckan som en linje med unik färg. En informationsruta visar användarnamnet tillsammans med den tilldelade färgen. Användaren har här även möjlighet att zooma till var och en av de uppkopplade användarna för att få en mer detaljerad bild av området. Vidare kan ett meddelande skickas till en användare genom att skriva in text i en textruta och därefter välja vilken användare texten ska skickas till. Inkommande meddelanden från enheterna samt information om förändringar i anslutningar presenteras i specifika textrutor. Figur 5.8 visar *GUI* -t med tre uppkopplade enheter vars positioner visas med markörer.



Figur 5.8. *GUI* –t för insideklienten med tre uppkopplade mobila enheter och deras positioner som markörer i *Google Maps*.

6 Testning av prototyp

I detta kapitel beskrivs de tester som har gjorts av prototypen samt den testmiljö som har använts.

Under utvecklingen av grundlösningen har kontinuerliga tester gjorts på de olika delarna av lösningen. För att testa klasser och flöden i grundlösningen har små och enkla *GUI* -s utvecklats vilka visar vad som händer vid exekvering. Dessa *GUI* –s är inte dokumenterade då deras syfte endast är att testa en del av lösningen och när denna del fungerar som den skall fortsätter utvecklingen med nästa del (se Appendix C för sammanställning av löpande tester).

Prototypen installeras och testas i *Landskrona kommuns* miljö för att därefter genomföra en *tracking*. Rutten som *trackas* består av en promenad i Landskronas hamnområde. Mobilklienten installeras på kommunens standardtelefon för utomhusarbete, *Nokia 5500 Sport,* som hämtar positioneringsinformation från den *bluetooth* –anslutna *GPS* –mottagaren *Holux GPSlim 236*. Både telefonen och abonnemanget stöder uppkoppling mot Internet via *GPRS*. Vidare installeras insideklienten på en standarddator med operativsystemet *Microsoft Windows XP* som befinner sig på *Landskrona kommuns* intranät, det vill säga innanför deras brandvägg. Navet befinner sig utanför intranätet och installeras och körs därför för enkelhetens skull på en av examensarbetarnas privata dator, även denna en standarddator med operativsystemet *Microsoft Windows XP*. En anledning att använda denna konfiguration är att navet enkelt kan fjärrstyras och eventuella problem kan åtgärdas.

Såväl installering som *tracking* (se resultat i Figur 6.1) fungerar tillfredsställande. Det går även bra att skicka textmeddelanden till och från mobilklienten och insideklienten. Det enda som har vållat problem är att kryptering av data på *MIDlet* –versionen av mobilklienten inte går att använda samt att tecknen å, ä och ö inte visas korrekt.

Figur 6.1. Skärmbild från insideklienten som visar resultatet av den *tracking* som utfördes hos *Landskrona kommun*.

7 Diskussion

I kapitlet görs en utvärdering av prototypen där för- och nackdelar diskuteras. Även förestående tekniska förändringar tas upp och hur dessa kan komma att påverka grundlösningen.

Syftet med projektet var att skapa en grundlösning för att integrera mobila enheter i ett verksamhetssystem. Med utgångspunkt från den grundlösning som presenterades i kapitel 3 diskuteras här de frågeställningar som togs upp i projektets problemformulering (avsnitt 1.2). Därefter behandlas aspekter som framkommit under arbetets gång.

Hårdvara och utveckling

Resultaten från tester indikerar att vår systemdesign fungerar väl för olika mobila enheter. Prototypen är testad för såväl en enklare mobiltelefon som en handdator vilka använder olika operativsystem. Detta tillsammans med att lösningen bygger på att minsta gemensamma nämnare används för dataöverföringen, gör att det är rimligt att anta att även andra mobila enheter kan användas. Valet av hårdvara blir därför beroende av vilken funktionalitet som krävs och vilken utvecklingsmiljö som föredras. Med en mer avancerad enhet kan mobilklienten innehålla mer funktionalitet men blir samtidigt dyrare. En fördel med t.ex. en *laptop* är dock att de utvecklingsmässiga begränsningarna som finns i mindre enheter inte existerar. Detta samtidigt som en *laptop* får begränsat användningsområde på grund av sin storlek och låga batterikapacitet. En mer intressant jämförelse är den mellan mindre mobila enheter och vilken typ av applikationer som kan utvecklas till respektive enhet:

Att utveckla *MIDlets* för *JME* upplevs av rapportförfattarna som mer begränsande jämfört med *C#* -applikationer i *.NET CompactFramework*. Detta beroende på att dokumentationen känns mer bristfällig samtidigt som de standardklasser som ingår är färre. Därför känns *.NET CompactFramework* som det mest naturliga valet att utveckla i om valmöjligheten finns, inte minst för att det är mycket enklare att hitta färdigimplementerade och fungerande krypteringsbibliotek. Det innebär dock inte att *JME* och *MIDlets* helt bör förkastas. De har istället en klar fördel genom att kunna installeras och användas på en mängd olika billiga mobila enheter. Prototypen utvecklades t.ex. för ändamålet att *tracka* en rutt vilket en mobiltelefon passar utmärkt för. Dels är den billig, men är också något som många redan bär med sig. *GUI* -t för en sådan tillämpning kan också göras väldigt litet och lämpar sig väl för en mobiltelefons lite mindre skärmar.

Valet mellan en inbyggd eller separat *GPS* –mottagare beror på varje enskilt fall och vilka behov som finns. Enheterna ansluter i nuläget till en *GPS* – mottagare via *bluetooth*, något som har både för- och nackdelar. En stor fördel är att *GPS* –mottagaren kan placeras i en annan position än den mobila enheten för att få bättre sikt till satelliterna. Vidare har en separat *GPS* –mottagare ett eget batteri vilket ger en längre batteritid i jämförelse med en kombinerad *GPS* –mottagare och mobil enhet. Nackdelen är att flera enheter tar större plats. Dessutom finns det viss risk att *bluetooth* – anslutningen kan brytas vilken skapar luckor i överföringen. Rent

utvecklingsmässigt är det dock ingen större skillnad om en inbyggd eller extern *GPS* används.

