Vektorbaserade kartor för mobila terminaler

Författare: Magnus Janlert

Intern handledare: Jerry Eriksson
Extern handledare: Peter Jacobson 23 augusti 2004

Innehåll

1 Inledning **1**

1.1 Introduktion 1

1.2 Problembeskrivning 1

1.3 Upplägg 2

2 Bakgrund **5**

2.1 Kartor 5

2.1.1 Koordinatsystem 5

2.1.2 Objekt 6

2.1.3 Metadata 6

2.2 Representation 6

2.2.1 Rastergrafik 8

2.3 Vektorgrafik 9

2.3.1 Lagerhantering 11

2.4 Mobila kartor 11

2.4.1 Nätverk 12

2.4.2 Cacheing 13

2.4.3 GPS 13

2.4.4 Carteus 14

3 Tekniker **15**

3.1 Tekniker, protokoll och produkter 15
3.1.1 W3C 15
3.1.2 Open GIS 16
3.1.3 ESRI 19

4 Implementationen **23**

4.1 Introduktion 23
4.2 Arkitektur 23
4.2.1 Klienten 23
4.2.2 Servern 24
4.3 Systembeskrivning 26
4.3.1 Cacheing 26
4.3.2 Klienten 29
4.3.3 Serv 32
4.3.4 Connector 32

5 Avslutning **33**

5.1 Begränsningar 33
5.1.1 Prestanda 33
5.1.2 Vidareutveckling 33
5.2 Tack 34
5.3 Referenser 35

Abstract

Traditional maps can easily be carried anywhere, but computer based maps offer many advantages; better coverage and more details. Adding to that is some new features such as searching. With mobile terminals (such as the Palm or Pocket PC) it is possible to combine the two. And a network connection make it possible to store the data on a central server, removing any limit on the data size.

One such system for mobile maps is Carteus, developed by Visualiseringscentrum. Carteus is however only capable of raster graphics. This report is the result of an attempt at adding vector graphics capabilities to Carteus. The systems uses the open standards WFS and GML and the client caches data locally to increase speed and availability.

Sammanfattning

Traditionella papperskartor kan lätt tas med varhelst man går, men datoriserade kartor har många fördelar över papperskartor; större omfång och detaljrikedom samt möjlighet till ny funktionalitet som till exempel sökning. Med mobila terminaler (exempelvis Palm och Pocket PC) finns möjligeten att kombinera fördelarna från båda typerna. En nätverksanslutning ökar funktionaliteten ännu mer, datan kan då lagras centralt, vilket mer eller mindre tar bort de praktiska begränsningarna på datamängdens storlek.

Ett sådant kartsystem för mobila terminaler är Carteus, som utvecklats av Visualiseringscentrum. I grunden hanterar Carteus dock bara rastergrafik. Denna rapport är resultatet av ett examensarbete som har som mål att lägga funktionalitet till Carteus för att hantera vektorgrafik. Systemet använder de öppna protokollen/formaten WFS och Open GIS för hantering av geografisk data. Klienten använder sig av cacheing av vektordatan för att öka hastighet och tillgänglighet för klienten.

Kapitel 1

Inledning

1.1 Introduktion

Denna rapport tar upp ett examensarbete, utfört av Magnus Janlert vid Umeå universitet. Arbetet, som är på 20 poäng (Examensarbete D, kurskod TDBD01), utfördes under vårterminen 2003 vid Visualiseringscentrum i Umeå. Intern handledare vid den Datavetenskapliga institutionen vid Umeå universitet är Jerry Eriksson och extern handledare vid Visualiseringscentrum i Umeå är Peter Jacobson.

Visualiseringscentrum är en del av Lantmäteriet som utvecklar internetbaserade karttjänster. Detta innefattar både försäljning av kartor på webben och mobila kartor.

1.2 Problembeskrivning

En av Visualiseringscentrums produkter heter Mapmate och är en vidareutveckling av ett tidigare examensarbete [5]. Mapmate, som bygger på Carteus, är en produkt för mobila kartor som låter användaren navigera i sina kartor från en mobil terminal.

Figur 1.1 visar några skärmutskrifter från Carteus som visar hur det kan se ut. Programmet (i fortsättningen kallad klienten) körs på användarens mobila terminal (exempelvis en handdator), medan kartorna ursprungligen lagras centralt på Lantmäteriets kartserver. När användaren navigerar till ett område som inte tidigare besökts skickar klienten en förfrågan till servern om den del av kartan som saknas. Klienten sparar sedan denna information, så när användaren nästa gång besöker samma position igen behöver

klienten inte ha någon kontakt med servern, vilket ökar både hastighet och tillgänglighet.

Det finns i princip två olika sätt att representera grafik; som rastergrafik och som vektorgrafik. Både kartors uppbyggnad samt användningsmönster lämpar sig för vektorgrafik. Informationen kan representeras effektivare, användaren får bättre kontroll över presentationen och framförallt är den skalningsinvariant, samma data kan alltså användas i flera olika skalor. En närmare förklaring av raster- och vektorgrafik och vad de två teknikerna innebär finns i avsnittet Representation.

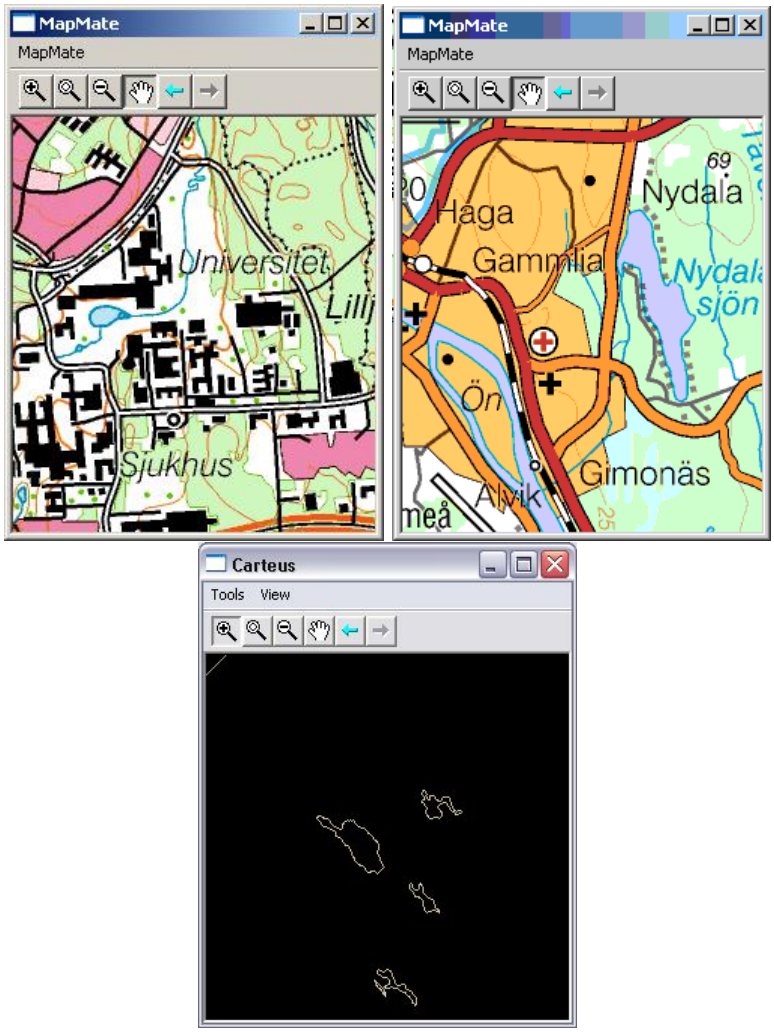
I grundutförande klarar dock Carteus bara av att hantera rastergrafik (båda bilderna i figur 1.1 baseras på rastergrafik). Uppgiften gick ut på att implementera en utbyggnad av Carteus för hantering av vektorgrafik.

Problemet har delats upp i tre huvuddelar. Första steget är att upprätta en kartserver, sedan ska informationen överföras från kartservern till klienten, där ska datan lagras på något effektivt sätt så att åtkomst blir så snabb som möjligt. Till slut ska datan presenteras på skärmen. Under detta arbete måste val av protokoll, filformat och representation göras.

Från användarens synvinkel blir resultatet en mer flexibel karta och i många fall snabbare svarstider. Det blir även tekniskt möjligt för användaren själv att ”komponera” sina egna kartor, istället för som med rastergrafik välja mellan ett antal fördefinierade kartor och skalor.

1.3 Upplägg

Efter introduktionen kommer en allmän översikt av datorbaserade kartor och hur de är uppbyggda. Istället för att beskriva de teknologier som använts allt eftersom så kommer först en beskrivning (en slags ordlista) över de standarder, protokoll, organisationer och format jag senare kommer att nämna. Sedan kommer en beskrivning över det system jag har konstruerat. Sist tas några avslutande frågor upp, bland annat begränsningar och framtida utvecklingsmöjligheter.



