Integration of Icon and Text Positioning Algorithms in Web Map Service

Peter Ringberg

Department of Real Estate Science
Lund Institute of Technology

ISRN LUTVDG/TVLM 05/5116 SE

Avdelningen för
FASTIGHETSVETENSKAP
Lunds Tekniska Högskola
Lunds Universitet
Box 118
221 00 LUND

Department of
REAL ESTATE SCIENCE
Lund Institute of Technology
Lund University
Box 118
SE-221 00 LUND
SWEDEN

Integration of Icon and Text Positioning Algorithms in Web Map Service

Master of Science Thesis written by:
Peter Ringberg
Education Programme in Engineering, Land Surveying and Management
Lund Institute of Technology

Supervisor:
Lars Harrie, Department of Real Estate Science, Lund Institute of Technology

Examinator:
Klas Ernald Borges, Department of Real Estate Science, Lund Institute of Technology

June, 2005

ISRN LUTVDG/TVLM 05/5116 SE

Keywords: real-time maps, icon and text placement, WMS servers, WMS clients

Language: English

Preface

This Master of Science thesis completes my studies in land surveying at the Lund Institute of Technology. I must say that I am happy that I got the chance to enter the exciting GIS world through this study. The work has been interesting and the atmosphere at the GIS-centre has been inspiring.

Firstly I would like to thank my supervisor Lars Harrie for all his help and excellent feed-back throughout the process. Mehmet Bozkurt, Tobias Lennartsson and Peter Segerstedt (they are all referred to later on in this study) have played a major role throughout the project. Thank you! Roger Groth and Hanna Stigmar have provided valuable support at work. This study's examinator, Klas Ernald Borges, and opponent, Rasmus Lindberg, have presented good suggestions in order to complete the report. Jessika Nilsson improved my English with good proof-reading. Thank you.

Finally, I would like to thank everybody at the GIS-centre and all friends in Lund for a really enjoyable autumn semester. My memories from Lund will remain happy!

Härnösand, June, 2005

Peter Ringberg

Abstract

A map is nowadays more often a digital product than a paper product. Digital maps demand a standardized way of communication between the map provider and the user. There is also a need for maps that are of good cartographic quality. Since digital maps are automatically rendered and presented on a screen or a display the map-making process has been altered. The making of a paper map concentrates on a good overview and excellent cartographic quality. A digital map is restricted to its presentation device and therefore loses the overview property and resolution of a paper map. On the other hand it enables a set of new functions on a map, such as zooming between scales and different levels of interaction.

Well-defined, standardized interfaces between servers and clients are necessary when digital maps are rendered in real-time. The *Open Geospatial Consortium* (OGC) is an international standardization organization, which has developed standards that define the interface between a map-providing server and a client. Two of these are *Web Map Service* (WMS) and *Web Feature Service* (WFS). Briefly, WMS defines requests on the layer level whereas WFS defines requests on the feature level. There are several implementations of both servers and clients featuring these standards. Examples of both WMS servers and clients are given in this report.

The positions of icons and text are essential for the readability of a map. When a map is automatically generated in real-time, algorithms need to optimise these positions so that icons and text do not overlap objects in the map. This study works with symbol and text placing algorithms. These algorithms were developed within the *GiMoDig* project and this study documented them and made some modifications of the algorithms' programme structure.

Neither the WMS nor the WFS standard deals with icon and text placement matters. Thus, it is a task for developers of WMS servers to include such functionalities. The algorithms used within this study are effective and guarantee a high map quality. This report discusses an approach on how these algorithms could be implemented in a WMS/WFS service.

Contents

Appendix

An input file for ParmeterReader

Part I

-

Theory

1 Introduction

1.1 Background

Maps have been important for hundreds of years. Often, they were reserved for the king or commander-in-chief. In the course of time maps started to play a greater role for a larger amount of users. The increasing mobility of people made road atlases and hiking maps essential. On a public level, maps became more important in the fields of e.g. land registration and urban planning.