Ekonomiska aspekter

Lösningen som har presenterats är kostnadseffektiv för både kund och leverantör. För kunden yttrar sig detta genom att startsträckan för att komma igång med tester är mycket kort. De saker som behövs är installation av programvara på ett antal mobila enheter samt installation av insideklienten, något som inte tog oss mer än en halv dag att göra i *Landskrona kommuns* testmiljö. Eftersom den mobila enheten kan vara en enkel telefon är det dessutom stor sannolikhet att kunden redan äger en eller flera enheter som kan användas. För själva dataöverföringen beror kostnaden på det enskilda abonnemanget. Flera mobiltelefonbolag erbjuder nu fri mängd dataöverföring till en fast månadskostnad. Ett sådant abonnemang kan vara av intresse för de kunder som använder lösningen frekvent. Leverantören, i vårt fall *SWECO*, kan också dra nytta av att lösningen är så enkel att komma igång med. Det blir lättare att visa upp funktionalitet i kundens miljö och därmed även öppna för en diskussion kring vad som kan åstadkommas. Vidare kan stora delar av utvecklingen för de mobila klienterna återanvändas från tillämpning till tillämpning. Det medför låga utvecklingskostnader för *SWECO* samtidigt som det öppnar för möjligheten att slå ut kostnaden på flera kunder för att kunna erbjuda ett lågt pris. Det finns även möjlighet att vidareutveckla navet för att klara av ett många till många förhållande mellan mobilklienter och insidesklienter. Navet kan då installeras på *SWECO* och det blir då möjligt för flera kunder att använda detta.

Kryptering av känslig data

De data som skickas skyddas från obehöriga genom kryptering. Detta är en potentiellt mycket viktig egenskap eftersom data kan vara av känslig natur för en kund. Uppgifter som kan vara känsliga är t.ex. en persons position eller resultatet av en undersökning som ska sändas till en databas. Att de data som skickas är krypterade hela vägen från avsändare till mottagare är en extra trygghet om navet skulle bli *hackat*. För även om avlyssning skulle ske på navet behöver krypteringen knäckas om någon läslig information ska kunna tas fram. De delar av protokollet som inte är krypterade innehåller ingen lägesbunden information utan endast uppgifter om till vilken klient data ska skickas. Detta gör att ingen värdefull information kan tolkas. En ytterligare säkerhetsåtgärd kan dock vara att använda *SSL* i *TCP/IP* – anslutningen och *HTTPS* (*HyperText Transfer Protocol Secure*) istället för *HTTP* i anslutningen mellan insideklienten och navet. Det skulle innebära att även protokollets okrypterade information skulle skyddas. Trots det behövs ändå den ursprungliga krypteringen för att omöjliggöra avlyssning i navet.

Kommunikationsavbrott

Vid bristfällig täckning för den mobila enheten kan data sparas lokalt tills en anslutning återupprättats. Lösningen uppmärksammar att en förändring i en anslutning har skett och meddelar detta. Vad som händer därefter beror på

vilken tillämpning lösningen används i. Om det enbart är intressant att veta positionen i realtid behöver ingenting göras. Är det däremot viktigt att kunna spåra hela rutten för en enhet kan data *buffras* lokalt på enheten tills anslutningen återfås. Därefter kan samtliga data sändas in. Det viktiga är dock att lösningen meddelar anslutningens status när den förändras så att klienterna kan vidta nödvändiga åtgärder.

Kommunikation med intranät

Mobilenheterna kommunicerar med ett nav utanför organisationens intranät för att undkomma brandväggsproblem. Lösningens kärna ligger i att den är enkel och snabb att komma igång med. Tack vare att använda ett nav undkommer man de vanligaste brandväggsproblemen hos organisationen. I en skarp version kan navet placeras antingen hos *SWECO* eller på organisationens *DMZ*. Den för kunden snabbaste lösningen är att *SWECO* har ett nav uppe konstant som flera olika organisationer kan nyttja. Det krävs då mycket få installationer för att använda lösningen. Om organisationen därefter t.ex. väljer att utöka antalet mobilklienter kan det vara intressant att installera ett nav på sin egen *DMZ*.

Säkerhetsaspekter

Bortsett från att omöjliggöra för utomstående att ta del av sänd data finns andra säkerhetsaspekter att ta hänsyn till. En sådan är att om flera illasinnade människor ansluter till navet finns möjlighet att överbelasta systemet genom att sända data ett stort antal upprepade gånger. Stora resurser kommer då att läggas på dessa anslutningar och det finns risk för förseningar i de korrekta anslutningarna. I värsta fall kan servern belastas så hårt att den kraschar. Hotbilden kan avhjälpas genom att varje användare måste godkännas innan en anslutning sker. Detta kan t.ex. göras genom att använda en *RADIUS* –server där denna godkänner eller förkastar klienternas certifikat. Detta har inte undersökts ingående för den utvecklade prototypen men eftersom *RADIUS* –servern använder vanliga *HTTPS* –certifikat bör inga hinder finnas för att använda metoden med vare sig *.NET CompactFramework* eller *JME*.