Figur 1.1: Tre bilder på MapMate över delar av Umeå, de två första är från standardversionen med rastergrafik, den tredje visar sumpmarksområden utritade med vektorgrafik (rastergrafiken är där borttagen för tydlighetens skull)

Kapitel 2

Bakgrund

2.1 Kartor

Nedan kommer en beskrivning av de olika delarna som utgör en karta. För olika ändamål behövs olika kartor och detta uppnås genom att kombinera olika varianter och data från nedanstående beståndsdelar.

2.1.1 Koordinatsystem

För att kunna orientera sig i en karta behövs någon form av koordinatsystem som kan ange en position på kartan. Det största problemet är givetvis att jorden inte är platt, till skillnad från kartorna man försöker avbilda den på, så det krävs någon form av projektion. Vid en projektion placeras kartan i önskat förhållande till en modell av jorden som sedan får avbildas mot kartan genom linjer som går från ett projektionscentrum och genom jordytan tills dess att de träffar kartan.

Två saker påverkar hur koordinatsystemet (och därmed jorden avbildning på kartan) ser ut; projektionsplanet och perspektivet. Projektionsplanet är formen på kartan vid projektionen. De tre vanligaste är ett plan (planet nuddar jordytan vid en punkt, även kallad azimutal projektion), en cylinder (planet nuddar jordytan i en linje) och konisk (även här nuddas jordytan i en linje).

Perspektivet avgör var projektionsstrålarna utgår ifrån, exempelvis jorden mittpunkt eller vinkelrätt mot projektionsplanet.

Kartor över Sverige använder sig nästan alltid av ”Rikets nät”, även kallat RT 90 [7], som är en cylindrisk projektion. Medelmeridianen (linjen där

cylindern nuddar jordytan) ligger något väst om Stockholm ($15°48'29.8''$ öst om Greenwich) och för att undvika negativa koordinater har origo förskjutits 150 mil västerut.

2.1.2 Objekt

Objekten (eng. feature) är det användaren ser, till exempel vägar, städer, byggnader, skog och andra geografiska objekt. I digitala kartor kan varje objekt ha en mängd attribut. En del är normalt synliga för användaren (typ av objekt, position, färg, form mm.) och en del inte (exempelvis åt vilket håll en flod rinner eller hur hårt trafikerad en väg är). Beroende på tekniken som presenterar kartan kan det finnas möjlighet att direkt påverka presentationen av kartan, vilka objekt som skall vara med på kartan och den information om dessa som skall synas, på det sättet kan kartor konstrueras dynamiskt efter olika behov.

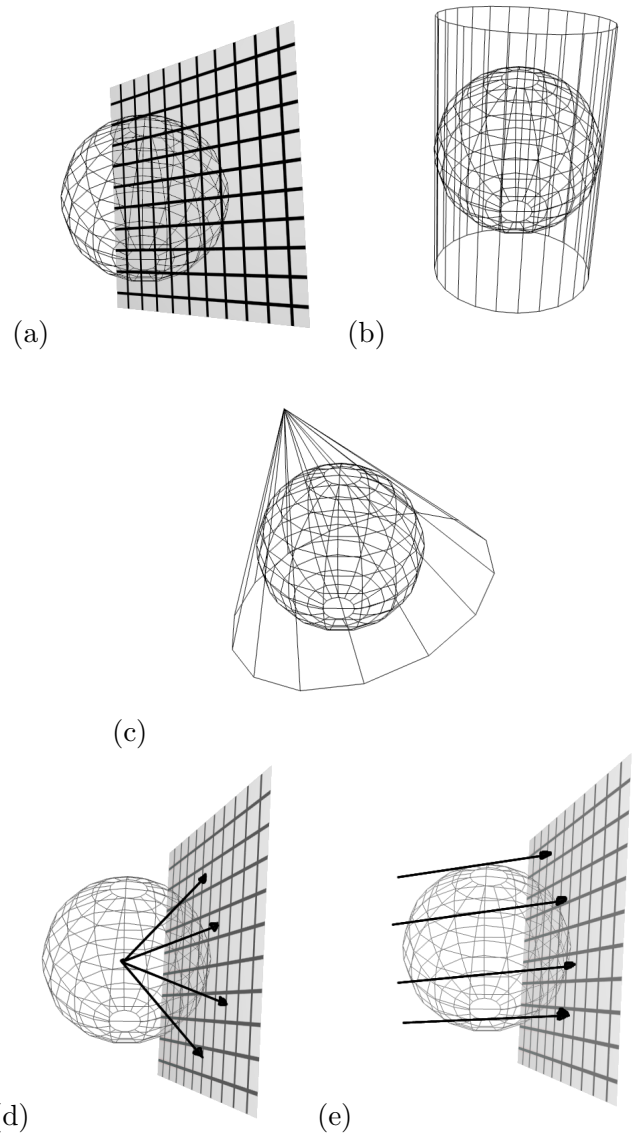
2.1.3 Metadata

Metadata är data om data, i detta fall information om datan som utgöra kartan. Detta är ofta nödvändigt både för den mjukvara som ska presentera kartan för användaren och användaren själv. Metadata kan vara information om vilket koordinatsystem som används, utgivningsår, kartans namn, skala, kartans gränser och pris för tjänsten. För vektorkartor behövs även en hel del informationen om de olika lager som finns (vad ett lager är förklaras senare).

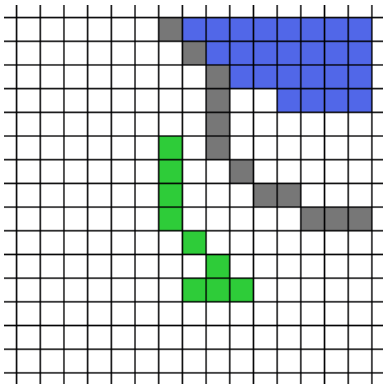
2.2 Representation

Det finns två huvudsakliga sätt att representera kartor digitalt (här menas intern representation och inte hur de presenteras för användaren, även om den interna representationen ibland kan påverkar hur användaren kan interagera med kartan), som raster- eller vektorkartor. Som så ofta är ingen av dessa två tekniker avgjort bäst i alla lägen, utan de fyller olika funktioner. Här följer en beskrivning av de olika metoderna, samt ett avsnitt om lagerhantering.

De två teknikerna utesluter dock absolut inte varandra, utan kan med fördel kombineras. Till exempel går det att välja flygfotokarta som bakgrund och sedan rita upp vektorlager ovanpå denna.



Figur 2.1: Olika projektioner och perspektiv
(a) plant projektionsplan
(b) cylindriskt projektionsplan
(c) koniskt projektionsplan
(d) perspetiv med projektionsstrålar utgående från jorden centrum
(e) parallella projektionsstrålar



Figur 2.2: En rasterbild med de flesta pixlarna i bakgrundsfärg

2.2.1 Rastergrafik

Rastergrafiken är den vanligaste och enklaste metoden. Bilden lagras i en matris, där varje värde i matrisen (kallad ”pixel”) representerar en punkt i bilden. Varje pixel representeras med ett eller flera värden, exempelvis ett gråskalevärde $v \in (0 - 15)$ eller en färg $\{r, g, b\} \in (0 - 255)$. Dess enkla struktur liknar mycket en dators minne vilket gör den enkel att hantera, lagra och överföra i datorer. Formatet passar bra för fotografier (i kartsammanhang oftast flygfotografier) och liknande där hela bilden kan vara fylld med små detaljer utan att detta påverkar bildens egenskaper (exempelvis storlek och kostnad att bearbeta). Många vanliga operationer är enkla och går att genomföra effektivt, exempelvis att slå samman flera bilder till en (bara att skapa en större matris och fylla denna med de mindre) och att bearbeta delar av bilden. Då i princip alla dagens bildskärmar bygger på rastergrafik är det oftast trivialt att rendera den till bildskärmen.

Om bilden har stora ytor av samma färg (vilket kartor ofta har) så är det inte utrymmeseffektivt, stora delar av matrisen innehåller samma värde. Även mycket detaljrika rasterbilder går oftast bra att komprimera.

Beroende på hur kartan ser ut bör olika komprimeringsmetoder användas. Om kartan innehåller stora ytor med samma färg (som i figur 2.2) fungerar en förlustfri (eng. lossless) komprimering bäst. Den enklaste varianten är Run Length Encoding (RLE), där bilden kodas med par av (*pixelvärde*, *antal*). Denna triviala metod kan vid vissa fall ge bra resultat, trots sin enkelhet. 2.2 är en bra kandidat för RLE. Mer avancerade algoritmer (exempelvis Huffmankodning) bygger på att byta ut vanligt förekommande sekvenser med kortare koder och ovanliga sekvenser med längre koder.

