Redigering av punktdata i webbkarta med WFS-teknik
– Implementation och användartest

Therése Karlsson

Avdelning för Fastighetsvetenskap
Lunds Tekniska Högskola

ISRN/LUTVDG/TVLM 06/5136

Redigering av punktdata i webbkarta med WFS-teknik

– Implementation och användartest

Therése Karlsson

Avdelning för Fastighetsvetenskap
Lunds Tekniska Högskola

Titel:
Redigering av punktdata i webbkarta med WFS-teknik – Implementation och användartest

Engelsk titel:
Editing point data in a web map using WFS – Implementation and usability test

Författare:
Therése Karlsson, Civilingenjörsutbildningen i Lantmäteri, Lunds Tekniska Högskola

Handledare:
Lars Harrie, GIS-centrum, Lunds universitet
Jonas Andréasson, Lantmäteriavdelningen, Lunds kommun

Examinator:
Klas Ernald Borges, Avdelning för Fastighetsvetenskap, Lunds Tekniska Högskola

Opponent:
Lina Johansson, Civilingenjörsutbildningen i Lantmäteri, Lunds Tekniska Högskola

Sökord:
Karttjänst, WFS, Java, Servlet, ArcSDE, Redigering, Användargränssnitt

Keywords:
Map service, WFS, Java, Servlet, ArcSDE, Editing, User Interface

Förord

Examensarbetet på 20 poäng är den sista delen av Lantmäteriutbildning på LTH. Arbetet har utförts på Lunds kommuns Lantmäteriavdelning under hösten 2006 (augusti till december). Handledare på kommunen var Jonas Andréasson och handledare från LTH var Lars Harrie, GIS-centrum. Klas Ernald Borges, Avdelning för Fastighetsvetenskap, var examinator. Opponent på examensarbetet var Lina Johansson, student på Lantmäteriutbildningen vid LTH.

Jag riktar ett stort tack till mina handledare, min examinator samt min opponent. Även till alla arbetskamrater på Lunds kommun riktas ett tack för att jag har fått utföra mitt examensarbete på en så trevlig arbetsplats. Där vill jag särskilt tacka Anna-Stina Munsin och Tomas Åkerholm. Dessutom vill jag tacka Anders Andersson och Per Selin på ESRI support som hjälpt mig med de problem jag haft för att få Java API att fungera tillsammans med ArcSDE. Sist vill jag även tacka Christina Nilsson på *Gatu och trafikkontoret* på Lunds kommun som ställt upp som testperson.

Therése Karlsson

Lund, december 2006

Sammanfattning

Examensarbetet har utförts på Lantmäteriavdelningen på Lunds kommun. Syftet var att utöka kommunens interna karttjänst med funktionalitet för redigering av punktlager. Anledningen till att denna funktion efterfrågades var för att på ett enkelt sätt göra det möjligt för varje förvaltning att själva underhålla sina data. Tidigare sköttes allt underhåll av data av Lantmäteriavdelningen. Då en annan förvaltning upptäckte att redigeringar behövde göras i deras data fick de kontakta Lantmäteriavdelningen som sedan fick utföra redigeringen. Eftersom de tänkta användarna av den nya redigeringsfunktionen till stor grad saknar tidigare GIS-erfarenhet krävs ett enkelt och tydligt användargränssnitt.

Kommunens karttjänst är byggd med programmet ArcIMS och för lagring av data används ArcSDE. Eftersom ArcIMS saknar funktionalitet för redigering av data måste direkt kommunikation med ArcSDE ske för att kunna redigera data. För att göra detta har en servlet som använder ESRI:s Java API implementerats. Servleten tar emot WFS-anrop av typerna *GetCapabilities*, *DescribeFeatureType* samt *Transaction* och returnerar svar i något XML-format enligt standarden för WFS 1.1.

Anrop till servleten görs från en klient skriven i ASP.NET och C#. Redigering av data kan delas in i tre olika operationer där varje operation kräver olika typ av information från användaren. Därför har tre olika formulär skapats. Det finns ett formulär för insättning, ett för uppdatering och ett för borttagning av punkter. JavaScript har använts för att passa in dessa formulär i karttjänsten så att användaren exempelvis kan välja vilken punkt som skall tas bort genom att klicka på den.

För att få en uppfattning om hur pass användarvänliga de nya funktionerna är för personer ur målgruppen gjordes ett användartest. Endast en testperson användes: en trafikmiljörådgivare från *Gatu och trafikkontoret* på Lunds kommun. Hon hade ingen tidigare erfarenhet av GIS men hade använt kommunens interna karttjänst för att titta på fastigheter samt mäta avstånd. Testet gick till så att hon fick sätta in, uppdatera och slutligen ta bort en parkeringsplats. Under testet gavs viss ledning och en diskussion om formulärens utformning fördes.

Syftet med examensarbetet har uppfyllts i och med att de önskade funktionerna för redigering av punktdata har implementerats. En begränsning med lösningen är att för att data i ett punktlager skall kunna redigeras krävs att tabellen inte är versionshanterad. För att kunna redigera data via programmet ArcMap, som används av kommunen idag, krävs dock att tabellen är versionshanterad. Detta medför att man får välja om ett lager skall kunna redigeras via ArcMap eller med hjälp av den nya redigeringsfunktionen i karttjänsten.

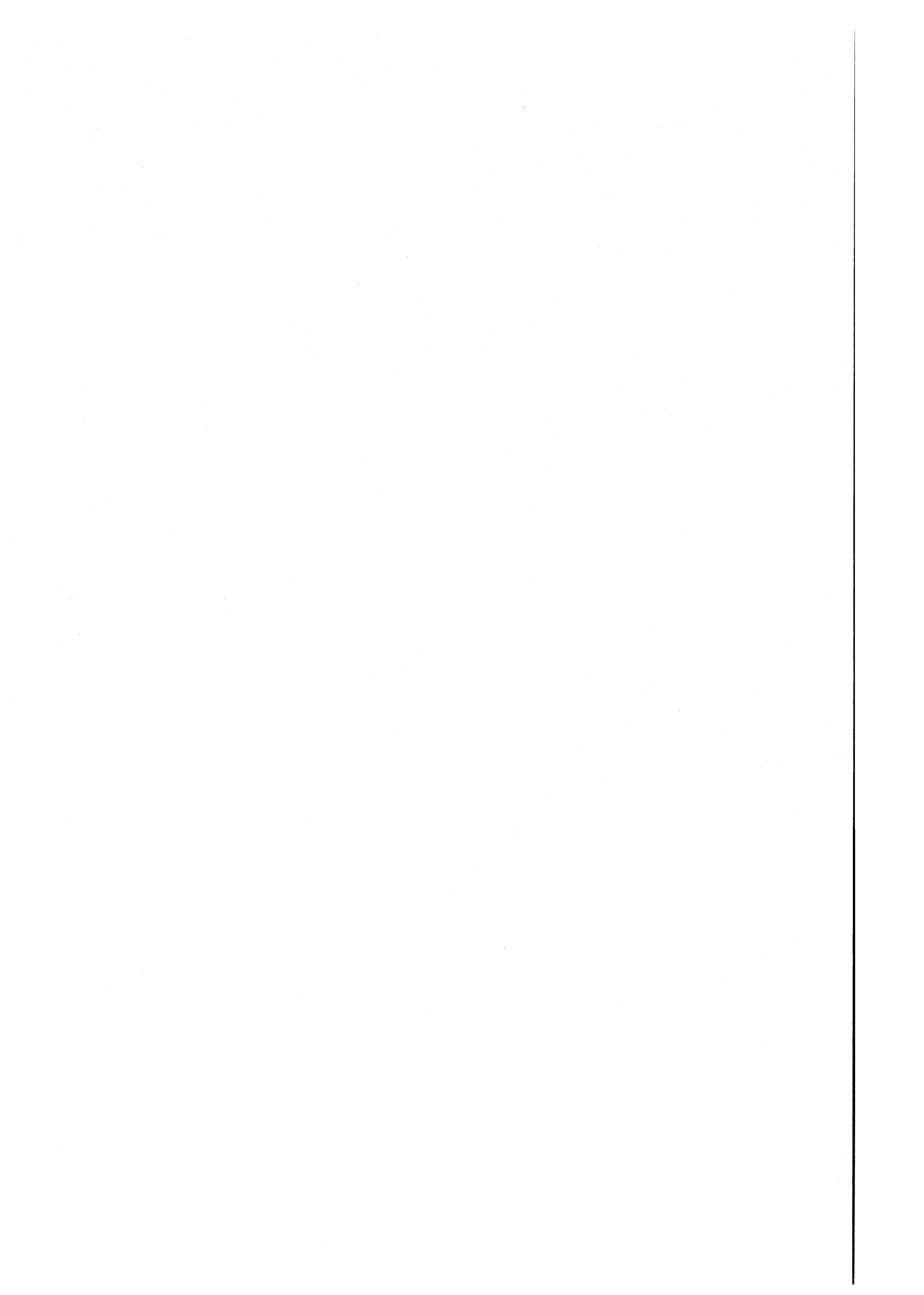
Abstract

This Master Thesis has been performed at the department of surveying at Lunds kommun. The aim was to increase the functionality of the internal map service of the municipality with functionality for editing point layers. The reason why this functionality was requested was to in an easy way make it possible for every department in the municipality to maintain their own data. Earlier all data maintenance was performed by people at the department of surveying. Many of the new users don't have any experience of GIS, therefore it is important to make a simple and clear user interface.

The map service is built with the program ArcIMS, and data are stored with ArcSDE. There is no functionality for editing data with ArcIMS, therefore direct communication with ArcSDE is needed to be able to edit data. To do this a servlet which uses ESRI Java API is implemented. WFS-calls are sent to the servlet and answers in XML-format are returned. The types of calls that can be handled are *GetCapabilities*, *DescribeFeatureType* and *Transaction*. The calls to the servlet are made from a client written in ASP.NET and C#. Editing data can be divided into three different operations: insert, update and delete. One form for each type of operation is made. JavaScript have been used to integrate these forms with the map service and make it possible for example to choose a point to delete by clicking it in the map.

A smaller usability test was carried out to get an idea of how easy the new functions are to use for a person from the target group. Only one person was used for the study. She had no earlier experience of GIS, but had used the internal map service to look at properties and to measure distances. During the study she first inserted a new car park, then updated it and finally deleted it. A discussion about the user interface and the functionality was held during the test.

The aim of the Master Thesis has been reached. The functions have been implemented and are working as planned. There is however one limitation of the solution: to be able to edit data in a layer the table can't be versioned. But to be able to edit data from the ESRI program ArcMap the table has to be versioned. This means you have to decide in which way data in a table shall be edited: from ArcMap or from the new functions in the internal map service.



Innehållsförteckning

1 Inledning ... 1

1.1 Bakgrund ... 1
1.2 Syfte ... 1
1.3 Metod ... 1
1.4 Avgränsningar ... 1
1.5 Disposition ... 2

2 GIS – Geografiska InformationsSystem ... 5

2.1 Desktop-GIS ... 7
2.2 Distribuerat GIS ... 7

3 Standardiserade märkspråk ... 9

3.1 XML – eXtensible Markup Language ... 10
3.1.1 DTD – Document Type Definition ... 11
3.1.2 XML Schema ... 13
3.1.3 XML Namespace ... 13
3.2 GML – Geography Markup Language ... 14
3.3 SVG – Scalable Vector Graphics ... 15

4 Standardiserade karttjänster ... 19

4.1 WMS – Web Map Service ... 19
4.1.1 GetCapabilities ... 20
4.1.2 GetMap ... 21
4.1.3 GetFeatureInfo ... 23
4.2 WFS – Web Feature Service ... 24
4.2.1 GetCapabilities ... 25
4.2.2 DescribeFeatureType ... 26
4.2.3 GetFeature ... 26
4.2.4 GetGmlObject ... 26
4.2.5 LockFeature ... 27
4.2.6 Transaction ... 27

5 Webbteknologi ... 31

5.1 HTML ... 31
5.2 Programmering ... 32
5.2.1 Java ... 32
5.2.2 C# ... 33
5.3 Dynamik på serversidan ... 34
5.3.1 Servlets ... 34
5.4 Dynamik på klientsidan ... 36
5.4.1 JavaScript ... 36

6 Databaser ... 39

6.1 Normalformer ... 41
6.1.1 Första normalformen ... 42
6.1.2 Andra normalformen ... 42

6.2 SQL – Structured Query Language 43
6.2.1 Skapa tabell 43
6.2.2 Sätta in ny post 43
6.2.3 Uppdatera post 44
6.2.4 Ta bort post 44
6.2.5 Uppdatera tabell 44
6.2.6 Ta bort tabell 44
6.2.7 Visa information 45

7 Programvaror – ESRI 47

7.1 ArcSDE 47
7.1.1 Java API till ArcSDE 48
7.2 ArcIMS 48

8 Intrakartan – Lunds kommuns interna karttjänst 53

8.1 Funktionalitet 53
8.1.1 Intrakartan Bas 53
8.1.2 Intrakartan Plus 54
8.2 Teknik 55
8.3 Användningsområden 56

9 Användaraspekter av klienten 61

9.1 Insättning av punkt 63
9.2 Uppdatering av punkt 63
9.3 Borttagning av punkt 65
9.4 Användartest 65

10 Tekniska aspekter – Planerad teknik 69

11 Tekniska aspekter – Använd teknik 71

11.1 Klienten 72
11.1.1 Insättning 72
11.1.2 Uppdatering 74
11.1.3 Borttagning 76
11.1.4 Integrering i karttjänsten 76
11.2 Servleten 79
11.2.1 doGetCapabilities() 79
11.2.2 doDescribeFeatureType() 80
11.2.3 doTransaction() 80

12 Diskussion 83

12.1 Kommentarer om tekniken 84
12.2 Kommentarer om användartestet 86

13 Slutsatser 87

Referenser 89

Bilaga 1 – Ordlista 93

Bilaga 2 – Specifikation av servleten 95

1 Inledning

1.1 Bakgrund

Lunds kommun har en intern karttjänst som kallas *Intrakartan.* Tillgång till tjänsten ges via kommunens Intranät. Karttjänsten är skapad med hjälp av programmet ArcIMS, därefter har tjänsten manuellt utökats med diverse funktionaliteter speciellt anpassade för kommunens ändamål. Gemensamt för all funktionalitet är dock att de inte ändrar någon information i databasen.

Den karta som visas i *Intrakartan* skall hållas uppdaterad. Olika förvaltningar inom kommunen är ansvariga för olika delar av de data som lagras. I dagsläget är det enbart ett fåtal personer på Lantmäteriavdelningen som har möjlighet att redigera data eftersom särskilda GIS-program används till detta ändamål. Detta medför att dessa personer måste sköta all redigering av data, oavsett vilken förvaltning som är ansvarig för förändringarna. Genom att utöka *Intrakartan* med funktionalitet för redigering av data skulle varje förvaltning själv kunna utföra redigeringar vid behov. På så sätt skulle konceptet med registrering vid källan uppfyllas, dvs. att informationsägaren själv underhåller sina data.

1.2 Syfte

Examensarbetets syfte är att utöka den karttjänst som finns på Lunds kommuns Intranät, *Intrakartan,* med funktionalitet för redigering av geografiska punktobjekt med hjälp av WFS-teknik.

1.3 Metod

Arbetet består dels av en litteraturstudie och dels av praktiskt arbete. Litteraturstudiens syfte är att sätta sig in i teorin för att kunna skriva den teoretiska delen av rapporten samt för att lättare kunna utföra det praktiska arbetet. En stor del av litteraturstudien består av att studera specifikationerna för standarderna WMS och WFS, men även böcker och Internet används.

Det praktiska arbetet inleds med att söka information om vad som redan gjorts med WFS-teknik samt vilket stöd som finns för användande av WFS i den programvara som kommunen använder. Detta för att kunna bestämma vilken teknik som ska användas för att utöka karttjänsten med funktionalitet för redigering. En klient i ASP.NET och C# implementeras sedan. Klienten kommunicerar med hjälp av WFS med en servlet som i sin tur använder ESRI:s Java API för kommunikation med ArcSDE, där de geografiska objekten lagras.

Den sista delen av det praktiska arbetet består av ett mindre användartest. Endast en testperson används. Syftet med studien är att få en bild av användargränsnittets användarvänlighet för personer ur målgruppen. För att få en bättre bild borde givetvis flera testpersoner ha användas, men tid till detta saknas och endast en mindre studie utförs.

1.4 Avgränsningar

För att avgränsa arbetet bestämdes att endast den del av WFS-standarden som är nödvändig för att kunna utföra redigering av data skulle implementeras. För att utföra själva redigeringen implementeras operationen *Transaction.* För att klienten ska kunna få information om vilka tabeller som kan redigeras samt vilka kolumner varje tabell innehåller implementeras även

operationerna *GetCapabilities* och *DescribeFeatureType*. Den obligatoriska operationen *GetFeature* implementeras dock inte då denna inte anses behövas för att kunna utföra redigering. Detta innebär att WFS-standarden inte uppfylls, men denna avgränsning är nödvändig för att inte arbetet skall bli allt för omfattande. Ytterligare en avgränsning är att i första hand enbart punktlager ska kunna redigeras. En avgränsning gällande användartestet är att endast en testperson används. Även detta för att inte arbetet skall bli allt för omfattande.

1.5 Disposition

Efter rapportens inledning (kapitel 1) följer en teoretisk del (kapitel 2-8). Kapitel 2 ger en kortare beskrivning av geografiska informationssystem (GIS) medan kapitel 3 tar upp standardiserade märkspråk. Kapitel 4 bygger vidare på standarder och beskriver två typer av standardiserade karttjänster. I kapitel 5 gås olika sätt att tillföra dynamik till en webbtjänst igenom. Kapitel 6 tar upp databaser medan kapitel 7 beskriver två programvaror från ESRI: ArcIMS och ArcSDE. Den teoretiska delen avslutas sedan med kapitel 8 där Lunds kommuns interna kartjänst, *Intrakartan*, beskrivs.

Efter den teoretiska delen följer en del som beskriver det praktiska arbetet. Den delen inleds med kapitel 9 som beskriver användaraspekterna av klienten. De följande två kapitlen beskriver de tekniska aspekterna, kapitel 10 tar upp den teknik som först var tänkt att användas samt varför inte denna teknik fungerade medan kapitel 11 beskriver den teknik som istället har använts. Rapporten avslutas med diskussion, slutsatser samt referenser.

I bilaga 1 finns en ordlista där de flesta förkortningar finns med, samt ytterligare några begrepp. Första gången dessa ord förekommer i rapporten har de markerats med fet stil. Bilaga 2 innehåller en specifikation av den servlet som skrivits som en del av det praktiska arbetet.

Teoridel

2 GIS – Geografiska InformationsSystem

GIS är en förkortning för Geografiska InformationsSystem och kan förklaras som ett datoriserat informationssystem som används för att lägga in, hantera, lagra, analysera samt presentera geografiska data [1]. Att de data som systemet innehåller är geografiska innebär att alla objekt i systemet är knutna till minst en geografisk punkt. Att ett system innehåller en karta medför alltså inte automatiskt att det är ett GIS [2]. För att knyta ett objekt till en geografisk punkt, dvs. en punkt på jordens yta, används koordinater. För alla objekt i systemet skall koordinater anges i samma referenssystem och samma projektion skall användas.

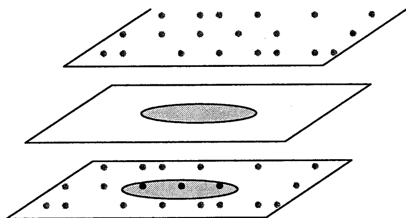
För att lagra geografisk data delas informationen upp i lager. Det finns tre olika typer av data som kan lagras i ett GIS. Dessa är kartdata, attributdata samt bilddata [3]. Ett kartlager består antingen av bilddata eller av kartdata, med eventuell attributdata kopplad till sig. Bilddata är i rasterformat och kan t.ex. vara ett flygfoto eller en inskannad karta. Kartdata är istället vektorformat och innehåller information om geografiska objekts position och utbredning. Attributdata innehåller information om icke-geografiska egenskaper som knyts till geografiska objekt och därmed får en geografisk position. De geografiska objekten lagras som punkter, linjer eller ytor (polygoner) och alla objekt inom ett lager måste vara av samma typ. Dessutom har alla objekt inom ett lager samma typ av attribut kopplade till sig. Detta medför att varje lager beskriver en viss typ av geografiska objekt med gemensamma egenskaper, exempelvis vägar. Attributdata som skulle kunna kopplas till väg-objekt är t.ex. vilken hastighet som är tillåten på vägen.

Eftersom världen ständigt förändras och aktuell information eftersträvas bör det vara enkelt att lägga till, uppdatera och ta bort information. Dessutom är det en fördel om det är lätt att söka data, utifrån såväl geografiska som icke-geografiska egenskaper. Data lagras därför vanligen i databaser. För att hantera attributdata används sedan databashanterare som antingen kan vara en del av det övriga systemet eller ett externt program [2].

Systemet bör även kunna hantera metadata. Metadata beskrivs ofta som ”data om data” vilket innebär att det är information om de data som används. Informationen behövs för att använda data [4]. Exempelvis kan metadata innehålla definitioner av namn och datatyper, en datamodell samt information om hur data har samlats in och bearbetats. Att använda rumslig metadata är viktigt eftersom det underlättar hanteringen, och kan minska mängden, av rumslig data [4].

Analyser av flera slag kan utföras på både geografiska och icke-geografiska data med hjälp av GIS. Även i andra typer av system kan sökningar på icke-geografiska data göras (se avsnitt 6.4), men det är de geografiska sökningarna och analyserna som är speciellt för GIS. Hur avancerade analyser som är möjliga beror på de kartdata som finns. Om exempelvis alla vägar är sammanbundna till ett nätverk kan nätverksanalyser utföras för att hitta kortaste väg från en punkt till en annan. Om det finns attributdata som anger tillåten hastighet för varje vägsegment kan även snabbaste väg beräknas. En annan typ av analys är buffertzoner. Med hjälp av buffertzoner och överlagring är det möjligt att exempelvis beräkna hur många människor som bor inom ett visst avstånd från ett givet objekt. Detta kan vara användbart för att t.ex. beräkna hur många människor som har närmare än 2 kilometer till ett planerat köpcentrum, se exempel i figur 2.1. Den buffertzon som skulle användas i det fallet är en cirkulär yta med centrum i köpcentrumet och radien 2 kilometer. Sedan överlagras buffertzonen med ett kartlager över befolkning för att välja ut den del av befolkningen som

bor inom området. Andra typer av analyser kan syfta till att hitta olika topologiska samband eller beräkna längder och arealer med mera [5].



Figur 2.1. Överlagring mellan ett punktlager och ett lager med en buffertzon. Det översta lagret symboliserar befolkning. I mitten visas ett lager som innehåller en buffertzon på 2 km runt ett planerat köpcentrum. Längst ned har lagren överlagrats för att visa befolkning inom buffertzonen. Den sökta befolkningen har markerats med en mörkare färg.

Det är viktigt att resultatet av analyser kan presenteras på ett bra sätt. För att göra presentationen tydlig kan både kartor, diagram och tabeller användas. Diagram och tabeller kan antingen placeras på kartan eller visas separat. Det är även en fördel om de tabelldata som analysen genererade enkelt kan exporteras till andra system för vidare analyser och presentation [2]. I och med att informationen lagras i kartlager är det lätt att anpassa kartans utseende samt vilken information som visas utifrån användarens önskemål. Detta är en stor fördel gentemot papperskartor. Färger och symboler i kartan bör väljas så att betraktaren intuitivt förstår vad som visas, men kartan bör ändå ha en legend.

Beroende på användarens kunskaper och vad användaren vill använda systemet till behövs olika avancerade system, tre olika nivåer brukar nämnas [2]. Den lägsta nivån är ett system som endast klarar av enklare uppgifter, för de användare som inte har någon eller endast liten kunskap av hur ett GIS fungerar och hur data lagras. Anledningen till att systemet då inte skall innehålla för många och för avancerade funktioner är för att inte förvirra användaren med funktioner som ändå inte skulle användas. Den högst nivån är istället ett öppet system där användaren själv får programmera. Ett sådant system kan behövas för att göra riktigt avancerade analyser och operationer. Mellantinget är dock vanligare. Ett system som kan utföra de flesta operationer som efterfrågas men som kräver att användaren har en viss kunskap av hur data lagras och hur operationerna fungerar.

Förutom att dela in GIS efter hur avancerade systemen är och vilken typ av användare de riktar sig till kan GIS delas in i tre olika teknologier. Dessa kallas stordator- (*mainframe*), desktop- respektive distribuerat GIS [6]. Utvecklingen inom GIS-teknologier har följt utvecklingen av datorer. Stordator-GIS var därför den fösta tekniken av GIS. GIS-programmen låg då på en stordator och analyser och åtkomst av data skedde via terminaler inom det lokala nätverket (LAN) [6]. Då utvecklingen inom datorindustrin gick framåt kom först desktop- och sedan distribuerat GIS. Det är dessa tekniker som är vanliga idag och i följande avsnitt ges en beskrivning av de två teknologierna.

2.1 Desktop-GIS

Desktop-GIS kan delas in i två varianter beroende på om det är LAN-baserat (lokalt nätverk) eller inte [6]. Är det inte nätverksbaserat ligger allting, såväl data som program, på den lokala datorn och ingen kommunikation med andra datorer görs. I det andra fallet används vanligen en tvålagers klient/server-teknik där GIS-programmen på de lokala datorerna kommunicerar med servern. En nackdel med desktop-GIS är att GIS-program måste installeras på varje användares dator. I och med att licens krävs för varje installation begränsas antalet användare. En annan nackdel är att det blir ett slutet system i och med att åtkomst endast är tillgänglig från de datorer inom nätverket som har installerat programvaran.

2.2 Distribuerat GIS

Distribuerat GIS brukar kallas för geografiska informationstjänster istället för geografiska informationssystem. Desktop GIS är ett stängt system med ett begränsat antal användare, men så är inte fallet med distribuerat GIS. Med Internet har det blivit möjligt att göra ett GIS tillgängligt för alla med tillgång till en Internet-uppkoppling. En fördel med denna teknik är att den är plattformsoberoende [6].

Distribuerat GIS kan delas in i två typer, mobilt GIS och Internet GIS [6]. För Internet GIS består klienterna i huvudsak av stationära eller bärbara datorer medan klienterna för mobilt GIS, även kallat trådlöst GIS, även kan bestå av handdatorer och mobiltelefoner. Det brukar även talas om webb-GIS, som kan beskrivas som en delmängd av Internet GIS [6]. Istället för att enbart säga att kommunikationen mellan server och klient ska ske via Internet specificeras det i webb-GIS ytterligare genom att säga att World Wide Web protokoll (www) skall användas.

Vid Internet GIS består klienten av en vanlig webbläsare. Att inget GIS-program behöver installeras gör tekniken lättåtkomlig och mängden användare begränsas inte som med de äldre teknikerna. Fastän inget särskilt program behöver installeras kan analyser utföras. För att utföra en analys kan klienten antingen be servern om data och analysverktyg och själv utföra analysen, eller så ber klienten servern att utföra analysen och endast svaret returneras. Vilket alternativ som väljs styr hur avancerad klienten behöver vara, det talas då om tjocka och tunna klienter. En tunn klient visar endast statisk information och klarar inte av några beräkningar medan en tjock klient ber servern om data som den själv bearbetar. En tjock klient kan även visa information som ändras dynamiskt utan ytterligare server-anrop (se kapitel 5). Det går dock inte att skilja helt på server och klient, ett program kan t.ex. fungera som en tjock klient gentemot en server men som en server gentemot andra (tunna) klienter [6].

3 Standardiserade märkspråk

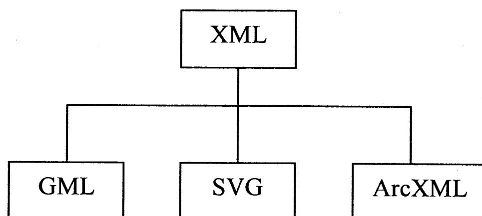
Ett märkspråk (*markup language*) har regler för hur textinformation som lagras i ett dokument ska märkas upp. Anledningen till att märka upp texten kan vara för att strukturera upp den information som lagras eller för att ange hur informationen skall presenteras. Det finns flera metoder som kan användas för att märka upp innehållet, ett vanligt sätt är att använda element och attribut. Ett element har en starttagg, i vilken attribut kan placeras, och en sluttagg. Mellan elementets start- och sluttagg kan det finnas fler element eller enbart vanlig text. Att ett dokument har en så enkel men ändå strikt struktur medför att det är läsbart för människor. Det medför även att det inte behövs något specialprogram för att hantera dokumentet utan en vanlig texteditor, t.ex. *Notepad*, kan användas.

Det finns ett flertal standardiserade märkspråk. En fördel med att använda sig av standardiserade märkspråk för exempelvis utbyte av data är att man då inte blir beroende av en viss programleverantör och deras format. Om alla använde samma standardiserade märkspråk skulle det bli lättare både att använda sig av flera olika programleverantörer och att byta från en programleverantör till en annan. Två organisationer som tagit fram standarder för märkspråk är W3C och OGC.

W3C, som står för World Wide Web Consortium, bildades 1994 och tar fram generella webbstandarder [7]. Konsortiumet har mer än 400 medlemmar som hjälps åt att finansiera dess verksamhet tillsammans med statsbidrag [8]. Medlemmarna består av såväl regeringar och EU som ledande industriföretag och forskningsinstitut.

OGC, Open Geospatial Consortium, är en internationell organisation som tar fram GIS-inriktade webbstandarder. Organisationen, som inte är vinstdrivande, grundades den 25 september 1994 och består idag av mer än 300 företag, regeringar och universitet [9].

Exempel på två vanliga standardiserade märkspråk är **HTML** och **XML** som båda har tagits fram av W3C. En kortare beskrivning av HTML ges i kapitel 5. Det finns ett flertal dialekter av XML, se figur 3.1. En dialekt av XML är XML som är giltig enligt en viss **DTD** eller **XML Schema** (se avsnitt 3.1). Två av dessa dialekter är **GML** och **SVG** som har tagits fram av OGC respektive W3C, dessa beskrivs senare i kapitlet. Även **ArcXML** är en dialekt av XML. Språket har tagits fram av ESRI och används bl.a. vid kommunikation mellan webbklienter och karttjänster skapade med programmet ArcIMS (se avsnitt 7.2).



Figur 3.1. Olika dialekter av märkspråket XML.

3.1 XML - eXtensible Markup Language

Teorin i avsnittet om XML baseras i huvudsak på två källor. Dessa är hemsidan för W3C [10] samt boken *XML in a Nutshell* [11].

XML är ett metaspråk och standarden för det togs fram av W3C 1998. I ett XML-dokument lagras data på ett väl strukturerat sätt. Den ursprungliga tanken med XML var att det skulle användas för att skicka data mellan klient och webbserver, men idag används det även ofta som t.ex. konfigurationsfiler till datorprogram.

Både XML och HTML är märkspråk, men de har stora olikheter. I HTML är alla element som får användas samt vilka attribut dessa får ha fördefinierade. I XML är det istället tillåtet att skapa egna typer av element och ge dem vilka attribut man vill. En annan skillnad är att HTML används till att presentera data medan XML enbart är till för att lagra data på ett strukturerat sätt. XML saknar därför helt information om hur data skall presenteras. Dessa två skillnader medför att man i XML brukar låta elementets namn beskriva vilken typ av information som lagras i det, som i exempel 3.1.

```
<title>GIS - A Computing Perspective</title>
```

Exempel 3.1. Ett XML-element som innehåller en boks titel.