Systemoptimering

Den enskilt största flaskhalsen i systemet är den mobila uppkopplingen. Därför är det här arbete bör läggas för att överföra så lite data som möjligt. Det innebär, som tidigare nämnts, att meddelandeuppbyggnaden utvecklas för varje tillämpning. Att använda ett stort färdigskrivet protokoll för att täcka in så många användningsfall som möjligt är inte optimalt. Om endast en *tracking* ska utföras är det inte lämpligt att använda t.ex. *GML* som överföringsformat. *GML* skulle förvisso beskriva punkterna på ett korrekt sätt men skulle använda en allt för stor datamängd för att göra det. En stor datamängd innebär längre tid för överföring och högre kostnader vilket inte är önskvärt. En tillämpning där *GML* bör användas är istället om större mängder kartmaterial ska överföras till eller från den mobila enheten. Detta eftersom standarden är väl utvecklad och definierad vilket sparar en stor mängd utvecklingsarbete. Besparingen uppväger därmed den extra kostnad och tid som läggs på överföringen.

Framtida tekniska förändringar

Eftersom teknikutvecklingen framskrider mycket fort, inte minst på mobil – sidan är det viktigt att resonera kring hur eventuella tekniska förändringar kan påverka grundlösningen. Det som vi med säkerhet vet är att samtliga mobila enheter kommer att bli mer avancerade och klara av fler uppgifter. I skrivande stund anser vi att *.NET CompactFramework* ligger före *JME* som utvecklingsspråk. *JME* utvecklas dock stadigt liksom de klassbibliotek som finns att tillgå. Därför är det sannolikt att krypteringen fungerar inom en snar framtid även för *JME*.

Även snabbare och billigare mobila uppkopplingar är att förvänta. Detta gör att större datamängder kan överföras snabbare och därmed öka pålitligheten och prestanda i systemet.

8 Slutsats

I detta kapitel dras slutsatser från projektet.

Rapporten har visat hur en enkel grundlösning kan skapas för att integrera mobila enheter i ett verksamhetssystem. Genom att använda en generisk lösning som grund för kommunikationen är startsträckan för projekt där en mobil anslutning krävs kort. Den generiska lösningen bygger på öppna gränssnitt och inga licenser krävs. Detta skapar bra förutsättningar för smarta lösningar där en mobil användare har en direkt anslutning till verksamheten.

För att utvärdera grundlösningen har en prototyp utvecklats. Denna testades i *Landskrona kommuns* miljö och fungerade tillfredställande. Ett potentiellt säkerhetsproblem är att en port i brandväggen måste hållas öppen mot Internet. I rapporten har detta diskuterats och i en skarp version bör en *RADIUS* -server användas för att på så sätt godkänna enskilda klienters *IP* – nummer genom att använda certifikat.

En annan slutsats från prototyputvecklingen är att ur utvecklingssynpunkt kan man välja vilken mobil hårdvara som helst så länge den har en trådlös anslutning till Internet. Hårdvara kan därför väljas efter tillämpning. Men utvecklingsarbetet för de olika plattformar som används till mobila enheter skiljer sig åt. Det är betydligt svårare att komma igång med utveckling av *MIDlets* än med utveckling för *Windows Mobile*. Då val av hårdvara sker blir det en avvägning mellan inköpskostnad, utvecklingskostnad och funktionalitet.

Att bygga upp grundlösningen genom att definiera gränssnitt mellan dess olika delar gör det möjligt att enkelt byta ut en del. Att skapa en flexibel lösning innebär att den skall passa så många syften som möjligt. Det är svårt att förutse vad som kommer att hända inom teknikutvecklingen då allt går väldigt fort. Det är därför viktigt att kunna ändra en del av lösningen utan att behöva ändra hela.

Slutligen konstateras att det är möjligt att utforma en grundlösning vilken kan fungera som bas för vidare utveckling av systemintegrerande applikationer. Den initiala utvecklingskostnaden kan på detta sätt fördelas på olika applikationer och projekt. Detta tillsammans med att en stor del av applikationerna redan är utvecklad genom grundlösningen gör att startsträckan för att testa ett nytt system förkortas och en kostnadseffektiv lösning åstadkoms.

Appendix A

Nummer	1
Krav	Grundlösningen skall åtminstone kunna användas med *Windows Mobile* och operativsystem som stödjer *JME*.
Beskrivning	Det finns en mängd olika operativsystem för mobila enheter och det är viktigt att systemet kan användas med de vanligaste mobila enheterna.

Nummer	2
Krav	Den mobila enheten skall kunna hämta positioneringsinformation från en inbyggd eller *bluetooth* ansluten *GPS* –mottagare.
Beskrivning	Alla de tillämpningar som är intressanta att utveckla behöver kunna hålla reda på den mobila enhetens position. Genom att tillåta både inbyggda och *Bluetooth*-anslutna *GPS* –mottagare införs inga begränsningar i grundlösningen och den blir på så sätt mer flexibel.

Nummer	3
Krav	Grundlösningen skall garantera att data som skickas inte är tillgänglig för obehöriga.
Beskrivning	Lägesbunden information kan i många fall vara känslig, speciellt om det anger en persons läge. På grund av det är det viktigt att kunna garantera att informationen endast är tillgänglig för den tänkta mottagaren.

Nummer	4
Krav	Grundlösningen skall ge verifikation på att data som skickats når den avsedda mobila enheten samt möjliggöra bekräftelse på att användaren av enheten har tagit del av skickad data.
Beskrivning	Olika tillämpningar ställer olika krav på tillförlitlighet. För att systemet ska kunna anpassas till varje tillämpning med geografisk koppling bör det högsta kravet på tillförlitlighet gälla. Därför är det viktigt att ge information om huruvida data har nått sitt mål eller ej. Ofta är det även viktigt att få en bekräftelse på att användaren av enheten har tagit del av skickad data. T.ex. läst en arbetsorder.