För mer verkliga bilder (fotografier, eller kartor med hög detaljnivå och mjuka färgövergångar) används oftast komprimering med förlust. Priset som

får betalas är som namnet antyder att den återskapade bilden inte blir en exakt kopia av originalet, förtjänsten är att mycket högre kompressionsgrad kan uppnås. Det absolut vanligaste formatet/algoritmen för denna typ av kompression är JPEG. Algoritmen ser till att utnyttja information som ögat inte ser och ofta kan en bild komprimeras upp till 75% innan ögat börjar märka någon skillnad. Den återskapade bilden innehåller dock små defekter, mest märkbar är små ”ringar” omkring skarpa kanter (dessa kan med nöd och näppe urskiljas i bilden till vänster i 1.1).

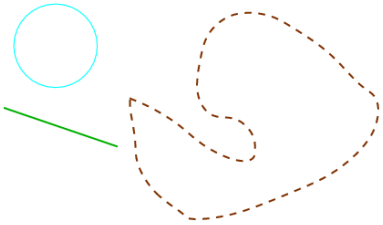
Den största nackdelen med rastergrafik när det gäller representation av kartor är att datan inte är skalnings- eller rotationsinvariant. Resultatet blir att samma data måste lagrad flera gånger, i olika skalor. Förutom det uppenbara problemet att detta resulterar i större utrymmeskrav så innebär det att användaren bara kan se kartan i de skalor som erbjuds. Det innebär även att varje gång användaren zoomar in eller ut måste en helt annan datamängd användas. Om användarens terminal är kraftfull nog att kunna lagra all data lokalt är detta kanske inget problem, men om användarens terminal bara kan lagra en liten del av den totala datan lokalt (eller av någon annan anledning en central server måste användas) så måste eventuellt ny data överföras från en server, vilket tar tid och kan dessutom vara (ekonomiskt) avgiftsbelagt.

Givetvis kan man använda någon av de många mer eller mindre avancerade algoritmerna som finns för att skala om och rotera rastergrafik. Men även de bästa algoritmerna resulterar förutom i några specialfall alltid i en kvalitetsförlust, det första som blir lidande är texter (ortsnamn med mera). Dessa operationer är dessutom ofta relativt dyra, då det ofta krävs flyttalsoperationer för att få ett något i närheten av användbart resultat. Skalning av rastergrafik är ändå användbart i vissa lägen, i Carteus används en omskalning som förhandsbild medan den nya datan överförs från servern.

2.3 Vektorgrafik

Alternativet till rastergrafik är alltså vektorgrafik, som utnyttjar att bilden har någon slags struktur. Bilden representeras nu som en mängd visuella objekt av olika typer. Exempel på objekt kan vara linjer, cirklar, punkter och polygoner. Till varje objekt hör informationen som definierar objektet, detta innebär både geometrisk information (position, skala, rotation, punkter) och manér (dess utseende, färg och stil). Till exempel kan vi ha ett objekt ”linje”, denna linje har två ändpunkter (start och slutpunkt) (4, 9) och (2, 8), linjens tjocklek (1mm) och färg (blå).

Rendering (det vill säga konvertering till rastergrafik för visning på skärmen)



Figur 2.3: Bild baserad på vektorgrafik. Figuren består av tre objekt med olika manér (färg, mönster, stil, storlek osv); en cirkel, en linje och en polygon med tio punkter.

är något mer komplicerat än för rastergrafik, det behövs rutiner för att rita ut linjer, cirklar och andra objekt. Det bör även ske en kontroll av objektets position, om hela objektet är utanför det synliga fönstret så kan det objektet ignoreras helt och hållet.

För rastergrafik är kostnaden för rendering enbart beroende av bildens storlek, två bilder med samma storlek har samma kostnad oberoende av innehåll. Vektorgrafikens renderingskostnad beror däremot på komplexiteten i bilden; antalet objekt, typ av objekt och objektens komplexitet, så det går inte att direkt säga något om skillnaderna i hastighet mellan de två metoderna. En konsekvens av detta är att uppdateringsfrekvensen för vektorgrafik varierar beroende på innehållet, vilket kan upplevas som mycket negativt av användaren.

Vektorgrafik ger dock slutanvändaren möjlighet att påverka utseendet på grafiken. Med rastergrafik ges bilden explicit av datan, på sin höjd kan användaren påverka kontrast och ljusstyrka. Med vektorgrafik kan varje slutanvändare själv bestämma hur slutresultatet ska se ut; ”sjöar ska vara lila och skogsvägar svarta linjer” eller ”placera ortnamnen så de inte täcker över några vägar”. Det är sedan renderingsprocedurens jobb att följa denna specifikation.

Den största fördelen är att vektorgrafik är skalnings- och rotationsinvariant. Till skillnad från rastergrafik där det exempelvis finns en färdigrenderad linje så innehåller vektordatan bara den information som är nödvändig för att återskapa denna linje. Det ger möjlighet att återskapa denna linje med godtycklig precision samt möjlighet att transformera (framförallt skala och rotera) linjen innan den renderas.

Rotation är ofta inte nödvändigt, utan snarare en bekvämlighet, men skalning är ofta en nödvändig funktion. Med rastergrafik så är presentationen låst till de skalor som kartan finns lagrad i, ett fixt antal fördefinierade skalor. Varje skala måste lagras för sig och tar därmed mer plats. Med vektordata kan samma data användas för alla skalor och användaren kan rotera och välja

skalan steglöst utan kvalitetsförlust.

För att detta ska fungera så måste bilden ursprungligen vara i vektorformat. Konverterting från vektorgrafik till rastergrafik är relativt enkelt, men att gå från raster- till vektorgrafik är betydligt svårare och innebär i princip alltid en del manuellt arbete. Givetvis skulle det gå att konvertera en rasterbild till vektorbild genom att ersätta varje pixel i rasterdatan med en motsvarande punkt i vektorbilden, men då har i princip alla vektorgrafikens fördelar gått förlorade.

2.3.1 Lagerhantering

Digital kartor delas in i lager, där varje lager innehåller olika slags information. Detta låter användaren själv komponera ihop sin "egen" karta över den information som behövs just nu, exempelvis en karta där bara lövskog och vägar visas.

För rasterkartor uppnås detta genom att lägga till en alpha-kanal, som bestämmer transparensen för varje pixel och lagren ritas sedan upp ovanpå varandra. Tyvärr så ökar utrymmeskostnaden proportionellt mot antalet lager.

Med vektorgrafik grupperas istället objekten efter vilket lager de tillhör, det är sedan en enkel sak att välja ut vilka av dessa lager som skall ritas ut och inte.

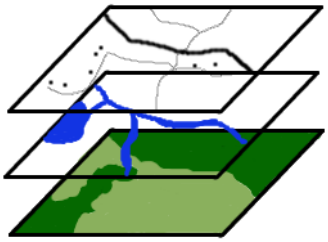
En annan användning av lager är att begränsa i vilka skalor de skall synas, så systemet automatiskt kan stänga av ett lager när användaren flyttar skalan utanför en visst intervall. Ju mindre detaljer desto lägre intervall bör de vara synliga i, det är ingen hjälp att visa alla små skogsvägar i en skala 1:1000000.

2.4 Mobila kartor

Med mobila kartor menas i detta sammanhang kartor som presenteras av en liten bärbar dator dator (även om vanliga papperskartor även de är mobila), det ger beräknings- och presentationsfördelarna från datorbaserade kartor kombinerat med rörelsefriheten hos papperskartor.

Ofta kombineras den mobila kartan med ett trådlöst nätverk, exempelvis GSM eller 802.11. Ibland används även positionsbestämning, ofta genom en GPS-modul (beskrivs senare), genom denna får datorn information om var användaren befinner sig.

I dagsläget är denna teknologi framförallt riktad mot yrkesutövare, då det



Figur 2.4: Flera lager i en vektorbild. Första lagret innehåller skog och annan markinformation, andra lagret vatten och vattendrag och översta lagret vägar och byggnader. Man kan på detta sätt komponera ”egna” kartor efter behov.

krävs en del relativt dyrbar hårdvara (handdator eller mobiltelefon med GPS-modul) utöver mjukvaran. I framtiden är det mycket troligt att denna mjukvara kan köras direkt på telefonen samt att telefonen har en inbyggd GPS-modul, vilket gör mobila kartor mycket mera tillgängligt för den stora allmänheten.

Nedan följer en genomgång av några viktiga egenskaper hos mobila kartor.