Det finns flera olika sätt att lagra samma information, det är t.ex. möjligt att välja att lagra all information med enbart element och text och helt låta bli att använda attribut. I exempel 3.2 och 3.3 visas två olika sätt att lagra samma information om en bok, i det första fallet utan och i det senare fallet med attribut. Det går inte att säga vilket sätt som är bäst, det finns fördelar med båda. En fördel med att inte använda attribut är att all information som lagras ligger som text mellan start- och sluttagg vilket kan upplevas som mer lättläst. En fördel med att använda attribut är istället att filen blir kortare samt att de element som saknar textinnehåll kan stängas på en gång. Hur detta görs syns på elementen med namnet *author* i exempel 3.3. Det går även bra att använda en kombination av de två metoderna så att attribut används ibland och text ibland.

```
<book>
  <title>GIS - A Computing Perspective</title>
  <author>
    <firstname>Michael</firstname>
    <lastname>Worboys</lastname>
  </author>
  <author>
    <firstname>Matt</firstname>
    <lastname>Duckham</lastname>
  </author>
  <ISBN>0415283752</ISBN>
</book>
```

Exempel 3.2. Ett XML-dokument där informationen lagras utan användande av attribut.

```
<book>
  <title value="GIS - A Computing Perspective"/>
  <author firstname="Michael" lastname="Worboys"/>
  <author firstname="Matt" lastname="Duckham"/>
  <ISBN value="0415283752"/>
</book>
```

Exempel 3.3. Ett XML-dokument där informationen lagras med hjälp av attribut.

Alla XML-dokument måste vara välformulerade (*well-formed*). För att vara det krävs att ett antal strikta regler angående dokumentets uppbyggnad följs. De fyra viktigaste reglerna (baserat på *XML in a Nutshell* [11]) är:

- Varje XML-dokument måste ha exakt ett rotelement.
- Varje element måste ha en start- och en sluttagg.
- Alla attributvärden måste ha citattecken runt sig.
- Element får inte vara nästlade.

Dessa strikta regler medför att det blir svårare att skriva ett XML-dokument. Men fördelen med dem är att de underlättar läsning av XML. Det är även viktigt att komma ihåg att det i XML är skillnad mellan stora och små bokstäver, *a* och *A* är alltså inte samma sak.

3.1.1 DTD – Document Type Definition

Förutom de regler som gör ett XML-dokument välformulerat finns det möjlighet att själv definiera regler för hur dokumentet får byggas upp. Exempel på sådana regler kan vara vilka element som får/måste finnas samt vilka attribut ett visst element får/måste ha. Dessa regler kan definieras i en DTD, Document Type Definition. Att följa en DTD gör att XML-dokumentet även blir giltigt (*valid*).

En DTD kan ligga i en egen fil eller vara en del av XML-dokumentet. Ett exempel på hur en DTD kan se ut visas i exempel 3.4 där varje rad beskriver ett element. Det första elementet som beskrivs heter *book* och ska innehålla exakt ett element av typen *title* följt av ett eller flera element *author* och sist exakt ett element *ISBN*. Om XML-dokumentet i exempel 3.2 använder denna DTD blir XML-dokumentet giltigt.

```
<!ELEMENT book (title,author+,ISBN)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (firstname,lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
```

Exempel 3.4. Om XML-dokumentet i exempel 3.2 använder denna DTD blir dokumentet giltigt.

En DTD innehåller en beskrivning av varje element som får förekomma i XML-dokumentet. I vilken ordning elementen beskrivs i DTD:n saknar betydelse. Hur en beskrivning av ett element byggs upp visas i exempel 3.5. *Element_namn* byts ut mot ett passande namn och *innehåll* specificerar vad elementet får innehålla. Innehållet beskrivs med hjälp av en lista

med namnen på de element som får förekomma i elementet. För att ange att elementet får innehålla text används strängen *#PCDATA*. Den ordning elementen har i listan har betydelse då det är samma ordning som de ska uppträda i inne i XML-dokumentet.

```
<!ELEMENT element_namn (innehåll)>
```

Exempel 3.5. Exemplet visar hur ett element beskrivs i en DTD.

Om inget annat anges skall exakt ett element förekomma. Med hjälp av en asterix efter elementnamnet i listan anges att det ska finnas noll, ett eller flera sådana element. För att ange att det får förekomma noll eller ett element används istället ett frågetecken och ett plustecken används för att tala om att det skall finnas ett eller flera element av en viss typ.

Med hjälp av parenteser och | kan mer komplicerade listor specificeras. Tecknet | används mellan två elementnamn för att tala om att antingen det första eller det andra elementet ska förekomma. Exempel 3.6 beskriver ett element *cirkel* som dels ska innehålla ett element *punkt* samt antingen ett element *radie* eller ett element *diameter*. För att beskriva det används både parenteser och tecknet |.

```
<!ELEMENT cirkel (punkt,(radie|diameter))>
```

Exempel 3.6. Elementet cirkel ska innehålla elementet punkt samt ett av elementen radie och diameter.

I de fall då ordningen mellan elementen saknar betydelse kan en asterix efter parentesen runt listan med element användas. Detta tar dock bort möjligheten att ange hur många gånger varje element skall förekomma.

Hittills har endast nämnts hur elementen specificeras. En DTD kan även innehålla information om vilka attribut som skall/får finnas till respektive element. Ett exempel på hur attributen till ett element av typen *Person* kan se ut visas i exempel 3.7. *Person* måste där ha ett attribut *firstname* och får även ha ett attribut *lastname*. Båda attributen har typen *CDATA* (character data) dvs. text.

```
<!ATTLIST Person firstname CDATA #REQUIRED
                 lastname CDATA #IMPLIED
>
```

Exempel 3.7. En definition av vilka attribut elementet Person ska ha. #REQUIRED innebär att elementet måste ha attributet firstname medan #IMPLIED betyder att det är valfritt att ange attributet lastname samt att inget default-värde för attributet finns.

3.1.2 XML Schema

Ett alternativ till att använda sig av en DTD för att göra XML-dokument giltiga är att använda XML Schema som är ett nyare format att lagra regler för XML-dialekter. En fördel med XML Schema framför DTD är att XML Schema följer XML-standarden, dvs. det är ett välformulerat XML-dokument [12]. Det medför att en vanlig XML-parser kan användas för att läsa reglerna vilket inte fungerar om de skrivs i en DTD.

En annan fördel med XML Schema framför DTD är att det ger större flexibilitet samtidigt som kontrollen över elementinnehåll och datatyper ökar [13]. Hur informationen i den DTD som visades i exempel 3.4 skulle beskrivas med hjälp av XML Schema visas i exempel 3.8. Där syns att attributen *minOccurs* och *maxOccurs* används för att ange hur många element av typen *author* som får finnas. Att antalet förekomster anges med attribut ger större flexibilitet än vad DTD erbjuder och att inte ange attributen medför att de får värdet ett.

```
<schema>
  <element name="book">
    <complexType>
      <sequence>
        <element name="title" type="string"/>
        <element name="author" minOccurs="1"
         maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="firstname" type="string"/>
              <element name="lastname" type="string"/>
            </sequence>
          </complexType>
        </element>
        <element name="ISBN" type="string"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

Exempel 3.8. Ett XML Schema som uttrycker samma sak som den DTD som visas i exempel 3.4.

3.1.3 XML Namespace

XML Namespace är en standard som W3C har tagit fram för att hantera de namnkonflikter som kan uppstå som en följd av att namn på element och attribut i XML inte är fördefinierade [14]. Flera XML Scheman kan ge olika definitioner på ett element med ett visst namn. Om alla XML-dokument enbart följer ett av dessa scheman är det inget problem. Men om en definition av ett element skall användas på en plats i XML-dokumentet och en annan definition av samma element används på ett annat ställe i dokumentet uppstår ett problem eftersom elementen har samma namn men skall vara uppbyggda på olika sätt. För att skilja elementen åt används då *XML Namespace.*

Vilka namnrymder, *namespaces*, som används i ett XML-dokument anges i dokumentets rotelement. Där anges vilket namn som används för att referera till namnrymden senare i dokumentet samt en **URI** som identifierar namnrymden [15]. Det är även möjligt att ange en namnrymd som *default namespace*, denna namnrymd används då i XML-dokumentet i de fall där ingen annan namnrymd anges. I exempel 3.9 visas hur två namnrymder anges i ett rotelement. Den första namnrymden har angetts som *default namespace* medan den andra namnrymden har getts namnet *exempel*. För att använda ett element från den första namnrymden anges då enbart elementets namn medan element från den andra namnrymden anges med *exempel:* följt av elementets namn.

```
<?xml version="1.0"?>
<root
xmlns="http://www.someserver.com/default namespace"
xmlns:exempel="http://www. someserver.com/exempel namespace">
 <element>Element från default_namespace</element>
 <exempel:element description="Element från
  exempel_namespace"/>
</root>
```

Exempel 3.9. Exempel på användning av XML Namespace.

3.2 GML – Geography Markup Language

Följande avsnitt bygger främst på boken Geography Mark-Up Language: Foundation for the Geo-Web [12].

GML är en dialekt av XML som har tagits fram av OGC. Den är främst framtagen för att distribuera geografiska vektordata även om formatet kan användas för datalagring.

Ett GML-dokument innehåller beskrivningar av ett eller flera geografiska objekt (*features*). Varje beskrivning innehåller information om objektets geografiska koordinater samt eventuella attribut. För att beskriva objektens geometriska egenskapar används olika geometriska element. Dessa element kallas geometrier.

Det finns flera versioner av GML-standarden. GML2 innehåller enbart linjära geometrier, dvs. geometrier som enbart är uppbyggda av raka linjer. Exempel på geometrier som finns i GML2 är *Point*, *LineString*, *Polygon* och *Multipolygon*. I den nyare standarden GML 3.0 finns mer komplexa geometrier och det är inte längre enbart raka linjer som används. Exempelvis finns geometrierna *Curve* och *Surface* med i GML 3.0.

Samma typ av geometri kan användas för att beskriva olika egenskaper beroende på vilken typ av objekt som beskrivs. För att tala om vilken egenskap hos objektet som geometrin beskriver används geometriegenskaper (*GML geometry-valued properties*). Några exempel på geometriegenskaper är *position*, *centerOf*, *centerLineOf* och *extentOf*. När ett GML schema specificeras finns dock möjlighet att definiera ytterligare geometriegenskaper.

I exempel 3.10 visas ett exempel på hur en del av ett GML-dokument kan se ut. Ett objekt av typen *Roads* beskrivs med attributen *id*, *speedlimit* och *shape_len*. Geometrin *LineString*

används för att beskriva vägens mittlinje, vilket geometriegenskapen *centerLineOf* anger. Linjen beskrivs i sin tur av ett antal koordinater, den delen har dock utelämnats i exemplet.

```
...
<Feature>
  <featuretype>Roads</featuretype>
  <property name="id">101</property>
  <property name="speedlimit">90</property>
  <property name="shape_len">105.75</property>
  <property name="gml2_coordsys"></property>
  <gml:centerLineOf>
    <gml:LineString
     srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:Coordinates>
        ...
      </gml:Coordinates>
    </gml:LineString>
  </gml:centerLineOf>
</Feature>
...
```

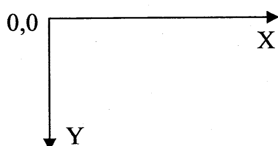
Exempel 3.10. En del av ett GML-dokument som beskriver ett väg-objekt. Attributet srsName som finns till elementet gml:LineString använder ***EPSG*** *för att ange vilket koordinatsystem som används. Koden 4326 svarar mot WGS84 Latitud/Longitud med koordinater i grader och Greenwich som medelmeridian.*

3.3 SVG – Scalable Vector Graphics

SVG är en dialekt av XML som har tagits fram av W3C och används för att presentera vektordata. Formatet kan exempelvis användas vid visning av en vektorkarta istället för vanliga rasterformat såsom JPEG, GIF och PNG.

Det finns både fördelar och nackdelar med att använda sig av SVG istället för de vanliga bildformaten. En nackdel är att det krävs att en plugin är installerad för att en klient ska kunna läsa SVG, medan bildformaten JPEG, GIF och PNG stöds av alla webbläsare. Fördelen med SVG framför vanliga bildformat är att SVG är ett vektorformat och därmed inte förlorar i kvalitet vid zoomning, vilket rasterformat gör. En annan fördel är att SVG-dokumentet blir en del av webbläsarens DOM (Document Object Model) [16]. Enskilda element i SVG-dokumentet kan därför nås genom att t.ex. använda sig av **JavaScript**. Detta i kombination med att SVG är en typ av XML och därmed lätt att läsa (och modifiera) medför att de enskilda objekten kan modifieras på ett relativt enkelt sätt. Ytterligare en fördel är att en SVG-fil blir mindre än exempelvis JPEG- och GIF-bilder.

Till skillnad från GML innehåller inte SVG någon information om de geografiska koordinaterna utan enbart koordinater i filens eget system. Detta koordinatsystem har origo i övre vänstra hörnet med stigande X-värden åt höger och stigande Y-värden nedåt (se figur 3.2) [17].

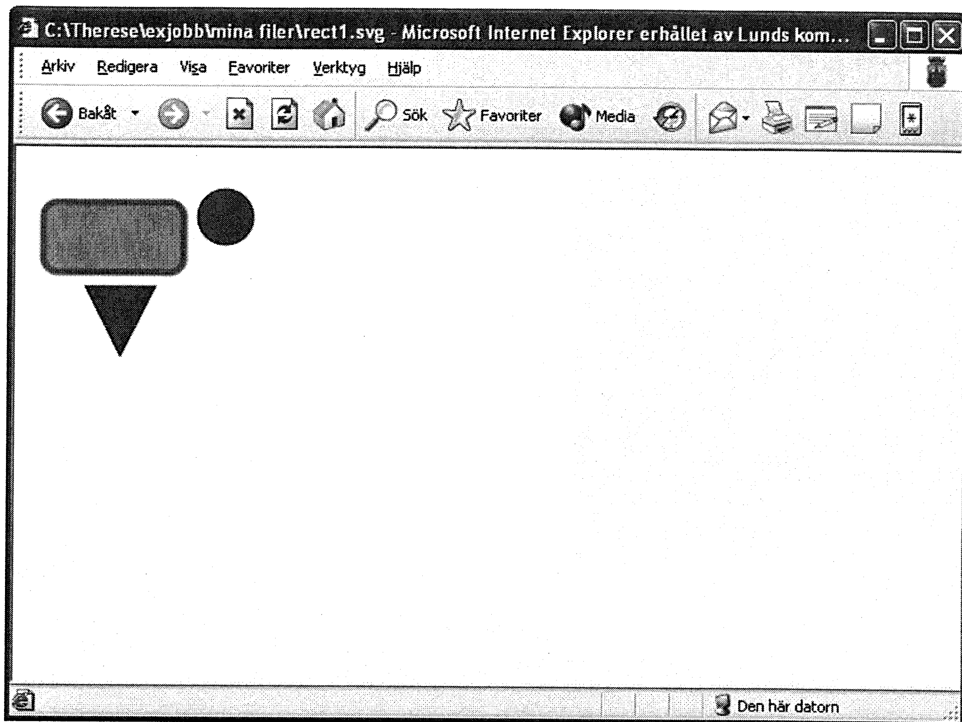


Figur 3.2. Koordinatsystem i ett SVG-dokument.

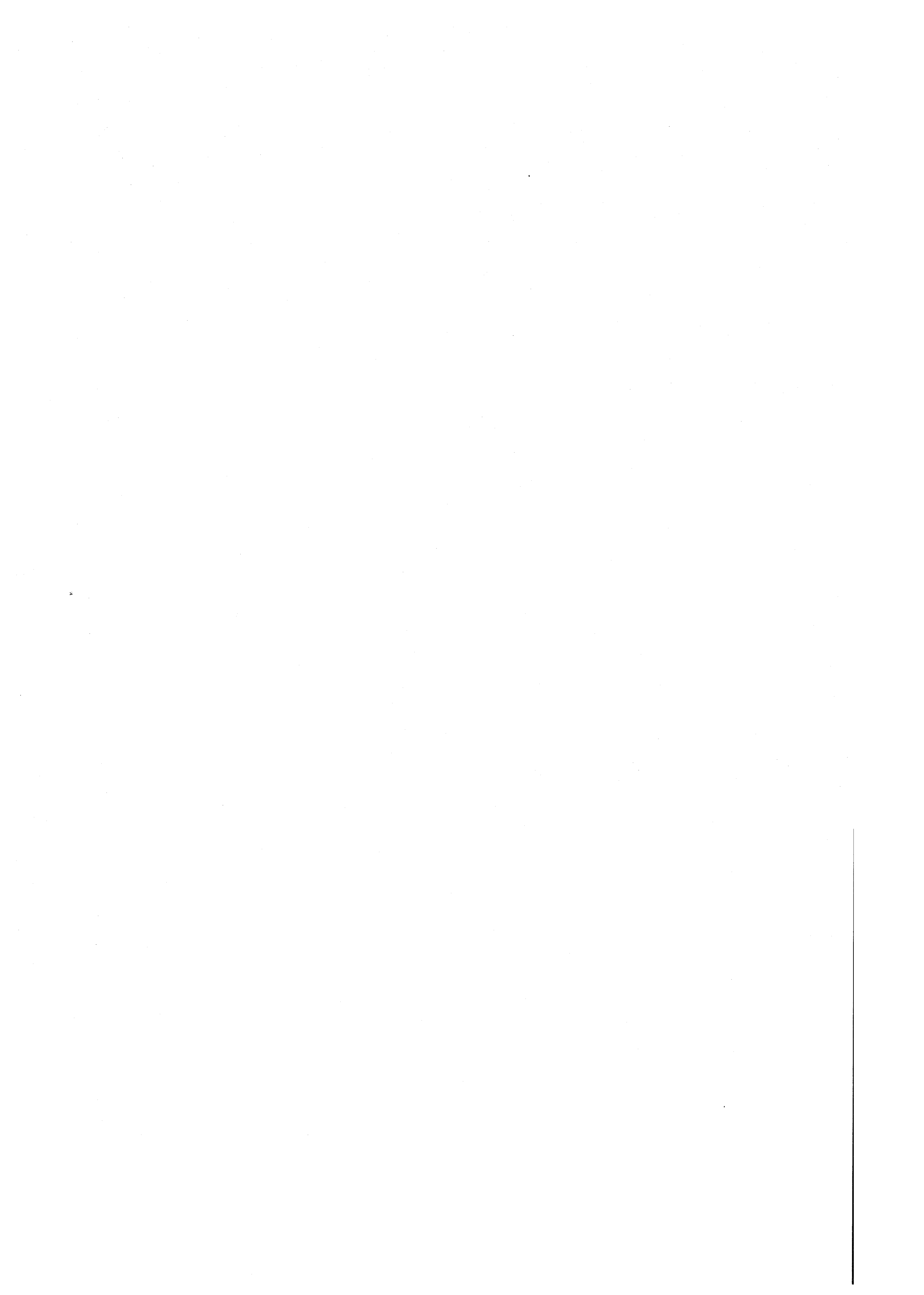
Det finns en mängd element med tillhörande attribut som kan användas för att rita bilder med SVG. I exempel 3.11 visas hur ett SVG-dokument med några av de enklare elementen kan se ut, och i figur 3.3 visas den bild som genereras. SVG-dokumentets rotelement heter *svg* och har attribut som bl.a. talar om storleken på den ruta där objekten ligger. Storleken kan anges i pixlar, som i exemplet, eller i procent. Sedan följer tre olika element som beskriver varsitt geometriskt objekt. Attributen *x* och *y* i elementet *rect* (rektangel) anger var övre vänstra hörnet skall vara placerat medan *rx* och *ry* anger att hörnen skall vara rundade. Attributet *style* innehåller information om hur objektet skall presenteras grafiskt, ett alternativ till att använda *style* är att istället använda attributen *stroke*, *fill* etc. på så sätt som visas i elementet *circle*. Alla tre objekt har fått samma färg men som syns i figur 3.3 ser rektangeln ljusare ut, detta beror på att den har opaciteten 0,5 medan de andra objekten (automatiskt) fått opaciteten 1. För cirkeln anges koordinater för centrum samt cirkelns radie. Elementet *path* kan användas för att rita egna geometriska objekt med såväl raka som böjda linjer. Attributet *d* anger objektets utsträckning där *M* anger startpunkt, *L* betyder linje till och *z* innebär linje tillbaka till startpunkten.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="400" height="300" version="1.1"
xmlns="http://www.w3.org/2000/svg">
  <rect x="20" y="40" rx="10" ry="10" width="100" height="50"
  style="fill:red;stroke:black;stroke-width:5;opacity:0.5"/>
  <circle cx="150" cy="50" r="20" stroke="black" fill="red"/>
  <path d="M50 100 L100 100 L75 150 z"
  style="fill:red;stroke:black;stroke-width:1"/>
</svg>
```

Exempel 3.11. Exempel på hur koden i en SVG-fil kan se ut. Hur bilden blir visas i figur 3.3.



Figur 3.3. Bilden som SVG-filen i exempel 3.11 ger. Alla figurer har getts samma färg men eftersom rektangelns opacitet är 0,5 istället för 1,0 ser färgen annorlunda ut.



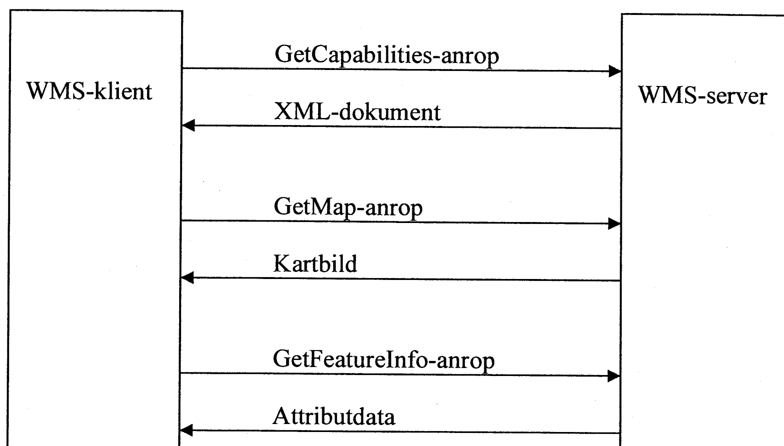
4 Standardiserade karttjänster

Kapitlet behandlar de två standardiserade karttjänsterna **WMS** (Web Map Service) och **WFS** (Web Feature Service). Båda standarderna är framtagna av OGC (se inledningen till kapitel 3) och används vid överföring av geografiska data via Internet. WMS är till för att presentera kartor och attributdata medan WFS används för att kunna analysera och uppdatera den geografiska informationen. Därför returnerar en WFS-server geografisk data i GML-format (se avsnitt 3.2) medan en WMS-server levererar kartdata i olika raster- och vektorformat [18] [19].

4.1 WMS – Web Map Service

Det här avsnittet bygger till stor del på Open Geospatial Consortiums specifikation av Web Map Service 1.3.0 [18].

WMS är en OGC-standard som används för att skicka information mellan en klient och en kartserver. Det är tre olika typer av information som klienten kan fråga efter. Först måste klienten ta reda på vad servern kan dvs. vilken typ av data kartservern tillhandahåller, detta görs genom ett anrop av metoden *GetCapabilities*. Sedan kan klienten med hjälp av denna information fråga kartservern efter en kartbild (*GetMap*) eller fråga om information om objekten i kartan (*GetFeatureInfo*). Hur denna kommunikation mellan klient och kartserver fungerar syns i figur 4.1.



Figur 4.1. Kommunikationsflöde mellan WMS-klient och WMS-server.

Som nämnts ovan finns tre olika typer av anrop, det är dock endast de två första som är obligatoriska då WMS-standarden implementeras. Att enbart implementera *GetCapabilities* och *GetMap* kallas *Basic WMS* medan det kallas *Queryable WMS* om även *GetFeatureInfo* implementeras.

WMS-tekniken fungerar så att klienten ställer frågor till kartservern med hjälp av **CGI-parametrar**. Vilka CGI-parametrar som skall finnas med i en fråga till WMS-servern definieras i standarden och det är enbart två parametrar, *service* och *request*, som alltid är obligatoriska. Parametern *service* används för att tala om att det är WMS-tjänsten som anropas. Anledningen till att det måste anges är att samma server kan erbjuda flera olika typer av tjänster. Parametern *request* används för att tala om vilken typ av fråga som ställs och kan ha värdet *GetCapabilities*, *GetMap* eller *GetFeatureInfo*.

4.1.1 GetCapabilities

GetCapabilities används för att ta reda på vad WMS-servern tillhandahåller för information. Svaret ges, om inget annat anges, i form av ett XML-dokument.

Strukturen på ett *GetCapabilities*-anrop är enkel. Förutom de två obligatoriska parametrarna *service* och *request* finns det tre valfria parametrar. Parametern *format* används för att tala om i vilket format svaret önskas. Det enda format som servern måste klara är XML, det är därför det format som returneras om inget format anges eller om servern inte tillhandahåller det efterfrågade formatet. Parametern *version* används för att tala om vilken version av tjänsten som frågas. Anges inte denna parameter skall servern returnera den senaste versionen. Den tredje valfria parametern heter *updatesequence*, vad den används till tas dock inte upp i denna rapport. Hur en fråga av typen *GetCapabilities* kan se ut visas i exempel 4.1.

```
http://www2.demis.nl/mapserver/request.asp?
service=WMS&
version=1.1.1&
request=GetCapabilities
```

Exempel 4.1. En fråga av typen GetCapabilities till WMS-servern http://www2.demis.nl/mapserver/request.asp.

Även vilka element det returnerade XML-dokumentet ska innehålla finns specificerat i WMS-standarden. Några viktiga delar av innehållet i XML-dokumentet som behövs för att senare kunna konstruera korrekta GetMap- och GetFeatureInfo-anrop är elementen Request och Layer. Elementet Request är viktigt eftersom det bl.a. innehåller information om vilka filformat som de andra två operationerna tillhandahåller. Elementet Layer innehåller istället information om de kartlager WMS-servern tillhandahåller.

Det finns ett element *Layer* för varje lager som kan efterfrågas med någon av de andra operationerna. Varje sådant element måste innehålla information om lagrets titel, namn, vilket geografiskt område som täcks samt vilka referenssystem som finns tillgängliga. Skillnaden mellan titel och namn är att titeln är till för oss människor och skall vara ett beskrivande och lättläsligt namn. Lagrets namn behöver däremot varken vara lättläsligt eller beskrivande, men det används för att referera till lagret. Det är valfritt att även låta elementet innehålla information om vilken skala som är lämplig att visa lagret i. Dessutom kan elementet *Layer* ha ett antal attribut, ett av dessa är attributet *queryable*. Attributet har värdet 1 om WMS-servern kan leverera attributdata om lagret och värdet 0 annars. Om lagret innehåller flera element *styles* finnas möjlighet att visa samma lager med olika utseende.

4.1.2 GetMap

Efter att ha tagit reda på vilka lager som finns och vilka filformat som är tillgängliga kan en fråga av typen *GetMap* konstrueras.

Utöver de två parametrar som alltid är obligatoriska vid WMS-anrop, *service* och *request*, finns ytterligare sju obligatoriska parametrar och sex valfria som kan finnas med i ett *GetMap*-anrop. Vilka de valfria parametrarna är kommer inte att tas upp här. Parametern *version* som var valfri för ett *GetCapabilities*-anrop är här obligatorisk.

Vilka lager som den efterfrågade kartan skall innehålla anges i en kommaseparerad lista till parametern *layers*. Vilken ordning lagernamnen anges i har betydelse för den returnerade kartans utseende, det lagernamn som är sist i listan kommer att ritas upp över de andra lagren. Även vilket utseende, *styles*, som skall användas för respektive lager anges i en kommaseparerad lista. Det första namnet i listan hör till det första lagret i listan med lager, det andra till det andra osv. Även om det är obligatoriskt att ha med parametern *styles* måste den dock inte ges ett värde. Om värdet lämnas blankt, som i exempel 4.2, kommer servern använda det utseende som är default.

Vilket referenssystem kartan skall returneras i anges med parametern *crs* (Coordinate reference system). I den tidigare versionen WMS 1.1.1 användes istället parametern *srs* (Spatial reference system). Båda varianterna syftar till att tala om i vilket referenssystem koordinaterna i parametern *bbox* anges.

Parametern *bbox* (bounding box) innehåller information om vilket geografiskt område den returnerade kartbilden skall täcka. Koordinaterna anges i en kommaseparerad lista i ordningen minsta x-koordinat, minsta y-koordinat, största x-koordinat samt sist största y-koordinat. För att tala om hur stor bild som önskas och i vilket format används parametrarna *width*, *height* och *format*. Bildens storlek anges i pixlar och vilka format som finns tillgängliga kan utläsas ur svaret på *GetCapabilities*-anropet.

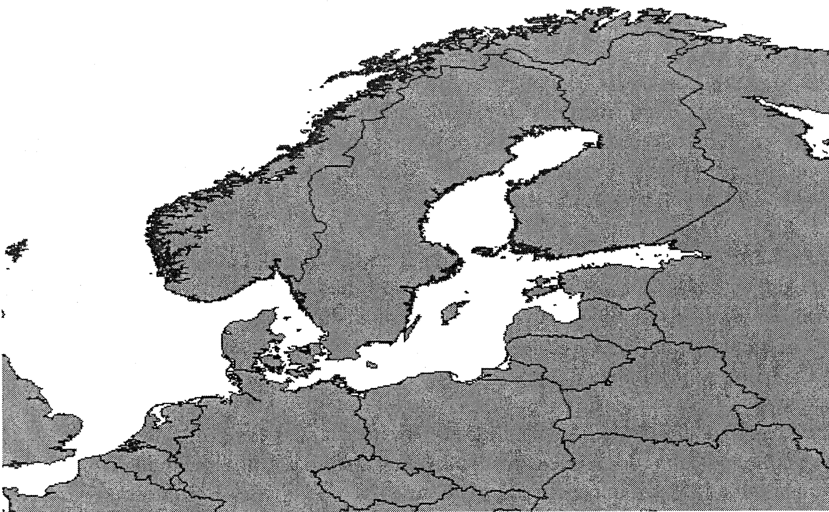
Om format som stöder transpararens finns tillgängliga kan svar från flera *GetMap*-anrop överlagras. Detta kan användas för att i klienten sammanställa en karta med information från flera olika servrar. Det finns även WMS-servrar som sammanställer data på detta sätt genom att fungera som en klient gentemot andra WMS-servrar och som en server gentemot andra WMS-klienter. En sådan server kallas *Cascading Map Server*.

Svaret på anropet består av en karta i angivet format. Om någon parameter ges ett felaktigt värde returneras istället ett felmeddelande (*exception*). Formatet på felmeddelandet är normalt XML, annat format kan anges vid anropet med hjälp av en av de valfria parametrarna.

Hur ett *GetMap*-anrop till en WMS-server kan se ut visas i exempel 4.2. Resultatet av anropet visas i figur 4.2.

```
http://www2.demis.nl/mapserver/request.asp?
service=WMS&
request=GetMap&
version=1.3.0&
layers=Countries,Borders,Coastlines&
styles=&
crs=EPSG:4326&
bbox=-2,48,36,76&
height=450&
width=600&
format=image/gif
```

Exempel 4.2. En fråga av typen GetMap till WMS-servern http://www2.demis.nl/mapserver/request.asp. Att parametern csr har värdet EPSG:4326 innebär att WGS84 används.



Figur 4.2. Resultatet av GetMap-anropet i exempel 4.2.

4.1.3 GetFeatureInfo

Ett anrop av typen GetFeatureInfo används för att ta reda på attributdata om objekten i kartan. Eftersom det inte är obligatoriskt för en WMS-server att implementera den här operationen är den inte alltid tillgänglig. Även om operationen stöds av servern behöver det inte vara möjligt

att få reda på attributdata om alla lager. Endast de lager som i svaret på *GetCapabilities*-anropet har värdet 1 på attributet *queryable* kan frågas. Om ett annat lager efterfrågas i anropet returnerar servern ett felmeddelande (*exception*).