Nummer	5
Krav	Grundlösningen skall göra det möjligt att förse de mobila enheterna med data utan att användaren av enheten aktivt begär det.
Beskrivning	Det kan i olika tillämpningar vara viktigt att ge användaren av den mobila enheten information utan att denne begär det. Det gör att det är viktigt att enheten konstant är beredd på att få data så länge denna är uppkopplad.

Nummer	6
Krav	En verksamhet skall kunna kommunicera med de mobila enheterna med minimal konfigurering av en eventuell brandvägg.
Beskrivning	De flesta organisationer har ett intranät med en brandvägg som skiljer detta från Internet. Det innebär att endast de portar som systemadministratören tillåter får användas för kommunikation. Enligt *SWECO* -s erfarenhet kan detta skapa problem när nya applikationer, som är tänkta att använda portar vilka är stängda, ska installeras. För att undvika den tidskrävande processen att få en applikation godkänd och se till att brandväggen tillåter trafiken är det viktigt att alla fasta klienter med tillgång till standardportar kan kommunicera med de mobila enheterna.

Nummer	7
Krav	Grundlösningen skall möjliggöra överföring av såväl strängar som datafiler.
Beskrivning	Den kommunikation som sker mellan de mobila enheterna och verksamhetssystemet kan bestå av enkla medelanden om vart en enhet befinner sig. För detta krävs att strängar kan överföras. Det kan finnas tillämpningar då man vilka skicka mer komplex information, t.ex. en bild.

Appendix B

MobileToServerCommunication

Denna klass är till för att sköta kommunikationen på en mobil enhet. Den kan skapa en anslutning till ett nav, skicka och ta emot, kryptera och avkryptera, packa och packa upp meddelande. Klassen kan implementeras både i *C#.NET Compact Framwork* och *JME*.

```
MobileToServerCommunication(string hostIP, string port)
```

Klassens konstruktor, här initieras de objekt som används av klassen. Inparametrarna bestämmer till vilket *IP* -nummer och till vilken port som skall anslutas till.

```
public void connect(string username, string password)
```

Metoden försöker att ansluta till den maskin som angetts då objektet skapats. Detta görs genom att använda en standardklass för *TCP* -klienter som finns både i *C#.NET Compact Framework* och *JME*. Om anslutningen lyckas kastas en *connectionChanged* händelse som säger att en anslutning skapats. Därefter sparas en ström för ingående data och en ström för utgående data som privata attribut i klassen. Efter detta startas en tråd för att lyssna efter inkommande data. Tråden skapas för att kunna köra andra processer samtidigt, t.ex. kan man både skicka och ta emot data simultant. Då inkommande data anländer packas den upp och dekrypteras, skickas vidare som en *dataRecived* -händelse. Om anslutningen skulle misslyckas kastas en *communicationChanged* -händelse som anger att anslutningen misslyckades.

```
void disconnect()
```

Metoden kan användas då man arbetat klart och vill stänga ner anslutningen. Först skickas ett meddelande iväg för att ange till navet att anslutningen kommer att stängas ner. Därefter stängs strömmarna för inkommande och utgående data, den tråd som lyssnar efter inkommande data avslutas och slutligen avslutas det objekt som har hand om *TCP* -uppkopplingen.

```
bool isConnected()
```

En metod för att kontrollera om en anslutning fortfarande existerar. Kontrollen sker genom att försöka skicka ett meddelande. Om försöket lyckas returneras *true*. Om försöket misslyckas returneras *false*, samt att metoden *disconnect* körs för att stänga ner eventuella objekt som körs trots att anslutningen inte existerar.

```
void sendData(byte[] data)
```

Metoden används för att skicka iväg meddelanden. Den får in ett meddelande, krypterar och packar det för att sedan skicka iväg det med -1 som sista byte för att visa att meddelandet är slut.

```
event dataRecivedEventHandler dataRecived
```

Denna händelse används för att skicka vidare inkommande data till den klass som skapat detta kommunikationsobjekt och därmed fungerar som meddelandehanterare. Metoden i meddelandehanteraren som tar hand om

händelsen måste kunna tyda meddelandet och utföra de instruktioner som finns. T.ex. skicka vidare meddelandet till insideklienten.

```
event communicationChangedEventHandler communicationChanged
```

Denna händelse används för att göra den ägande klassen uppmärksam på att en förändring i anslutningen har skett, t.ex. att anslutningen till en klient har förlorats.

ServerToMobilesCommunication

Klassen lyssnar efter nya mobila klienter och kontrollerar de anslutningar som är aktiva. Det gör att de meddelanden som skickas kan komma till rätt klient. Informationen som skickas mellan de mobila klienterna och insideklienten är krypterad med en nyckel som navet inte känner till. Det gör att informationen är säker under hela kedjan. Den enda information som navet behöver är vilket klient –id informationen kommer från. Således är meddelandena som skickas uppdelade i två delar. En del som innehåller systeminformation och en del som är själva meddelandet. Navet implementeras endast i *.NET Framework 2.0.*

```
ServerToMobilesCommunication(string port)
```

Klassens konstruktor, här skapas en länkad lista för att hålla reda på de klienter som ansluter. Datorns lokala *IP* –nummer hittas och ett *TCP-listener* –objekt skapas.

```
void start()
```

I denna metod startas en ny tråd som lyssnar efter nya klienter. Då en ny klient ansluter skapas ett nytt *MobileClient* -objekt med ett unikt id. Objektet läggs i en länkad lista för att man skall kunna skicka ut meddelanden till de olika anslutna klienterna samt för att göra det möjligt att stänga ner respektive anslutning. Därefter skickas ett systemmedelande till insideklienten med det ID –nummer navet gav klienten samt den krypterade datan. Insideklienten kan sedan avkryptera meddelandet och knyta användarnamn till navets id. Tråden upprepas till dess att man väljer att stoppa den.

```
void stop()
```