2.4.1 Nätverk

Om ett trådlöst nätverk kombineras med den mobila kartan uppnås två effekter. Dels så kan själva karttjänsten förbättras, hela kartan behöver inte lagras på den mobila terminalen utan på en central server, vilket ger möjlighet för mycket större och mer detaljerade kartor. Vidare finns även möjlighet att använda nätverket för nya funktioner, exempelvis dynamiskt hämta tredjepartsinformation om platser eller följa andra personers position.

I dagsläget är GSM-nätet oftast det bästa alternativet. I storstadsområdena har GSM god täckning, men i glesbygd och obebodda områden varierar täckningen, inte minst mellan olika operatörer. En satellittelefon ger fullständig täckning, men är väldigt dyr både i inköp och användning.

När man överför data med en GSM-telefon finns flera tillvägagångssätt. Precis som med fast telefoni går det att använda ett modem och ringa upp en modempool. Detta går dock relativt långsamt och den ekonomiska kostnaden är densamma som för ett vanligt samtal. Alternativt kan GPRS användas, en typ av paketbaserad dataöverföring för GSM-nät. Högre hastigheter kan uppnås och kostanden är proportionell mot mängden överförd data, inte mot tiden.

802.11 (som är avsett för trådlösa lokala nätverk) har en ytterst begränsad räckvidd (upp till några hundra meter), så det är endast i undantagsfall detta är intressant. I framtiden kan 3G/UMTS bli ett tredje alternativ.

2.4.2 Cacheing

Cacheing (eng. "cache") innebär att man lagrar undan data som man hämtat från ett långsammare minne i ett snabbare minne. När det gäller mobila kartor innebär det att klienten lagrar data den mottagit från servern i förhoppning om att detta undviker behovet av denna långsamma överföring i framtiden, med dagens GSM-nät är överföringshastigheten bara ett fåtal kilobyte per sekund.

Men det är inte bara mängden överförd data som är avgörande för den upplevda hastigheten. Round Trip Time (RTT) är den kortaste möjliga tiden för ett paketet att ta sig från klienten till servern och sedan tillbaka igen. För en GSM-telefon är den avsevärd, upp emot en sekund, så det är även högst angeläget att reducera antalet förfrågningar, eller åtminstone antalet förfrågningar där användaren märker av någon fördröjning.

Cacheing är mycket viktigt för användbarheten då det ur användaren perspektiv ökar hastigheten på systemet dramatiskt, väntetiden på klienten minskas från par sekunder ner till någon tiondels sekund. De exakta tiderna beror givetvis på en mängd faktorer: hur bra GSM-mottagningen är, prestanda på terminalen, kvaliteten och storleken på kartan och så vidare.

Den andra mycket viktiga funktionen är att tjänstens tillgänglighet ökar. Om servern av någon anledning inte är kontaktbar, om till exempel GSM-nätet inte har täckning i området eller om servern är ur funktion, så kan klienten fortsätta att fungera genom att använda cachen. Det kan givetvis uppstå "gråa ytor", områden som inte finns lagrade i cachen. För att undvika detta kan användaren navigera förbi området i förväg när det finns kontakt med servern.

2.4.3 GPS

GPS, som är en förkortning för Global Positioning System, är ett system för positionsbestämning. Systemet använder ett antal satelliter som skickar ut signaler som fångas upp av en mottagare nere på marken. Utifrån dessa signaler kan mottagaren beräkna sin position i princip var som helst på jorden med en precision på ett tiotal meter.

När en GPS-mottagare kopplas ihop med en mobil karta får man ett flertal nya intressanta funktioner. Kartan kan automatiskt följa med användarens

förflyttningar, så att användarens position hela tiden är centrerad i mitten av skärmen. Användaren kan även lagra och överföra positioner för att sedan lätt hitta tillbaka till samma ställe, mjukvaran ger information om riktning och avstånd till målet.

Även avståndsberäkningar och hastighetsbedömningar är möjliga. Då det relativa felet i positionsberäkningen är litet uppnås bra precision vid hastighetsberäkningar, med fel ned mot 0.5km/h. En uppskattning av hastighet och färdriktning kan vara användbar både för användaren och systemet, exempelvis kan delar av kartan som inte redan finns i cachen kan hämtas i förväg så att användaren helt slipper någon fördröjning medan datan överförs.

2.4.4 Carteus

Den produkt som bygger på Carteus heter Mapmate, men teknologin och mjukvaran som utgör Mapmate heter Carteus. I denna rapport kommer namnet Carteus att användas, då det i detta fall är tekniken bakom produkten som är intressant. Systemet bygger på ett tidigare examensarbete på Visualiseringscentrum [5].

Carteus körs på Pocket PC och är skrivet i C++. Det har licensehantering, cacheing, sökning, hanterar rastergrafik och har integrerat stöd för GPS (det kan lagra positioner och stigar). Arbete har även gjorts för att köra programmet på andra typer av mobila terminaler (och därmed slippa den ganska dyra handdatorn), både som Java-program och direktöversättning som C++-program.

När Carteus nämns i denna rapport, exempelvis vilka funktioner som finns eller hur något fungerar så gäller det hur produkten MapMate fungerade när jag avslutade mitt praktiska arbete.

Kapitel 3

Tekniker

3.1 Tekniker, protokoll och produkter

För att bygga ihop ett system för mobila karto, i detta fall vektorkartor, finns det ett antal format, protokoll, standarder och produkter som kommer att vara intressanta..

Här en en förklaring av de som använts i detta arbete, vad de har för uppgift och vilket problem de har löst. Hur dessa har använts och passar in i systemet förklaras i nästa kapitel.

3.1.1 W3C

W3C (World Wide Web Consortium, `http://www.w3c.org`) är en organisation som arbetar med standarder relaterade till WWW.

- eXtensible Markup Language (XML) är basen för många av andra format och protokoll som använts. XML är en standard för att lagra information i textformat, oftast läsbar även för människor. Formatet är trädbaserat och består av *element* som kan ha *attribut* och innehåll. Ett exempel kan vara ett element `paragraph` med ett attribut `align` som har värdet `left`, innehållet är sedan paragraftexten. Specifikationen säger ingenting om trädets uppbyggnad eller dess semantik, utan är bara regler för hur denna struktur och information skall lagras.

  Vad som gör XML så attraktivt är att man kan skriva generella bibliotek för att läsa och skriva XML-dokument. Programmeraren behöver då inte skriva en ny parser för varje protokoll som används, utan kan återanvända samma XML-parser varje gång. Vidare kan XML-

dokument valideras, man kan alltså på ett enkelt sätt kontrollera att ett dokument är syntaktiskt korrekt. Med hjälp av DTD (Document Type Definition) eller XML-schema kan man även garantera viss semantisk korrekthet. Resultatet är att programmeraren avlastas stora delar av felhanteringen.

- Hyper Text Transfer Protocol (HTTP) är överföringsprotokollet för WWW, och därmed mycket vida spritt. HTTP överför filer från en server efter en förfrågan från en klient. Protokollet är textbaserat, enkelt, tillståndsfritt och stödjer cacheing och autentifiering. Tillståndsfritt (eng. stateless) innebär att protokollet i sig inte har något stöd för tillstånd, det är dock möjligt för klienten själv att hålla reda på tillstånd mellan olika förfrågningar. Tack vare att det har så stor spridning är mjukvarustödet mycket bra och för utvecklare finns det i princip alltid tillgång till ett färdigt bibliotek som tillhandahåller en HTTP-klient.

  Det har även fördelen att det tar sig igenom det mesta, det vill säga brandväggar och liknande.

- XSL, eXtensible Stylesheet Language, är ett språk som definierar en transformation (XSLT) från ett XML-språk till ett annat, exempelvis från DocBook till XHTML. Det används ofta för att presentera samma data på olika sätt för olika användare (HTML, WML, röstuppläsning etc), men är egentligen bara en generell transformation mellan två XML-språk. XSL innefattar fyra delar: ett källdokument (indata), en stilmall (eng. stylesheet) som specificerar hur transformationen ska gå till, en transformator som utför själva transformationen samt ett måldokument. XSL är mallbaserat (eng. template driven), stilmallen beskriver ett mönster, om det mönstret passar källdokumentet så byts den delen av dokumentet ut enligt stilmallen.

  Figur 3.1 är ett exempel, källan är ett fragment DocBook som transformeras till ett HTML-dokument.

3.1.2 Open GIS

Open GIS Consortium är en ickekommersiell samarbetsorganisation för företag som jobbar med GIS-produkter. GIS står för Geographical information system och är ett samlingsnamn för mjukvara som hanterar geografisk data. Syftet med OGC är att förenkla och möjliggöra kommunikation mellan olika GIS-produkter från olika tillverkare. Open GIS Consortium (OGC) har därför producerat ett antal specifikationer som tar upp olika delar av hantering av GIS. Alla specifikationer är offentliga och fria att implementera.