Anropet skall innehålla alla obligatoriska delar av *GetMap*-anropet för att tala om vilken karta frågan gäller. Parametern *request* skall dock ha värdet *GetFeatureInfo* och *version* skall syfta på nuvarande anrop. Förutom dessa parametrar finns ytterligare fyra obligatoriska parametrar. Två av dessa, *I* och *J*, används för att tala om vilken pixel i kartan som frågas. *I* specificerar kolumn-pixel medan *J* specificerar rad-pixel. Övre vänstra hörnet i bilden har bildkoordinaterna 0,0.

För att tala om vilka lager frågan gäller används parametern *query_layers*, värdet skall vara en kommaseparerad lista med lagernamnen. Den sista obligatoriska parametern, *info_format*, används för att ange i vilket format svaret önskas. Tillgängliga format finns att läsa i svaret på *GetCapabilities*-anropet.

Det finns även två valfria parametrar. *Feature_count* används för att ange maximalt antal objekt per lager som attributdata skall returneras för. Att inte ange den här parametern ger samma resultat som att ge den värdet 1. Den andra valfria parametern används för att tala om önskat format på eventuellt felmeddelande. I exempel 4.3 visas hur ett *GetFeatureInfo*-anrop kan se ut och i figur 4.3 visas resultatet av anropet.

```
http://www2.demis.nl/mapserver/request.asp?
service=WMS&
request=GetFeatureInfo&
version=1.3.0&
layers=Countries,Borders,Coastlines&
styles=&
crs=EPSG:4326&
BBOX=-2,48,36,76&
height=450&
width=600&
format=image/gif&
I=300&
J=200&
query_layers=Countries&
info_format=text/html
```

Exempel 4.3. En fråga av typen GetFeatureInfo till WMS-servern http://www2.demis.nl/mapserver/request.asp.

Layer	ID	Description	Value
Countries	SW	Sweden	

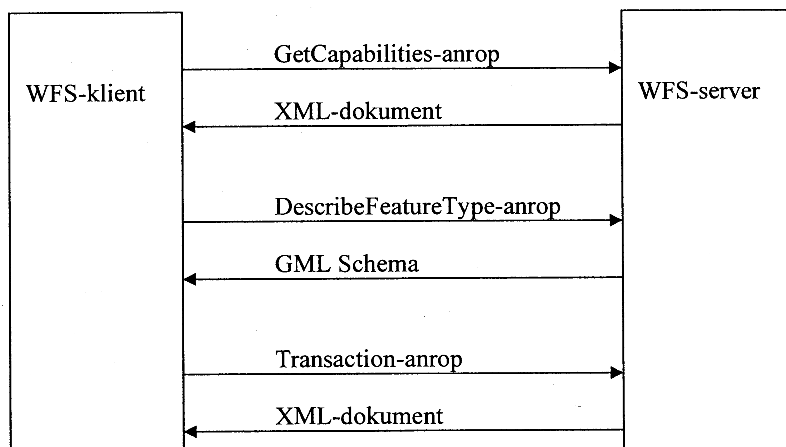
Figur 4.3. Svaret på GetFeatureInfo-anropet i exempel 4.3.

4.2 WFS – Web Feature Service

Det här avsnittet bygger till stor del på Open Geospatial Consortiums specifikation av Web Feature Service 1.1.0 [19].

WFS-standarden innehåller sex olika operationer varav tre är obligatoriska. De obligatoriska operationerna heter *GetCapabilities*, *DescribeFeatureType* samt *GetFeature*. Om endast de obligatoriska delarna implementeras kallas det *Basic WFS*, denna nivå stöder endast hämtning av data. En WFS-server som dessutom implementerar operationen *GetGmlObject* kallas för *XLink WFS*. Den sista typen av WFS-server kallas *Transaction WFS* och skall förutom de obligatoriska operationerna implementera operationen *Transaction*. Det är valfritt för en *Transaction WFS* att även implementera operationerna *GetGmlObject* och *LockFeature*.

I figur 4.4 visas hur kommunikationen mellan en klient och en WFS-server av typen *Transaction WFS* kan se ut. För att få reda på information om tjänsten anropas först *GetCapabilities*. Sedan anropas *DescribeFeatureType* för att ta reda på hur de olika objektstyperna skall beskrivas. Därefter kan *Transaction* anropas för att sätta in, uppdatera eller ta bort objekt. Eventuellt anropas först *LockFeature* för att låsa de objekt som skall uppdateras.



Figur 4.4. Exempel på kommunikationsflöde mellan WFS-klient och WFS-server av typen Transaction WFS.

Enligt WFS-standarden kan antingen HTTP GET eller HTTP POST användas för att skicka frågor från en WFS-klient till en WFS-server. Då GET används specificeras frågan med hjälp av CGI-parametrar på samma sätt som en WMS-fråga. Om POST används finns två alternativ. Det första alternativet är att frågan skickas som XML med *content type* satt till *text/xml*. Det andra alternativet är att skicka frågan med nyckelvärdespar, vilket motsvarar den del av en GET-fråga som står efter frågetecknet. Om det senare alternativet används ska

content type vara satt till *application/x-www-form-urlencoded*. För anrop av operationen *Transaction* med begäran om insättning eller uppdatering av objekt måste frågan alltid skickas som XML med HTTP POST. Gäller anropet dock enbart borttagning av objekt kan valfri anropsmetod användas. Endast hur frågor som skickas som XML med HTTP POST konstrueras kommer att tas upp i rapporten. Undantaget är avsnittet om operationen *GetCapabilities*.

Alla WFS-anrop måste innehålla parametrarna *service* och *version*. *Service* skall ha värdet WFS för att tala om att det är WFS-tjänsten som anropas. *Version* anger vilken version av tjänsten klienten önskar. Flera av frågorna innehåller även elementet *Filter* som används för att tala om vilka objekt anropet gäller. Hur filtret ska vara uppbyggt anges i en särskild standard, OpenGIS® Filter Encoding Implementation Specification [20]. Där anges att ett filter är uppbyggt av spatiala operationer, jämförelseoperationer, logiska operationer samt id. Spatiala operationer används för att jämföra geografiska egenskaper, två exempel på denna typ av operation är *intersects* och *overlaps*. Jämförelseoperationer används istället för att jämföra icke-geografiska egenskaper och exempelvis se om två värden är lika eller om ett värde ligger mellan två givna värden. Det finns tre logiska operationer, dessa är AND, OR och NOT. En logisk operation är alltid sann eller falsk och används för att kombinera uttryck (*expression*). Exempel på uttryck är de aritmetiska operationerna addition, subtraktion, multiplikation och division. Ytterligare exempel på uttryck är *PropertyName* och *Literal* som används för att ange värdet (*Literal*) på ett visst attribut (*PropertyName*).

Vid flera typer av WFS-anrop, exempelvis vid *Transaction*, kan attributet *handle* användas. Syftet med attributet är att göra det möjligt för klienten att associera ett visst namn med en fråga. Attributet kan sedan användas vid felhantering så att klienten i händelse av ett felmeddelande lättare kan identifiera var i frågan något var fel. Om ett WFS-anrop består av flera delar, t.ex. information om flera insättningar som skall göras, kan varje del ges ett eget *handle* för att i händelse av fel enkelt kunna ta reda på vid vilken del felet uppstod.

4.2.1 GetCapabilities

För att ta reda på vad WFS-servern klarar av anropas *GetCapabilities*. Standarden för WFS anger att denna operation måste kunna frågas med hjälp av nyckelvärdespar, dvs. med hjälp av CGI-parametrar och HTTP GET. Anledningen till det är att klienter alltid ska veta hur de kan ta reda på information om en WFS-tjänst. Det finns dock inget som hindrar att en *GetCapabilities*-fråga dessutom kan ställas genom att använda XML och HTTP POST.

Ställs frågan med hjälp av CGI-parametrar skall parametern *service* ha värdet *WFS* och parametern *request* värdet *GetCapabilities*. Dessutom kan parametern *version* anges, men det är inte obligatoriskt. Om frågan istället ställs med hjälp av XML ska dokumentet ha ett rotelement *GetCapabilities*. Rotelementets attribut skall bland annat innehålla information om att det är WFS-tjänsten som frågas.
Svaret ges i form av XML och innehåller information om tjänsten och vem som driver den. Dessutom skall en lista över vilka objektstyper som finns samt vilka operationer respektive objektstyp stödjer finnas med.

4.2.2 DescribeFeatureType

Operationen *DescribeFeatureType* anropas för att få information om hur de objektstyper WFS-servern tillhandahåller skall beskrivas. Servern förväntar sig att klienten beskriver

respektive objektstyp på angivet sätt vid t.ex. anrop av operationen *Transaction*. Det är även på detta sätt servern beskriver objekt vid *GetFeature-* och *GetGmlObject*-anrop.

En *DescribeFeatureType*-fråga som skickas i form av XML skall innehålla ett rotelement *DescribeFeatureType*. Rotelementet ska i sin tur innehålla attribut som bland annat talar om att det är WFS-tjänsten som frågas samt vilken version som önskas. Det är valfritt att ta med attributet *outputFormat* som talar om i vilket format svaret önskas. Rotelementet kan innehålla ett valfritt antal element av typen *TypeName*. De elementen anger vilka objektstyper frågan gäller. Om frågan saknar *TypeName*-element anses frågan gälla alla tillgängliga objektstyper.

Svaret fås som GML3 application Schema, en typ av XML Schema, om inget annat angetts. Då förutsätts dock att frågan är ställd på ett korrekt sätt, i annat fall returneras ett felmeddelande (*exception*).

4.2.3 GetFeature

GetFeature-operationen används för att få information om ett eller flera objekt i GML-format.

Rotelementet i frågan skall vara av typen *GetFeature* och bland annat innehålla attribut som talar om att det är WFS-tjänsten som frågas samt vilken version som önskas. I rotelementet finns ett eller flera element av typen *Query*. Varje sådant element beskriver en fråga genom att tala om vilken objektstyp som frågan gäller samt vilka attribut som skall returneras för objekten. Varje element *Query* innehåller även ett filter som anger vilka objekt anropet gäller.

Svaret ges, om inget annat angetts, i form av ett GML3-dokument som är giltigt gentemot det schema som gavs som svar på *DescribeFeatureType*-anropet. Finns det fel i frågan returneras istället ett felmeddelande (*exception*). Om frågan är korrekt ställd kan två olika typer av svar erhållas. Vilket svar som ges kan anges med hjälp av det valfria attributet *resultType* i frågans rotelement. Om attributet har värdet *Results* eller inte finns med i frågan ges ett komplett svar. Har attributet värdet *Hits* erhålls enbart information om hur många objekt som matchade frågan.

4.2.4 GetGmlObject

För att fråga efter ett visst objekt eller element anropas operationen *GetGmlObject*. Frågan skickas som XML med rotelementet *GetGmlObject*. Svaret ges, om inget annat angetts, i GML-format giltigt enligt det schema som gavs som svar på *DescribeFeatureType*-anropet.

4.2.5 LockFeature

Operationen *LockFeature* är valfri och implementeras eventuellt av en WFS-server av typen *Transaction WFS*. Operationen används för att låsa objekt hos WFS-servern i samband med uppdateringar av dessa objekt.

Anledningen till att låsa objekt är att förhindra att följande situation uppstår. En klient hämtar information om ett objekt som skall uppdateras. Under tiden som klienten förbereder det *Transaction*-anrop som skall användas för att utföra uppdateringen utför en annan klient en uppdatering av samma objekt. Då sedan den första klienten skickar sitt anrop till servern går de ändringar den andra klienten gjort på objektet förlorade. För att förhindra detta låses

objektet med hjälp av operationen *LockFeature* innan information om objektet hämtas. När objektet sedan uppdateras genom anrop av operationen *Transaction* låses objektet automatiskt upp och blir tillgängligt för andra klienter. Om ingen uppdatering görs bör objektet ändå låsas upp efter en viss tid. Tidens längd kan anges i frågan med det valfria attributet *expiry*. I standarden finns dock inget förutbestämt värde som skall användas om frågan saknar information om hur länge låsningen ska gälla. Detta är istället upp till varje WFS-server att bestämma.

Frågans rotelement *LockFeature* innehåller ett eller flera element av typen *Lock*. Varje sådant element används för att låsa en mängd av objekt som definieras med hjälp av ett filter.

Svaret på anropet av *LockFeature* ges i form av ett XML-dokument som innehåller information om vilka objekt som har låsts samt ett id. Id-numret används sedan vid anrop av operationen *Transaction* för att servern ska veta att det är samma klient som har låst objektet som nu vill uppdatera det.

4.2.6 Transaction

För att göra uppdateringar i de data WFS-servern tillhandahåller anropas operationen *Transaction*. Det finns tre olika typer av uppdateringar som kan göras. Dessa är insättning av nytt objekt, uppdatering av befintligt objekt samt borttagning av objekt. Flera av dessa kan göras med samma anrop av *Transaction*, anropet betraktas uppifrån och ner varför den ordning i vilken uppdateringarna anges har betydelse.

Anropet av *Transaction* ska innehålla ett rotelement av typen *Transaction* samt valfritt antal element av typerna *Insert*, *Update* och *Delete*. Flera objekt kan sättas in med ett element *Insert*. Detsamma gäller vid uppdatering och borttagning. Ett element för uppdatering ska innehålla element som anger vilka attribut som ska uppdateras samt till vad. Alla objekt som uppdateras med hjälp av ett *Update*-element måste vara av samma typ och för att ange vilka objekt som skall uppdateras används ett filter. Även vid borttagning av objekt används ett filter.

Om ett anrop av *LockFeature* gjorts innan anropet av *Transaction* måste frågan innehålla det id som då returnerades. Efter uppdateringen kommer objekten att åter göras tillgängliga för andra klienter. Ett attribut kan användas för att ange om alla objekt som låstes vid det tillfället skall låsas upp eller enbart de nu uppdaterade objekten.

I exempel 4.4 visas hur ett anrop av *Transaction* som sätter in ett nytt objekt i lagret *Stompunkter* skulle kunna se ut. I exemplet antas att WFS-servern inte implementerar *LockFeature*.

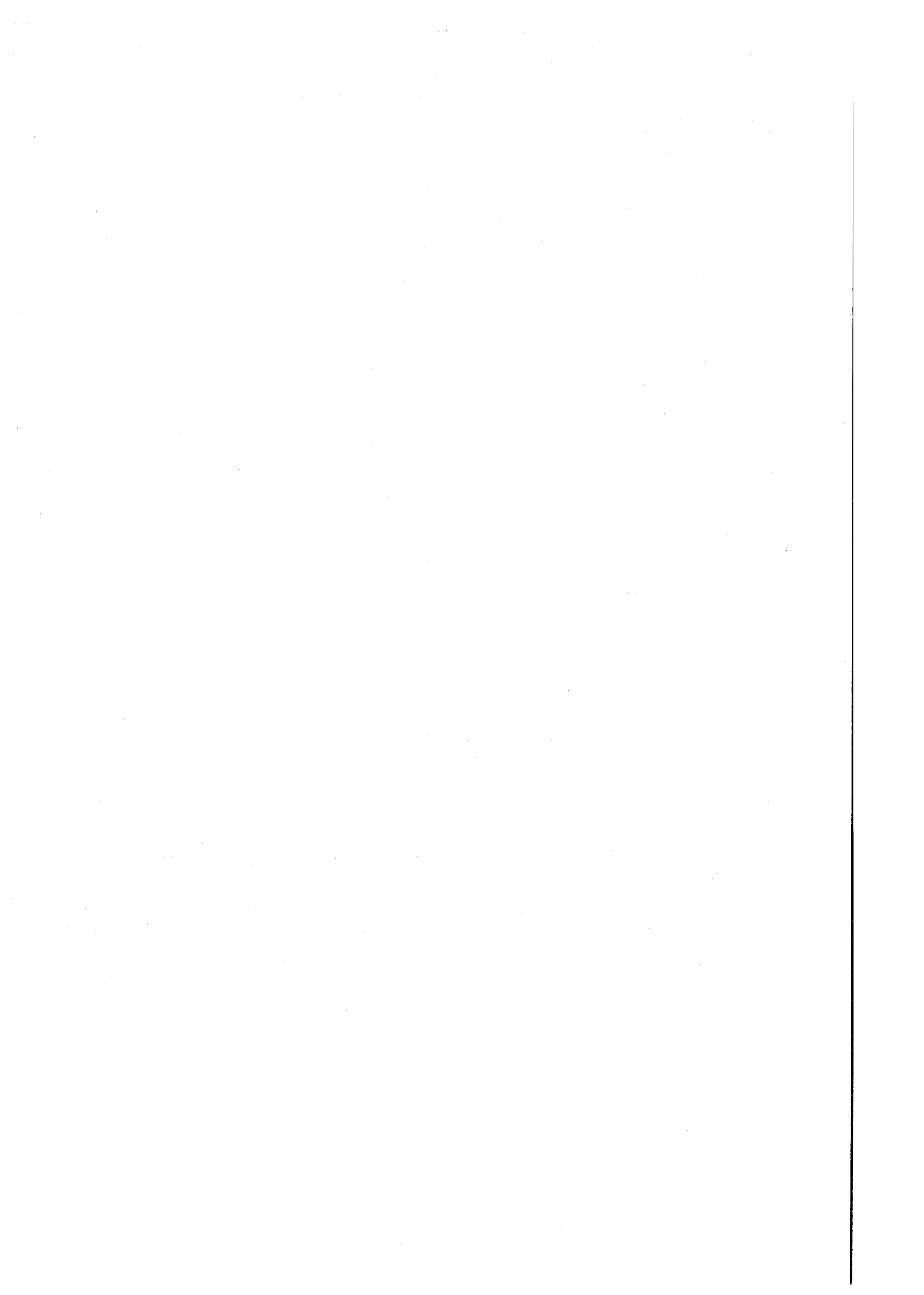
```xml
<?xml version="1.0"?>
<wfs:Transaction
version="1.1.0"
service="WFS"
xmlns="http://www.someserver.com/myns"
xmlns:gml="http://www.opengis.net/gml"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:wfs="http://www.opengis.net/wfs"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.someserver.com/myns
http://www.someserver.com/wfs/cwwfs.cgi?
request=describefeaturetype&amp;typename=InWaterA_1M.xsd
http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd">
  <wfs:Insert>
    <Stompunkter>
      <location>
        <gml:Point gid="e33" srsName="...">
          <gml:pos>-98.5485 24.2633</gml:pos>
        </gml:Point>
      </location>
      <id>23644</id>
      <type>plan</type>
    </Stompunkter>
  </wfs:Insert>
</wfs:Transaction>
```

Exempel 4.4. En WFS-fråga av typen Transaction. Värden på attributen till elementet wfs:Transaction kommer från ett exempel i specifikationen för WFS-standarden [19].

Svaret returneras i form av ett XML-dokument som anger huruvida redigeringarna av data lyckats. Vid insättning skall dessutom objektsidentifierare till de nyinsatta objekten returneras, detta görs i elementet *InsertResults*. Ifall någon operation misslyckades innehåller svaret dessutom elementet *TransactionResponse* där information om felet finns. Hur svaret på frågan i exempel 4.4 skulle se ut om insättningen lyckades visas i exempel 4.5.

```
<?xml version="1.0" ?>
<wfs:TransactionResponse
version="1.1.0"
xmlns:wfs="http://www.opengis.net/wfs"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wfs
../wfs/1.1.0/WFS.xsd">
  <wfs:TransactionSummary>
    <wfs:totalInserted>1</wfs:totalInserted>
  </wfs:TransactionSummary>
  <wfs:InsertResults>
    <wfs:Feature>
      <ogc:FeatureId fid="Stompunkt.1001"/>
    </wfs:Feature>
  </wfs:InsertResults>
</wfs:TransactionResponse>
```

Exempel 4.5. Svar på WFS-frågan i exempel 4.4 om insättningen lyckades. Värden på attributen till elementet wfs:TransactionResponse kommer från ett exempel i specifikationen för WFS-standarden [19].



5 Webbteknologi

En hemsida levereras av en webbserver och visas i en webbläsare. Webbservern är ett program som väntar på en begäran om en hemsida från en webbläsare. Då servern tagit emot begäran skickas hemsidan till webbläsaren som då visar sidan på datorskärmen. Webbläsaren kan även kallas för en klient, vilken kan förklaras som ett program som används för att be webbservrar om hemsidor, tolka svaret och sedan presentera det på datorns skärm [21].

En hemsida som enbart består av HTML är helt statisk. Det innebär att så fort användaren gör något som innebär att sidan som visas ska förändras måste klienten be servern om en ny sida. Detta medför att det blir mycket kommunikation mellan server och klient, vilket belastar nätverket.

Ofta önskas mer dynamiska hemsidor. Det finns då flera alternativ. En lösning är att all dynamik läggs på serversidan. Vid en begäran om en hemsida genererar då servern HTML-kod som sedan skickas till klienten. Att koden genereras vid förfrågan innebär att den alltid är aktuell. En annan fördel är att om det som skickas till klienten fortfarande är ren HTML så kan alla klienter visa informationen på ett korrekt sätt. En nackdel är dock att det fortfarande behövs lika mycket kommunikation mellan klient och server.

För att minska kommunikationen mellan klient och webbserver kan istället dynamiken ligga hos klienten. En nackdel med detta är att det är svårt att skapa dynamik som alla webbläsare klarar av. Ett sätt att lösa detta på är att göra olika varianter av hemsidan, en variant anpassad speciellt för varje typ av webbläsare. Vid första anrop efter sidan tar servern reda på vilken typ av webbläsare som används och sedan returneras "rätt" variant av sidan. Ett tredje alternativ är att kombinera de två teknikerna och placera en del av dynamiken hos servern och en del hos klienten.

5.1 HTML

HTML är ett standardiserat märkspråk (se inledningen till kapitel 3) som används för att skapa hemsidor [22]. Information om sidans utseende ges i textformat i en fil med ändelsen *html* eller *htm*. För att märka upp texten används element och attribut. Det finns regler för vilka element och attribut varje element får innehålla. HTML-dokument har ett rotelement som heter just *html*. Rotelementet innehåller två element, *head* och *body*. Elementet *head* innehåller i sin tur bl.a. information om sidans titel dvs. den text som visas högst upp i fönstrets ram. Elementet *body* innehåller information om allt det som visas på sidan. Sedan finns en mängd element som används för att tala om hur texten på sidan skall presenteras samt för att infoga t.ex. bilder.

De flesta element i HTML ska ha start- och sluttagg. I vissa fall behövs dock ingen sluttagg. Exempel på sådana element är element för bilder och radbrytning, dvs. tomma element som inte kan innehålla ytterligare element eller text mellan start- och sluttagg. I nästa version av HTML kommer det att vara obligatoriskt att använda sluttagg för alla element [22], därför kan det vara en bra ide att redan nu ta för vana att alltid använda sluttagg.

För att skapa ett HTML-dokument kan en vanlig texteditor användas. Det finns dock flera mer avancerade program som är anpassade för att skapa just HTML-dokument. Ett exempel på ett sådant program är *Visual studio*. Där är det möjligt att genom att välja designläge se hur sidan kommer att se ut och även göra ändringar i utseendet därifrån. För den som är ovan att

programmera kan ett sådant program vara till stor hjälp vid byggandet av hemsidor. Nackdelen är att det är svårt att lära sig använda HTML genom att enbart använda designläget.

5.2 Programmering

Programmering används för att få t.ex. en dator att utföra en rad uppgifter vid ett visst kommando eller en viss händelse. Programmet kan kommunicera med användaren genom att be användaren om information och sedan använda denna information i det fortsatta arbetet.

Datorns processor förstår enbart maskinspråk som är en binär form av kod. Det innebär att kommandon till datorn måste ges som en följd av ettor och nollor. Att konstruera ett sådant kommando skulle vara mycket svårt, om inte omöjligt, för de flesta människor. Därför har programmeringsspråk utvecklats där programmeraren kan skriva kommandona på ett lättare sätt. När programmet är färdigskrivet måste det kompileras innan det kan köras (exekveras). Kompileringen innebär att kommandona översätts till ettor och nollor som sedan kan förstås av datorns processor [23].

Oavsett vilket programmeringsspråk som används är det mycket viktigt att helt följa den struktur som koden skall skrivas i. Att missa ett tecken innebär att det blir fel. Kompilatorn som ska översätta koden förstår då inte kommandona den ska översätta och meddelar det. Programmeraren får då hitta och rätta felet för att sedan försöka kompilera igen. Att ett program går att kompilera behöver inte nödvändigtvis vara samma sak som att programmet är korrekt. Det finns nämligen olika typer av fel. Dels skrivfel som kompilatorn hittar men också logiska fel. Logiska fel är fel som inte bryter mot den struktur i vilken koden ska skrivas men som ändå leder till ett oönskat resultat. Om ett program innehåller logiska fel hittas inte de av kompilatorn. Förhoppningsvis upptäcks de dock vid provkörning av programmet.

Både för att kompilera och för att exekvera ett program behövs det program som förstår den kod som skrivits. Eftersom koden skrivs på olika sätt i olika språk behövs en särskild kompilator som förstår just det språket för att kompilera ett program. På samma sätt måste det program som används för att exekvera koden vara anpassad för aktuellt språk. Båda dessa program ingår i språkets Software Development Kit (SDK) [23]. Dessutom ingår någon form av dokumentation av det klassbibliotek som hör till språket. Klassbiblioteket innehåller ett antal färdiga klasser som kan användas vid programmeringen.

5.2.1 Java

Java är ett objektorienterat programmeringsspråk som har funnits sedan 1995 [24]. Att språket är objektorienterat innebär att man försöker modellera den del av verkligheten som programmet rör och skapa en klass för var och en av de saker man vill modellera.

Om exempelvis ett program för kontaktuppgifter till kompisar skall skrivas skulle först en klass *Person* skapas som modellerar en person. Sedan skapas en lista som kan innehåll instanser, objekt, av klassen *Person*. För varje ny kompis kan en ny instans av klassen *Person* skapas och läggas till i listan. Till varje klass hör ett antal attribut, såsom namn och telefonnummer, samt ett antal metoder. Exempel på metoder som skulle kunna finnas i klassen *Person* är *getNumber()* och *setNumber()* för att ta reda på respektive ändra telefonnumret till en person.

För att kunna köra ett program räcker det inte med att skriva ett antal klasser som beskriver olika objekt. Programmet måste ha en main-metod där objekt skapas och anropas. Denna main-metod är unik för varje program medan de klasser som skrivs för ett program även kan användas av andra program. Det leder till att mängden programkod minskas i och med att samma kod kan användas flera gånger genom enkla anrop.

I Javas klassbibliotek finns en mängd klasser som kan användas rakt av. Vad klasserna har för attribut och metoder finns att läsa i dokumentationen på Javas hemsida (nås via http://java.sun.com/). Förutom färdiga klasser innehåller klassbiblioteket även abstrakta klasser och interface.

En abstrakt klass måste ärvas för att kunna användas. Det beror på att abstrakta klasser inte innehåller en komplett implementation. Även vanliga, icke-abstrakta klasser kan ärvas. Genom att ärva en (abstrakt) klass kan klassen utökas med önskad funktionalitet samtidigt som grunden i den ärvda klassen kan användas. En klass som ärver kallas subklass medan klassen som ärvs kallas superklass. Om superklassen innehåller metoder som inte fungerar på önskat sätt kan dessa metoder överskuggas (*overriding*) [25]. Det innebär att en ny metod med samma namn och parametertyper, men med en annan funktionalitet, skapas i subklassen. Alla eventuella abstrakta metoder i superklassen måste implementeras i subklassen.

Inte heller interface kan användas direkt som de är. Ett interface innehåller ett antal metoder med namn och parametertyper, men ingen av dessa metoder är implementerade. Ett interface kan därför inte ärvas utan implementeras istället. En klass som implementerar ett interface måste implementera alla interfacets metoder. Medan en klass bara kan ärva från en superklass kan samma klass implementera flera interface. Detta kan vara en fördel om den klass som skrivs ska användas i flera olika sammanhang.

I Java brukar klasser och interface ges namn som börjar med en stor bokstav. Variabler som refererar till en instans av en klass brukar istället få ett namn som börjar med en liten bokstav. Detsamma gäller namn på attribut och metoder i klasser. Det är inget som hindrar att en programmerare bryter mot detta. Men eftersom Java gör skillnad mellan stora och små bokstäver, dvs. *a* är inte samma sak som *A*, underlättar det att alltid skriva på det sättet.

5.2.2 C#

C# är ett objektorienterat programmeringsspråk som har utvecklats av Microsoft. Den första versionen av språket släpptes i juli 2000 och i september samma år satte **ECMA** samman en grupp för att utveckla en standard för språket [26]. Enligt standarden känner programmerare som är vana vid språken C eller C++ sig hemma även i C#. Språket är även likt Java. Vissa skillnader i syntax finns, exempelvis *using* istället för *import* vid importering av klasser, men stora delar är lika.

I C# finns stöd för att skapa avancerade användargränssnitt. Förutom enklare delar såsom textboxar och rullgardinsmenyer finns även stöd för att skapa menyer med undermenyer [27]. Genom att använda programmet *Visual Studio* vid implementeringen kan användargränsnittet skapas i ett designläge. Kod för att skapa de olika kontrollerna genereras då automatiskt i C#-filen. Även händelsehantering kan anges utifrån designläget. På så sätt blir också händelsehanteringen lättare att implementera i C# jämfört med i exempelvis Java.

5.3 Dynamik på serversidan

Genom att låta servern genera HTML-kod vid klientanrop kan en hemsida göras mer dynamisk samtidigt som det undviks att krav ställs på klienterna. Detta är ett bra alternativ i de fall då det är viktigare att alla klienter kan visa sidan på ett korrekt sätt än att minska belastningen på servern. Det finns flera olika metoder att använda sig av för att skapa dynamik på serversidan. Några av dessa kommer här att beskrivas kortfattat medan en av teknikerna, *servlets*, beskrivs mer ingående.

Den första tekniken som beskrivs heter Active Server Pages (**ASP**) och är en Microsoft-teknologi. ASP är till stor del detsamma som vanlig HTML. Skillnaden är, förutom filtypen, att en ASP-fil kan innehålla skript som körs på servern. Det som returneras till klienten är dock ren HTML-kod. Skripten läggs i speciella taggar, i exempel 5.1 visas hur en sådan tagg som skriver ut texten "Hello world!" på sidan ser ut. En fördel med ASP är att klienten aldrig kan se ASP-koden utan enbart den HTML-kod som genereras vid anrop från en klient [28].

```
<%response.write("Hello World!")%>
```

Exempel 5.1. En skripttagg i ASP.

ASP.NET är en nyare version av ASP [29]. Den nya versionen stöder fler språk, förutom skriptspråk finns nu möjlighet att skriva kod i t.ex. C# eller C++. Koden kan då läggas i en separat fil. Första gången en klient frågar efter sidan kompileras koden innan den körs. Nästa gång sidan frågas behöver inte koden kompileras. Användargränssnittet finns i en fil med ändelsen *aspx*. Denna fil kan även innehålla funktionalitet. Om funktionaliteten skrivs i språket C# och läggs i en separat fil skapas en klass som representerar sidan och bland annat innehåller en metod *PageLoad()*. Denna metod anropas varje gång en klient frågar efter sidan. Klassen kan även innehålla metoder som anropas vid vissa händelser, som t.ex. då användaren klickar på en knapp. En fördel med ASP.NET och C# är att det finns ett stort klassbibliotek som bl.a. kan användas för att skapa webb- och HTML-kontroller (se även avsnitt 5.2.2). Ett exempel på en webbkontroll är objektet *Repeater* som kan användas för att visa svar från en databasfråga på ett enkelt sätt. Från en metod i C#-filen kan en tabell skapas, t.ex. genom att ställa en fråga till en databas, och sedan kan tabellen anges som källa till *Repeater*-objektet. Genom att i aspx-filen, som innehåller användargränssnittet, ange hur en rad av tabellen ska visas gör sedan objektet så att detta automatiskt repeteras för varje rad i tabellen.