Metoden avslutar först den tråd som lyssnar efter nya klienter. Därefter gås listan med anslutna klienter igenom och varje anslutning stängs. Metoden används då man vill stänga av navet på ett kontrollerat sätt och därmed frigöra alla resurser som används.

```
void sendActiveClients()
```

Metoden kastar en händelse för varje aktiv klient. Används då den inre klienten ansluter för att den skall få reda på vilka mobila klienter som är uppkopplade mot navet och deras unika ID -nummer.

```
void sendData(byte[] message)
```

Metoden sänder ett meddelande från insideklienten till en mobil klient. Inparametern är meddelandet vilket innehåller både systemmeddelandet, med vilket klient –ID meddelandet ska skickas till, samt den krypterade informationen. Rätt klient letas upp och det krypterade meddelandet sänds.

```
event datareceivedEventHandler datareceived
```

```
event communicationChangedEventHandler communicationChanged
```

Dessa händelsehanterare används för de fall där data tas emot eller när kommunikationsförhållanden förändras. T.ex. då en anslutning förloras.

MobileClient

Klassen innehåller en anslutning till en mobil klient och skapas när en ny klient ansluter till navet.

```
MobileClient(TcpClient connection, int id)
```

Klassens konstruktor, som inparameter får den ett *TcpClient* -objekt som skapas vid en ny anslutning och ett unikt id nummer som används för att identifiera en anslutning. Strömmar för ingående och utgående data definieras och en ny tråd för att lyssna efter inkommande data initieras. Då data tas emot kastas en händelse vilken hanteras av en metod i *ServerToMobileCommunication.*

```
void close()
```

Metoden stänger strömmarna för ingående och utgående data samt avslutar *TcpConnection* -objektet.

```
string getFirstString()
```

Returnerar det första meddelandet som kom in efter att en klient anslutit. Det krypterade meddelandet sparas under hela objektets livstid och innehåller ett användarnamn. Metoden används då klienten på insidan behöver få reda på vilka klienter som är anslutna till navet. Insideklienten dekrypterar meddelandet och parar ihop användarnamnet med det id navet använder.

```
int getId()
```

Returnerar det unika ID -nummer som tilldelats då objektet skapats.

```
void sendData(string data)
```

Sänder en sträng till denna mobila klient över den ström som är kopplad till objektet. Inparameter är informationen som skall skickas vilket är en krypterad sträng.

```
event datareceivedEventHandler datareceived
```

```
event communicationChangedEventHandler communicationChanged
```

Dessa händelsehanterare används för de fall där data tas emot eller när kommunikationsförhållanden förändras, t.ex. då en anslutning förloras.

InsideToServerCommunication

Klassen hanterar kommunikationen från insideklienten till navet och använder *BOSCH* –teknik för att simulera tvåvägskommunikation.

```
InsideToServerCommunication()
```

Konstruktor, skapar objektet och dess medelmsvariabler.

```
void connect(string uri)
```

Metoden skapar en tråd för inkommande data. Tråden börjar med att ställa en *HTTP* -fråga till den *URI* -address som skickats med som inparameter. Om ett svar återfås existerar en anslutning och en *communicationChanged* -

händelse kastas. *HTTP* -svaret hålls sedan öppet för att ta emot inkommande data och används därmed som en vanlig nätverksström. Då data kommer in kontrollerar man om denna innehåller ett systemmeddelande, t.ex att en ny mobilklient har anslutit till navet. Om det är ett systemmeddelande kastas en *communicationChanged* –händelse. Om inte packas den mottagna datan upp och dekrypteras för att sedan kastas som en *dataRecived* -händelse. Efter detta inväntar tråden ny data. Om ett svar på den första *HTTP* –frågan uteblir kastas en *communicationChanged* –händelse som anger detta och ett nytt försök att kontakta navet initieras.

```
void disconnect()
```

Metoden skickar ett systemmeddelande där det anges att klienten avslutar anslutningen. Navet kommer då att avsluta den svarsström som finns mot klienten. Den tråd hos klienten som tar emot data från navet upptäcker detta och kastar en *communicationChanged* –händelse för att sedan avslutas.

```
bool isConnected()
```

För att enkelt kunna ta reda på om klassen har en anslutning till ett nav används en medlemsvariabel. Denna sätts till *true* då en anslutning lyckats och till *false* då en anslutning misslyckas eller upphör. Genom att kontrollera variablen undviker man att försöka ta emot eller skicka data om en anslutning inte existerar. Metoden returnerar värdet på den medlemsvariablen.

```
void sendData(byte[] data)
```

För att skicka data ställs en ny *HTTP* -fråga, denna gång utnyttjas den inbyggda *POST* -metoden för att få tillgång till en utgående ström. Data krypteras, komprimeras och skickas iväg över strömmen för att därefter stängas. För att följa *HTTP* -protokollet skall ett svar alltid tas emot, navet svarar därför med ett "OK" vilket kan ses som en garanti på att frågan och därmed den skickade datan kommit fram till navet.

```
event dataRecivedEventHandler dataRecived;
```

```
event communicationChangedEventHandler communicationChanged;
```

Dessa händelsehanterare används för de fall där data tas emot eller när kommunikationsförhållanden förändras. T.ex. då en anslutning förloras.