- Geography Markup Language (GML) är ett XML-baserat språk för att

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<sect1>
<title>Exempel</title>
<para>Lorem ipsum...</para>
</sect1>
```

+

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head><title>Sample</title></head>
<body>
<xsl:apply-templates />
</body>
</html>
</xsl:template>
<xsl:template match="sect1/title">
<h1><xsl:apply-templates /></h1>
</xsl:template>
<xsl:template match="sect1/para">
<p><xsl:apply-templates /></p>
</xsl:template>
</xsl:stylesheet>
```

=

```
<html>
<head>
<meta http-equiv="Content-Type"
  content="text/html; charset=UTF-8">
<title>Sample</title>
</head>
<body>
<h1>Exempel</h1>
<p>Lorem ipsum...</p>
</body>
</html>
```

Figur 3.1: XSL-processen

lagra och överföra geografisk information [1]. Det tillhandahåller ett komplext språk för att uttrycka geografiska föremål och förhållanden mellan dessa. Förutom de fördefinierade datatyperna (bland annat punkter, kurvor, polygoner) finns det möjlighet att kombinera dessa till egna datatyper. GML hanterar även olika externa referenser, skalor, längdenheter, koordination, positonssystem, riktningar, temporala data och mycket mer.

GML gör det möjligt att dela data mellan olika mjukvaror från olika tillverkare, så länge båda använder det gemensamma språket GML. I Carteus används det för att undvika att låsa in sig på någon speciell kartserver.

- GML specificerar dock bara hur geografisk information skall representeras, inte hur man ska få ut den önskade informationen från en karttjänst. Web Feature Service (WFS) definierar ett HTTP-baserat förfrågningsprotokoll för en tjänst som tillhandahåller GML [2]. Om man jämför GML med HTML så motsvaras WFS av HTTP. Förutom viss metadata, såsom de olika funktioner tjänsten tillhandahåller samt information om de olika kartorna som finns tillgängliga, så är den viktigaste funktionen att hämta GML-data som beskriver en mängd objekt.

  Notera att GML inte specificerar något om hur informationen skall lagras, oftast lagras informationen i något annat format av effektivitetsskäl (exempelvis en SQL-databas), för att sedan genereras direkt från resultatet av en fråga.

  Urvalet av vilken data som skall returneras kan göras på ett par olika sätt. Direkt i WFS går det att begränsa typen (”alla busshållplatser”) eller antalet (”de första 400”). För mer avancerade urval används ett annat minispråk: filter.

- Filter används för mer avancerad filtrering på svaret från en WFS-förfrågan. Spatiala begränsningar och jämförelsekrav är vanligast. Ett geometriskt objekt (oftast en rektangel, på engelska kallad bounding box) avgränsar det spatiala område som frågan gäller. De filter som används i GML är specificerade i Filter Encoding [3].

  Flera av dessa krav kan sedan kombineras med logiska operationer, det geografiska område som är intressant och andra egenskaper hos de objekten som är intressanta (exempelvis ”alla landsvägar inom Västerbotten” eller ”alla älvar som rinner ut i Bottenhavet”). Figur 3.2 visar ett enkelt exempel på hur WFS och Filters används i Carteus, här anges ett enkelt rektangulärt område från det angivna lagret, och resultatet returneras som ett GML-dokument. Ett exmepel på resultatet från en sådan fråga finns i figur 3.3.

```
<GetFeature
    version="1.0.0"
    service="WFS"
    maxFeatures="10"
    xmlns="http://www.opengis.net/wfs"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/wfs
../wfs/1.0.0/WFS-basic.xsd">
  <Query typeName="A1">
    <ogc:Filter>
      <ogc:BBOX>
        <ogc:PropertyName>_SHAPE_</ogc:PropertyName>
        <gml:Box
    srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
          <gml:coordinates>
    1710448,7455517 1712848,7457917
  </gml:coordinates>
        </gml:Box>
      </ogc:BBOX>
    </ogc:Filter>
  </Query>
</GetFeature>
```

Figur 3.2: Exempel på WFS/GML-förfrågan, A1 är typen av objekt som söks, för att ytterligare begränsa sökningen används ett filter.

- Web Map Service (WMS) är liknande WFS. Till skillnad från WFS så tillhandahåller inte denna tjänst GML-data utan resulterar direkt i en bild, då detta precis som WFS är HTTP-baserat så innebär det att en vanlig webbläsare kan användas för att använda tjänsten, frågan ställs i URL:en och svaret kommer som en bild direkt i webbläsaren.

3.1.3 ESRI

ESRI är ett av dom större företagen inom GIS, de två produkterna nedan är de ESRI-produkter som används i detta system.

- ArcIMS är en serverprogramvara för GIS. Den tillhandahåller tjänster för att publicera GIS-data lokalt och över nätverk. Användaren kan ställa frågor till databasen med hjälp av en programbibliotek eller få

```
<?xml version="1.0" ?>
<wfs:FeatureCollection
    xmlns:wfs="http://www.opengis.net/wfs"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
      "http://www.opengis.net/wfs/1.0.0/WFS-basic.xsd">
  <gml:boundedBy> <gml:Box
      srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:coordinates>
        1478542.375,6639516 1478640.625,6639821
      </gml:coordinates>
  </gml:Box> </gml:boundedBy>
  <gml:featureMember>
    <A1>
      <KOORDX>6639492</KOORDX> <KOORDY>1478707</KOORDY>
      <INTERVALL>1</INTERVALL>
      <KARTID1>66394921478707</KARTID1>
      <KARTID2>110575211</KARTID2> <_ID_>1</_ID_>
      <_SHAPE_>
        <gml:MultiPolygon
srsName="http://www.opengis.net/gml/srs/espg.xml#">
          <gml:polygonMember> <gml:Polygon>
          <gml:outerBoundaryIs> <gml:LinearRing>
          <gml:coordinates decimal="." cs="," ts=" ">
          1478579.125,6639516 1478597.5,6639546
          1478614.625,6639566 1478635.375,6639603
          1478640.625,6639633 1478638.25,6639664
          1478627.5,6639687.5 1478600,6639689.5
          1478577.625,6639672 1478562.375,6639657.5
          1478548.875,6639658.5 1478542.375,6639666.5
          1478561.875,6639699 1478569.25,6639729.5
          1478574.125,6639753.5 1478567.625,6639789.5
          1478537.625,6639821 1478579.125,6639516
          </gml:coordinates> </gml:LinearRing>
          </gml:outerBoundaryIs> </gml:Polygon>
          </gml:polygonMember>
        </gml:MultiPolygon>
      </_SHAPE_>
    </A1>
  </gml:featureMember>
</wfs:FeatureCollection>
```

Figur 3.3: Exempel på WFS/GML-svar, en featureMember för varje objekt, här en polygon.

```
<ARCXML version=''1.1''>
<REQUEST>
  <GET_FEATURES featurelimit=''100'' outputmode=''xml''
geometry=''false'' envelope=''true'' globalenvelope=''true''>
    <LAYER id=''a1'' />
      <SPATIALQUERY>
        <SPATIALFILTER relation=''area_intersection''>
        <ENVELOPE minx=''-10'' miny=''-10''
           maxx=''10'' maxy=''10''/>
      </SPATIALFILTER>
    </SPATIALQUERY>
  </GET_FEATURES>
</REQUEST>
</ARCXML>
```

Figur 3.4: Exempel på en AXL-förfrågning. Användaren frågar efter objekt inom ett givet geometriskt område.

ut kartor direkt i sin webbläsare.

- ArcXML (eller AXL) är ett XML-baserat språk som ArcIMS använder [6]. Det används både för att definiera scheman, kartor och GIS-data samt för att ställa frågor mot dessa, så i viss mån uppfyller det samma funktion som GML.

  I exemplet i figur 3.4 frågar användaren efter objekt från ett lager `a1` med funktionen `GET_FEATURES`. Metadata erhålles genom funktionerna `GET_SERVICE_INFO` och med `GET_IMAGE` returneras en färdig bild i form av en JPEG.

Kapitel 4

Implementationen

4.1 Introduktion

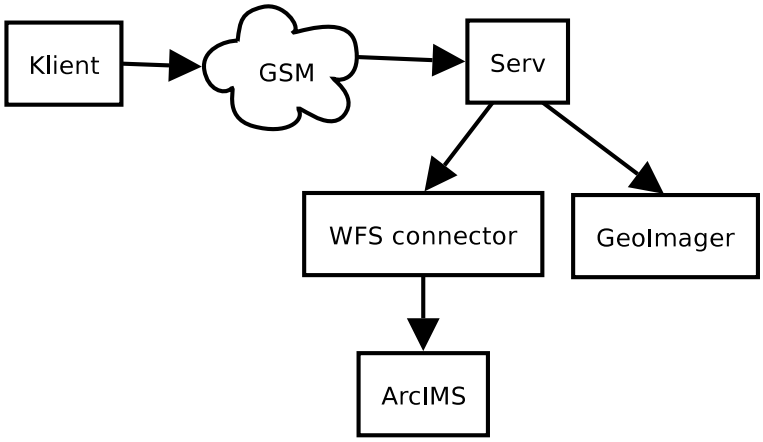
Här följer en beskrivning av det system som har implementerats. Först presenteras den övergripande arkitekturen och dess beståndsdelar, sedan tas överföring och lagring upp. Då detta bara är en prototyp avslutas kapitlet med en beskrivning av möjliga vidareutvecklingar av systemet. En del av dessa är nödvändiga för att kunna använda systemet som en del av en färdig produkt och andra är bara önskvärda funktioner som är möjliga att implementera.