Today, with the introduction of Internet technologies, maps are more used than ever. The number of applications and situations where maps are used is diverse. Maps are used for presentation purposes, but also as a base for decisions. The Internet provides several different ways to present cartographic information. Most often, a single map image is presented in a browser, but it might as well be presented in tables or in other ways. Depending on the technique, different level of interaction can be used.

Aleady developed and recognized standards for the exchange of cartographic data are important when several actors want to interact with each other. The standardizing organisation *Open Geospatial Consortium* (OGC) has developed the *Web Map Service* (WMS) and the *Web Feature Service* (WFS) interfaces. These are standards for a client-to-server dialogue that define how maps can be requested, rendered and transferred over the Internet. WMS deals with whole maps whereas WFS deals with the data that constitute a map.

The WMS and WFS interfaces do not define rules of generalising or placing map objects in a certain order. This could be a problem since a map is always rendered from scratch each time that it is requested. This rendering process has to be fast and result in understandable maps. Therefore, adjustments might have to be carried out by the map service provider. Examples of such adjustments are that only a selection of important objects is shown on a map and that symbols and text are placed in a logical manner.

This study concerns firstly the WMS interface. A WMS server has been set up. A mobile phone WMS client developed within a master thesis study at Sony-Ericsson has used this server to acquire maps. Sony-Ericsson has then informed which improvements they expect from the server-side. This study also concerns the modification and integration of two web WMS clients.

Secondly, this study has worked with implemented algorithms for text and icon placement. These implementations were carried out through the *Geospatial Info-Mobility service by real-time Data-Integration and Generalisation* project (GiMoDig), founded by the European Union. The contribution of this study is a better structure of the programme containing the algorithms. A future task would be to integrate this programme into a WMS service.

1.2 Objectives

The main objectives of this master thesis are to set up a functional WMS server and to modify a program handling icon and text placement algorithms. A secondary objective is to describe how to integrate these algorithms in a WMS service.

1.3 Methods

This master thesis is divided into a theoretical and a practical part.

A literature study concerning the standards and the techniques used within this project was carried out. Used sources were e.g. specifications from OGC, technical articles, books and websites. The first part of this report contains some of the literature study.

The practical part was divided into three main areas: The installation and maintenance of a WMS server, the development of a new WMS client and some work with text and icon placing algorithms. For the WMS server, this meant installing a server software and adjusting cartographic data to it. The development of a new WMS client was partly carried out in a team, where the author of this thesis played a role as constructor of a java script file. The final result of this part was therefore submitted the time that the others could engage to this. Most of the practical work was carried out on a programme for text and icon placing algorithms. The structure and the usability of the programme were improved. For example, values of constants and variables were lifted out from the code in order to enable an easy testing environment of the algorithms. The program was also documented through new comments in the code and through this report.

1.4 Report form

This report consists of three major parts: theory, practical part and final analysis.

Part I – Theory - concerns the literature study carried out. It contains the following chapters.

1 Introduction: (This introduction.)

2 Internet fundamentals: Introduces some basic Internet concepts, as well as brief presentations of standards used within this study.

3 Maps and GIS: Introduces the reader to digital maps in general as well as different approaches how to use Internet as a platform for *geographical information systems.*

4 Standardization of Internet GIS: Describes the WMS and WFS interfaces, two standards for geographical information transfer on the Internet.

5 Text and icon placement: Summarizes the theory behind the implemented algorithms for text and icon labelling.

Part II – Practical Part – describes what this study has accomplished practically.

6 Introduction to practical part: Describes how the different parts of the study are related and how they should be seen from an overview perspective.

7 WMS clients: Presents two clients; one developed within this study and one developed by a master thesis at Sony-Ericsson.

8 WMS servers: Presents the servers used within this study.

9 Programs for Text and Icon Placement: Presents the developing environment for the used algorithms and what modifications have been accomplished by this study.

10 Model of integration: Describes how integration methods (like text and icon labelling) can be integrated with a WMS server.

Part III – Final Analysis – Summarizes the report and discusses the results of this study.