ServerToInsideCommunication

En klass för att sköta kommunikationen från navet till klienten på insidan. Det är klienten som inleder kommunikationen så denna klass startar en tråd och väntar sedan på att en klient skall anslutas. Den är utvecklad så att bara en klient kan ansluta till den.

```
ServerToInsideCommunication(string uri)
```

Klassens konstruktor, medlemsvariabler initieras och ett *HTTP* -lyssnarobjekt skapas, med prefix för vilken adress den skall lyssna på vilket är en inparameter till metoden.

```
void start()
```

Metoden startar en tråd för att lyssna efter inkommande *HTTP* -frågor den första frågan som kommer in skapar man en svarsström till, denna hålls sedan öppen så länge en anslutning finns den används för att skicka ut

meddelanden. En ny tråd skapas för att lyssna efter inkommande meddelanden, även här lyssnar man efter *HTTP* -frågor, varje gång en fråga ställs läses *HTTP* -frågans *POST* -meddelande av för att sedan kasta en *dataReceived* -händelse och sedan skicka iväg . Efter att denna tråd skapats startar man ytterligare en tråd vilken skickar iväg ett meddelande var fjärde minut, vilket används för att hela tiden hålla anslutningen vid liv, och kontrollera så att inte det finns några problem. Efter att dessa trådar skapats avslutas den tråd som väntat efter en första anslutning.

```
void stop()
```

Metoden avslutar de trådar som körs samt stänger de strömmar som är öppna.

```
bool isConnected()
```

En medlemsvariabel håller reda på om en klient är ansluten till navet. Denna används även för att avsluta trådar.

```
bool sendData(byte[] data)
```

För att skicka data till klienten på insidan används den ström som öppnas då klienten ställer sin första fråga. För att inte skicka ut data på en stängd ström kontrolleras först att en anslutning finns.

```
event dataRecivedEventHandler dataRecived;
```

```
event communicationChangedEventHandler communicationChanged;
```

Händelsehanterare vilka måste implementeras i den ägande klassen, de kastas då data kommer in en förändring i anslutningen sker.

Connection

Navets huvudklass.

```
Connection()
```

Klassens konstruktor, skapar ett *ServerToInside* -objekt samt ett *ServerToMobile* -objekt och definierar vilka metoder som skall ta hand om deras *dataReceived-* och *communicationChanged-* händelser. Dessa är privata metoder till klassen vilka måste implementeras, i dessa byggs funktioner för buffring som skall ske då ena sidan ej är uppkopplad mot navet.

```
void start()
```

I denna metoden startar man de trådar vilka lyssnar efter nya anslutningar från insideklienten och mobila klienter. Detta sker genom de underliggande objekt vilka skapats i konstruktorn.

```
void stop()
```

I denna metoden anropas de metoder i de underliggande klasserna som avslutar de anslutningar som finns och stänger de trådar som lyssnar efter nya.

Crypto

En klass för kryptering av data. Används som inparameter till kommunikationskomponenterna på klienterna för att där användas för kryptering och avkryptering av data. Gränssnittet är menat att användas som

en omslutande (*wrapper*) klass för att på ett enkelt sett kunna byta krypteringsalgoritm.

En lämplig krypteringsalgoritm är *Blowfish* -algoritmen är en symmetrisk 128 -bitars krypteringsalgoritm som i dagsläget saknar någon känd kryptoanalyslösning. Algoritmen finns implementerad i ett projekt med kryptografiska bibliotek kallat *bouncycastle*. *Bouncycastle* är ett projekt med öppen källkod för kryptografiska lösningar i *Java*, *JME* och *C#*.

```
Crypto(string key)
```

Klassens konstruktor, här skapas de objekt som sedan används för att kryptera och avkryptera meddelanden. Nyckeln är inparameter.

```
string enCrypt(string data)
```

En metod för att kryptera meddelanden, tar en sträng som inparameter krypterar denna och sedan returneras den krypterade strängen.

```
string deCrypt(string data)
```

En metod för att avkryptera meddelanden, tar en sträng som inparameter avkrypterar denna och sedan returneras den avkrypterade strängen.

Compress

En klass för komprimering av data, används i kommunikationskomponenterna. Gränssnittet skall användas som en *wrapper* -klass för redan implementerade komprimeringsmetoder.

En möjlig komprimeringsmetod är *SharpZipLib* vilket är ett projekt med öppen källkod som implementerar bland annat *GZIP* och kan användas på både klienten på insidan och en mobil klient med *Windows Mobile*. Däremot har ingen öppen källkod för komprimering kunnat hittas för mobila klienter skrivna i *JME* där det således inte finns möjlighet att använda komprimering i dagsläget.

```
Compress()
```

Klassens konstruktor, här skapas de objekt som sedan används för att komprimera och avkomprimera meddelanden.

```
string zip(string data)
```

En metod för att komprimera meddelanden, tar en sträng som inparameter komprimerar denna och sedan returneras den komprimerade strängen.

```
string unzip(string data)
```

En metod för att avkomprimera meddelanden, tar en sträng som inparameter avkomprimerar denna och sedan returneras den avkomprimerade strängen.

NMEAParser

En klass som tar emot *NMEA* -meddelanden över en *COM* -port, tolkar dessa och sparar information om position och noggrannhet. För att se den senaste positionen anropas metoden *getLatestLogg* där man får tillbaka en positionsstruktur vilken är definierad för longitud och latitud men det är enkelt att lägga till variabler för *DOP* -värde och antal aktiva satteliter m.m.

```
NMEAParser(string portName, bool )
```

Klassens konstruktor. Här skapas ett objekt som kan lyssna på en *com* –port och kallas i *.NET Compact framework* för *SerielPort*. Inparametern anger vilken port man skall lyssna på. Då man arbetar med midlets ser detta lite anurlunda ut då man använder en uri istället för en com port för att hitta *GPS* som skickar ut *NMEA* -meddelanden. I *Windows Mobile* måste man först para ihop en *COM* -port med en *GPS* -mottagare. Man definierar även vilken metod som skall användas då data kommer in på porten. Denna metod börjar med att kolla så att meddelandet är giltigt genom att beräkna dess kontrollsumma och jämföra det med den som står i meddelandet. Därefter kollar man vilken meddelandetyp detta är. Efter det är gjort kan man spara informationen från de fält som är intressanta, t. ex *GPGLL* -fält för longitud och latitud.

```
void startRecive()
```

I denna metod anropas det objekt som skall lyssna på porten och får instruktionen att börja lyssna efter inkommande data.

```
void stopRecive()
```

Här stängs porten så att ingen ny data kommer in.

```
Position getLatestLogg()
```

Här returneras den positionsstruktur som innehåller den senaste mottagna datan från *GPS* -en.