4.2 Arkitektur

Hittills har bara två stora delar nämnts, server och klient. I praktiken så blir det dock lite mer komplicerat, servern består i själva verket av flera program. Nedan följer en beskrivning av de olika delarna och deras funktion, vi börjar med klienten och rör oss mot servern.

4.2.1 Klienten

Klienten är den programvara som användaren kör på sin mobila terminal. Dess uppgift är i första hand att hämta information från en server och visa den för användaren. För att detta ska fungera på bästa möjliga sätt cacheas datan som hämtas från servern. Förutom dessa grundläggande funktioner finns även en del annan funktionalitet som inte tas upp i detta arbete, exempelvis GPS- och licenshantering.



Figur 4.1: De olika komponenterna och hur de kommunicerar

När klienten behöver en del av en karta som inte finns i cachen så skickar den en förfrågan till nästa komponent i ledet; Serv. Förfrågan består av koordinater för området (ett rektangulärt område), vilka lager/kartbaser det gäller samt i vilket format klienten vill ha tillbaka svaret.

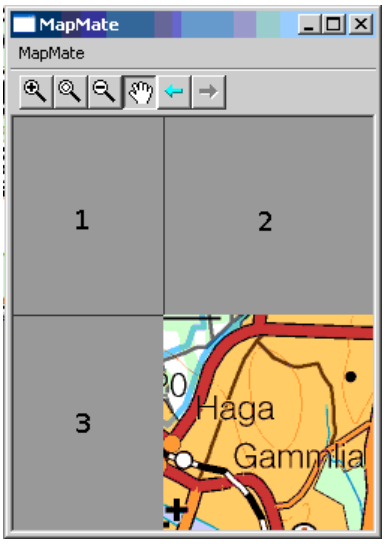
När klienten kommer till ett läge där något saknas i cachen behövs det dock oftast mer än bara en rektangulär yta av en karta. Som figur 4.2 visar krävs det oftast två eller tre rektangulära ytor för att täcka den saknade ytan, och om kartan består av flera lager behövs det eventuellt en fråga per lager. För att spara så mycket tid som möjligt packas alla frågorna ihop till ett paket innan det skickas iväg till servern. Alternativet är att kasta bort den data som redan finns i cachen och ta hem hela det område som besöks.

4.2.2 Servern

Det som fram tills nu har kallats server är alltså i själva verket ett antal olika servrar. Dessa är för närvarande Serv, OGCWFS, ArcIMS och GeoImager.

Serv

Syftet med Serv är att lägga in ett abstraktionslager mellan klienten och specifika karttjänster. Serv är det enda klienten ser av servern, dess huvuduppgift är att agera proxy/tolk mellan klienten och kartservrarna. I varje förfrågan specificerar klienten exakt vad den är ute efter, vilken tjänst som efterfrågas (exempelvis Sverigekartan, sumpmark eller flygfoto). Serv håller en tabell över dessa tjänster och vilken server som tillhandahåller dessa. När en förfrågan tas emot från klienten skickas denna vidare till avsedda



Figur 4.2: Användaren har förflyttat sig nordväst, till ett område som inte finns i cachen (som internt består av tre olika områden)

kartserver och Serv returnerar resultatet till klienten.

Kommunikationen mellan Serv och klienten sker med ett egenkonstruerat HTTP-baserat protokoll. Idén är att det ska vara ett uniformt protokoll för Carteus-systemet, samma protokoll används av en version av Carteus skriven i Java för mobiltelefoner som ett annat examensarbete [4]. Serv har två huvudfunktioner för klienten, dels så kan den ta emot flera kartförfrågningar per förfrågan från klienten, och dels befriar den klienten från att behöva veta hur eller varifrån en karta kommer. Klienten talar bara om vilken tjänst, skala, koordinater, lager och övrig metadata som behövs, sedan får Serv sköta resten.

Serv håller reda på vilka kartservrar som finns tillgängliga, ställer frågorna till rätt server med hjälp av de olika bakstycken som finns tillgängliga (i dagsläget finns det bakstycken för Open GIS samt GeoImager), samlar sedan ihop svaren och ser till att svaren kommer tillbaks i ett enda paket till klienten.

Delar av denna komponent utvecklades gemensamt med Magnus Hall, då han hade behov av liknande funktionalitet i sitt examensarbete.

Connector

Som namnet antyder är AXL specifikt för ArcIMS, och även om ArcIMS fungerar väl så är det dumt och onödigt att göra sig beroende av det. Från

avsnittet om OGC bör det ha framgått att det är önskvärt att göra systemet fritt från beroenden på specifika implementationer. För att slippa kommunicera via AXL med ArcIMS så tillhandahåller ESRI en ”connector”, denna connector är ett fristående program som tillhandahåller en WFS.

Connectorn översätter alla förfrågningar till AXL och skickar vidare till ArcIMS, och översätter svaret tillbaka från AXL till WFS/GML.

GeoImager

GeoImager är serverprotokollet för rasterbaserade kartor i Carteus. Det har sessioner (servern håller reda på tillståndet; position och skala med mera) och bygger på HTTP-protokollet, så det går att använda en vanlig webbläsare för att kommunicera med servern.

4.3 Systembeskrivning

4.3.1 Cacheing

När ny data har hämtats hem från servern ska den cacheas för senare bruk. Den datan kommer antagligen att läsas många fler gånger än den skrivs så det är viktigt att åtkomsten är så snabb som möjlig under så många omständigheter som möjligt. Den ”naiva” lösningen är att dela in kartan i ett rutnät där alla objekt som är inne i en ruta (”cell”) lagras i en fil för sig. När sedan en del av kartan ska användas beräknas vilka av dessa celler som överlappar det sökta området, dess innehåll läses in och unionen av dessa innehåller de sökta objekten. En bieffekt är att det sannolikt kommer med ett antal objekt som innefattas av någon av dessa celler men ej av det sökta området. Antingen gallras de bort redan nu, eller så lämnas det över till uppritningsrutinerna att upptäcka det.

Denna lösning har två uppenbara problem. Dels så är den totala mängden data i kartan okänt, om det är totalt hundra objekt så är det antagligen meningslöst att ha mer än tio rutor, men om vi har en miljon objekt så skulle tio rutor vara tämligen dåligt val. Det andra problemet är att datan kan vara väldigt ojämnt fördelat, till exempel har en karta över fastigheter alla objekten starkt koncentrerade till tätorter. De flesta cellerna kommer att vara helt eller nästan helt tomma, förutom några som kommer att innehålla en stor mängd objekt.

Dessa båda problem resulterar i samma sak, en ineffektiv och ojämn fördelning av antalet objekt i varje cell. Det är alltså önskvärt med något som

anpassar sig efter mängden data och dess fördelning.

Quadtrees

Ett quadtree är en rekursiv datastruktur som delar in rymden i fyra identiska rektanglar (nära släkt med octree [8]). Det använder två tröskelvärden för att balansera mängden data mellan sina löv. Alla noder har ett par koordinater som definierar vilket område noden täcker. För bekvämlighetens skull så är de fyra delarna namngivna Q1, Q2, Q3 respektive Q4, detta kan sedan användas för att ange en sökväg till de olika noderna.