11 Discussion: presents some thoughts on WMS in general, WMS as a means for mobile applications and text and icon placement..

12 Conclusions: Concludes the study.

Each chapter begins with a short introduction where the content of the chapter is very briefly summarized.

2 Internet fundamentals

In this chapter, information about some of the most important Internet standards used in this project are presented. The goal of this chapter is to provide the reader with enough basic information for the understanding of the following chapters. It deals with what the Internet is, the HTTP protocol, the XML metalanguage and the client-server model.

2.1 The Internet

The history of the Internet goes back to the 1960s, when the first protocols for transmitting data through a telephone line were created. In 1969 four computers (at Stanford Research Institute, University of California in Los Angeles, University of California in Santa Barbara and University of Utah) were connected, and the first configuration of the Internet's predecessor APRANET saw the light of the day. The birth of the Internet, as we know it, is usually considered to have taken place on January 1, 1983. Then, the APRANET switched its protocol to the TCP/IP (Transmission Control Protocol/Internet Protocol), which is still used today. However, the exact definition of the term 'Internet' was formed as late as 1995. Then the Federal Network Community agreed that Internet is computers linked with IP addresses and that these computers support communication through the TCP/IP with derived protocols. (ISOC 2003)

The Internet is to be seen as a fundamental infrastructure, on which information is distributed in several ways. The visiting of web pages, downloading of files and emailing are examples of different information channels that use the Internet. Certain standards are therefore developed to facilitate the exchange of information. The most common standards are the Hyper Text Transfer Protocol (HTTP) to request and send a web page, the File Transfer Protocol (FTP) to download a file and the Simple Mail Transfer Protocol (SMTP) to send and receive e-mails.

2.1.1 URI / URL

URI stands for Uniform (or Universal) Resource Identifier. It is a global naming scheme that contains information about where all different Internet Resources are to be found. Normally, a URI consists of four parts: the name of the protocol, the server name or domain name, the port number and the location of target resources. The server name is either given as its IP-address, consisting of four numbers between 0 and 255, separated by decimal points, or as a domain name. Through the Domain Name Servers (or System) – DNS – logical and hierarchical domain names are used instead of IP-addresses. However, a computer connected to the Internet always has an IP-address, and the DNS is a system that redirects domain names to their corresponding IP-addresses. (Peng and Tsou 2003).

URL stands for Uniform (or Universal) Resource Locator and is often used instead of URI when HTTP is the protocol. Even though it is an informal term, it is commonly used. In this text, the notation URL is used.

```
http://www.someserver.com/resource/file.html
http://100.150.200.250/resource/file.html
```

Figure 2.1: Example of two URLs leading to the same resource.

2.1.2 Common Gateway Interface (CGI)

CGI is a standard that is used so that a client can transfer parameters to a (CGI) program on the server. Once the server executes this program, it will respond with the processed result according to the transferred parameters. Mostly, this is in the form of a real-time generated document. (NCSA)

The two methods HTTP-GET and HTTP-POST (see 2.2.2) use the CGI parameters transfer. A CGI-string begins with a question mark (?) at the end of the URL. After the question mark the parameter names follow with their values. If there are two or more parameters, they are separated by a &. The equal sign is the separator between the parameter and its value.

```
http://www.domain.xx/page.cgi?parameter1=value1&parameter2=value2
```

Figure 2.2: An example of a CGI query string

2.2 Hypertext Transfer Protocol (HTTP)

2.2.1 The HTTP and the World Wide Web

The HTTP is a standard developed by the World Wide Web Consortium (W3C) and its purpose is to offer a standard way of requesting data over the Web. The first version was used in 1990. With the version of 1.1 the standard is considered fully developed (W3C 1999). A browser requests a certain page using the HTTP notation in the URL. Thus, the providing server replies with a document (most often HTML).

Two nouns that are often used synonymously are the Internet and the World Wide Web (WWW, 'the Web'). However, this is not quite correct. As described above, the Internet is to be seen as a fundamental (base) infrastructure. The WWW runs on top of the Internet. To request a page on the WWW, a browser, for example, requests a URL using the HTTP as protocol. Thus, the WWW is to be seen as the part of the Internet that uses the HTTP whereas, e.g., FTP-services are a part of the Internet, but not a part of the Web.