En annan typ av teknik är Common Gateway Interface (CGI). CGI är ett program som exekveras på servern i realtid vilket medför att en dynamisk sida returneras. Vilket språk som används för att skriva programmet är valfritt och såväl skript- som programmeringsspråk kan användas. Då ett CGI-program skrivs är det viktigt att tänka på att det ska gå snabbt att köra så att inte klienten behöver vänta för länge på svar [30].

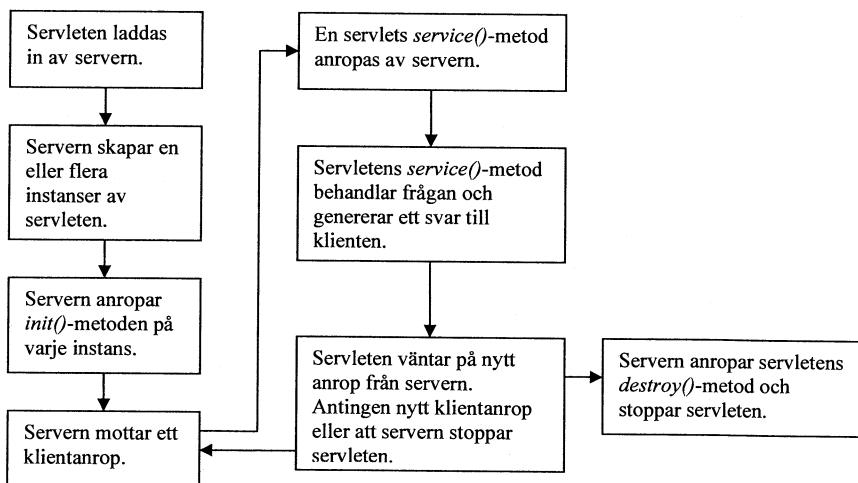
5.3.1 Servlets

En servlet är ganska lik ett CGI-program men är alltid skriven i programmeringsspråket Java. Servleten får anrop från klienter, bearbetar anropet och genererar ett svar. Svaret kan exempelvis bestå av en HTML-sida. En fördel med att använda servlets istället för ett CGI-program är att ett CGI-program startas som en ny process för varje klientanrop vilket inte en

servlet gör. Det medför att servlets blir effektivare. En annan fördel med servlets är att de är plattformsoberoende, vilket inte CGI är [31].

En servlet är en klass som implementerar interfacet *Servlet*. Det är dock inte helt lätt att implementera interfacet från grunden. För att göra det enklare att skriva en servlet innehåller Javas klassbibliotek två abstrakta klasser, *GenericServlet* och *HttpServlet*, som implementerar interfacet. Det medför att en servlet kan skrivas genom att ärva någon av dessa klasser, vilket också är det vanligaste sättet att skriva en servlet på [32].

Hur en servlets livscykel ser ut visas i figur 5.1. Först laddar servern in servleten och skapar en instans av den. Eventuellt skapas flera instanser, i så fall görs följande på var och en av instanserna. Metoden *init()* anropas för att initiera servleten. Metoden finns i superklassen men kan överskuggas för att anpassas till just denna servlet. Ett exempel på vad som kan göras i metoden är att skapa en databasuppkoppling så att samma uppkoppling kan användas vid varje nytt klientanrop. Vid ett anrop från en klient anropas metoden *service()* på en instans av servleten. Klientanropet behandlas och ett svar genereras till klienten. Servlet-objektet är då redo att ta emot ett nytt klientanrop. Förloppet fortsätter till dess att servern stoppar servlet-objektet och anropar metoden *destroy()* på den. Även denna metod finns i superklassen och kan överskuggas. Om en databasuppkoppling skapades i metoden *init()* är det lämpligt att överskugga även *destroy()* för att stänga uppkopplingen.



Figur 5.1. En servlets livscykel. Baserat på Inside Servlets [32] sidan 83.

Vad servleten skall göra vid ett klientanrop bestäms i metoden *service()*. Metoden kan överskuggas, men om den inte överskuggas kommer den anropa andra metoder beroende på vilken typ av klientanrop det är. Om anropet till exempel är av typen HTTP GET kommer *doGet()* att anropas och är anropet av typen HTTP POST kommer *doPost()* anropas. Det kan

därför vara lämpligt att istället överskugga dessa metoder om man vill skilja på de olika typerna av anrop [33].

Till *service()*-metoden ges två inparametrar. Den första parametern, *request*, används för att få information om frågan, dvs. den innehåller all information som klienten skickade till servern. Den andra parametern, *response*, används för att generera ett svar till klienten. Innan svaret genereras bestäms svarets format genom ett anrop av metoden *setContentType()* på objektet *response*.

Som nämnts tidigare i avsnittet behövs en webbserver för att skapa instanser av den servlet-klass som implementerats samt för att anropa instansens *init()*-, *service*- och *destroy*-metod. Det är dock inte alla webbservrar som klarar av att köra servlets. *Apache **Tomcat*** är ett exempel på en gratis webbserver som gör det [34].

5.4 Dynamik på klientsidan

Det finns flera olika tekniker för dynamik på klientsidan. Två av dessa är skript och applets. HTML innehåller två olika element för att infoga skript eller applets till sidan.

Det finns flera skriptspråk att välja mellan, elementet *script* används för dem alla. Vilket språk som används anges med hjälp av attributet *language*. Skriptspråk kan ses som en enklare form av programmering. Dels är de enklare eftersom det inte är lika mycket att lära sig för att komma igång med att skriva kod och dels kan de anses enklare eftersom det inte krävs någon kompilering innan skriptet kan köras. Ett skriptspråk är dock inte lika kraftfullt och har inte samma möjligheter som t.ex. Java.

Applets är skrivna i programmeringsspråket Java (se avsnitt 5.2.1). Det är kompilerad kod som skickas till klienten och exekveras där [35]. Med elementet *applet* anges i HTML-dokumentet vilken fil som skall hämtas och köras. Ligger program-filen på en annan server än hemsidan anges även det med hjälp av attribut. Dessutom används attribut för att bestämma storleken på en ruta där appleten ska visas. En nackdel med applets är dock att det krävs att klienten har en Java™ Plug-in för att programmet ska kunna exekveras.

5.4.1 JavaScript

Huvuddelen av det som tas upp i detta avsnitt tas även upp i W3Cs handledning för JavaScript [36].

JavaScript kan infogas i HTML-dokument genom att använda elementet *script* samt ange elementets attribut *language* till *JavaScript*. Allt som står mellan elementets start- och sluttagg skall då vara JavaScript-kod. Det finns ingen begränsning för hur många *script*-element som kan finnas på en sida och de kan ligga såväl i dokumentets *head* som i dess *body*.

En av de enklare saker som kan göras med JavaScript är att skriva ut text i webbläsaren. Detta görs med hjälp av variabeln *document* som refererar till webbläsarens Document Object Model (DOM). Variabeln är alltid tillgänglig och det går bra att anropa metoder såsom *write()* för att t.ex. skriva ut text.

Det finns möjlighet att skapa egna variabler och funktioner. Genom att skapa en funktion som anropas vid en viss händelse, till exempel vid klick på en knapp, kan sidan göras mer

dynamisk. Med hjälp av en rullgardinsmeny, en knapp, en funktion som anropas vid knapptryck samt variabeln *document* kan användaren t.ex. ändra bakgrundsfärg på sidan utan att klienten behöver kommunicera med servern. Hur detta görs visas i exempel 5.2.
För att inte äldre webbläsare som inte förstår JavaScript ska skriva ut skriptet som text i webbläsaren kan HTML kommentars-element användas. Även detta visas i exempel 5.2.

```
<html>
  <head>
    <title>Byt bakgrundsfärg</title>
    <script language="javascript">
      <!--
        function changeBgColor(){
          document.body.bgColor =
          document.getElementById("color").value;
        }
      //-->
    </script>
  </head>
  <body>
    <form>
      <select id="color">
        <option value="blue">Blå
        <option value="green">Grön
        <option value="yellow">Gul
        <option value="red">Röd
        <option value="white">Vit
      </select>
      <input type="button" value="Byt färg!"
       onclick="changeBgColor()">
    </form>
  </body>
</html>
```

Exempel 5.1. Exempel på en JavaScript-funktion som ändrar dokumentets bakgrundsfärg.

Ett alternativ till att lägga JavaScript direkt i HTML-dokumentet är att lägga det i en separat fil med ändelsen *js*. Genom att lägga in ett *script*-element i HTML-dokumentet och låta attributet *srs* peka på JavaScript-filen kan skripten nås från dokumentet. En fördel med detta är att flera HTML-dokument kan använda sig av samma skript. En annan fördel är att det går lätt att byta vilka skript som används samt att HTML-dokumentet blir mer lättläst då det inte ligger några skript inbäddade i HTML-koden.

En nackdel med JavaScript är att även om de nyare webbläsarna klarar av att tyda JavaScript så fungerar skripten inte helt lika i alla webbläsare. Ibland kan det därför vara nödvändigt att ta reda på vilken typ av webbläsare som används och länka vidare till en ny sida som är anpassad för just den typen av webbläsare. Anledningen till kompabilitetsproblemen är att JavaScript ursprungligen utvecklades av Netscape och att Microsoft har sin egen version som kallas JScript. Det finns visserligen en standard, *ECMAScript Edition 3*, som både JavaScript

och JScript implementerar, men ändå finns vissa skillnader och språken är inte helt kompatibla [37][38].

6 Databaser

En databas används för lagring av data på ett sätt som gör det enkelt att lägga till, uppdatera och hämta information. Oavsett vilken databas som används (t.ex. Oracle eller Microsoft **SQL** Server) är principerna de samma. Exempelvis uppdateringar görs på samma sätt med undantag för mindre skillnader i syntax.

En databas består av ett antal tabeller där varje tabell i sin tur består av poster (rader) och kolumner [2]. Området där en post och en kolumn möts kallas för fält. Om en tabell t.ex. innehåller information om adresser motsvarar varje post en adress. Vidare innehåller varje fält en del av den information som finns om adressen, t.ex. postnummer (figur 6.1).

Kolumn

Fält

Post

Gatunamn	Gatunummer	Postnummer	Postort
Nygatan	2	22229	Lund
Bangatan	7	22221	Lund
Kungsgatan	40	21149	Malmö
Kanalgatan	6	24130	Eslöv

Figur 6.1. En tabell består av ett antal poster som i sin tur innehåller ett antal fält.

Det finns en mängd olika datatyper som kan användas för att lagra data i tabellerna. Vilka datatyper som finns kan dock skilja sig något mellan de olika databaserna, exempelvis representeras datum och tid med en datatyp i vissa databaser och med två datatyper i andra. En beskrivning av de vanligaste datatyperna finns i tabell 6.1. Vilken datatyp som skall användas bestäms då en ny kolumn skapas eftersom alla fält i en viss kolumn måste ha samma datatyp. För att lagra text kan t.ex. datatypen *char* användas. Rent tekniskt skulle den datatypen kunna användas för nästan all sorts data, dock inte för objekt. Men det görs normalt inte eftersom en tydligare specifikation om vilken typ av data som ska lagras i en viss kolumn motverkar att fel sorts data läggs in i tabellen. För att lagra postnummer vore det därför lämpligare att använda datatypen *int* (heltal), medan det för datum går att specificera ännu noggrannare och använda just datatypen *date*.

Datatyp	Beskrivning
Char(n)	En textsträng med längden *n*, fylls ut med blanksteg.
Varchar(n)	En textsträng med varierande längd, dock maximalt *n* tecken.
Boolean	Har värdet *true*, *false* eller *unknown*. Finns inte i t.ex. Oracle.
Int, integer	Heltal.
Float, real	Flyttal.
Decimal(n,d)	Reellt tal med *n* siffror varav *d* decimaler.
Date, time	Datum och tid. Kan ibland representeras med en datatyp som innehåller både datum och tid.
Blob	”Binary large objects” t.ex. bilder.
Clob	”Character large objects” t.ex. referens till en fil.

Tabell 6.1. De vanligaste datatyperna.

Förutom att ange vilken typ av data som skall lagras i en viss kolumn måste det för vissa datatyper (de för text) även anges hur många tecken som skall få plats i varje fält. Här fungerar de olika datatyperna på olika sätt. Datatypen *char* som nämndes i föregående stycke får alltid den längd som angetts oavsett vilken längd det attribut som läggs in har. Om t.ex. kolumnen gatunamn tilldelats datatypen *char(20)* och ”Nygatan” läggs till kommer det som lagras i fältet ha längden 20 och inte 7. Anledningen till detta är att datatypen *char* fungerar så att textsträngen fylls ut med blanksteg tills den specificerade längden uppnås. Detta kan medföra problem vid sökning. Antag att alla adresser med gatunamn ”Nygatan” skall sökas. För att hitta dessa adresser krävs då att söksträngen fylls ut med rätt antal blanksteg innan jämförelsen görs. För att undvika problemet kan istället datatypen *varchar* användas. Om datatypen för en kolumn sätts till *varchar(20)* innebär det att längden kan vara maximalt 20 tecken. Om det är en kortare textsträng som läggs in fylls inte fältet ut med blanksteg utan behåller den ursprungliga längden.

Varje post i en tabell ska vara unik så att den kan identifieras. För att säkerställa att så är fallet måste varje tabell ha en primärnyckel [39]. Om det finns en kolumn, t.ex. personnummer, som gör varje post unik kan den kolumnen användas som primärnyckel. Ibland kan det vara så att det inte finns någon naturlig primärnyckel. Det finns då två alternativ. Antingen kan flera kolumner tillsammans fungera som primärnyckel, förutsatt att de tillsammans gör varje post unik. Det andra alternativet är att skapa en extra kolumn vars enda uppgift är att vara primärnyckel.

All information skulle kunna lagras i en enda tabell, men detta skulle troligen leda till redundans. Vi återgår till exemplet om adresser. Om det finns flera adresser med samma postnummer skulle vi vara tvungna att lagra vilken postort som hör till det postnumret flera gånger, dvs. vår tabell innehåller redundans (tabell 6.2). Redundans är ett problem på flera sätt. Det tar extra lagringsutrymme och i händelse av en uppdatering av den dubbellagrade informationen måste samma uppdatering göras på flera ställen. Risken finns att man missar att uppdatera på något ställe och då innehåller databasen istället motstridiga uppgifter. Dessutom kan redundans i en tabell vara ett hinder för lagring av data [2], med tabell 6.2 kan inte ett postnummer lagras utan att lagra en adress som hör till postnumret. För att undvika redundans i det här exemplet kan informationen delas upp i två tabeller, en med gatunamn, gatunummer och postnummer samt en med postnummer och postorter (tabell 6.3). Önskas hela adresser

skrivas ut kan tabellerna kopplas samman temporärt med hjälp av kolumnen postnummer, mer om hur detta görs finns att läsa i avsnitt 6.2.

Gatunamn	Gatunummer	Postnummer	Postort
Nygatan	2	22229	Lund
Bangatan	5	22221	Lund
Bangatan	7	22221	Lund
Kungsgatan	40	21149	Malmö
Kanalgatan	6	24130	Eslöv

Tabell 6.2. Exempel på en tabell som innehåller redundans. Informationen om att postnumret 22221 hör till Lund lagras två gånger.

Gatunamn	Gatunummer	Postnummer
Nygatan	2	22229
Bangatan	5	22221
Bangatan	7	22221
Kungsgatan	40	21149
Kanalgatan	6	24130

Postnummer	Postort
22229	Lund
22221	Lund
21149	Malmö
24130	Eslöv

Tabell 6.3. Genom att dela upp informationen i tabell 6.2 i två tabeller undviks redundans.

För att säkerställa att det inte av misstag läggs till någon adress med ett postnummer som inte finns med i tabellen med postorter kan främmande nycklar användas. En främmande nyckel är en kolumn som refererar till en primärnyckel i en annan tabell [39]. Antag att kolumnen postnummer i tabellen med postorter från det tidigare exemplet är en primärnyckel. Då kan kolumnen postnummer i tabellen adresser bli en främmande nyckel som refererar till postnummer i tabellen med postorter. Användandet av främmande nycklar gör det nu omöjligt att lägga till en adress med ett nytt postnummer utan att först lägga till motsvarande postnummer i tabellen med postorter. Om istället en post i postortstabellen tas bort kommer de poster i adresstabellen som refererar till denna också att försvinna.

6.1 Normalformer

Normalformer bygger på funktionella beroenden [2]. Ett funktionellt beroende innebär att om värdena på ett antal fält är givna så kan även värdena på ytterligare fält bestämmas entydigt. Funktionella beroenden kan indelas i triviala och icke-triviala funktionella beroenden. Ett trivialt funktionellt beroende är t.ex. att gatunamn och gatunummer entydigt bestämmer gatunummer. Detta är trivialt eftersom gatunummer finns med på båda sidor i beroendet, dvs. högerledet är en del av vänsterledet. Om inte detta gäller är det ett icke-trivialt funktionellt beroende. Ett exempel på ett icke-trivialt funktionellt beroende är att om gatunamn, gatunummer samt postnummer på en adress är givet så kan adressens postort bestämmas

entydigt. Hur detta skrivs visas i exempel 6.1. Eftersom en primärnyckel har som uppgift att göra varje post unik bestämmer den entydigt värdena i övriga fält.

```
gatunamn, gatunummer, postnummer -> postort
```

Exempel 6.1. Ett funktionellt beroende.

Normalformer används för att tala om hur väl utformad databasen är bl.a. avseende redundans. Ju högre nivå av normalform som databasen uppfyller desto bättre. I detta avsnitt kommer endast de två lägsta normalformerna att beskrivas.

6.1.1 Första normalformen

Första normalformen innebär att varje fält endast får innehålla ett värde samt att varje post måste vara unik [40].

6.1.2 Andra normalformen

För att uppfylla den andra normalformen krävs att varje fält i tabellen som inte är en del av primärnyckeln måste bero på hela primärnyckeln. Om tabellens primärnyckel endast består av ett fält är det villkoret automatiskt uppfyllt, men i händelse av en sammansatt primärnyckel kan det bli så att villkoret inte uppfylls. Detta löses genom att tabellen delas upp i flera tabeller. Dessutom krävs att den första normalformen är uppfylld [40].

Ett exempel på en tabell som inte uppfyller andra normalformen är tabell 6.4 där primärnyckeln består av fälten *Studentid* och *Kurskod*. Anledningen till att andra normalformen inte uppfylls är att *Kurskod*, som endast är en delmängd av primärnyckeln, ensam ger *Kursnamn*. Om tabellen delas upp i två tabeller enligt tabell 6.5 uppfylls däremot den andra normalformen.

Studentid	Kurskod	Kursnamn	Betyg
100-12	EDA110	Algoritmteori	4
100-12	FMA420	Linjär algebra	4
100-25	EDA110	Algoritmteori	3
100-34	EDA110	Algoritmteori	4

Tabell 6.4. En tabell som inte uppfyller andra normalformen.

Studentid	Kurskod	Betyg
100-12	EDA110	5
100-12	FMA420	4
100-25	EDA110	3
100-34	EDA110	4

Kurskod	Kursnamn
EDA110	Algoritmteori
FMA420	Linjär algebra

Tabell 6.5. Efter uppdelning i två tabeller uppfylls andra normalformen.

6.2 SQL – Structured Query Language

SQL är en ISO-standard för hur kommandon till en databas ges. Kommandona kan röra såväl skapande av tabeller och insättning/uppdatering/borttagning av data som sökning av information. Nedan följer en enklare genomgång av språket, för att lära sig mer kan t.ex. W3Cs handledning användas [41].

6.2.1 Skapa tabell

Då en ny tabell skapas anges vilka kolumner som ska finnas i tabellen samt villkor för dessa (se exempel 6.2). Exempel på villkor kan vara att fält alltid måste fyllas i eller att alla heltal i en kolumn måste vara positiva. Det går även att ange default-värden för en kolumn, dvs. ett värde som fältet i en ny post får om inget annat anges. Dessutom måste primärnyckel och ev. främmande nyckel anges.

```
CREATE TABLE tabellnamn
(lista av definitioner för kolumner och nycklar)
```

Exempel 6.2. Kommando för att skapa en tabell.

6.2.2 Sätta in ny post

Då en ny post läggs till i en tabell kan det väljas ifall värden för alla fält skall anges eller inte. Om värden för alla fält anges behöver inte kommandot innehålla kolumnnamnen. Det kan ändå vara bra att ange kolumnnamnen för att minska risken för fel. Om inte värden för alla fält anges måste alltid kolumnnamnen finnas med i kommandot. Ett värde kan även sättas till default (kräver att ett default-värde för kolumnen har angetts) eller null. I exempel 6.3 visas strukturen för kommandot för insättning av en ny post.

```
INSERT INTO tabellnamn (kolumnlista)
VALUES ( DEFAULT | NULL | värden )
```

Exempel 6.3. Kommando för att sätta in en ny post i en tabell.

6.2.3 Uppdatera post

För att specificera vilken/vilka poster som skall uppdateras används ett WHERE-villkor. Om alla poster ska uppdateras kan villkoret uteslutas. Strukturen för kommandot som uppdaterar poster i en tabell visas i exempel 6.4.

```
UPDATE TABLE tabellnamn
SET kolumnnamn = ( DEFAULT | NULL | värde )
WHERE villkor
```

Exempel 6.4. Kommando för att uppdatera en post i en tabell.

6.2.4 Ta bort post

Även vid borttagning av poster används ett WHERE-villkor för att tala om vilken/vilka poster som ska tas bort. Anges inget WHERE-villkor tas alla tabellens poster bort. Hur strukturen för ett kommando som tar bort poster ser ut visas i exempel 6.5.

```
DELETE FROM tabellnamn
WHERE villkor
```

Exempel 6.5. Kommando för att ta bort poster ur en tabell.

6.2.5 Uppdatera tabell

En tabell kan uppdateras genom att lägga till eller ta bort kolumner. Oavsett om kolumner skall tas bort eller läggas till inleds instruktionen med kommandot ALTER TABLE. Borttagning anges sedan med kommandot DROP (se exempel 6.6) medan insättning anges med kommandot ADD (se exempel 6.7).

```
ALTER TABLE tabellnamn
DROP kolumnnamn
```

Exempel 6.6. Kommando för att ta bort en kolumn från en tabell.

```
ALTER TABLE tabellnamn
ADD kolumnnamn datatyp
```

Exempel 6.7. Kommando för att lägga till en kolumn till en tabell.

6.2.6 Ta bort tabell

Kommandot *drop table* tar bort en hel tabell samt hela dess innehåll (se exempel 6.8).

```
DROP TABLE tabellnamn
```

Exempel 6.8. Kommando för att ta bort en tabell.

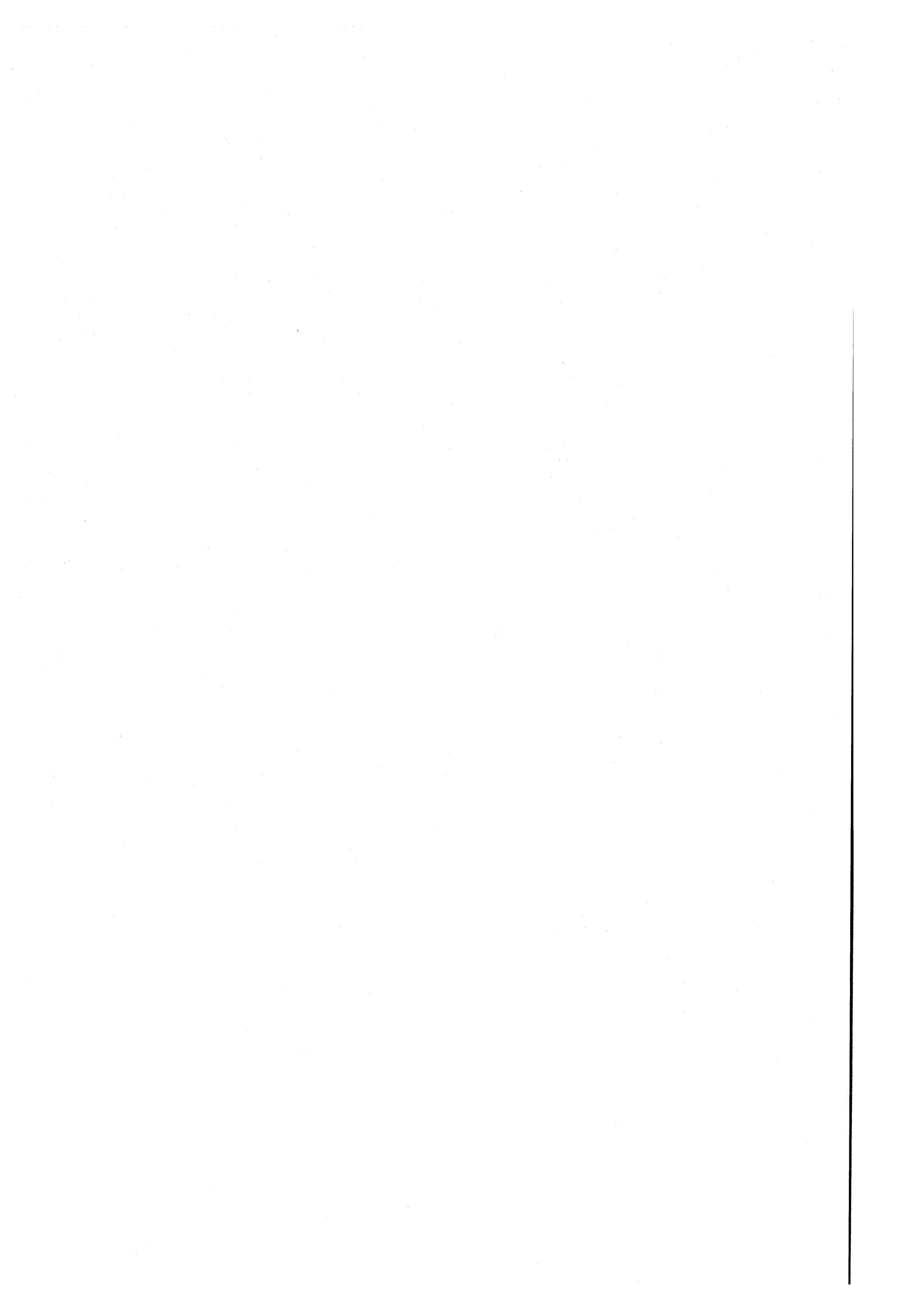
6.2.7 Visa information

De tidigare kommandona har endast behandlat data i *en* tabell, men vid visning av information kan flera tabeller kopplas samman. Det enda som måste ingå i kommandot är vilka kolumner (SELECT) som skall visas och från vilka tabeller informationen skall hämtas (FROM). Sedan kan WHERE-villkor användas för att även specificera från vilka poster informationen skall hämtas. WHERE-villkoret kan bestå av flera villkor med AND eller OR emellan. Dessutom går det att med ORDER BY ange hur resultatet skall sorteras (efter vilken kolumn). I exempel 6.9 visas strukturen för ett kommando som hämtar information från tabeller i databasen.

```
SELECT tabellnamn.kolumnnamn
FROM tabellnamn
WHERE villkor
ORDER BY sorteringsvillkor
```

Exempel 6.9. Kommando för att hämta information från en tabell.

Det går även att koppla samman tabeller på olika sätt (temporärt) och det finns ett flertal funktioner som kan användas för att sammanställa data på olika sätt. Hur detta går till beskrivs dock inte i denna rapport.

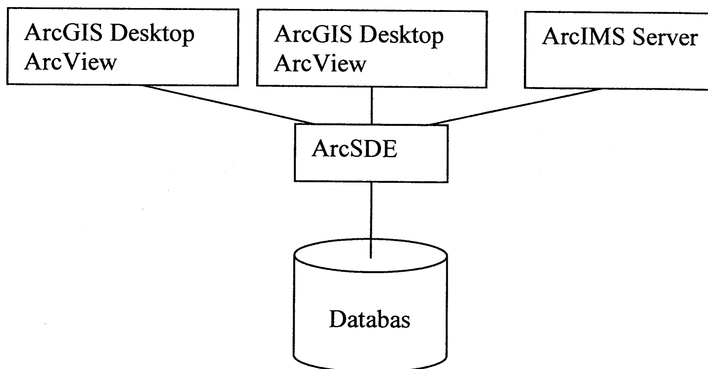


7 Programvaror – ESRI

ESRI, Environmental Systems Research Institute, grundades 1969 och är idag ett av världens största företag inom geografisk informationsteknik. 1982 lanserade ESRI sitt första kommersiella GIS-program ARC/INFO. Programmet visade geografiska objekt och innehöll en databashanterare för att kunna hantera attribut till objekten, programmet var därmed det första moderna GIS-programmet [42]. Från början av 90-talet och fram till idag har flera nya programvaror i Arc-familjen lanserats. Två av dessa är ArcSDE och ArcIMS, vilka beskrivs i kommande avsnitt.

7.1 ArcSDE

ArcSDE är en spatial dataserver som används för att klienter, som exempelvis ArcIMS, skall kunna komma åt och hantera geografiska data lagrade i en relationsdatabas (se figur 7.1). Olika typer av relationsdatabaser kan användas, funktionaliteten och kapaciteten är densamma oavsett vilken som används [43]. Två exempel på databaser som kan användas är Microsoft SQL Server och Oracle. Den första versionen av ArcSDE lanserades 1994 [43]. I detta avsnitt beskrivs version 9.1.



Figur 7.1. Klienter kan kommunicera med databasen via ArcSDE. Baserat på ArcGIS® 9 What is ArcGIS® 9.1? [44] sidan 57.

Om data lagras i flera olika databaser underlättar det att använda ArcSDE. Eftersom olika databaser har olika datatyper och fungerar lite olika kan inte kommunikationen med två olika databaser ske på exakt samma sätt. Med ArcSDE som mellanlager behöver inte klienten ge anrop på olika sätt beroende på från vilken databas informationen kommer från utan kommunikationen kan skötas på ett enhetligt sätt. ArcSDE sköter sedan kommunikationen med respektive databas och ser till att rätt datatyper används. All typ av GIS-data stöds av ArcSDE oavsett vilken typ den underliggande databasen är av. Dessutom klarar ArcSDE av mycket stora datavolymer liksom långa transaktioner och många samtidiga användare [44].

Förutom att använda något program från ESRI mot ArcSDE finns även stöd för att utveckla egna program [44]. Genom en **API** för programmeringsspråken Java och C ges möjlighet att komma åt de underliggande rumsliga tabellerna. Detta gör att ArcSDE är mycket flexibelt.

7.1.1 Java API till ArcSDE

Avsnittet baseras på dokumentationen till ESRI:s Java API version 8.3 [45].

Genom ESRI:s Java API är det möjligt att kommunicera med ArcSDE med hjälp av Java-objekt för att exempelvis söka och uppdatera information i databasen. Först måste en koppling till ArcSDE skapas, detta görs genom att skapa ett objekt av typen *SeConnection*. Som inparametrar till konstruktorn anges förutom vilken databas som skall anslutas till även ett användarnamn och lösenord. Dessa är samma inloggningsuppgifter som används för att kunna uppdatera databasen via exempelvis ArcCatalog. Efter att uppkopplingen skapats kan information sökas och såväl tabeller som den information som lagras i tabellerna kan redigeras.