Appendix C

Testbeskrivning, mobilklient	JME	.NET CompactFramework
Kan ansluta till navet	Ja	Ja
Kan skicka data till navet	Ja	Ja
Tar mot data från navet	Ja	Ja
Upptäcker om anslutningen till navet försvinner	Ja	Ja
Kan kryptera data	Nej	Ja
Kan dekryptera data	Nej	Ja
Kan komprimera data	Nej	Nej
Kan dekomprimera data	Nej	Nej
Kan ansluta till *GPS* –mottagare	Ja	Ja
Kan läsa *NMEA* –strömmen från *GPS* –mottagare.	Ja	Ja
Tolkar inkomna meddelanden korrekt	Ja (om ej krypterade)	Ja
Presenterar tolkade meddelanden	Ja	Ja

Tabell C.1. Utfall för löpande testning av mobilklient.

Testbeskrivning, nav	Utfall
En eller flera mobilklienter kan ansluta till navet	Ja
Tar mot data från mobilklient	Ja
Kan skicka data till mobilklient	Ja
Upptäcker om anslutningen till en mobilklient försvinner	Ja
Insideklienten kan ansluta till navet	Ja
Tar mot data från insideklienten	Ja
Kan skicka data till insideklienten	Ja
Upptäcker om anslutningen till insideklienten försvinner	Ja (inom 30 sekunder)
Meddelar insideklienten vilka mobilklienter som är anslutna när insideklienten ansluter	Ja
Meddelar insideklienten om en ny mobilklient ansluter.	Ja
Vidarebefordrar data från mobilklienter till insideklienten	Ja
Vidarebefordrar data från insideklienten till rätt mobilklient	Ja

Tabell C.2. Utfall för löpande testning av nav.

Testbeskrivning, insideklient	Utfall
Kan ansluta till navet	Ja
Kan skicka data till navet	Ja
Kan ta mot data från navet	Ja
Kan kryptera data	Ja
Kan dekryptera data	Ja
Kan komprimera data	Ja

Kan dekomprimera data	Ja
Upptäcker om anslutningen till navet försvinner	Ja
Tolkar inkomna meddelanden korrekt	Ja
Presenterar inkommande positioner i ett kartfönster	Ja
Hanterar situationen då flera mobilklienter är anslutna och uppdaterar positionen för rätt markör	Ja
Presenterar inkommande textmeddelanden	Ja
Kan upprätta anslutning genom brandvägg	Ja

Tabell C.3. Utfall för löpande testning av insideklient.

Referenser

Tryckta källor

Faison T (2006). Event-Based Programming: Taking Events to the Limit. ISBN 1 59 059643 9

Ford W (1994). Computer Communications Security. ISBN 0 13 799453 2

Forouzan B (2007). Data Communications and Networking. Fjärde upplagan. ISBN 978 007 125442 7

Hofmann-Wellenhof B, Lichenegger H, Collins J (2001). GPS Theory and Practice. Femte upplagan. ISBN 3 211 83534 2

Johnson G, Scholes K, Whittington R (2005). Exploring Corporate Strategy. Sjunde upplagan. ISBN 0 273 68734 4

Jonsson P, Mattsson S-A (2005). Logistik – Läran om effektiva materialflöden. ISBN 91 44 04182 9

Lundmark A, Tisén T (2006).
Analysis and adaptation of component-based software engineering for system integration.

McClure S, Scambray J, Kurtz G (2003). Hacking i fokus. Fjärde upplagan. ISBN 91 636 0774 3

Nagel C, Mungale A, Kumar V, Laghari N, Krowczyk A, Parker T, Sivakumar S, Serban A (2004). Pro .NET 1.1 Network Programming. Andra upplagan. ISBN 1 59059 345 6

Oaks S, Wong H (1999). Java Threads. Andra upplagan. ISBN 1 56592 418 5

Peng Z-R, Tsou M-H (2003). Internet GIS. ISBN 0 471 35923 8

Sayood K (2000). Introduction to Data Compression. Andra upplagan. ISBN 1 55860 558 4

Sommerville I (2001). Software Engineering. Sjätte upplagan. ISBN 0 201 39815 X

Strand L (2001). UML & RUP – Att lyckas med projekt. ISBN 91 7882 571 7

TietoEnator (2001). Praktisk ProjektStyrning

Weiss M A (2006). Data Structures & Problem Solving Using Java. Tredje upplagan. ISBN 0 321 31255 4

Internet

Baddeley G (2006). GPS – NMEA sentence information. Hämtad 2007-05-18. http://www.werple.net.au/~gnb/gps/nmea.html

Internet Engineering Task Force (2000). Remote Authentication Dial In User Service (RADIUS). Hämtad 2007-06-01. http://tools.ietf.org/html/rfc2865

Telia (2007). Täckningskarta. Hämtad 2007-04-27. http://www.telia.se/privat/treramar.do?_EXTERN=http://mobiltackning.telia.se/KARTAN_V3/&channelId=-103502&tabId=0

OGC (2007). About OGC. Hämtad 2007-05-23. http://www.opengeospatial.org/ogc

OGC (2005). Web Feature Service Implementation Specification. Hämtad 2007-05-19. http://www.opengeospatial.org/standards/wfs