2.2.2 Request methods

For the concern of this study, there are two methods to request the server. Those are the HTTP-GET and HTTP-POST methods.

The HTTP-GET method consists of an ordinary URL followed by a query string (see figure 2.2). A long string in the address field of a web browser (with several &-characters) is often an example of a get method. Chapter 4 about WMS shows specific examples of such strings.

The HTTP-POST method is not visible in the address field in a web browser. It is hidden in the head of the message. The post method is used when a lot of text has to be transmitted, e.g. on an Internet guest book. Thus, a post method is preferred when a request is either long or consists of special characters.

2.3 The Extensible Markup Language (XML)

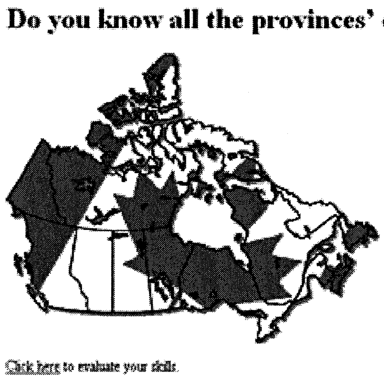
2.3.1 An introduction to the XML based on HTML

The hypertext markup language (HTML) is a standard developed by the W3C (W3C 2004b). Its purpose is to standardize the presentation of a document. An HTML document will have the same (or almost the same) appearance whatever software is used to present it.

HTML consists of different tags, used as markers. There are two types of tags that always appear in pairs: an opening tag followed by a closing tag. Whatever is written in between will be presented according to the specification of the tag. In figure 2.3 <H1>Canadian provinces</H1>, means that 'Canadian provinces' will be presented as a headline (the font size will be larger and bold). Some other basic tags are also shown in the figure. The browser displays the document according to the tags. This is shown in figure 2.4.

```
<HTML>
   <TITLE>Canadian provinces</TITLE>
   <BODY bgcolor=#FFFFFF>
     <H1>Do you know all the provinces' capitals?<H1>
     <IMG SRC="Canada_map.gif">
      <P><A HREF="test.html">Click here</A> to evaluate your skills.</P>
   </BODY>
</HTML>
```

Figure 2.3: An HTML document



Do you know all the provinces' capitals?

Click here to evaluate your skills.

Figure 2.4: How the browser displays the HTML document.

Even though HTML is a well-spread and stable standard, it has several disadvantages. One is that the tags are pre-defined and there are no means to create new tags with new meanings, because there is just one set of tags defined (Hunter et al. 2001). Another disadvantage is that the structure is not hierarchic. Thus, it is the author's choice how to place his or her tags, e.g. by first placing a small headline followed by a big headline. Note, that this might sometimes be the intention of the author, so why refer to this as a big disadvantage for the HTML? For several presentation purposes HTML is the proper approach. However, when the data has to follow a certain structure, the distributor of the data needs to define her or his own tags, HTML is not enough. For these situations the Extensible Markup Language (XML) has been developed.

2.3.2 XML Structure

XML is developed under the authority of the World Wide Web Consortium. Their recommendation on XML (W3C 2004a) is a stable document and should be considered as a standard. XML – like HTML – is a part of the ISO standardized Standard Generalized Markup Language (SGML). In this, XML follows the syntax and rules of the much wider SGML (Hunter et al. 2001).

The XML allows hierarchic storage of data. Whatever data are needed, the XML author is free to define special elements for this data in each single XML document. The relationship between the elements is specified in either a Document Type Definition document (DTD) or an XML schema document. The latter document was introduced by W3C in 2000 and it has several advantages. Unlike DTD, an XML schema is written using XML notation, and therefore it supports almost any XML tool. The schema is at the same time stronger, more stable and more extensible (since it is XML) (Hunter et al. 2001).