För att representera ett kartlager finns två klasser, *SeTable* och *SeLayer*. *SeTable* är ett objekt som används för att skapa, redigera, ta bort eller fråga en databastabell genom ArcSDE medan *SeLayer* är ett objekt som hör till en databastabell men definierar en rumslig kolumn till den. Finns det inte redan en rumslig kolumn till tabellen skapar ArcSDE en sådan. Båda klasserna har konstruktorer som kan användas för att få tag på en tabell eller ett lager som redan existerar genom att ange vilken uppkoppling som används samt id-nummer eller namn på tabellen respektive lagret. Alternativt kan en metod i klassen *SeConnection* användas för att få alla tabeller respektive lager som finns i databasen.

Då information skall sökas skapas ett nytt *SeQuery*-objekt. För att kunna göra detta krävs ett objekt av typen *SeSqlConstruct* som innehåller information om vilken tabell frågan gäller samt eventuellt where-villkor (se avsnitt 6.2). Dessutom behövs en String-vektor som anger vilka av tabellens kolumner som svaret skall innehålla samt en referens till uppkopplingen. Genom anrop av två metoder förbereds och ställs frågan, sedan kan metoden *fetch()* anropas. Metoden returnerar ett objekt som representerar en rad ur svaret och får därför anropas en gång för varje rad.

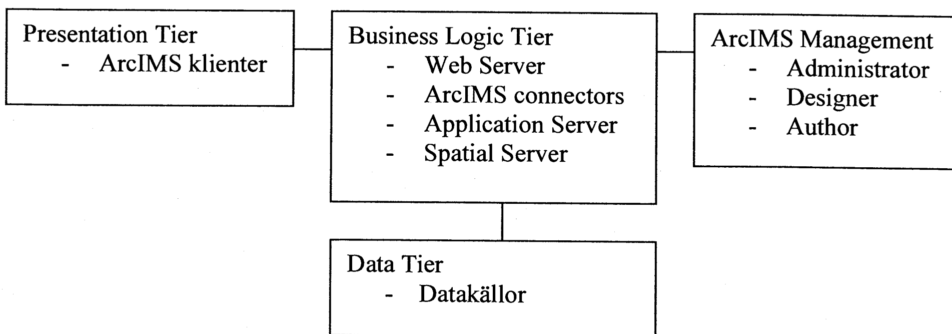
Om en ny rad ska läggas till i en tabell skapas ett objekt av typen *Insert* med en referens till det objekt som representerar uppkopplingen som inparameter till konstruktorn. Sedan anropas metoden *intoTable()* för att ange i vilken tabell samt i vilka kolumner information skall infogas. För att ange de värden som skall sättas in används ett objekt som representerar en rad, ett sådant objekt fås som svar på ett anrop av metoden *getRowToSet()*. Efter att värden har angetts anropas först *execute()* och sedan *close()* på *Insert*-objektet för att först utföra insättningen och sedan stänga den ström som öppnas i och med att ett objekt av typen *Insert* skapas. Uppdatering och borttagning sker på liknande sätt.

7.2 ArcIMS

ArcIMS är ett program som används för att skapa och hantera karttjänster som levererar dynamiska kartor och geografiska data. Klienten kan vara en vanlig webbläsare, men det kan även vara exempelvis programmet ArcGIS Desktop eller en mobiltelefon [46]. Tjänsten syftar till att ge användaren tillgång till geografiska data, kartdata och metadata. Flera olika datakällor kan lätt kombineras, såväl lokala datakällor (ArcSDE, shape mm.) som datakällor

på Internet [46]. För att skapa en karttjänst med hjälp av ArcIMS krävs ingen programmering och det finns stora möjligheter att anpassa karttjänsten efter olika avancerade klienter.

ArcIMS är uppbyggt av tre olika lager (*tier*), se figur 7.2. Dessa är *Business Logic Tier, Data Tier* samt *Presentation Tier*. Dessutom finns *ArcIMS Management* med ett antal applikationer som används för att hantera tjänsten [47]. För kommunikation mellan klient och ArcIMS samt för kommunikation inom ArcIMS används ArcXML. ArcXML är en dialekt av XML framtagen av ESRI (se avsnitt 3.1) anpassad speciellt för att hantera geografiska data. Genom att använda ArcXML är det även möjligt för utvecklare att påverka och utöka funktionaliteten i webbtjänsten [44].



Figur 7.2. Bilden visar de delar ArcIMS är uppbyggt av. Baserat på ArcIMS white paper [47] sidan 1.

Business Logic Tier tar emot frågor från klienter, behandlar frågorna och skickar sedan ett svar [6]. Lagret består av flera komponenter. Klienten anropar webbservern som skickar frågan vidare till en applikationsserver (*Application Server*). På vägen passeras en *connector* (*ArcIMS Application Server Connector*) som översätter frågan till ArcXML och skickar den vidare till applikationsservern. ArcIMS erbjuder flera olika *connectors*, den som normalt används är en *Servlet Connector* vilken fungerar med alla de plattformar som ArcIMS stöder [47]. Applikationsservern ser sedan till att rätt spatial server anropas. En spatial server hämtar data och sätter ihop en karta efter klientens önskemål. Svaret översätts av *connectorn* innan det skickas vidare till klienten. Det finns sju olika komponentservrar till ArcIMS spatiala server varav fyra är publika och tre interna [47]. De interna komponentservrarna används automatiskt av ArcIMS då de behövs medan de publika kan kommas åt via ArcIMS interface. De fyra publika komponentservrarna är:

- *Image Server* som levererar rasterkartor.
- *Feature Server* som levererar vektordata. Denna ger klienten mer funktionalitet än en *Image Server* men kräver att klienten kan köra Java Applets (se avsnitt 5.4).
- *ArcMap Image Server* som är en tjänst som levererar rasterkartor, liksom en *Image Server*, men som konfigureras via ArcMap.
- *Metadata Server* som används för att leverera metadata.

Data Tier innehåller en eller flera datakällor som är tillgängliga för ArcIMS. Vilka datatyper som finns tillgängliga beror på vilken typ av tjänst som används [6]. En *Feature*-tjänst klarar av minst antal format, den kan exempelvis inte hantera rasterformat, och en *ArcMap Image* flest. Exempel på lokala datakällor är Shape-filer och ArcSDE. Om ArcSDE används behövs en uppkoppling för varje spatial server-instans. Det medför att det kan behövas många uppkopplingar, detta är dock inget problem då ArcSDE tillåter ArcIMS att ha obegränsat antal uppkopplingar [6].

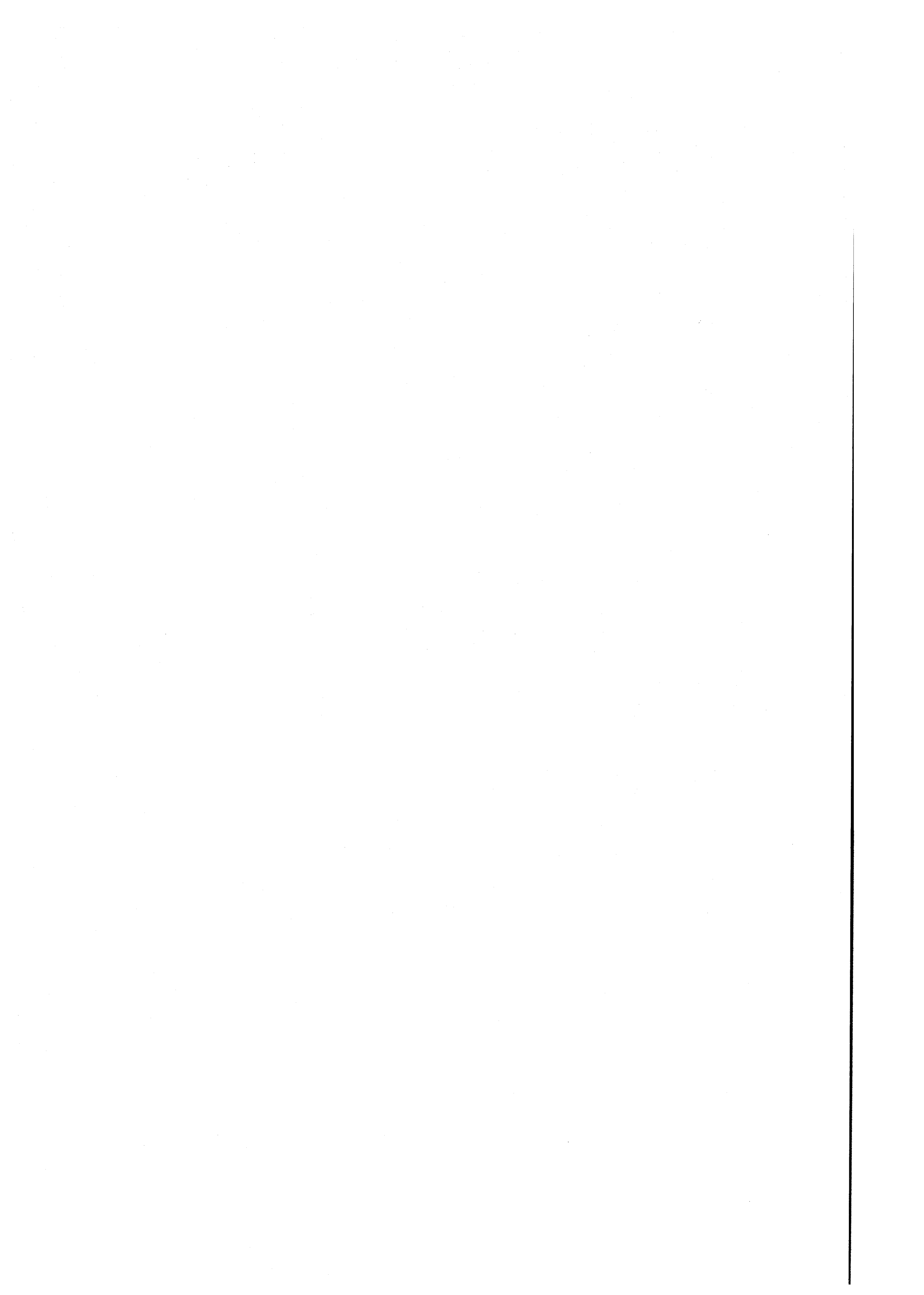
Presentation Tier innehåller klienter som används för att titta på och manipulera geografiska data. Vanligtvis innehåller klienten en kartbild samt ett antal mer eller mindre avancerade funktioner att använda med kartan, som zoomning eller få information om objekt i kartan. Som nämnts tidigare kan klienterna bestå av en vanlig webbläsare, program som exempelvis ArcGIS Desktop eller mobila enheter som mobiltelefoner. De karttjänster som kan skapas med *ArcIMS Designer* visas i en webbläsare. *HTML Viewer* används för tjänsterna *Image* eller *ArcMap Image* och består av HTML, **DHTML** och JavaScript [6]. JavaScript används för att skapa frågor och hantera svar, för att klara av det krävs att webbläsaren är *Internet Explorer* eller *Netscape®* version 4.x eller högre. *Java Viewer* kan användas för tjänsterna *Image*, *ArcMap Image* samt *Feature* och flera olika tjänster kan kombineras i samma klient. Det krävs att klienten kan köra Java Applets (se avsnitt 5.4) för att den ska kunna visa information och hantera frågor. Data laddas till cache-minnet och frågor behandlas i första hand hos klienten vilket minskar antalet anrop till servern [6]. När webbläsaren stängs tas data som laddats till cache-minnet bort. Det finns två varianter av *Java Viewer* som båda har samma funktionalitet [47]. Den ena varianten använder JavaScript för kommunikation med appletsen och fungerar endast i *Internet Explorer*. Den andra varianten använder inte JavaScript och fungerar i både *Internet Explorer* och *Netscape®* (version 4.x eller högre).

ArcIMS Management består av de tre delarna *Author*, *Administrator* samt *Designer* [47]. Är karttjänsten av typen *Image* eller *Feature* används *Author* för att definiera vilka data som ska vara tillgängliga i vilka skalor samt hur de olika kartlagren ska visas. Dessa inställningar lagras sedan i en ArcXML-fil. Inställningar kan även göras direkt i denna fil genom att exempelvis använda en vanlig texteditor. För att ange dessa inställningar för en tjänst av typen *ArcMap Image* används istället programmet *ArcMap* och inställningarna lagras i en binär fil. För Metadata-tjänster måste en konfigurationsfil skapas med hjälp av en text- eller XML-editor. *Administrator* används för publicering och administrering av tjänsterna. Vid publicering av en tjänst anges den konfigurationsfil som skapades med exempelvis *Author*. Efter att tjänsten har publicerats kan *Designer* användas för att generera en webbsida. *Designer* guidar steg för steg och låter användaren göra en rad val. Exempelvis väljs vilken tjänst som skall användas, hur sidan ska se ut och vilka funktioner som ska erbjudas. Det finns även tre typer av klienter att välja mellan, en *HTML Viewer* eller en av två *Java Viewer*. Då guiden är slutförd genereras en rad HTML-sidor som antingen kan användas direkt som de är eller modifieras efter egna önskemål.

ArcIMS arbetar med Javamiljö och för att en ArcIMS-tjänst ska fungera behövs tre externa komponenter [47]. Det behövs en webbserver som kommunicerar med klienterna och eftersom Javamiljö används måste **Java VM** vara installerad. Sedan behövs också en Servlet Engine, ett Java-tillägg som behövs för att kunna köra servlets. Denna fungerar som en länk mellan webbservern och Java VM.

Det finns ett antal tillägg till ArcIMS som kan användas för att utöka funktionaliteten [44]. Exempel på tillägg är *Route Server* som utökar karttjänsten med funktionalitet för

ruttplanering. Andra tillägg är WMS- och WFS-*connector* som gör det möjligt att kommunicera med karttjänsterna via de standardiserade gränssnitten WMS och WFS (se kapitel 4). Den WFS-connector som finns stöder dock inte redigering av data utan är av typen *Basic WFS*. Det finns inte heller någon annan metod för att redigera data via ArcIMS [48].



8 Intrakartan – Lunds kommuns interna karttjänst

Lunds kommun har en intern karttjänst, *Intrakartan*, som nås via kommunens Intranät. Karttjänsten, som är byggd med ArcIMS, används för att distribuera geografiska data till olika användare inom kommunen och finns i två varianter. Den första varianten, *Intrakartan Bas*, innehåller mindre funktionalitet än den andra varianten, *Intrakartan Plus*. *Intrakartan Bas* är gratis att använda för alla som arbetar inom kommunen, men plus-varianten kräver interndebitering [49].

8.1 Funktionalitet

Avsnittet baseras på den information som finns om karttjänsterna på Lunds kommuns Intranät [49] samt på egen erfarenhet av karttjänsterna.

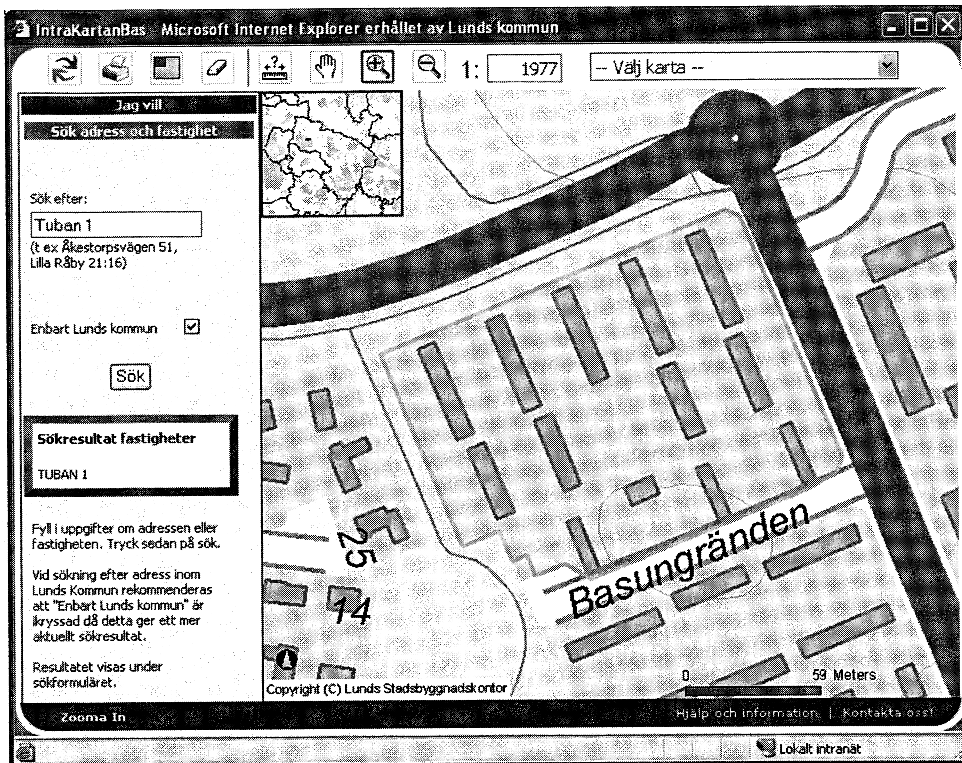
Gemensamt för de båda tjänsterna är att de inte innehåller någon funktionalitet för redigering av data i databasen. Tjänsterna kan enbart användas för att titta på data av olika slag samt för att utföra enklare analyser, såsom mätning och sökning.

8.1.1 Intrakartan Bas

Intrakartan Bas innehåller sju olika kartor som användaren kan växla mellan. Två av dessa täcker hela Skåne, Skånekartan samt flygfoto från 1940. Övriga kartor täcker olika mindre områden, men alla täcker Lunds kommun. Två av kartorna är äldre, Skånska rekognoseringskartan från 1812 samt Hushållningssällskapets karta från 1912. Det finns även ett flygfoto från 2004 samt en terrängkarta. Den sista kartan är fastighetskartan som enbart täcker Lunds kommun, denna karta uppdateras en gång i veckan medan exempelvis Skånekartan endast uppdateras två gånger per år.

Det finns möjlighet att söka efter adresser inom hela Skåne samt efter fastigheter inom Lunds kommun. Det finns även funktionalitet för utskrift av kartor där användaren har möjlighet att ange en rubrik till kartan samt att välja upplösning. Andra funktioner som finns i karttjänsten är zoomning, mätning av längder samt att visa teckenförklaring.

I figur 8.1 visas en bild på klienten då en sökning efter fastigheten *Tuban 1* har gjorts. Den sökta fastigheten har markerats i kartan.



Figur 8.1. Resultat av en sökning efter fastigheten Tuban 1 i Intrakartan Bas.

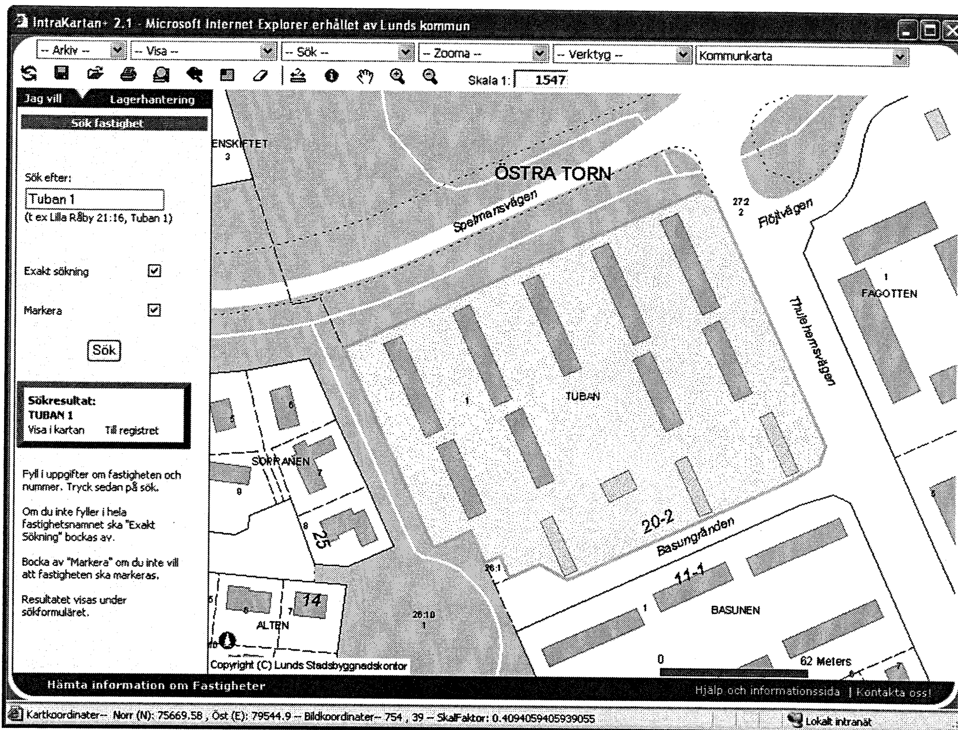
8.1.2 Intrakartan Plus

Intrakartan Plus har elva olika bakgrundskartor varav fyra av dessa är flygfoton. Dessutom finns här valet *Ingen bakgrundskarta.* Anledningen till detta är att det förutom att välja bakgrundskarta finns möjlighet att via *Lagerhanteraren* välja vilka kartlager som skall vara synliga. Även här täcker olika kartor/lager olika geografiska områden, som mest täcks hela Skåne och som minst enbart Lunds kommun.

I denna version av karttjänsten har användaren möjlighet att spara vissa inställningar. Användaren får då välja om det är geografisk utbredning, valda lager eller båda som skall sparas. Varje användare kan spara cirka fem olika inställningar.

Förutom funktionalitet för zoomning och mätning av längder innehåller *Intrakartan Plus* även funktionalitet för att mäta areor samt för att få information om en viss punkt genom att klicka i kartan. Precis som i *Intrakartan Bas* finns funktioner för utskrift av karta, här finns även möjlighet att skriva kartan till **pdf-format**. Dessutom finns en mängd sökfunktioner, exempelvis kan en adress, fastighet eller detaljplan sökas. I och med att kartan är kopplad till olika register kan t.ex. information från fastighetsregistret visas om en vald fastighet.
I figur 8.2 visas hur det ser ut när en sökning efter fastigheten *Tuban 1* har gjorts. Sökbegrepp anges i textrutan till vänster om kartbilden där även sökresultatet visas. För att visa

information från fastighetsregistret används länken *Till registret.* I figuren framgår även att denna karttjänst har mer funktionalitet än *Intrakartan Bas* (jämför med figur 8.1).



Figur 8.2. Resultat av en sökning efter fastigheten Tuban 1 i Intrakartan Plus.

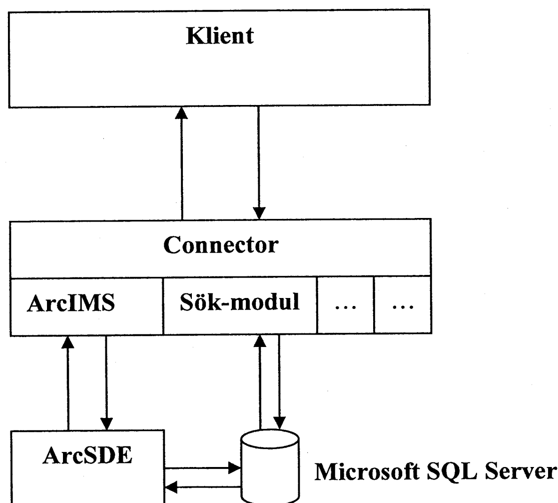
Användarna av *Intrakartan Plus* är indelade i olika grupper som har tillgång till olika stor del av plus-utbudet. Detta beror på att vissa handläggare kan behöva tillgång till känslig information som inte ska vara tillgänglig för andra användare. Vilka rättigheter en användare har samt vilken användargrupp den tillhör finns lagrat i en databas [50].

8.2 Teknik

Lunds kommun använder sig av produkter från ESRI och karttjänsterna har därför byggts med ESRI:s program ArcIMS (se avsnitt 7.2). En HTML-viewer används vilket innebär att klienten består av HTML och JavaScript. Den klient som generades av *ArcIMS Designer* har modifierats både vad det gäller utseende och funktionalitet för att anpassas efter kommunens önskemål och behov [50]. Dels har funktioner som programmet genererade modifierats och dels har nya funktioner skapats, som exempelvis nya sökfunktioner. De nya funktionerna skrevs först i ASP med skript i språket VB, men flertalet av funktionerna har senare gjorts om så att den nyare tekniken med ASP.NET och kod i C# används.

För att lagra geografiska data används ArcSDE (se avsnitt 7.1) på en Microsoft SQL Server-plattform. Vissa punktlager ligger dock enbart i Microsoft SQL Server och finns alltså inte med i ArcSDE [50]. För att hantera geografiska data används ArcGIS-programmen ArcCatalog och ArcMap. ArcCatalog används för att administrera de tabeller som finns och eventuellt skapa nya, medan ArcMap används för att lägga till, uppdatera samt ta bort data från tabellerna.

I figur 8.3 visas en förenklad skiss över hur systemet är uppbyggt [51]. Klienten kommunicerar med *connectorn* som vidarebefordrar frågorna till ArcIMS och/eller de egenbyggda modulerna som används för t.ex. sökning. ArcIMS kommunicerar sedan med ArcSDE för att exempelvis hämta kartbilder som returneras till klienten. Modulerna för sökning kommunicerar direkt med de Microsoft SQL Server-databaser som även ArcSDE använder.



Figur 8.3. Figuren visar en förenklad skiss över systemets uppbyggnad.

8.3 Användningsområden

Följande avsnitt bygger på samtal med Jonas Andréasson, chef på Lunds kommunala Lantmäteri, den 25/10 2006 [51].

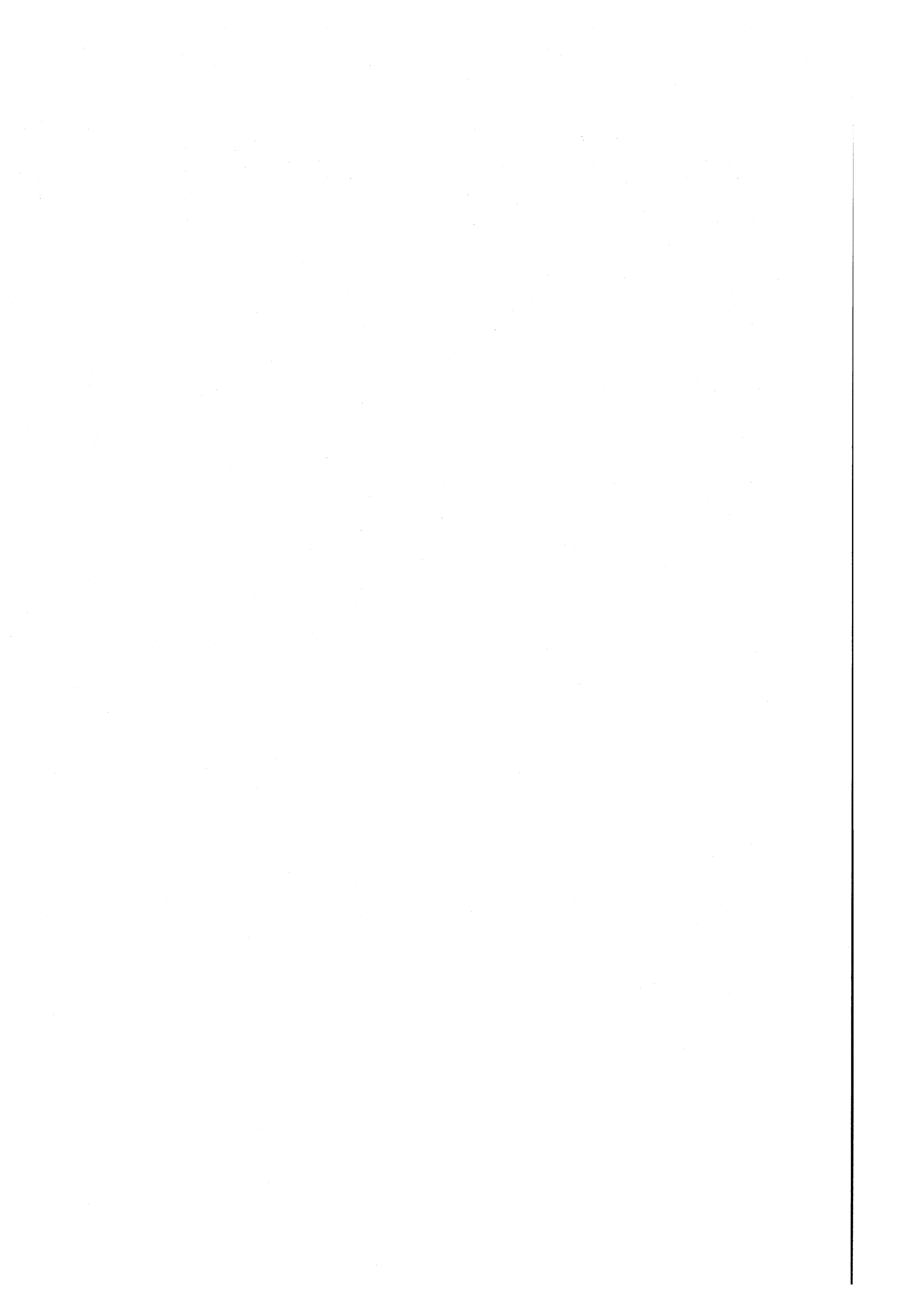
Det finns många olika användningsområden för *Intrakartan*. Ett flertal förvaltningar använder sig av karttjänsten, nedan ges några exempel på vad den används till idag.

- Stadsarkitekternas användning av planer och bygglov.
- Informationen om adresser används bl.a. för att planera renhållning.
- Adressättning.
- Fastighetsbildning.

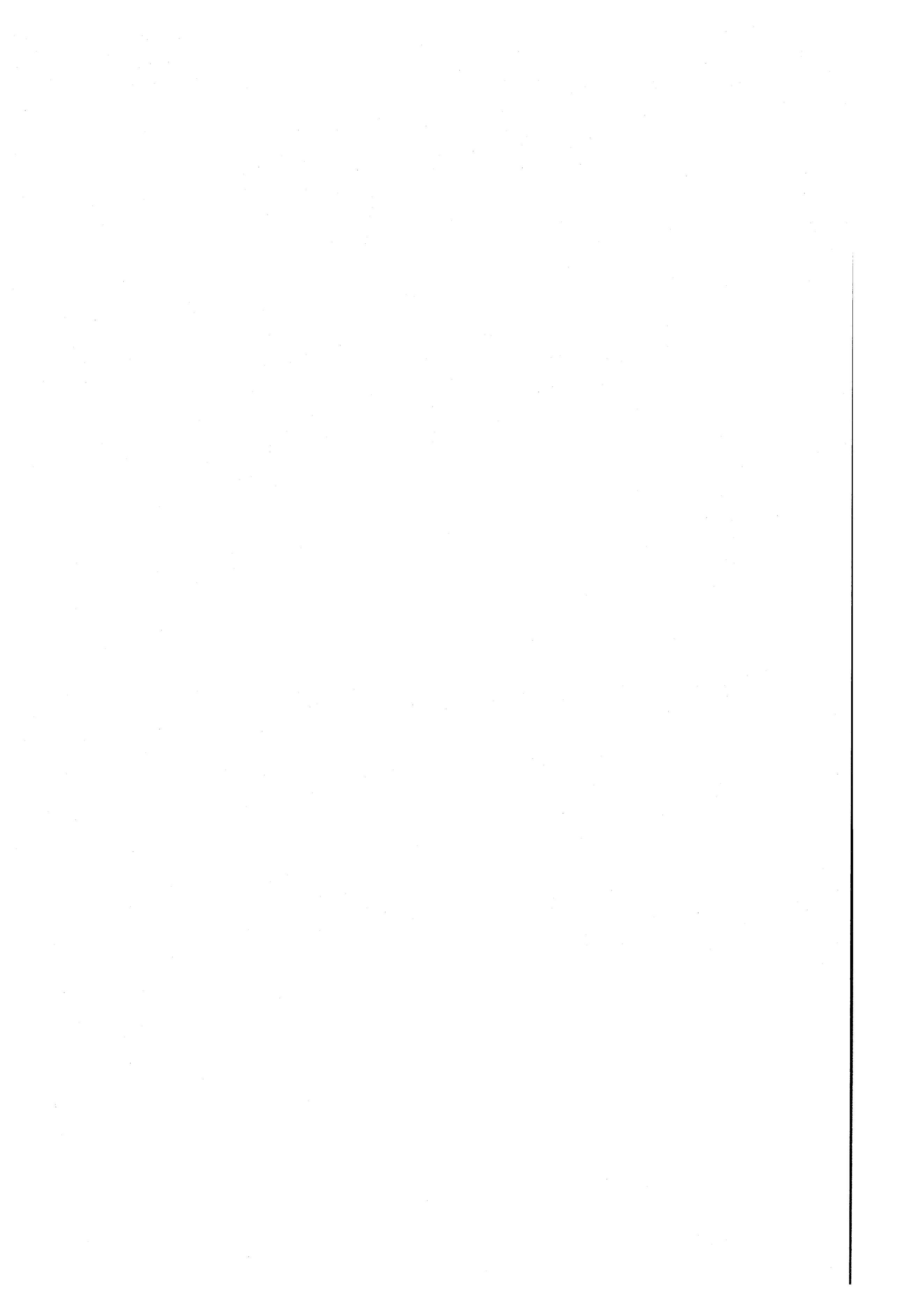
- Befolkningsprognoser.
- Kommunkontorets användning av näringsregistret.
- Tekniska kontorets och Miljöförvaltningens användning av karttjänsten för att ta fram fastighetsinformation.
- Framställning av detalj- och översiktsplaner.
- Skötselplaner för kommunalägd mark.