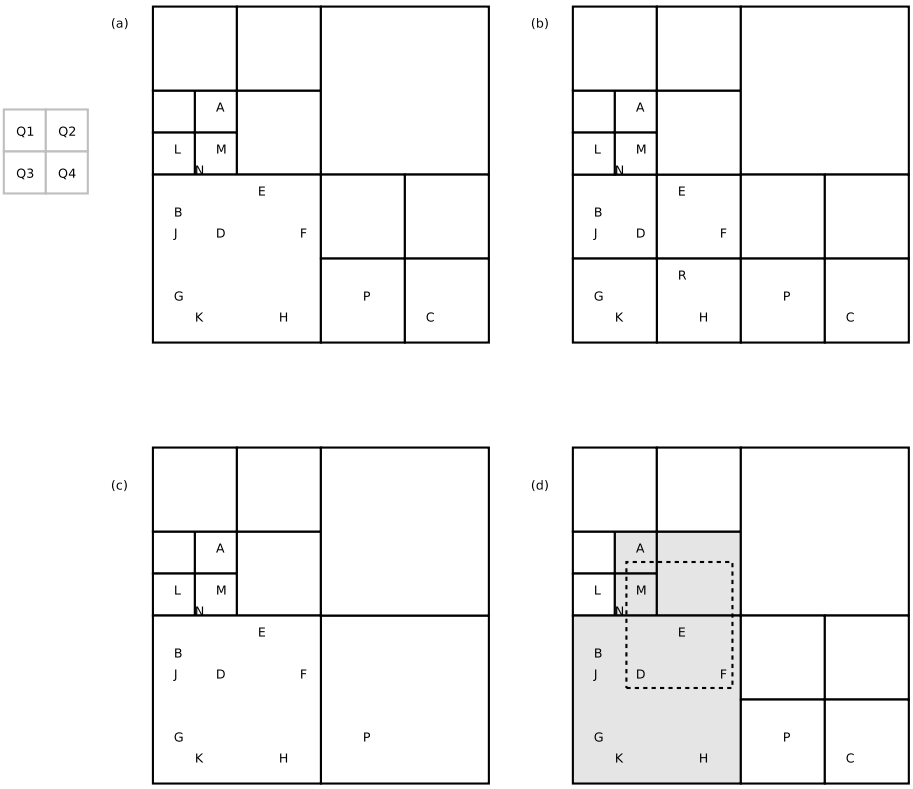
Från början innehåller trädet bara en lövnod, vars koordinater täcker upp hela rymden. När nya objekt läggs till utgår algoritmen från roten. Om den nuvarande noden är ett löv läggs objektet till lövets innehåll, om den nuvarande noden är en intern nod görs ett rekursivt anrop för det barn vars koordinater innefattar det nya objektet. Sökningen kommer alltså att gå ned mot mindre och mindre noder tills dess att de påträffar ett löv, för att där läggas till.

För att undvika ojämn fördelning i trädet så används två tröskelvärden; en övre (N_{max}) och en undre (N_{min}) gräns. Varje gång ett objekt ska läggas till i ett löv görs en kontroll om antalet objekt i lövet överstiger N_{max}. Om så är fallet delas lövet. Lövet görs om till en intern nod, barnens koordinater fås genom att dela det ursprungliga lövets rektangel i fyra lika stora barn. Det gamla innehållet läggs till enligt den vanliga algoritmen fast med den nya interna noden som rot. På detta sätt undviks täta ansamlingar av många objekt i några få noder.

Motsatsen (att undvika en väldig massa tomma noder) uppnås genom ett motsvarande tröskelvärde N_{min}. Om det finns någon nod i trädet som totalt (rekursivt räknat) innehåller färre än N_{min} noder slås dess barn (rekursivt) ihop och ombildas till ett löv.

En sökning utgår från roten, om noden är ett löv så returneras objekten i lövet, om det är en intern nod så görs ett rekursivt anrop på de barnnoder som överlappar med sökområdet.

Figur 4.3 visar några exempel. För att lokalisera ett objekt så anges en sökväg, till exempel så befinner sig objekt A i figuren i lövet [$Q1$ $Q3$ $Q2$]. Antag att exempelträdet har $N_{max} = 8$, om nu ett nytt objekt R läggs till någonstans i området som täcks av [$Q3$] så skulle antalet objekt där komma att överskrida N_{max}, detta löv skulle då dela sig och trädet skulle nu se ut som i (b). Antag att $N_{min} = 1$, på liknande sätt skulle då barnen till $Q4$ slås ihop och lämna $Q4$ som ett löv om objektet C togs bort (c).



Figur 4.3: Exempel på Quadtree. (a) Ett tiotal objekt (b) R har lagts till och [$Q3$] delats upp (c) C har tagits bort och [$Q4$] måste därför slås ihop (d) En sökning (streckad rektangel) returnerar de gråa löven

En sökning returnerar innehållet i de löv som överlappar sökområdet, i figuren representeras sökområdet med den streckade rektangeln, och de mörkare rutorna skulle returneras.

Utöver denna ”standardversion” av quadtree så användes några tillägg i denna tillämpning. För användning av lagring av cache räcker det inte med att veta *vad* som finns lagrat i en nod, utan även *om* det finns lagrat. Detta uppnås genom ett attribut på varje löv som bestämmer om lövet är ”grått” och har ett okänt innehåll, eller om dess innehåll verkligen finns lagrat i trädet.

Detta påverkar en del av funktionerna ovan. Vid åtkomst av trädet returneras gråa löv som tomma ytor, dock så skickar trädet iväg en förfrågan till en hjälprutin om att detta område behövs. Hjälprutinen ser sedan till att fylla på trädet när den fått svar från servern. Ihopslagning av noder (efter kontroll mot N_{min}) kan inte ske ifall någon av underliggande löv skulle vara gråmarkerad, då skulle gråmarkeringen gå förlorad och eventuella objekt i

detta område aldrig upptäckas.

Det finns två problem med tekniken ovan. När objekt läggs till trädet så är det inte alls säkert att det som i beskrivningen ovan överlappar med precis en av noderna, utan objektet kan ligga på gränsen mellan två noder (eller helt innesluta en eller fler noder), ju större objekt desto större chans för detta. Antingen kan objektet delas upp i två (eller fler) mindre objekt eller så kan objektet läggas till i samtliga noder det överlappar. Då det första alternativet snabbt blir komplext (speciellt då det rör sig om ihåliga polygoner) har den senare lösningen valts. Nackdelen med detta är att samma objekt kommer att återfinnas flera gånger när man söker i trädet. Antingen kan problemet ignoreras (och objektet kommer alltså att ritas ut flera gånger), eller så kan resultatet från en sökning filtreras för att ta bort alla eventuella dubbletter.

Problemet uppstår antingen när det i samma lager finns objekt med mycket stor skillnad i spatial storlek, så ett antal mindre objekt "tvingar" fram en delning av en nod då antalet objekt nått N_{max}. Andra möjligheten är när många objekt överlappar varandra, och på så sätt också tvingar fram en uppdelning av noder, på grund av denna överlappning så kommer en uppdelning inte att leda till någon förbättring av fördelningen av objekten, så den korrekta lösningen på detta problem är att öka N_{max} (detta är en av de få specialfall då trädet behöver justeras manuellt för att bibehålla prestanda). Då båda dessa fall är ett relativt sällsynta problem görs i nuläget ingen filtrering (olika typer av objekt delas upp i olika lagar och objekt av samma typ har oftast liknande storlek).

Den andra fallgropen är när en nod delar på sig, det behövs någon kontroll för att förhindra oändlig uppdelning. Detta kan uppstå om fler än N_{max} objekt är placerade på samma punkt eller om de helt innesluter en nod, då kommer inget antal uppdelningar av trädet resultera i ett löv med mindre än N_{max} objekt.

I implementationen så lagras trädet direkt i RAM-minnet, medan själva objekten av utrymmesskäl lagras på sekundärminne (Secure Digital eller Compact Flash). Det innebär att sökningen i trädet kan ske helt i RAM-minnet, men för att returnera objekten måste de läsas in från sekundärminnet.

4.3.2 Klienten

Klienten är ett bibliotek som ger Carteus extra funktionalitet, och är alltså precis som Carteus skrivet i C++. Utvecklingsmiljön har varit MSVC++ samt Embedded C++. Nedan följer en beskrivning av de viktigaste klasserna och hur de hänger ihop. Händelseschemat ger en uppfattning om vad som

händer när användaren navigerar runt i en karta. Listan är givetvis inte komplett, utan visar bara de viktigaste punkterna.

CImageBuffer tillhör kärnan i Carteus, det är en buffer dit olika metoder kan rita saker. När allt ritande är klart meddelas Carteus detta med en `notifyImageObservers` och innehållet i buffern kopieras till skärmen.

När användaren förflyttar sig på kartan anropas `getImageFromServer` och det är den som sätter igång kedja av händelser för att rita upp den nya vyn på skärmen, med anropet kommer en parameter med de nya koordinaterna.

VMaker är huvudklassen för vektorgrafiken i Carteus, och är vad som syns utåt i Carteus. Den har medlemsvariabler för de olika lagren, quadtrees, och ett objekt för serveråtkomst. VMaker har även en kö för de objekt som står på tur för att ritas ut, och det är genom denna kö quadtree kan returnera objekt efter en sökning i trädet.