OGC (2004). Geography Markup Language (GML). Hämtad 2007-05-19. http://www.opengeospatial.org/standards/gml

OGC (2004). OGC Web Map Service Interface. Hämtad 2007-05-22. http://www.opengeospatial.org/standards/wms

Sun Microsystems (2007). Java ME Technology. Hämtad 2007-07-09. http://java.sun.com/javame/technology/index.jsp

W3C (2006). Extensible Markup Language (XML) 1.0. Fjärde upplagan. Hämtad 2007-05-05. http://www.w3.org/TR/2006/REC-xml-20060816/)

XMPP Standards Foundation (2007). XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH). Hämtad 2007-05-14. http://www.xmpp.org/extensions/xep-0124.html

Bilaga

SWECO Position

EXAMENSARBETE
MOBIL SYSTEMARKITEKTUR

UPPDRAGSPLAN

Rev 0
2007-03-20
SWECO Position AB
Malmö

Joakim Antonsson
Johan Hasselström

Uppdragsnummer exj2007-GPS-GIS

SWECO POSITION
Hans Michelsensgatan 2
Box 286, 201 22 Malmö
Telefon 040-16 70 00
Telefax 040-15 43 47

SWECO

Uppdragsplan

Kund

SWECO Position
Fredrik Ekelund
Hans Michelsensg. 2
20122 MALMÖ

Organisation

Personer	Roller
Joakim Antonsson	Examensarbetare
Johan Hasselström	Examensarbetare
Fredrik Ekelund	Uppdragsgivare, SWECO
Ulf Månsson	Teknisk granskare, SWECO
Lars Harrie	Handledare, LTH
Klas Ernald-Borges	Examinator, LTH

Omfattning

Projektet syftar till att ta fram ett systemförslag för hur man med standardiserade öppna gränssnitt kan skapa kommunikation av geografiska data och annan information mellan mobil enhet och server. Förslaget skall verifieras mot ett användarfall med en klient och en server för att kunna dra slutsatser inför en eventuell skarp version.

Förutsättningar och krav

Projektet utförs inom ramen för examensarbeten på Lunds Tekniska Högskola och löper över tiden 19:e mars till 20:e augusti.

Då den mobila bandbredden blir bättre skapas nya förutsättningar för mobil kommunikation. Då tillverkare utav mobiler har börjat göra mobiler med GPS-chip finns stora möjligheter att integrera mobiler i geografiska informationssystem.

SWECO tillhandahåller samtlig utrustning och mjukvara som krävs för genomförande och testning. Arbetet utförs på kontoret i Malmö vilket skapar en bra kunskapsbas med stor samlad kompetens.

Tid och resursplanering

Projektet omfattar 20 högskolepoäng vilket motsvarar 20 veckors heltidsarbete för vardera examensarbetare.

Tidplan är bifogad på sida 5.

Kvalitets- och miljöstyrning

För att säkerställa kvalitén hos de utvecklade applikationerna skall kravspecifikationer upprättas och testning genomföras. För att säkerställa att projektet blir klart i tid sätts milstolpar och delleveranser upp.

Rapportens kvalité skall säkerställas genom att kontinuerligt stämma av rapportens innehåll med handledare genom möten samtidigt som materialet granskas av båda rapportskrivarna.

Varje vecka sker möten med handledare på SWECO och ungefär var tredje vecka med handledare på LTH.

Dokumentstyrning

Vi har tilldelats en nätverksmapp på SWECO server.

g:/6641/adm/data/exjobb/ex2007/GPS_GIS

Där finns underbibliotek med följande struktur:

Källkod: Backup från Sourcesafe och utvecklingsserver.

Underlag: Här sparas intressanta artiklar och rapporter

Rapport: Här sparas de olika dokument som används för rapporten.

Dokument: Olika dokument vilka framställs under projektet.

För att veta vilket som är det senaste dokumentet kommer vi att använda ett versionsnummer i filnamnet.

Vid implementering används SourceSafe.

Informationsstyrning

Eftersom examensarbetarna delar kontor är informationsflödet dem emellan inget problem. Inför möten med handledare skall överenskommet dokument levereras senast 24 timmar innan mötet äger rum.

Riskanalys

Klassificering:
A – allvarliga konsekvenser
B – medelsvåra konsekvenser
C – mindre konsekvenser

Risk	Klass	Åtgärd
Dataförlust	A	All information sparas på SWECO:s server där backup tas kontinuerligt. För sådan data SWECO inte tar backup på skall egna veckovisa backuper göras till USB-minne.
Omarbete på grund av ändrat syfte eller inriktning av arbetet.	A	Såväl projektplan och disposition skall tidigt godkännas av berörd handledare.
Projektets omfattning upptäcks vara för stort för tidsrymden.	A	En översyn och eventuell revidering av syfte och omfattning kommer göras enligt milstolparna.
Bristande handledning.	B	Alla initiativ ska tas av examensarbetarna. Är handledningen så bristfällig att arbetet inte kan utföras ska möte med samtliga handledare anordnas för diskussion.
Förseningar på grund av felaktigt uppskattad tidsåtgång för delmoment	B	Detaljplanering görs och 15% bufferttid finns av totala projekttiden.
Förseningar på grund av övriga uppdrag	B	Examensarbetet sätts som högsta prioritet. Om arbetet ligger efter tidplanen får inga nya uppdrag accepteras.
Tidsförlust p.g.a sjukdom.	B	Båda ska vara väl insatta i varandras arbeten. Buffertveckor finns inplanerade.
Nödvändig utrustning och/eller mjukvara saknas vid implementering och test.	B	Ett dokument innehållande utrustningslista (hårdvara och mjukvara) överlämnas enligt milstolparna.