Även om karttjänsten har många olika användningsområden redan idag finns ytterligare planer på vad den skulle kunna användas till i framtiden. Några exempel ges nedan.

- Lagra vem som är vårdnadshavare på en viss adress. Skulle kunna användas av Vård & Omsorgs-förvaltningen.
- Ruttplanering. Skulle t.ex. kunna användas för att planera skolskjuts. Denna funktion är under uppbyggnad.
- Dela in befolkningen i åldersgrupper och presentera i karttjänsten, kan exempelvis användas för skolplanering.
- Koppling till Arken (Lantmäteriets digitala arkiv).



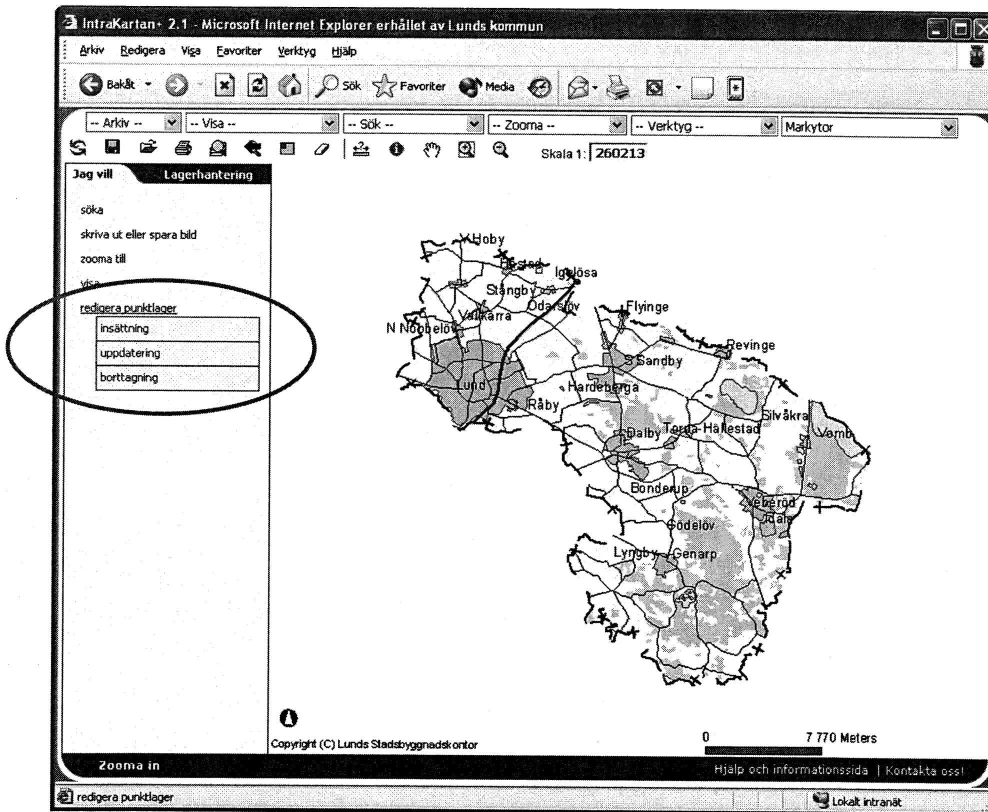
Praktikdel



9 Användaraspekter av klienten

I kapitel 8 gavs en beskrivning av *Intrakartan.* I fortsättningen beskrivs de nya delarna av karttjänsten som har implementerats som en del av examensarbetet. Användaraspekterna av klienten tas upp i detta kapitel medan den tekniska lösningen beskrivs i kapitel 11. Däremellan gås den planerade tekniken kort igenom i kapitel 10.

Redigering av data kan ske med hjälp av tre olika operationer. Antingen kan ett nytt punktobjekt läggas till eller så kan ett existerande objekt uppdateras eller tas bort. Eftersom den information användaren måste ange för att utföra redigeringen beror på vilken typ av operation som avses har tre olika formulär skapats, ett för var typ av operation. Formulären visas till vänster om kartan, dvs. i samma del av fönstret som formulär för exempelvis sökning visas (se figur 8.2). De tre formulären nås från den nya menyn *redigera punktlager* som markerats i figur 9.1. I framtiden borde en ny rullgardinsmeny för redigering läggas till överst på sidan (mellan *arkiv* och *visa*) så att funktionerna kan nås även därifrån.



Figur 9.1. Till vänster i bilden syns länkarna till de nya funktionerna för redigering av punktlager (se inringat område).

Överst i varje formulär finns en länk ”Hur gör jag?”. Då användaren klickar på länken öppnas ett nytt fönster med en beskrivning av hur de olika redigeringsfunktionerna skall utföras. I ett tidigare skede av utvecklingen fanns istället en förklaringstext nederst i varje formulär. Då dessa hjälptexter skulle göras utförligare ansågs det lämpligare att lägga dem i ett separat dokument, detta visas i figur 9.2.

Hjälp vid redigering av punkter

Insättning
Uppdatering
Borttagning

Insättning

Välj först i rullgardinsmenyn det kartlager till vilket den nya punkten skall läggas till. Ange sedan information om punkten i textrutorna. Istället för att fylla i värden för punktens koordinater i textrutorna går det bra att klicka i kartan för att ange punktens position. Koordinaterna för den punkt man klickade i fylls då automatiskt i textrutorna.

När värden angetts klickas knappen "Sätt in" för att utföra insättningen. Över rullgardinsmenyn visas då ett meddelande som talar om ifall insättningen lyckades eller inte.

Till sidans topp

Uppdatering

Välj först i rullgardinsmenyn det kartlager till vilket den punkt som skall uppdateras hör. Ange sedan vilken punkt som skall uppdateras genom att klicka på punkten i kartan. Punktens id-nummer fylls då i textrutan under rullgardinsmenyn. Alternativt kan punktens id-nummer anges manuellt.

Under textrutan finns två bilder varav den ena har en röd kant. Bilderna symboliserar två olika funktioner vid klick i kartan: ange punkt eller ange ny position. Den röda ramen anger vilken funktion som är aktiv, klicka på symbolerna för att växla mellan de två funktionerna. När den vänstra symbolen, "ID", har en röd kant kan punkt att uppdatera väljas genom klick i kartan, som beskrivits ovan. Om istället den högra symbolen, "XY", har en röd kant kan punktens nya position anges genom klick i kartan. Då fylls värden för koordinaterna i textrutorna.

Nya värden för attributen anges i textrutorna, om en ruta lämnas tom behålls det gamla värdet.

När värden angetts klickas knappen "Uppdatera" för att utföra uppdateringen. Över rullgardinsmenyn visas då ett meddelande som talar om ifall uppdateringen lyckades eller inte.

Till sidans topp

Borttagning

Välj först i rullgardinsmenyn det kartlager till vilket den punkt som skall tas bort hör. Ange sedan vilken punkt det är som ska tas bort genom att klicka på punkten i kartan. Alternativt kan punktens id-nummer anges i textrutan.

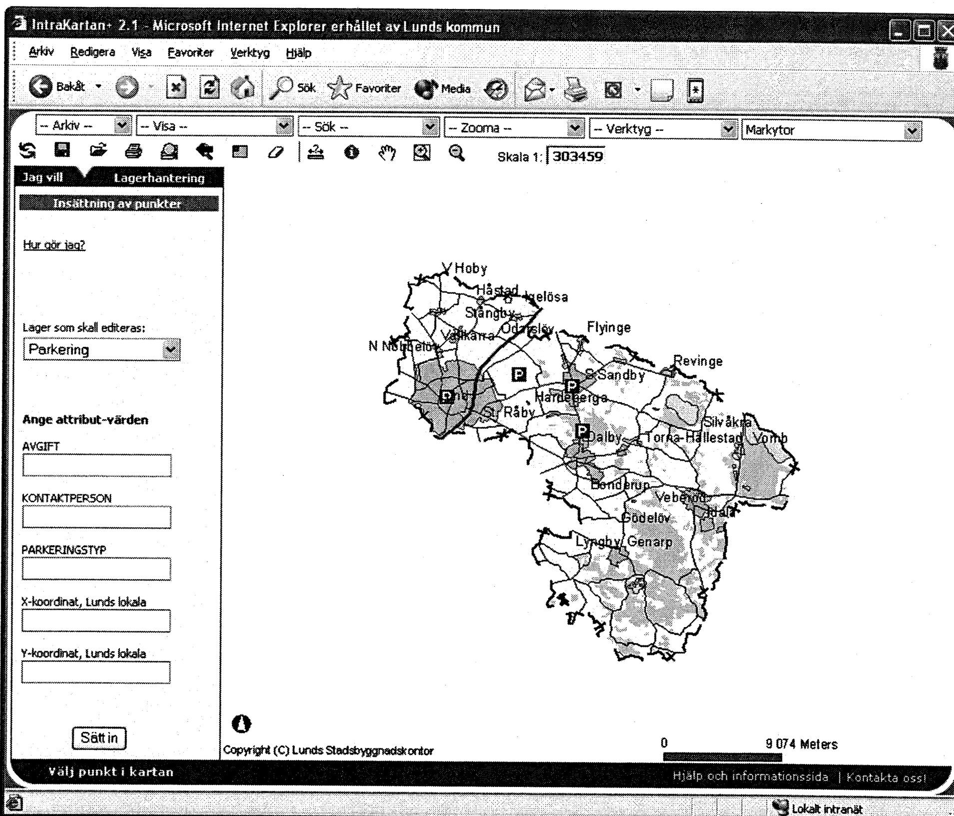
Klicka på knappen "Ta bort" för att utföra borttagningen. Över rullgardinsmenyn visas ett meddelande som talar om ifall borttagningen lyckades eller inte.

Till sidans topp

Figur 9.2. Bilden visar den hjälptext som finns till redigeringsfunktionerna. Dokumentet nås via en länk överst i respektive formulär.

9.1 Insättning av punkt

Formuläret för insättning syns till vänster i figur 9.3. Då en ny punkt skall sättas in anges först vilket lager punkten skall tillhöra. Detta görs genom val i rullgardinsmenyn överst i formuläret. Sedan kan punktens attribut anges i textrutorna. En punkts position lagras som ett attribut men har i formuläret delats upp i en ruta för x-koordinat och en ruta för y-koordinat. Ett alternativ till att skriva in värden för punktens koordinater är att ange dess position med ett klick i kartan, så att koordinaterna automatiskt fylls i textrutorna. Då alla värden har angetts och användaren klickar på knappen ”Sätt in” läggs punkten till i databasen och kartan laddas på nytt för att punkten skall kunna ses. Över rullgardinsmenyn i formuläret meddelas ifall insättningen lyckades vilket id-nummer den nyinsatta punkten fått. Om insättningen misslyckades meddelas istället detta.

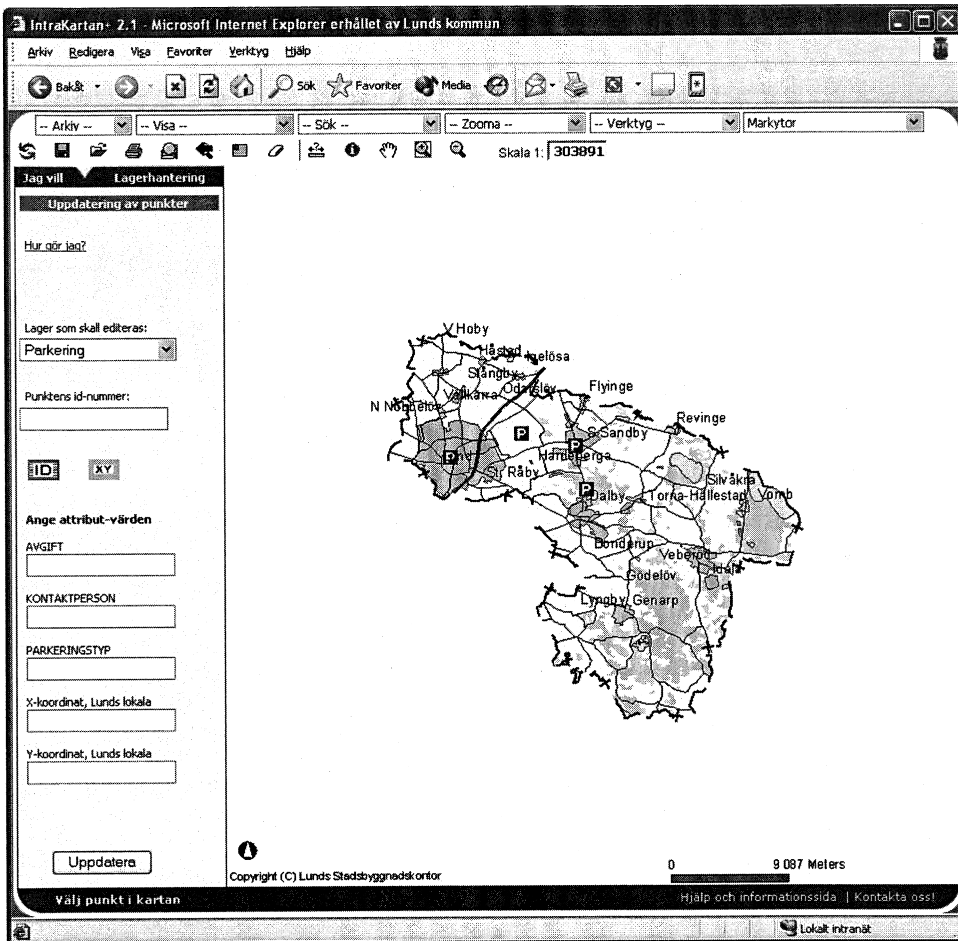


Figur 9.3. Till vänster i bilden syns det formulär som används vid insättning av punkter.

9.2 Uppdatering av punkt

I figur 9.4 visas det formulär som används vid uppdatering av punktobjekt. Först anges till vilket kartlager den punkt som skall uppdateras hör. Detta görs genom att användaren väljer ett punktlager i rullgardinsmenyn överst i formuläret. För att identifiera vilken punkt som skall uppdateras skall punktens id-nummer anges i den översta textrutan. Troligen vet inte

användaren vilket id-nummer punkten har men kan däremot identifiera punkten visuellt på kartan. Användaren kan då välja att ange vilken punkt som skall uppdateras genom att klicka på punkten i kartbilden. Det krävs då att rätt lager först har valts i rullgardinsmenyn.



Figur 9.4. Till vänster i bilden syns det formulär som används vid uppdatering av punkter.

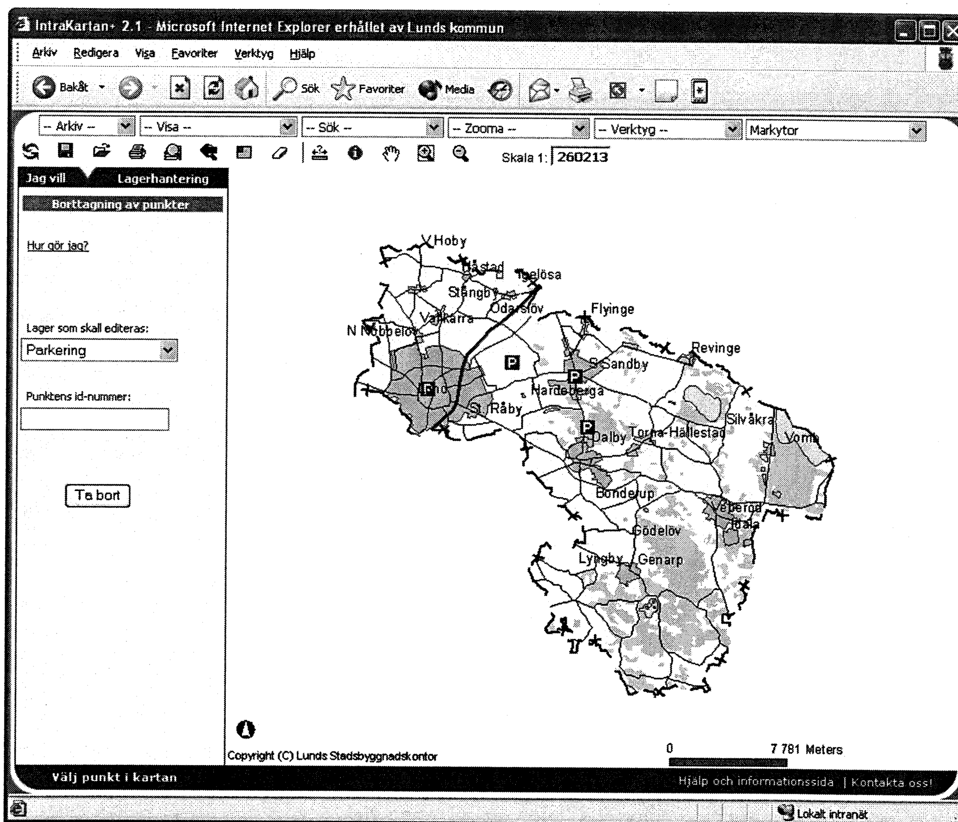
Över textrutorna för övriga attribut finns två bilder/knappar som visar vad som händer vid klick i kartan. Från början har den vänstra bilden en röd kant vilket innebär att den punkt som skall uppdateras väljs vid klick i kartan, som beskrivits ovan. Genom att klicka på den högra bilden/knappen får istället den en röd kant och punktens nya position anges vid nytt klick i kartan. Nya värden för övriga attribut kan sedan anges i textrutorna, ifall en textruta lämnas tom sker ingen uppdatering av motsvarande attribut.

Då användaren klickar på knappen "Uppdatera" utförs ändringarna i databasen och om punktens position ändrats laddas kartbilden på nytt för att ändringarna skall synas. Ett

meddelande som anger huruvida uppdateringen lyckades eller inte visas över rullgardinsmenyn i formuläret.

9.3 Borttagning av punkt

Det formulär som används för att ta bort punkter visas i figur 9.5. Först anges ett kartlager i rullgardinsmenyn högst upp i formuläret. Sedan skall den punkt som skall tas bort från det valda lagret identifieras. Precis som vid uppdatering kan detta göras antingen genom att ange punktens id-nummer i textrutan eller genom att klicka på punkten i kartan. Då användaren klickar på knappen "Ta bort" tas punkten bort ur databasen. Över rullgardinsmenyn i formuläret visas ett meddelande som talar om ifall borttagningen lyckades eller misslyckades. Om den lyckades laddas kartbilden på nytt för att göra ändringen synlig.



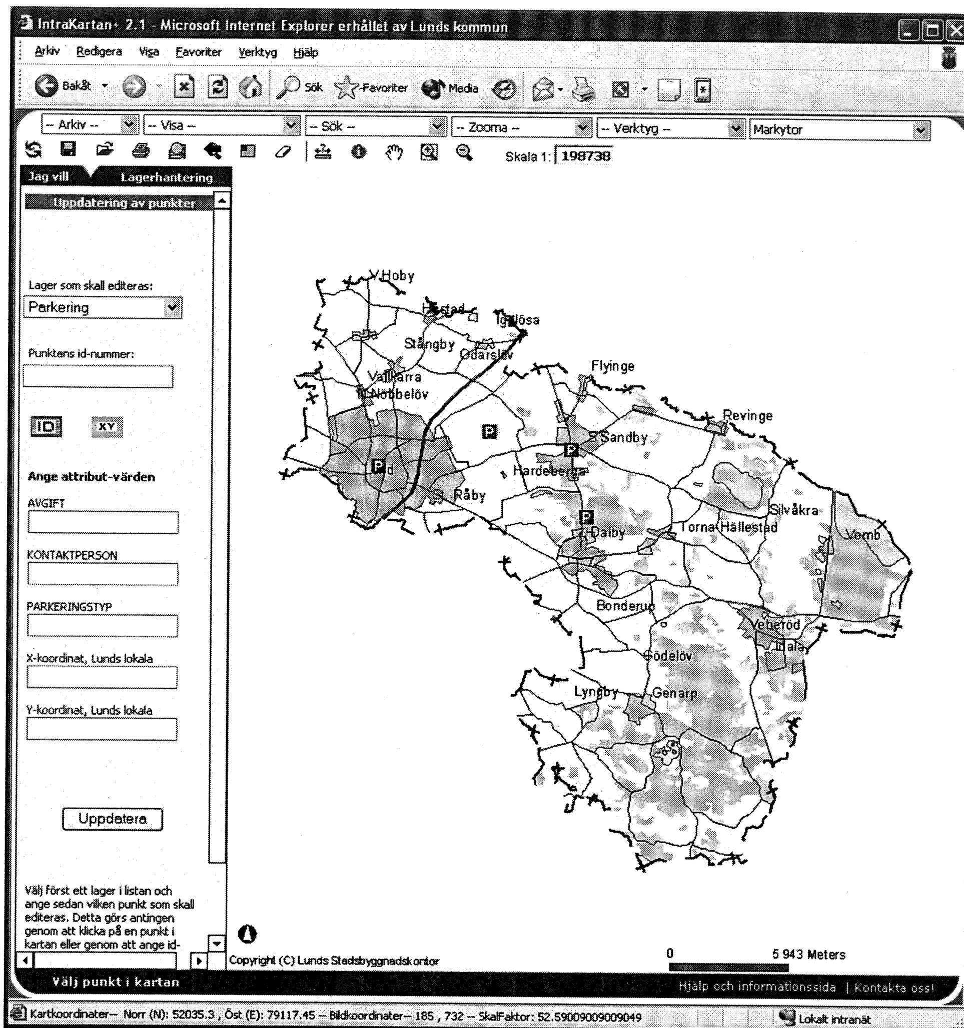
Figur 9.5. Till vänster i bilden syns det formulär som används vid borttagning av punkter.

9.4 Användartest

En studie av användbarheten har utförts för att få en bild av hur användargränssnittet skulle fungera för avsedd målgrupp, dvs. kommunanställda med ingen eller liten GIS-erfarenhet. En

person ur målgruppen fick testa funktionerna och komma med kommentarer. Trafikmiljörådgivaren Christina Nilsson på *Gatu och trafikkontoret* på Lunds kommun fungerade som testperson. Hon har akademisk bakgrund men hade ingen tidigare erfarenhet av GIS. Hon hade dock använt *Intrakartan* tidigare för att titta på fastigheter samt mäta avstånd.

Efter testets genomförande har en mindre ändring i användargränssnittet gjorts. Tidigare fanns information om hur redigeringen skulle utföras längst ned i respektive formulär, se figur 9.6. Vid testet upptäcktes att dessa texter inte syntes eftersom datorskärmen var mindre än den som använts vid utvecklingen. Ett önskemål framkom om att denna text istället borde visas högst upp samt att den borde vara utförligare. Detta är troligen bra för en ny/ovan användare men för de användare som vet hur redigeringen går till blir då nackdelen att varje gång behöva skrolla ned innan värden kan fyllas i. Därför gjordes en kompromiss genom att informationen lades i en separat fil som öppnas genom länken ”Hur gör jag?” överst i formuläret.



Figur 9.6. Till vänster i bilden syns det formulär som först var tänkt att användas vid uppdatering av punkter. Som synes kan inte hela hjälptexten läsas utan att först skrolla ned.

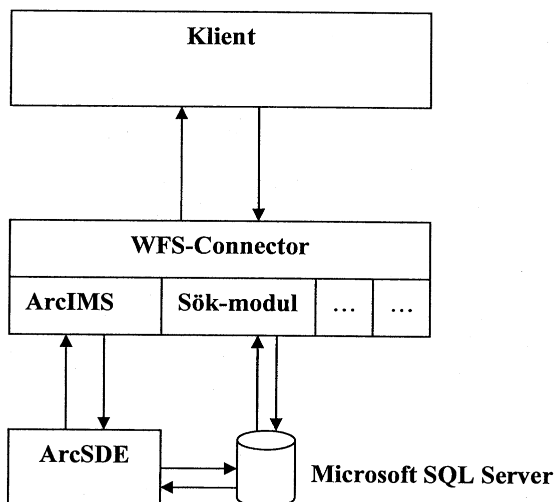
Christina Nilsson fick testa att sätta in en ny parkeringsplats, sedan uppdatera den och slutligen ta bort den. Hjälp gavs och användargränssnittet diskuterades under testets gång. Nedan listas önskemål om förbättringar som framkom, dessa diskuteras i avsnitt 12.2.

- Tydligare förklaringstexter.
- Förklaringar högst upp i formuläret.
- Alternativet att skriva in koordinatvärden behövs inte, det räcker med att klicka i kartan. Textrutorna för koordinater behövs då inte, vore bättre att markera punktens position i kartan som en temporär punkt.
- Vad menas med punktens id-nummer? Denna information behövs inte, det vore bättre om den punkt som är vald markeras i kartan.

- Vid uppdatering vore det bra om de gamla värdena var ifyllda så att användaren lättare ser vilka värden som skall ändras.
- Knapparna för att växla mellan att välja punkt och position begreps inte.
- Uppdatering kanske kan delas upp i två fall: uppdatera fakta respektive flytta punkt. Då en punkt flyttas borde detta synas i kartan på en gång.

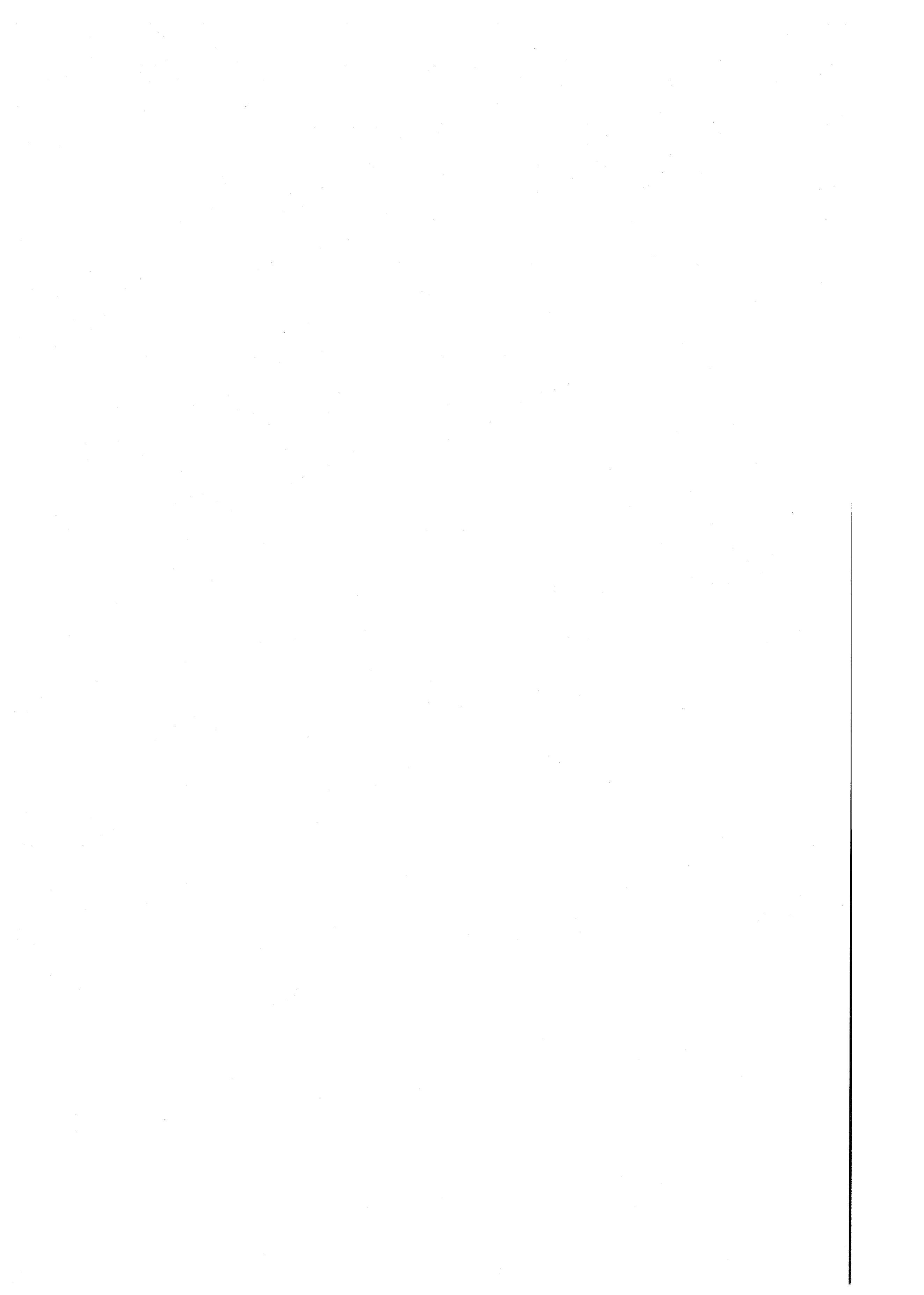
10 Tekniska aspekter – Planerad teknik

Vid de efterforskningar som inledde examensarbetet upptäcktes snart att det fanns ett tillägg till ArcIMS, en *WFS-connector*, som kunde installeras för att en klient skulle kunna kommunicera med karttjänsten via det standardiserade gränssnittet WFS (se avsnitt 7.2). Eftersom tanken med examensarbetet var att använda WFS för kommunikationen mellan klient och server lät detta mycket bra. Förhoppningen var att en klient som konstruerade WFS-anrop och skickade dem till *connectorn* skulle kunna skrivas för att utöka karttjänsten med funktionalitet för insättning, uppdatering samt borttagning av objekt. Sedan skulle *WFS-connectorn* behandla anropen, dvs. utföra begärda redigeringar, och skicka ett svar som talade om huruvida operationen lyckades, allt enligt WFS-standarden. I figur 10.1 visas hur det var tänkt att systemet skulle fungera efter att funktionalitet för uppdatering av geografiska data lagts till, jämför med figur 8.3. Skillnaden är att den *connector* som klienten kommunicerar med i det senare fallet kan hantera WFS-anrop som sedan översätts till ArcXML innan de skickas vidare till ArcIMS. Innan svaret skickas till klienten översätts det till det format WFS-standarden anger att svaret skall ges i, dvs. någon typ av XML.