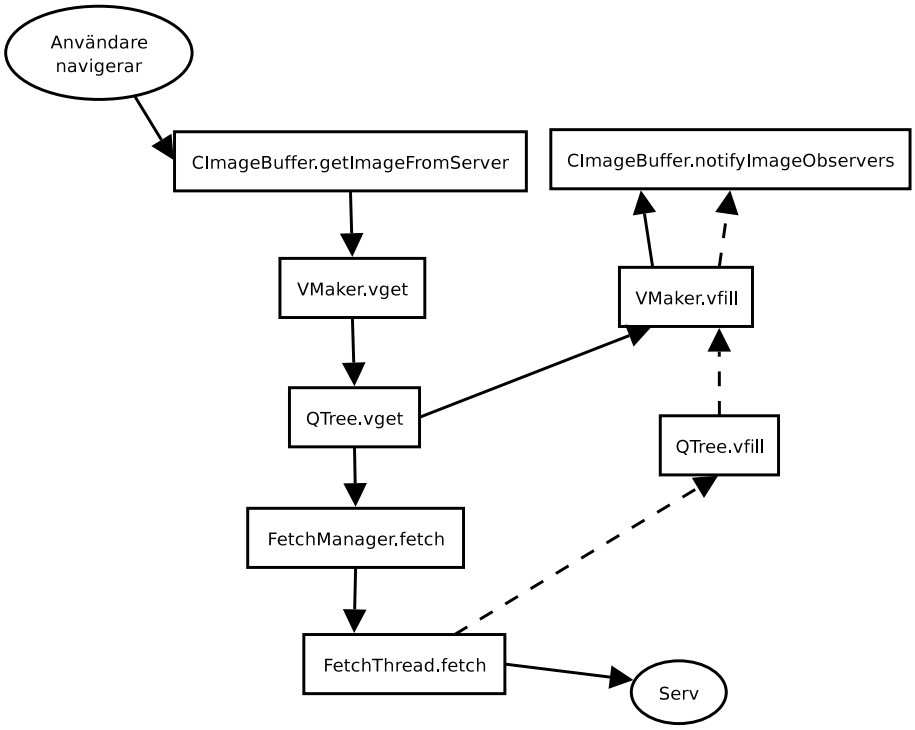
Utåt sett innehåller den funktioner för att hämta vektorgrafik (`vget`) och fylla på kön (`vfill`). `vfill` ser även till att objekten blir utritade på skärmen.

GMLParser Avkodar GML-data och skapar den interna representationen. Denna parser är dock långt från komplett, men tar hand om enklare punkter och polygoner.

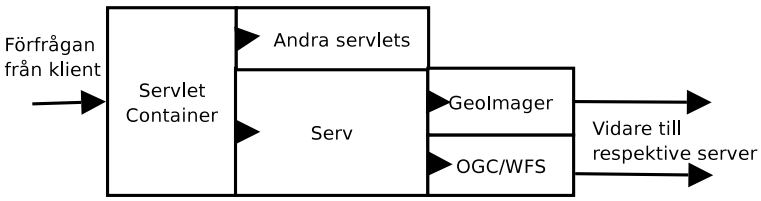
QTree är implementationen av datatypen quadtree. Trädet får anrop från Vmaker via metoden `vget`, som startar en sökning i trädet. De objekt som redan finns i trädet returneras omedelbart genom att läggas i kön i VMaker. För varje "grått" löv (som alltså inte finns lagrat i cachen) skickas ett jobb till FetchManager. När FetchManager har hämtat den efterfrågade informationen från servern meddelar den trädet som dels sparar informationen men också direkt skickar ett meddelande tillbaks till Vmaker om den nyinkomna informationen.

Uppritning på skärmen kan alltså ske vid två tillfällen, först direkt från innehållet i trädet och sedan en andra omgång när ny data har hämtat hem från servern och lagts in i trädet. Givetvis kan användaren hinna navigera runt ytterligare gånger medan en överföring från servern pågår, så det andra steget kan upprepas flera gånger.

FetchManager överser kontakten med servern. Då en del av dess operationer kan ta ansenlig tid (flera sekunder) så kör delar av FetchManager i en egen tråd. När det kommer en förfrågningar från trädet läggs dessa på en kö. En annan tråd samlar sedan ihop dessa förfrågningar och skickar iväg en förfrågan till servern. När resultatet är färdigt lägger FetchManager in de nya objekten i trädet.



Figur 4.4: Anropsstrukturen. De streckade linjerna indikerar anrop från tråden som har kontakt med Serv och kan ska med några sekunders fördröjning.



Figur 4.5: Uppbyggnaden av Serv

4.3.3 Serv

Serv är implementerad i Java och för att på bästa sätt tillhandahålla HTTP-gränssnittet körs Serv som en Java-servlet. En servlet är en speciell typ av Java-klass som tillhandahåller web-sidor enligt ett givet API. Ett ramverk som bland annat tar hand om HTTP tillhandahålls av en Java servlet container, som alltså innesluter en servlet. Den implementations som används i detta fall heter Tomcat som är en del av Apache-projektet.

Då WFS/GML inte är något kompakt språk och bandbredden är liten använder Serv en XSL-transformation för att transformera svarsdokumentet (som är i GML-format) till en kompaktare version. Transformationen plockar ut det intressanta (exempelvis typ av objekt samt koordinater) och lägger till en del information rörande förfrågan (en identifierare så att klienten kan associera svaret med den fråga som det hör till) innan svaret skickas vidare till klienten.

4.3.4 Connector

Connectorn ESRI tillhandahåller är ett fristående system och behöver alltså inte köras på samma dator som ArcIMS. Precis som Serv är connectorn implementerad som en Java-servlet, så även den behöver en Java servlet container för att fungera. Precis som Serv har Tomcat använts till detta.

I skrivandets stund så är denna produkt inte helt färdig, vissa delar av WFS (såsom transaktioner och uppdateringar) är ej implementerade, som tur var så var allt som krävdes för detta arbete färdigt (om än inte helt problemfritt).

Kapitel 5

Avslutning

5.1 Begränsningar

5.1.1 Prestanda

För närvarande finns det två områden som med enkla medel kraftigt kan optimeras. Att detta inte redan är gjort beror på tidsbrist.

Som nämnts så lagras själva datan i sekundärminnet i ett antal olika filer (en för varje löv i trädet). I nuvarande implementations så öppnas och stängs denna fil för varje åtkomst till trädet, vilket leder till ett stort antal onödiga filoperationer.

Den andra begränsningen är relaterad till den första. Ofta så kommer förflyttningar/zoomoperationer att innebära små förändrar i vilken data som visas, så genom att cachea det senaste resultatet i minnet skulle åtkomsten till det långsammare sekundärminnet minska kraftigt.

5.1.2 Vidareutveckling

Resultatet är endast en prototyp så här följer några av de möjliga eller nödvändiga förbättringar innan det blir färdigt för slutanvändaren.

Lagerhantering

För att vektorgrafiken skall vara praktiskt användbar måste användaren själv kunna definiera upp sina kartor, vilka lager som ska visas (och i vilken ordning), dess manér och inom vilka skalintervall lagret ska användas. Det-

ta gäller inte bara vektorkartor utan generellt för Carteus, så detta ligger egentligen utanför detta arbete. I nuläget anges vektorlagren vid kompilering som argument till konstruktorn för klassen som sköter lagerhanteringen.

Uppdatering

Viss vektordata går enkelt för användaren själv att samla in och konstruera, speciellt med hjälp av en GPS-modul, så det skulle vara önskvärt med sådan funktionalitet. Användaren skulle kunna mata in och uppdatera enkel data (punkter och stigar) som sparas på kartservern permanent. För närvarande pågår det ett examensarbete som har som mål att lägga till denna funktionalitet.

5.2 Tack

Tack till mina två handledare Jerry Eriksson och Peter Jacobson, samt alla på Visualiseringscentrum.

5.3 Referenser

[1] Simon Cox et. al: Geography Markup Language, 2003,
`http://www.opengis.org/docs/02-023r4.pdf` (2004-05-21)

[2] Panagiotis A. Vretanos: Web Feature Service, 2002,
`http://www.opengis.org/docs/02-058.pdf` (2004-05-21)

[3] Panagiotis A. Vretanos: Filter Encoding, 2001,
`http://www.opengis.org/docs/02-059.pdf` (2004-05-21)

[4] Hall, Magnus: Location based map system for mobile devices
(UMNAD 475/03), 2003

[5] Lindblom, Jonas, Körning, Anders: Kartklient för mobila plattformar
(UMNAD 425/02), 2002

[6] ESRI: AXL Programmers reference guide, 2002,
`http://downloads.esri.com/support/documentation/ims_/`
`ArcXML_Guide/Support_files/arcxmlguide.htm` (2004-05-21)

[7] Lantmäteriet: Swedish national map projections for RT 90,
`http://www.lm.se/geodesi/refsys/rt/rt_projections.htm`
(2004-05-21)

[8] Foley et. al.: Computer graphics, 2002, s. 550