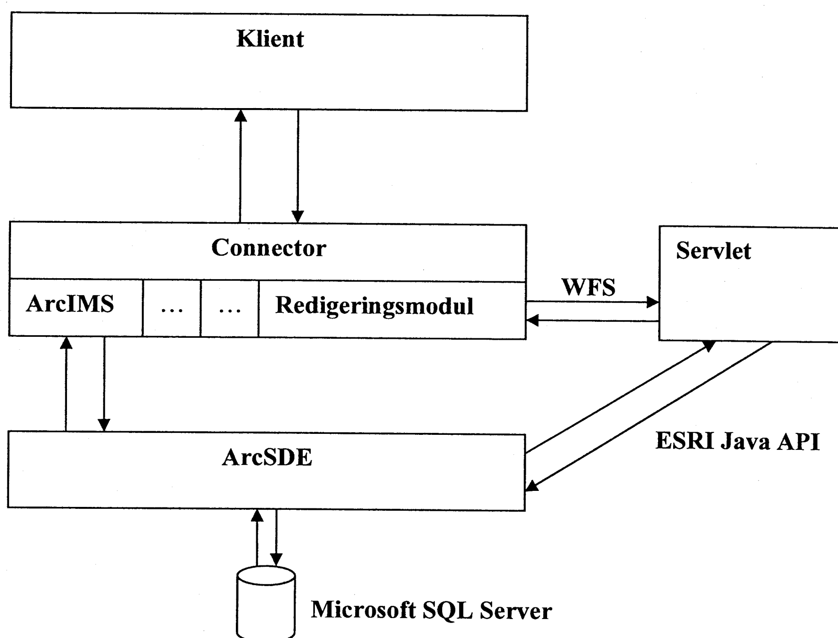
Figur 10.1. Figuren visar en förenklad skiss över hur det var tänkt att systemet skulle vara uppbyggt efter att funktionalitet för uppdatering av geografiska data lagts till.

Det visade sig dock att den *WFS-connector* som finns att installera till ArcIMS version 9.0 endast stöder *Basic WFS*. Detta medförde att *WFS-connectorn* inte kunde användas. Efter kontakt med ESRI support framkom att en ny beta-version skulle släppas i slutet av augusti 2006, efter ytterligare efterforskningar konstaterades dock att inte heller denna version har stöd för *Transaction WFS* [52]. Därmed kunde inte den planerade metoden användas.



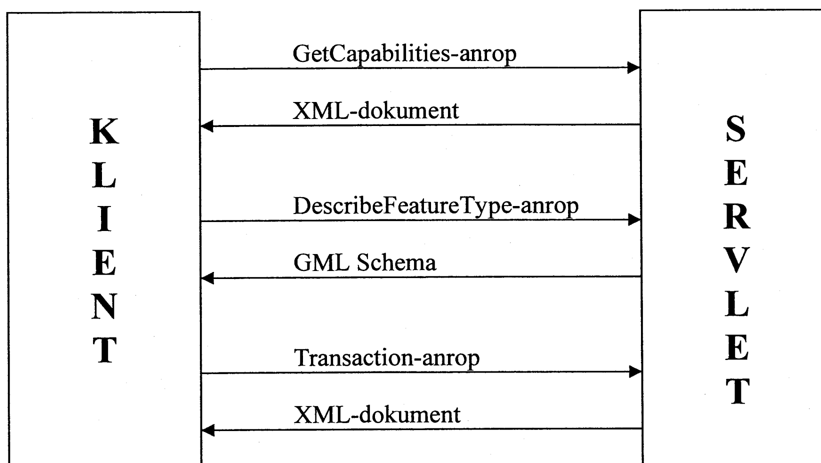
11 Tekniska aspekter – Använd teknik

ArcIMS saknar stöd för redigering av data (se avsnitt 7.2), därför måste redigeringen ske genom direkt kommunikation med ArcSDE. Det finns ett Java API som erbjuder kommunikation med ArcSDE och det bestämdes att en servlet som använder detta API för att utföra insättning, uppdatering samt borttagning av geografiska data skulle skapas. Servleten innehåller enbart funktionalitet och ett separat användargränssnitt skapas i ASP.NET och C#. För kommunikation mellan servleten och den modul som innehåller användargränssnittet används WFS-teknik. I figur 11.1 visas en förenklad skiss över hur systemet är uppbyggt.



Figur 11.1. Figuren visar en förenklad skiss över hur systemet är uppbyggt efter att funktionalitet för redigering av geografiska data lagts till.

En tydligare bild över det kommunikationsflöde som sker mellan klienten/redigeringsmodulen och servleten visas i figur 11.2. Först ställs en fråga av typen *GetCapabilities* till servleten. Svaret används för att presentera vilka lager som kan redigeras. Vid insättning och uppdatering behövs även information om vilka attribut som finns till objekt som tillhör ett visst lager, därför sänds en fråga av typen *DescribeFeatureType*. För borttagning behövs inte denna information. Då en insättning, uppdatering eller borttagning skall utföras skickas ett *Transaction*-anrop. Som svar på varje anrop ges ett XML-meddelande och informationen i detta läses för att presentera delar av den för användaren.



Figur 11.2. Kommunikationsflöde mellan klienten och servleten. Klienten i figuren motsvarar den del som benämns "redigeringsmodul" i figur 11.1. Vid borttagning utesluts anropet DescribeFeatureType.

11.1 Klienten

En klient för var typ av redigeringsoperation har skapats, dvs. en för insättning, en för uppdatering samt en för borttagning av punkter. Var och en av dem består av en aspx-fil med användargränssnitt samt en C#-fil med funktionalitet (se avnitt 5.3). Gemensamt för de tre användargränssnitten är att de innehåller en rullgardinsmeny med namn på de lager som kan redigeras samt en knapp för att utföra operationen. Dessutom finns en länk som öppnar ett nytt fönster med information om hur användaren går till väga för att utföra redigeringen.

11.1.1 Insättning

Det formulär som används vid insättning av data visas i figur 11.3. Under länken till hjälptexten syns den rullgardinsmeny som innehåller namnen på de kartlager där punkter kan läggas till, detta är en webbkontroll av typen *DropDownList*. Under denna syns en textruta för varje attribut som kan anges till det lager som är valt. Dessa textrutor samt attributnamnen som visas ovanför dem ligger i ett *Repeater*-objekt (se avsnitt 5.3). Längst ned syns den knapp som används för att utföra insättningen. Både knappen och textrutorna till attributen är vanliga HTML-element.

Insättning av punkter

Hur gör jag?

Lager som skall editeras:

Parkering

Ange attribut-värden

AVGIFT

KONTAKTPERSON

PARKERINGSTYP

X-koordinat, Lunds lokala

Y-koordinat, Lunds lokala

Sätt in

Figur 11.3. Formulär för insättning av punkter.

Metoden PageLoad() i C#-filen anropas varje gång sidan frågas. Därifrån görs anrop till metoderna getTables() samt getColumns(). Från getTables() görs sedan ett anrop till servleten. Anropet är av typen GetCapabilities och sker med HTTP GET. Ur svaret plockas information om namn och titel på de lager där punkter kan läggas till, denna information sätts in i en tabell som anges som källa till rullgardinsmenyn. Även från metoden getColumns() görs ett anrop till servleten, den här gången av typen DescribeFeatureType och med hjälp av HTTP POST. Anropet innehåller information om vilket lager som är valt, då sidan först laddas är det översta lagret valt. Svaret används för att skapa en tabell som innehåller attributnamn. Denna tabell sätts som källa till ett Repeater-objekt och en textruta för varje attribut skapas på så sätt. Attributet shape som beskriver punktens position delas upp i två textrutor, en för x-koordinat och en för y-koordinat. För att vid klick i kartan enkelt kunna fylla i koordinaterna på den punkt användaren klickat i har attributet id angetts för dessa textrutor. Då användaren ändrar vilket lager som är valt i listan skall även informationen om vilka attribut som finns ändras, därför anropas metoden getColumns() även då.

När attributvärden fyllts i och användaren klickar på knappen för att sätta in den nya punkten anropas en JavaScript-funktion. Funktionen sätter ihop en sträng med CGI-parametrar och

anropar sidan på nytt med dessa parametrar. Parametrarna innehåller information om vilket lager som är valt, vilka värden attributen har fått samt namnet på det attribut som beskriver punktens position. Det senare behövs för att kunna sätta samman parametrarna för x- och y-koordinat till ett attribut innan servleten anropas.

När metoden *PageLoad()* anropas och frågan innehåller CGI-paramterar sker förutom de anrop till metoderna *getTables()* samt *getColumns()* som redan nämnts även anrop till metoden *doInsert()*. Denna metod anropar metoden *computeXML()* som konstruerar ett XML-meddelande med information om den insättning som skall göras. Meddelandet skickas sedan med HTTP POST till servleten, dvs. ett *Transaction*-anrop görs. Svaret läses och högst upp på sidan visas ett meddelande som talar om huruvida insättningen lyckades eller inte. Om användaren försöker sätta in en punkt utan att ange punktens position visas ett felmeddelande överst på sidan och inget *Transaction*-anrop görs.

11.1.2 Uppdatering

Figur 11.4 visar det formulär som används vid uppdatering. Formuläret är uppbyggt ungefär på samma sätt som formuläret för insättning av punkter, skillnaden är att det finns en textruta för att ange id-nummer på den punkt uppdateringen gäller. Textrutan för id-nummer ligger utanför *Repeater*-objektet och kan därmed enkelt nås från C#-filen om den görs som en webbkontroll. Textrutorna för attribut ligger däremot inuti *Repeater*-objektet och är precis som vid insättning vanliga HTML-element. Det finns även två knappar som gör det möjligt för användaren att växla mellan att fylla i en punkts id-nummer respektive punktens nya koordinater vid klick i kartan. Vi klick på någon av dessa anropas en JavaScript-funktion, funktionen beskrivs i avsnitt 11.1.4.

Uppdatering av punkter

Hur gör jag?

Lager som skall editeras:

Parkering

Punktens id-nummer:

Ange attribut-värden

AVGIFT

KONTAKTPERSON

PARKERINGSTYP

X-koordinat, Lunds lokala

Y-koordinat, Lunds lokala

Uppdatera

Figur 11.4. Formulär för uppdatering av punkter.

När användaren klickar på knappen ”Uppdatera” anropas sidan på nytt med ett antal CGI-parametrar. Anropet sätts ihop i en JavaScript-funktion och innehåller den information användaren angett. Har användaren glömt att ange punktens id-nummer eller om x- eller y-koordinat har angetts meddelas detta. I annat fall konstrueras ett Transaction-anrop och resultatet av anropet visas överst på sidan. Fanns ingen punkt med angivet id-nummer meddelas också det.

Från metoden *PageLoad()* i C#-filen sker anrop till metoderna *getTables()* samt *getColumns()* varje gång sidan laddas. Dessa två metoder fungerar på samma sätt som vid insättning, dvs. rullgardinsmenyn fylls med namn på de kartlager som kan uppdateras och en textruta för varje attribut till valt lager skapas. Då anropet innehåller CGI-parametrar görs dessutom ett anrop

till metoden *doUpdate()* som ser till att uppdateringen utförs på motsvarande sätt som *doInsert()* såg till att en insättning gjordes.

11.1.3 Borttagning

I figur 11.5 visas det formulär som används vid borttagning av punkter. Även här används en webbkontroll av typen *DropDownList* för att låta användaren välja vilket lager punkten som skall tas bort tillhör. Under denna finns en textruta där punktens id-nummer skall anges. Textrutan och den knapp som klickas för att utföra borttagningen är båda webbkontroller.

Borttagning av punkter

Hur gör jag?

Lager som skall editeras:

Parkering

Punktens id-nummer:

Ta bort

Figur 11.5. Formulär för borttagning av punkter.

Från metoden PageLoad() görs anrop till metoden getTables() för att ta reda på från vilka lager punkter kan tas bort. Därifrån skickas en fråga av typen GetCapabilities till servleten och svaret läses för att plocka ut tabellnamnen och presentera dem i en rullgardinsmeny. Metoden *doDelete()* i C#-filen anropas när användaren klickar på knappen "Ta bort". Först kontrolleras att id-nummer har angetts, i annat fall visas ett felmeddelande. Om id-nummer har angetts konstrueras ett *Transaction*-anrop som skickas till servleten. Svaret läses och ett meddelande som anger ifall borttagningen lyckades eller inte visas över rullgardinsmenyn. Om användaren anger ett id-nummer som inte existerar innehåller meddelandet information om detta.

11.1.4 Integrering i karttjänsten

Den implementation av klienterna som hittills beskrivits har ingen koppling till karttjänsten, men fungerar för redigering som fristående klienter. Tanken var dock att en inpassning till karttjänsten skulle göras för att användare skulle kunna ange en punkts position eller välja en punkt att uppdatera eller ta bort genom att klicka i kartan. Detta har gjorts med hjälp av JavaScript. Kod har lagts till dels i de tre aspx-filerna som bygger upp formulären och dels i

de skript som ArcIMS genererade då karttjänsten skapades. Hur implementeringen av detta har gjorts beskrivs relativt detaljerat i fortsättningen av avsnittet.

När ett av formulären för redigering laddas sätts alla lager som kan redigeras synliga genom att ändra värden i vektorn *LayerVisible* som skapats i filen *aimsLayers.js*, vilken ArcIMS genererat. För att göra detta behövs de namn på lagren som ArcIMS använder. Dessa är tyvärr inte desamma som tabellnamnen, därför har namnen hårdkodats. För att se lagren måste kartan laddas om, detta görs genom ett anrop av funktionen *sendMapXML()* som finns i filen *aimsXML()*. För att ange vad som skall hända vid klick i kartan görs även ett anrop av funktionen *clickFunction()* i filen *aimsClick()*. Funktionen tar en inparameter som bestämmer vad som skall ske vid klick i kartan. Två nya värden för denna parameter har lagts till i funktionen, dessa heter *exjobb_xy* respektive *exjobb_id*. I figur 11.6 visas den kod som lagts till i funktionen. I båda fallen anges bl.a. att zoomning och panorering inte skall kunna ske samt hur muspekaren skall se ut. Sist anges attributet *toolMode*, det är detta attribut som avgör i vilket fall i funktionen *customMapTool()* i filen *aimsCustom.js* man hamnar vid klick i kartan.

```
case "exjobb_xy":
            panning=false;
            zooming=false;
            selectBox=false;
            shapeSelectBuffer = false;
            hideLayer("measureBox");
            modeBlurb = modeList[21];
            var theCanvas = document.getElementById("canvas");
            theCanvas.style.zIndex="3";
            if (isIE)    {
                        document.all.theTop.style.cursor = "hand";
                        theCursor = document.all.theTop.style.cursor;
            }
            toolMode = 1008;
break;

case "exjobb_id":
            panning=false;
            zooming=false;
            selectBox=false;
            shapeSelectBuffer = false;
            hideLayer("measureBox");
            modeBlurb = modeList[21];
            var theCanvas = document.getElementById("canvas");
            theCanvas.style.zIndex="3";
            if (isIE)    {
                        document.all.theTop.style.cursor = "hand";
                        theCursor = document.all.theTop.style.cursor;
            }
            toolMode = 1009;
break;
```

Figur 11.6. Kod som lagts till i funktionen clickFunction() i den ArcIMS-genererade filen aimsClick.js.

När användaren klickar i kartan anropas metoden *mapTool()* i filen *aimsClick.js*, om variabeln *toolMode* har ett värde som är större än 1000 anropas sedan *customMapTool()* i filen *aimsCustom.js*. Där har två fall lagts till. Om *toolMode* har värdet 1008 skall x- och y-koordinat för klicket fyllas i formuläret och är värdet 1009 skall istället id-nummer för den punkt användaren klickat på fyllas i. Den tillagda koden visas i figur 11.7. I och med att attributet *id* har angetts för textrutorna med koordinater kan värdena enkelt fyllas i. Koordinaterna fås från parametrarna *mapX* respektive *mapY*. För att få reda på id-nummer för

en punkt måste en fråga till servern skickas. Frågan sätts ihop i den nya funktionen *exjobbIdentify()* i filen *aimsIdentify()*, där anges bl.a. att alla attribut till hittade punkter skall returneras. Variabeln *XMLMode* ges ett nytt värde, 1009, för att kunna ta hand om svaret på rätt sätt i funktionen *useCustomFunction()* i filen *aimsCustom.js*. Servern letar endast efter matchningar inom ett specifikt lager, därför anges från aspx-filerna även vilket lager som är aktivt när sidan laddas samt när aktivt lager förändras. Detta görs genom anrop av funktionen *setActiveLayer()* i filen *aimsLayers.js*.

```
if (toolMode==1008) {        //exjobb_xy.
              parent.TextFrame.document.getElementById('x').setAttribute('value',mapX);
              parent.TextFrame.document.getElementById('y').setAttribute('value',mapY);
              return false;
}
if (toolMode==1009) {        //exjobb_id.
              exjobbIdentify(e);
              return false;
}
```

Figur 11.7. Kod som lagts till i funktionen customMapTool() i filen aimsCustom.js.

I funktionen useCustomFunction() i filen aimsCustom.js läses svaret på server-anropet. Koden för detta visas i figur 11.8. Först kontrolleras hur många punkter som hittades. Om exakt en punkt hittades plockas punktens id-nummer ut ur svaret och skrivs in i textrutan i formuläret. I annat fall visas en alert-ruta för användaren som informerar om att antingen ingen punkt eller flera punkter hittades.

```
} else if (XMLMode==1009) {              //exjobb_id
              replyArray[ActiveLayer]=theReply;
              var exInd = theReply.indexOf("<FEATURECOUNT count=");
              var exStart = exInd+20;
              var exEnd = exInd+23;
              var exCount = theReply.substring(exStart,exEnd);
              if(exCount=='"0"'){
                            alert("Klicka på en punkt.");
              }else if(exCount=='"1"'){
                            exInd = theReply.indexOf('OBJECTID="');
                            if(exInd!=-1) {
                                          exStart = exInd+10;
                                          exEnd = theReply.indexOf('"',exStart);
                                          var exID = theReply.substring(exStart,exEnd);
              parent.TextFrame.document.getElementById('id').setAttribute('value',exID);
                            }
              }else{
                            alert("Du klickade på flera punkter.");
              }
}
```

Figur 11.8. Kod som lagts till i funktionen useCustomFunction() i filen aimsCustom.js.

Beroende på vilken typ av redigering som är vald skall olika saker ske då användaren klickar i kartan. Detta anges då sidan laddas. Vid insättning skall koordinaterna för punkten användaren klickade i fyllas i textrutorna och vid borttagning är det id-nummer för den punkt som klickades som skall fyllas i. Vid uppdatering kan det vara bra att kunna använda sig av båda funktionerna så att användaren exempelvis först kan ange vilken punkt som skall uppdateras och sedan ange dess nya position. Därför finns det i formuläret för uppdatering möjlighet att välja vilken av dessa funktioner som skall vara aktiv. Två bilder används för att

byta och visa vilken av funktionerna som är aktiverad (se figur 11.4). Bilden som symboliserar den valda funktionen har en röd kant, vid klick på bilderna anropas JavaScript-funktioner som anger källa till bilderna så att bilden för den nu aktiva funktionen har röd kant. Dessutom anropas funktionen *clickFunction()* i filen *aimsClick()* för att ange vilken funktion som skall användas vid klick i kartan.

När en insättning eller borttagning lyckats anropas funktionen *sendMapXML()* för att ladda om kartan så att förändringarna syns. Anropet sker genom att med en if-sats kontrollera värdet på meddelandet i sidans topp. Om meddelandet börjar med ”Insättningen lyckades” respektive ”Borttagningen lyckades” laddas kartan om, i annat fall inte. Det tar tid att ladda om kartan så detta bör inte ske i onödan. Att en uppdatering lyckas behöver inte betyda att sidan behöver laddas om, därför bör inte samma metod användas här. Ändringarna kommer endast att synas i kartbilden om punktens koordinater har uppdaterats. För att hålla reda på om så är fallet används en textsträng med värdet *true* eller *false*. Textsträngen har gjorts osynlig för användaren. Istället för att kontrollera om meddelandet i sidans topp börjar med ”Uppdateringen lyckades” kontrolleras om den nya textsträngen har värdet *true*, i så fall laddas kartbilden på nytt.

11.2 Servleten

Den servlet som används för redigering av punktlager är klassen *sdeServlet* som ärver klassen *HttpServlet* (se avsnitt 5.2.1). I bilaga 2 finns en specifikation av klassen och i detta avsnitt kommer klassens uppbyggnad gås igenom. Metoderna *init()*, *doGet()*, *doPost()* samt *destroy()* har överskuggats och förutom dessa publika metoder finns även elva privata metoder. Klassen innehåller fyra attribut. Det första heter *conn* och är av typen *SeConnection*. Det används för att komma åt ArcSDE, kopplingen skapas i metoden *init()* och stängs i *destroy()*. De andra tre attributen är av typen *String* och används för att konstruera svaret på klientens anrop. Attributet *send_xml* används vid alla typer av anrop och innehåller det XML-meddelande som skall skickas som svar på anropet. De andra attributen, *insertRes* och *transactionRes*, används för att hålla reda på delar av svaret vid *Transaction*-anrop. *insertRes* innehåller den del av svaret som ska stå i elementet *InsertResults* medan *transactionRes* istället innehåller den del som ska stå i elementet *TransactionResponse* (se avsnitt 4.2.6).

Vid ett klient-anrop vidarebefordrar *service*-metoden i superklassen frågan till metoderna *doGet()* respektive *doPost()* i subklasssen. I *doGet()* kontrolleras om frågan innehåller parametern *request*. Om parametern finns och den har värdet *GetCapabilities* anropas den privata metoden *doGetCapabilities()*. Är inte parametern angiven eller om den har ett annat värde returneras ett felmeddelande i XML-format som svar på anropet. Metoden *doPost()* kontrollerar rotelementet i det XML-meddelande klienten skickat. Är rotelementet av typen *GetCapabilities*, *DescribeFeatureType* eller *wfs:Transaction* anropas metoden *doGetCapabilities()*, *doDescribeFeatureType()* respektive *doTransaction()*. I annat fall returneras även här ett felmeddelande i XML-format, detsamma gäller om inte frågan är korrekt formulerad.

11.2.1 doGetCapabilities()

Svaret på en fråga av typen GetCapabilities är ett XML-dokument som beskriver karttjänsten, dess operationer samt vilka lager som finns. Vilka lager som skall vara tillgängliga kan ändras genom att lager tas bort eller läggs till i ArcSDE. Denna information behöver därför genereras vid varje anrop för att säkert vara aktuell, övrig information ändras enbart om

implementationen av servleten ändras och behöver inte genereras vid varje nytt anrop utan kan lagras i en fil. Svaret består därmed av tre delar varav den första och sista är statiska och finns lagrad i filerna *cap_start* respektive *cap_end*. Mittendelen är den del som talar om vilka lager som kan redigeras och denna del genereras varje gång frågan ställs. För att läsa in de delar av svaret som lagras i filer används den privata metoden *fileToString()*. Metoden har sökvägen till en fil som inparameter och returnerar innehållet i filen som en textsträng. För att sätta ihop mittendelen av svaret används först metoden *getLayers()* i klassen *SeConnection* för att få reda på vilka lager som finns. Sedan stegas dessa igenom och om lagret skall vara tillgängligt adderas information om lagret till attributet *send_xml*.

För att ett lager ska vara tillgängligt för redigering krävs att det är ett punktlager. I dagsläget krävs dock även att lagret är ett av mina testlager som inte är versionshanterade. Anledningen till detta är att ESRI officiellt rekommenderar att redigering av data via Java API enbart sker på lager som inte är versionshanterade [53] och att de lager som egentligen skall kunna redigeras via karttjänsten är versionshanterade. En anledning till att göra ett lager versionshanterat är att det annars inte går att redigera data via ArcMap. Som nämnts i avsnitt 8.2 är det detta program som Lunds kommun använder för redigering av data i dagsläget, därför är de lager som skall kunna redigeras versionshanterade och testlager får istället användas.

11.2.2 doDescribeFeatureType()

Metoden har klientens fråge-XML som inparameter och börjar med att ta reda på om frågan gäller alla eller enbart en delmängd av de tillgängliga lagren. I det första fallet anropas metoden *getCols()* för varje lager som skall vara tillgängligt för redigering, i andra fallet enbart för de lager som angetts i frågan. Innan *getCols()* anropas adderas starttaggen på svarets rotelement till attributet *send_xml* och efter att alla anrop har gjorts läggs även sluttaggen till. Namnet på lagret ges som inparameter till metoden *getCols()*. Metoden tar sedan reda på vilka attribut som finns till lagret och stegar igenom dessa för att addera informationen till svaret.

11.2.3 doTransaction()

Metoden tar reda på alla element som ligger direkt under rotelementet i klientens XML-meddelande. Elementen stegas igenom och för varje element *Insert*, *Update* eller *Delete* görs anrop till metoden *doInsert()*, *doUpdate()* respektive *doDelete()* med elementet som inparameter. Dessa metoder utför redigeringen i ArcSDE och returnerar ett heltal som anger hur många redigeringar som har utförts. Denna information sparas i en heltalsvektor med en plats för respektive operation. Efter att alla element har gåtts igenom används informationen i heltalsvektorn för att konstruera svaret till klienten. Det kontrolleras även om attributen *insertRes* och *transactionRes* innehåller någon information, i så fall adderas denna information till svaret.

Ett *Insert*-element kan innehålla information för att sätta in punkter i flera olika lager genom att elementet innehåller ett element för varje lager där punkter skall sättas in. I metoden *doInsert()* gås dessa element igenom. För varje element hämtas en referens till lagret och skickas med som inparameter till metoden *makeInsertion()*. Även en lista med de element som beskriver den nya punkten skickas med som parameter. Det är i metoden *makeInsertion()* som insättningen av punkten görs. Klientens XML-meddelande innehåller information om namn och värde för de attribut som skall anges, men för att kunna göra insättningen krävs

även information om vilken datatyp attributet lagras som. För att ta reda på det används lagrets kolumndefinitioner, *SeColumnDefinition*, samt den privata metoden *findColIndex()*. Denna metod har vektorn med lagrets kolumndefinitioner samt namnet på ett specifikt attribut som inparameter och returnerar det index i vektorn där attributet finns lagrat. Finns inte angivet namn med i vektorn returneras istället *-1*. Denna information används sedan för att ange attributen på den punkt som skall sättas in i tabellen. Lyckas insättningen hämtas information om vilket unikt id-nummer ArcSDE har gett den nya punkten och adderas till attributet *insertRes* innan *true* returneras. Om operationen misslyckas adderas istället information om var felet uppstod till attributet *transactionRes* innan *false* returneras. Anledningen till att metoden *makeInsertion()* returnerar *true* eller *false* är för att i metoden *doInsert()* lätt kunna hålla reda på antalet insatta objekt för att sedan kunna returnera det till metoden *doTransaction()*.

För att utföra en uppdatering krävs information om vilka attribut som skall ändras, vad de nya attributvärdena är samt för vilka punkter i lagret uppdateringen gäller. Metoden *doUpdate()* hämtar först en referens till den tabell som skall uppdateras, sedan läses frågan för att ta reda på vilka attribut som skall ändras. För att ange attributens värden krävs precis som vid insättning information om vilken datatyp attributet lagras som, därför används den privata metoden *findColIndex()* även här. Information om vilka punkter uppdateringen gäller finns i elementet *Filter* (se avsnitt 4.2), är detta element tomt gäller uppdateringen alla punkter och inget where-villkor anges. I annat fall används den privata metoden *computeWhere()* för att läsa varje villkor och konstruera den textsträng med where-villkor som används för uppdateringen. Metoden *doUpdate()* innehåller en parameter för att hålla reda på hur många uppdateringar som har gjorts. Eftersom det är möjligt att en uppdatering påverkar flera punkter är detta inte nödvändigtvis detsamma som antalet uppdaterade punkter. Misslyckas en uppdatering adderas information om detta till attributet *transactionRes*. Efter att alla uppdateringar har gjorts returneras antalet uppdateringar till metoden *doTransaction()*.

Metoden *computeWhere()* är en **rekursiv metod** som används för att ta fram det where-villkor som används vid uppdatering och borttagning. Som inparameter anges det element som skall läsas, namnet på det attribut som innehåller lagrets unika id-nummer samt en parameter *not*. Denna parameter anger om elementet ligger i ett jämt eller udda antal *NOT*-element, dvs. om uttrycket skall vara som det är eller tvärtom. Om exempelvis ett element *ogc:PropertyIsEqualTo* ligger i ett element *NOT* skall det tolkas som "är inte lika" istället för som "är lika" och parametern *not* är därmed *true*. Metoden tar reda på vilken typ av element som skickats in för att kunna sätt ihop villkoret på rätt sätt. Om elementet innehåller ytterligare element görs rekursiva anrop på dessa. Som svar returneras en textsträng med den del av villkoret som finns i elementet. En begränsning är att metoden inte klarar av några rumsliga villkor. Följande element kan hanteras:

- AND
- OR
- NOT
- ogc:GmlObjectId
- ogc:FeatureId
- ogc:PropertyIsEqualTo
- ogc:PropertyIsNotEqualTo
- ogc:PropertyIsLessThan
- ogc:PropertyIsLessThanOrEqualTo
- ogc:PropertyIsGreaterThan
- ogc:PropertyIsGreaterThanOrEqualTo

- ogc:Add
- ogc:Sub
- ogc:Mul
- ogc:Div
- ogc:PropertyName
- ogc:Literal
- ogc:PropertyIsNull
- ogc:PropertyIsBetween

Metoden *doDelete()* som används för borttagning av punkter börjar med att ta reda på vilket lager punkterna skall tas bort ifrån. Sedan används precis som vid uppdatering den privata metoden *computeWhere()* för att sätta ihop det where-villkor som identifierar de objekt som skall tas bort. Lyckas en borttagning ökas den variabel som håller reda på antalet borttagningar och misslyckas operationen adderas istället information om detta till attributet *transactionRes*. Efter att alla borttagningar utförts returneras antalet lyckade borttagningar. Precis som vid uppdatering är det antalet lyckade anrop för borttagning som menas och inte antalet borttagna punkter. Om villkoret för vilka punkter som skall tas bort inte matchar några punkter anses det som en misslyckad borttagning och information adderas till attributet *transactionRes*.

12 Diskussion

Efter att det bestämts att redigeringarna av data i ArcSDE skulle utföras från en servlet med hjälp av ESRI:s Java API skulle servletens utformning bestämmas. Det fanns flera alternativ till hur servleten kunde vara uppbyggd. De två som beaktades var följande:

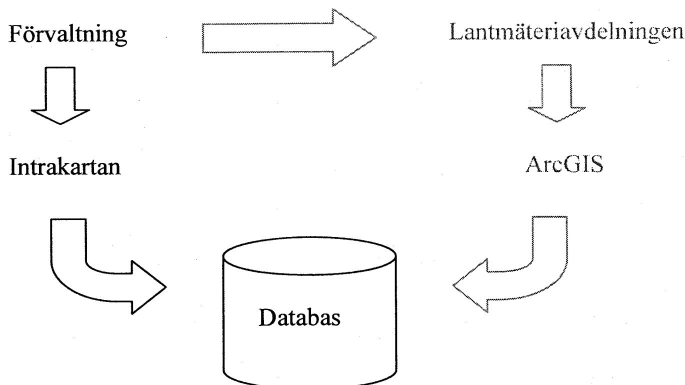
1. Servleten innehåller såväl funktionalitet som användargränssnitt och returnerar en HTML-sida som svar på ett klient-anrop.
2. Servleten innehåller enbart funktionaliteten för att utföra insättning, uppdatering samt borttagning. Det svar som returneras är i XML-format (enligt WFS-standarden).

Det senare alternativet är det som har använts. Anledningen till det är att det genom att skilja på användargränssnitt och funktionalitet blir lättare att enbart uppdatera en av delarna vid behov. Dessutom ger denna lösning möjlighet för flera olika klienter att anropa servleten för att utföra insättning, uppdatering eller borttagning av geografiska data. När det gäller antalet anrop till servleten borde det behövas ungefär samma mängd anrop oavsett vilken av dessa tekniker som används. Det är vid samma händelser servleten behöver anropas: vid laddning av sidan, vid byte av lager att redigera samt vid insättning/uppdatering/borttagning av punkt.

Då det bestämts att funktionalitet och användargränssnitt skulle skiljas åt återstod att bestämma teknik för användargränssnitt samt hur kommunikationen skulle ske. Anledningen till att ASP.NET och C# valdes för användargränssnittet var att denna teknik använts tidigare till exempelvis de sökfunktioner som lagts till karttjänsten och att tekniken har fungerat bra. För kommunikation mellan klienten med användargränssnittet och servleten med funktionaliteten används WFS. En anledning till att använda WFS är att det är en standardiserad teknik vilket medför att det finns specificerat hur frågor och svar skall vara uppbyggda. Detta medför att det blir enklare för andra att kommunicera med servleten eftersom det inte finns några tvivel om hur kommunikationen skall ske. En nackdel är att det krävdes en hel del tid till att sätta sig in i och tolka standarden samt att det ibland krävdes en del "onödig" implementation. Med det menas att för att följa standarden krävdes att specialfall togs om hand fastän de aldrig inträffar med den tänkta användningen. Exempelvis måste det kontrolleras att det koordinatsystem koordinaterna anges i är giltigt fastän klienten alltid använder sig av Lunds lokala koordinatsystem.

De textrutor som användaren kan fylla i attribut-värden i är vanliga HTML-element. Eftersom textrutorna skapas i ett *Repeater*-objekt kan de mig veterligen inte kommas åt från C#-filen på något enkelt sätt även om de görs som webbkontroller. Om de inte enkelt kan kommas åt från C#-filen ser jag ingen anledning till att ha dem som webbkontroller och därför gjordes de istället som vanliga HTML-element.

Den nya funktionaliteten i *Intrakartan* gör det möjligt för användare från olika förvaltningar att själva redigera data. Arbetet blir då i viss mening effektivare än tidigare då de var tvungna att be Lantmäteriavdelningen om hjälp för att utföra insättningar, uppdateringar samt borttagningar av punktobjekt. Det blir effektivare i den mening att ett steg i händelsekedjan elimineras, se figur 12.1. Utförandet av själva redigeringarna blir dock troligen inte effektivare i början eftersom de personer som nu utför redigering av data inte är vana vid arbetsuppgiften.



Figur 12.1. Händelsekedja vid redigering av punktdata med ny (visas i svart) respektive gammal (visas i grått) metod.

Varje förvaltning har egna punktlager som skall hållas aktuella. För att ingen skall kunna ändra i någon annans data vore det önskvärt om det gick att sätta rättigheter på lagren så att varje förvaltning endast kan redigera sina egna data. Hur detta skulle kunna göras är dock inget som har undersökts.

I ett tidigare skede innehöll formuläret information om vilka datatyper attributen tillhörde. Denna information togs bort för att inte riskera att förvirra användaren då det inte är troligt att personer i målgruppen vet vad t.ex. *int*, *double* och *string* innebär. Eventuellt skulle denna information kunna finnas med om namnen ändrades till exempelvis heltal, decimaltal och text.

12.1 Kommentarer om tekniken

Det finns flera tekniska brister med lösningen. Vissa beror på tidsbrist och vissa på att teknikstöd saknas. Detta innebär att flera av bristerna skulle kunna åtgärdas, men inte alla. Två brister i implementeringen av servleten som skulle kunna åtgärdas är att servleten inte kan hantera spatiala filter och inte heller *handle* (se avsnitt 4.2). Anledningen till att inte dessa saker implementerats är dels tidsbrist och dels att de inte prioriterats eftersom de inte används av den klient som skrivits. Att inte spatiala filter kan hanteras kan utläsas ur svaret på ett *GetCapabilities*-anrop. Där kan även utläsas att vid insättning av punkter genereras alltid ett nytt id-nummer till punkten, möjlighet att själv ange id-nummer för en ny punkt saknas därmed. Detta är naturligt eftersom ArcSDE automatiskt skapar en id-kolumn till en ny tabell och sedan genererar ett nytt id för varje ny punkt.

Versionshanterade tabeller kan inte redigeras via de nya funktionerna i karttjänsten. Detta medför att ett val måste göras då ett nytt kartlager skapas. Antingen görs lagret versionshanterat och redigering utförs via ESRI:s program ArcMap eller så görs inte lagret versionshanterat och redigering av data sker via *Intrakartan*. Oavsett vilket alternativ som väljs är data synlig i såväl *Intrakartan* som ArcMap. I ESRI:s Java API finns objekt som kan användas för att redigera en versionshanterad tabell. Vid fråga till ESRI Support om hur dessa

objekt skulle användas blev svaret att de officiellt inte rekommenderar att redigering av versionshanterade tabeller i ArcSDE sker med hjälp av deras Java API [53]. Istället rekommenderas att redigering av sådana data sker via ArcGIS Desktop eller en ArcObjects klient. Tyvärr framkom inte detta förrän i arbetets slutskede, i annat fall kunde möjligheten att använda sig av ArcObjects ha undersökts för att möjligen få en bättre lösning där redigering av data i en tabell kan ske från både *Intrakartan* och ArcMap. Alternativt kan en testdatabas skapas för att se om det fungerar att redigera data via Java API trots att detta inte är något som offficiellt rekommenderas av ESRI.

Om lösningen ska användas över Internet bör säkerheten förbättras. Servleten gör ingen kontroll av vem som anropar den och inte heller någon kontroll av att det är ett tillåtet kartlager som redigeras. Säkerheten är med andra ord låg, men eftersom tjänsten endast är tillgänglig via Intranätet ställs inte lika höga krav. Dessutom behövs alltid ett korrekt tabellnamn för att redigera data och endast lager som är tillåtna att redigera finns med i svaret på *GetCapabilities*. Det innebär att risken att ett otillåtet lager redigeras trots allt är ganska liten, de som har tillgång till lagernamnen har troligen även rättighet att redigera dessa. Att göra systemet säkrare i det avseendet kräver dock enbart att innan en insättning, uppdatering eller borttagning görs kontrollera att tabellen finns med i det svar som genereras vid *GetCapabilities*.

Två brister som nämnts tidigare (avsnitt 11.2.3) är att antalet uppdateringar och borttagningar inte syftar på antalet punkter som har påverkats av operationen utan antalet gånger operationen lyckats. Det innebär att om flera punkter identifieras med samma where-villkor räknas det ändå bara som en uppdatering/borttagning. För att svaret istället ska innehålla det antal punkter som har tagits bort skulle antalet rader i tabellen kunna räknas före och efter borttagningen. Att ta reda på antalet rader i en tabell verkar dock inte gå att göra på något effektivt sätt, en fråga som hämtar alla rader måste ställas och sedan får dessa stegas igenom för att räkna antalet rader. Denna metod skulle inte kunna användas vid uppdatering eftersom antalet rader då inte förändras. Genom att ställa en fråga med samma where-villkor som används vid uppdateringen och sedan stega igenom och räkna antalet rader skulle antalet rader som *borde* påverkas av uppdateringen ges. Men eftersom dessa sätt att räkna antalet rader som påverkats av operationen inte görs i samma ögonblick som operationen skulle det teoretiskt sett kunna bli så att en annan användare redigerar tabellen samtidigt. Om denna användare tar bort eller sätter in rader mellan räkningen av rader före och efter borttagningen skulle denna metod ge ett felaktigt resultat. Det bästa vore om det fanns metoder till objekten *SeUpdate* och *SeDelete* för att ta reda på antalet påverkade rader på samma sätt som det till *SeInsert* finns en metod för att ta reda på den senast insatta punktens id-nummer.

När ett fel uppstår vid redigering av data skall ett meddelande om felet adderas till svaret. Även information om var felet uppstod skall ingå. Detta görs inte på något bra sätt. Lokaliseringen av felet innehåller enbart information om vid vilken typ av operation felet uppstod dvs. om det var vid insättning, uppdatering eller borttagning. Det vore önskvärt att lokalisera felet ytterligare, exempelvis genom att ange vid vilket element i frågan felet uppstod. Meddelandet ger visserligen en liten förklaring till varför felet uppstod men om flera operationer av samma typ görs går det inte att utläsa vid vilken av dem ett fel inträffade. Om implementeringen ändrades så att *handle* användes skulle det kunna användas för lokalisering, under förutsättning att klienten använder sig av *handle*.

12.2 Kommentarer om användartestet

Hur användarvänlig klienten är har endast undersökts med en liten studie. Eftersom studien enbart utfördes av en person visar det endast vad den personen anser. För att få en bättre bild av användbarheten borde fler testpersoner användas. Eventuellt borde testpersonerna först ges en genomgång av hur redigering går till, under förutsättning att detta är tänkt att ske innan metoden tas i bruk. Sedan borde testpersonen få utföra uppgiften utan hjälp för att få en så realistisk situation som möjligt. Om samma person kommer att använda funktionen relativt mycket kommer personen att lära sig hur redigeringen går till. Därför borde eventuellt även testet undersöka användbarheten i de fall användaren har viss vana av arbetet. Detta för att exempelvis se hur effektiv funktionen upplevs.

För att tydligare identifiera brister med användargränssnittet behövs, som nämnts ovan, ett utförligare användbarhetstest. Studien som gjorts (se avsnitt 9.4) ger dock en indikation på vad som borde förbättras. En kommentar som framkom vid testet och som är gemensam för alla tre formulär var att placeringen av förklaringstexten borde flyttas. Denna har förändrats efter testets genomförande. Genom att placera förklaringarna i ett separat dokument fanns även plats för utförligare förklaringar. Framförallt har hjälpen till uppdatering utökats eftersom det var den operation som var svårast att använda för testpersonen.

Hur knapparna för att växla mellan att välja punkt för uppdatering respektive ny position för punkten genom klick i kartan fungerar förklaras nu tydligare i hjälpen. Men dessa knappar kanske inte behövs? Det kanske vore bättre om punkt att uppdatera väljs först och sedan när en punkt har valts ändras funktionen så att vid fortsatta klick i kartan anges istället en ny position för punkten. Frågan är även om det inte vore bättre att i kartan markera den punkt som valts samt punktens nya position. Detta blir tydligare än ett id-nummer och koordinatvärden i en textruta. Önskemålet om att punkten skall flyttas omgående känns dock lite osäkert, det är nog bättre att markera i kartan direkt men fortfarande låta användaren bekräfta med ett klick på knappen.

Beträffande önskemålet om att gamla attributvärden borde vara ifyllda vid uppdatering instämmer jag i att detta säkert underlättar för användarna. Det skulle dock kräva ganska mycket arbete. Om användaren alltid väljer punkt genom att klicka i kartan skulle attributvärdena kunna plockas ur server-anropet på samma sätt som punktens id-nummer hämtas. För att kunna fylla i dem i textrutorna krävs dock att attributet *id* för textrutorna är känt. Detta borde gå att lösa genom att ge rutorna samma *id* som namnet på det attribut de innehåller. Problemet blir i så fall att fylla i koordinatvärdena. Denna information finns inte i svaret från serverna eftersom de lagras som ett attribut *shape* istället för som en x-koordinat och en y-koordinat. Alternativt kan servleten implementera även WFS-operationen *GetFeature* och implementeringen av klienten ändras så att ett anrop av *GetFeature* görs då värdet i rutan för punktens id-nummer förändras.

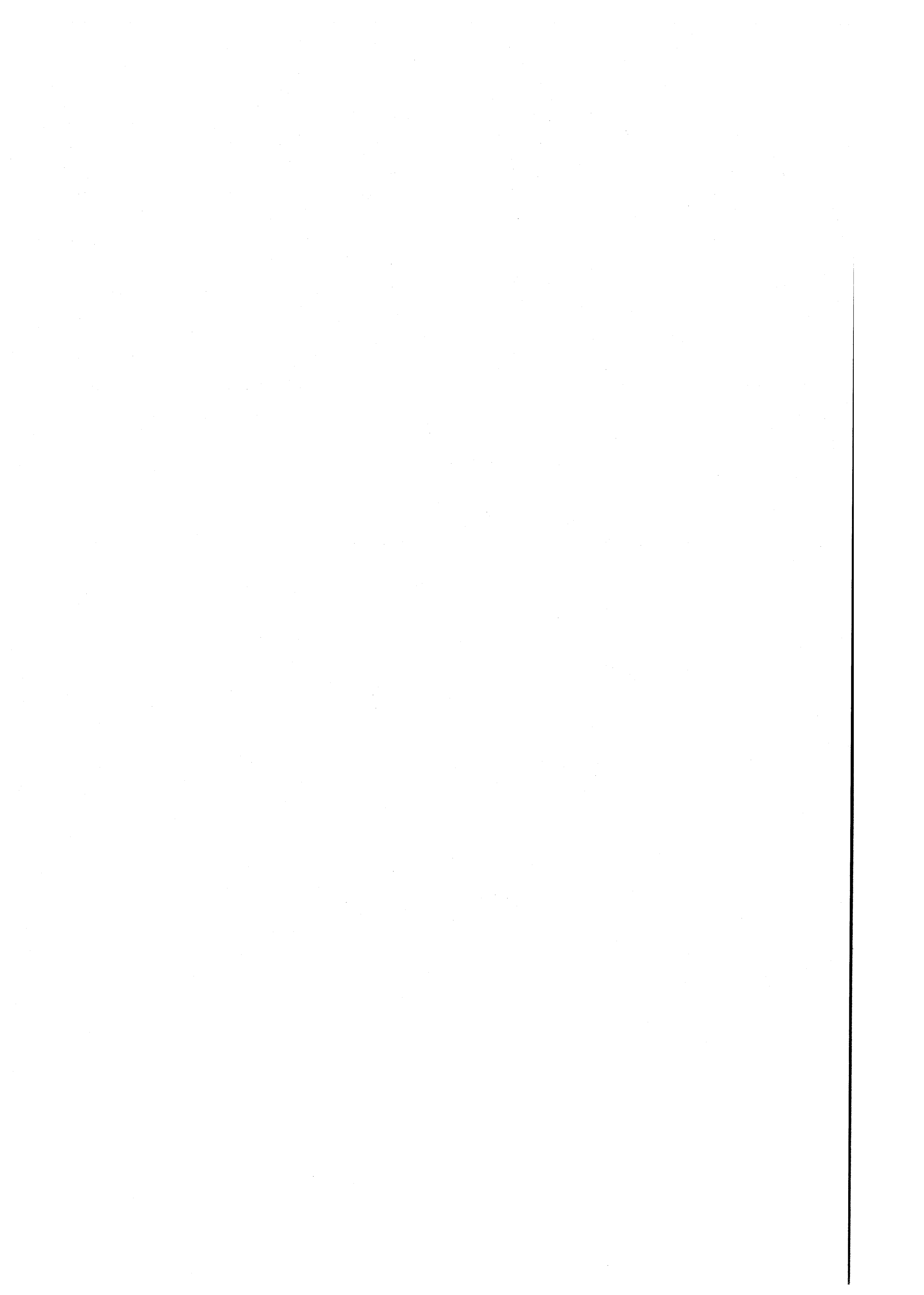
13 Slutsatser

Syftet med examensarbetet är att utöka *Intrakartan* med funktionalitet för redigering av punktlager med hjälp av WFS-teknik. Detta har utförts och syftet har därmed uppfyllts. En begränsning med lösningen är att för att data i ett punktlager skall kunna redigeras krävs att tabellen inte är versionshanterad. För att kunna redigera data via programmet ArcMap, som används av kommunen idag, krävs dock att tabellen är versionshanterad. Detta medför att den som skapar kartlagret måste välja om det skall kunna redigeras via ArcMap eller med hjälp av den nya redigeringsfunktionen i karttjänsten. En lärdom som kan dras av detta är att noga kontrollera möjligheterna med vald teknik innan arbetet påbörjas.

En avgränsning som gjordes av arbetet i ett tidigt skede var att enbart WFS-operationerna *GetCapabilities*, *DescribeFeatureType* samt *Transaction* skulle implementeras. Eftersom servleten inte skulle användas för att visa information, det används karttjänsten till, behövdes inte operationen *GetFeature*. Vid studien av användargränsnittet framkom dock önskemål om att de gamla attributvärdena borde vara ifyllda vid uppdatering. För att göra detta möjligt borde även *GetFeature* ha implementerats även fast den vid första anblick inte verkade behövas.

Ytterligare en slutsats drogs vid utformningen av den tekniska lösningen av klienten. Ett *Repeater*-objekt användes för att skapa textrutor som attributen skulle anges i vid insättning och uppdatering. Eftersom rutorna för koordinaterna behövde kommas åt från JavaScript-funktioner gavs dessa rutor attributet *id* med värdet *x* respektive *y*. För att göra detta skapades textrutan i en if-sats med ett fall för x-koordinat, ett för y-koordinat och ett för övriga fall. Tanken var att skapa textrutorna som webbkontroller så att värdena kunde kommas åt från C#-filen. Problemet var att fastän bara ett fall i taget var uppfyllt och därmed bara en textruta borde skapas så skapades tre textrutor varje gång. Detta ledde till problemet att veta vilken typ av textruta som användes och värdena kunde därmed inte enkelt kommas åt från C#-filen.

En viktig slutsats när det gäller användbarhet är att det är svårt att som utvecklare avgöra hur mycket funktioner som är optimalt. Det är lätt att se möjligheter som en användare ur målgruppen mest blir förvirrad av. Därför är det viktigt att använda sig av användbarhetstester, helst redan i ett tidigt skede för att slippa onödigt arbete. Det viktigaste är alltid att användarna blir nöjda med tjänsten eftersom de skall använda sig av den.



Referenser

[1] Geoforums hemsida.
http://www.geoforum.se/page/152/152/933 (2006-10-13)

[2] Eklund, Lars (red.) (2003), *Geografisk informationsbehandling Metoder och tillämpningar*. Tredje upplagan. Formas, Stockholm. ISBN 91-540-5904-6.

[3] GIS-portalen gis.com, skapad av *ESRI*.
http://www.gis.com/implementing_gis/data/data_types.html (2006-10-13)

[4] GIS-portalen gis.com, skapad av *ESRI*.
http://www.gis.com/implementing_gis/data/usingdata.html (2006-10-13)

[5] Nationalencyklopedin på Internet. Sökord: GIS.
http://www.ne.se.ludwig.lub.lu.se/jsp/search/article.jsp?i_art_id=182680&i_word=GIS (2006-10-13)

[6] Peng, Zhonh-Ren & Tsou, Ming-Hsiang (2003), *INTERNET GIS Distributed Geographic Information Services for the Internet and Wireless Networks*. John Wiley & Sons, Inc., Hoboken, New Jersey. ISBN 0-471-35923-8.

[7] Organisationen *World Wide Web Consortium*'s hemsida.
http://www.w3.org/Consortium/ (2006-10-13)

[8] Sökning på Wikipedias engelska hemsida. Sökord: W3C.
http://en.wikipedia.org/wiki/W3c (2006-10-13)

[9] Organisationen *Open Geospatial Consortium*'s hemsida.
http://www.opengeospatial.org/ogc (2006-10-13)

[10] Organisationen *World Wide Web Consortium*'s hemsida.
http://www.w3.org/XML/ (2006-10-13)

[11] Harold, Elliotte Rusty & Means, W. Scott (2001), *XML IN A NUTSHELL A Desktop Quick Reference*. O´Reilly & Associates. ISBN 0-596-00058-8.

[12] Lake, Ron; Burggraf, David S.; Trnini´c, Milan & Rae, Laurie (2004), *Geography Mark-Up Language: Foundation for the Geo-Web*. WILEY. ISBN 0-470-87153-9 (Cloth) ISBN 0-470-87154-7 (Paper).

[13] Konsultföretaget *XML Sweden*'s hemsida.
http://www.xml.se/xml/standarder.html#schema (2006-10-13)

[14] Handledning för XML skapad av *World Wide Web Consortium*.
http://www.w3schools.com/xml/xml_namespaces.asp (2006-11-01)

[15] Organisationen *World Wide Web Consortium*'s hemsida.
http://www.w3.org/TR/2006/REC-xml-names-20060816/ (2006-11-01)

[16] Handledning för SVG skapad av *World Wide Web Consortium*. http://www.w3schools.com/svg/default.asp (2006-10-13)

[17] Organisationen *World Wide Web Consortium*'s hemsida. http://www.w3.org/Graphics/SVG/ (2006-10-13)

[18] OpenGIS® Web Map Service (WMS) Implementation Specification. http://portal.opengeospatial.org/files/?artifact_id=14416 (2006-10-13)

[19] OpenGIS® Web Feature Service (WFS) Implementation Specification. https://portal.opengeospatial.org/files/?artifact_id=8339 (2006-10-13)

[20] OpenGIS® Filter Encoding Implementation Specification. http://portal.opengeospatial.org/files/?artifact_id=8340 (2006-11-01)

[21] Sökning på Wikipedias svenska hemsida. Sökord: webbläsare. http://sv.wikipedia.org/wiki/Webbl%C3%A4sare (2006-10-13)

[22] Handledning för HTML skapad av *World Wide Web Consortium*. http://www.w3schools.com/html/default.asp (2006-10-13)

[23] Webbstudio med information om olika typer av programmering, tillhandahålls av *International Data Group*. http://internetworld.idg.se/webbstudio/pub/avdelning.asp?id=21 (2006-10-13)

[24] *Sun*'s hemsida. http://www.sun.com/aboutsun/coinfo/history.html (2006-11-01)

[25] Flanagan, David (2005), *JAVA™ IN A NUTSHELL A Desktop Quick Reference*. Fifth edition. O'Reilly Media Inc. ISBN 0-596-00773-6.

[26] Specifikation av ECMA:s standard för programmeringsspråket C#. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

[27] Deitel, H. M.; Deitel, P. J.; Listfield, J. A.; Nieto, T. R.; Yaeger, C. H. & Zlatkina, M. (2003), *C# for Experienced Programmers*. Prentice Hall, New Jersey. ISBN 0-13-046133-4.

[28] Handledning för ASP skapad av *World Wide Web Consortium*. http://www.w3schools.com/asp/default.asp (2006-10-13)

[29] Handledning för ASP.NET skapad av *World Wide Web Consortium*. http://www.w3schools.com/aspnet/default.asp (2006-10-13)

[30] Hemsida om CGI skapad av *The National Center for Supercomputing Applications* (NCSA). http://hoohoo.ncsa.uiuc.edu/cgi/intro.html (2006-10-13)

[31] *Sun*'s handledning för servlets. http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html (2006-10-13)

[32] Callaway, Dustin R. (1999), *Inside Servlets*. Addison-Wesley.

[33] Javas dokumentation av klassen HttpServlet.
http://java.sun.com/products/servlet/2.2/javadoc/ (2006-10-13)

[34] Hemsida för *Apache Tomcat*.
http://tomcat.apache.org/ (2006-11-01)

[35] *Sun*'s hemsida.
http://java.sun.com/applets/ (2006-10-13)

[36] Handledning för JavaScript skapad av *World Wide Web Consortium*.
http://www.w3schools.com/js/default.asp (2006-10-13)

[37] *Mozilla*'s hemsida.
http://www.mozilla.org/js/ (2006-10-13)

[38] *Microsoft*'s hemsida.
http://www.microsoft.com/technet/security/Bulletin/MS03-008.mspx
(2006-10-13)

[39] Payne, Chris, översättning av Sjölander, Roger & Gillberg, Lars (2002), *Sams Lär dig ASP.NET på 3 veckor*. Pagina förlag, Sundbyberg. ISBN 91-636-0693-3

[40] Elmasari, Ramez & Navathe, Shamkant B. (1994), *Fundamentals of database systems*. Second Edition. ISBN 0-8053-1753-8.

[41] Handledning för SQL skapad av *World Wide Web Consortium*.
http://www.w3schools.com/sql/default.asp (2006-10-13)

[42] Företaget *ESRI*:s hemsida.
http://www.esri.com/company/about/history.html (2006-10-13)

[43] Företaget *ESRI*:s svenska hemsida.
http://www.esri-sweden.com/templates/Page.aspx?id=192&menu=2_1
(2006-10-13)

[44] *ArcGIS® 9 What is ArcGIS® 9.1?* ESRI, USA.
Copyright © 2001-2005 ESRI

[45] Dokumentation av *ESRI*:s Java API version 8.3.
http://edndoc.esri.com/arcsde/8.3/index.htm (2006-11-01)

[46] Företaget *ESRI*:s svenska hemsida.
http://www.esri-sweden.com/templates/Page.aspx?id=327&menu=2_0
(2006-10-13)

[47] ArcIMS White paper.
http://files.esri-sweden.com/Server_GIS/Whitepapers/arcims9-architecture.pdf
(2006-10-13)

[48] Team Support, *ESRI Sweden*, Personlig kommunikation 30/11 2006.

[49] Information om karttjänsterna som finns tillgänglig via Lunds kommuns Intranät.

[50] Munsin, Anna-Stina, Lunds kommunala Lantmäteri, Personlig kommunikation, augusti - november 2006.

[51] Andréasson, Jonas, Chef på Lunds kommunala Lantmäteri, Personlig kommunikation, augusti - november 2006.

[52] Selin, Per, Team Support, *ESRI Sweden*, Personlig kommunikation 23/8 2006.

[53] Selin, Per, Team Support, *ESRI Sweden*, Personlig kommunikation 1/11 2006.

Bilaga 1 – Ordlista

API	*Application Programming Interface*. Innehåller definitioner av klasser och interface.
ASP	*Active Server Pages*. En teknik för dynamik på serversidan.
ASP.NET	Nyare version av ASP.
ArcXML	En XML-dialekt framtagen av ESRI.
C#	Objektorienterat programmeringsspråk utvecklat av *Microsoft*.
CGI	*Common Gateway Interface*. En teknik för dynamik på serversidan.
DHTML	Dynamisk HTML. HTML som gjorts dynamisk genom att exempelvis använda skript på klientsidan och stilmallar.
DTD	*Document Type Definition*. Ett dokument som används för att definiera en XML-dialekt.
ECMA	*European Computer Manufacturers Association*. En organisation som tar fram standarder inom bl.a. skript- och programmeringsspråk.
EPSG	*European Petroleum Survey Group*. Har definierat koder som används för att ange i vilket referenssystem koordinater anges i.
GIS	Geografiska InformationsSystem. Ett datoriserat informationssystem som används till geografiska data.
GML	*Geography Markup Language*. En XML-dialekt som används för att distribuera geografiska vektordata.
HTML	*HyperText Markup Language*. En XML-dialekt som används för att visa information på en webbsida.
Java	Objektorienterat programmeringsspråk utvecklat av *Sun*.
JavaScript	En typ av skriptspråk som kan integreras i HTML-dokument för att göra en webbsida mer dynamisk.
Java VM	*Java Virtual Machine*. Den del av Java-programvaran som används för webbinteraktion.
Pdf-format	*Portable Document Format*. Ett plattformsoberoende dokumentformat utvecklat av *Adobe Systems*.

SQL	*Structured Query Language*. Ett frågespråk som används vid kommunikation med relationsdatabaser.
SVG	*Scalable Vector Graphics*. En XML-dialekt som används för att presentera vektordata.
Rekursiv metod	En medod som rekursivt anropar sig själv till dess att ett basfall nås.
Tomcat	Webbserver som kan hantera servlets.
URI	*Uniform Resource Identifier*. En teckensträng som identifierar en Internet-resurs.
WMS	*Web Map Service*. Webbtjänst för distribuering av kartor i raster- och vektorformat.
WFS	*Web Feature Service*. Webbtjänst för distribuering av geografiska data i GML-format.
XML	*eXtensible Markup Language*. Ett standardiserat märkspråk för lagring av data på ett strukturerat sätt.
XML Schema	XML-baserat alternativ till DTD.

Bilaga 2 – Specifikation av servleten

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.esri.sde.sdk.client.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.*;

public class sdeServlet extends HttpServlet
{

  SeConnection conn;               //koppling till ArcSDE
  String insertRes;                //innehåll till elementet
                                   //wfs:InsertResults vid Transaction-anrop
  String transactionRes;           //innehåll till elementet
                                   //wfs: TransactionResponse vid Transaction-
                                   //anrop
  String send_xml;                 //det XML-meddelande som ska skickas som
                                   //svar på anropet

  /** Anropar metoden i super-klassen samt initierar attributet conn. */
  public void init(ServletConfig config) throws ServletException;

  /** Innehåller frågan parametern request och parametern har värdet
      GetCapabilities anropas den privata metoden doGetCapabilities(), i
      annat fall returneras ett exception. */
  public void doGet(HttpServletRequest request, HttpServletResponse
  response) throws IOException, ServletException;

  /** Läser frågan och anropar doGetCapabilities() om rotelementet är av
      typen GetCapabilities, doDescribeFeatureType() om rotelementet är av
      typen DescribeFeatureType eller doTransaction() om rotelementet är av
      typen wfs:Transaction. I annat fall returneras ett exception. */
  public void doPost(HttpServletRequest request, HttpServletResponse
  response) throws IOException, ServletException;

  /** Sätter värdet på attributet send_xml. För att göra detta anropas även
      metoden fileToString(). */
  private void doGetCapabilities();

  /** Läser innehållet i filen med sökvägen file och returnerar detta som
      en textsträng. */
  private String fileToString(String file);

  /** Sätter värdet på attributet send_xml. Innehåller rotelementet i doc
      ytterligare element kommer send_xml innehålla information om dessa
      lager och i annat fall information om alla tillgängliga lager. För
      att ta reda på information om ett lager anropas getCols(). */
  private void doDescribeFeatureType(Document doc);

  /** Lägger till information om kolumnerna i tabellen str_table till
      attributet send_xml. */
  private void getCols(String str_table) throws SeException;
```

```
    /** Sätter värdet på attributet send_xml. Går igenom elementen i doc och
        anropar doInsert() för varje element av typen wfs:Insert, doUpdate()
        för varje element av typen wfs:Update samt doDelete() för varje
        element av typen wfs:Delete. */
    private void doTransaction(Document doc);

    /** Utför insättningar, genom anrop av makeInsertion(), efter vad som
        anges i elementet n. Returnerar antalet insatta element. Upptäcks fel
        i frågan rapporteras detta med hjälp av attributet transactionRes. */
    private int doInsert(Node n);

    /** Sätter in en punkt i tabellen table, information om punkten som skall
        sättas in finns i parametrern nl. Upptäcks fel i frågan rapporteras
        detta med hjälp av attributet transactionRes, information om insatta
        punkter anges till attributet insertRes. */
    private boolean makeInsertion(SeTable table, NodeList nl);

    /** Returnerar index på den plats i colDefs som har namnet name. */
    private int findColIndex(SeColumnDefinition[] colDefs, String name);

    /** Utför uppdateringar efter vad som anges i elementet n. Returnerar
        antalet uppdaterade element. Upptäcks fel i frågan rapporteras detta
        med hjälp av attributet transactionRes. */
    private int doUpdate(Node n);

    /** Utför borttagningar efter vad som anges i elementet n. Returnerar
        antalet borttagna element. Upptäcks fel i frågan rapporteras detta
        med hjälp av attributet transactionRes. */
    private int doDelete(Node n);

    /** Rekursiv metod som beräknar ett where-villkor utifrån elementet n.
        Parametern id anger namnet på den kolumn i aktuell tabell som
        innehåller det unika id som ArcSDE har hand om. Parametern not anger
        om elementet n finns i ett jämnt (false) eller udda (true) antal NOT-
        element, dvs. om uttrycket ska vara som det är eller tvärtom. not har
        normalt värdet false. */
    private String computeWhere(Node n, String id, boolean not);

    /** Stänger kopplingen till ArcSDE. */
    public void destroy();
}
```