Basic example documents of XML, DTD and XML schemas are found in figures 2.5 – 2.7. The document itself (figure 2.5) consists of a parent element called *country*. This element also has a *name* attribute. The parent element in this example consists of two child elements, *province_or_state*, which in their turn consist of two additional child elements, these are *name* and *capital*. The document is valid to the DTD shown in figure 2.6. The DTD states that the *country* element consists of a *province_or_state* element and the attribute *name*. Figure 2.7 shows the corresponding XML schema. Note that it uses hierarchic tags (XML notations).

```
<?xml version="1.0?>

<country name="Canada">
   <province_or_state>
      <name>
         Nova Scotia
      </name>
      <capital>
         Halifax
      </capital>
   </province_or_state>
   <province_or_state>
      <name>
         Newfoundland
      </name>
      <capital>
         Saint John's
      </capital>
    </province_or_state>
</country>
```

Figure 2.5: A simple XML-document

```
<!DOCTYPE country [

   <!ELEMENT province_or_state (name, capital)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT capital (#PCDATA)>

   <!ATTLIST country name #PCDATA>
]>
```

Figure 2.6: Corresponding DTD to figure 2.5

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="country" type="province_or_state"/>

   <xs:complexType name="province_or_state">
      <xs:sequence>
          <xs:element name="name" type="xsd:string"/>
          <xs:element name="capital" type="xsd:string"/>
      </xs:sequence>
      <xs:attribute name="name" type="xsd:string"/>
   </xs:complexType>

</xs:schema>
```

Figure 2.7: Corresponding XML Schema to figure 2.5

2.3.3 Parsing XML

As showed in figure 2.5, an XML document is plain text. Therefore it can be read in an ordinary text editor. If the names of the data are meaningful, an XML document should be rather self-explaining. It is easy to get an overview of the content just by viewing it in a text editor. But this is of course only to learn what an XML document looks like. The purpose of this standard is to enable transfer of data over the web. Therefore, a XML document is parsed by a special program (an XML parser). The parser utilises the XML schema in order to obtain the special rules it has to follow.

The parser goes through the XML document and could, for example, store the data in another environment. With this approach, powerful web applications can be created. Ordinary HTML documents could be rendered from the data environment that the parser has created.

Since the parsing of an XML document is one main task in dealing with XML, the concept of well-formed XML is important. A well-formed XML document has to follow the following rules (Hunter et al. 2001):

- Every start-tag must have a matching end-tag
- Tags can not overlap
- XML documents must have one and only one root element

If an XML document violates any of these rules, a parser will not be able to proceed its parsing. However, the other way round should be emphasized: As long as an XML document follows these simple rules, any parser should be able to parse that document.

2.3.4 XML applications

Since the XML syntax allows everybody to name their own elements and create rules of the relations between the elements, different standards within the XML have been developed to mark up special types of data. These "XML dialects" are specified by DTD or XML Schema documents. Thus, it becomes easier to transfer data since the receptor knows in advance how to interpret it. Any one of these standards within the XML could be parsed by an ordinary XML parser. However, with standardized element names and rules, it is much easier to create applications that know how to treat the expected data.

Two of these "dialects" are of concern to this study. *The Geography Markup Language* (GML) describes both the geometry and properties of geographic features. It was developed by the OGC to enable transfer and storage of geographic information (OGC 2003). *Scalable Vector Graphics* (SVG) is a language developed to describe two-dimensional graphics and graphical applications. It was developed by the W3C. (W3C 2004d)

2.4 Client / Server Model

A fundamental concept within this study is the client / server model. Also it is a concept commonly used on the Internet. The name refers to a situation where a client demands something from a servant. One example is when a web browser (the client) through a URL requests a web page from a web server that responds by sending the page to the browser.

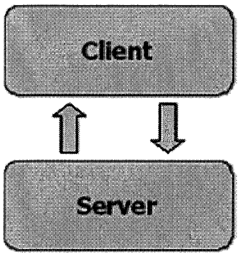


Figure 2.8: The Client / Server Model

A client / server model contains three elements or units: a presentation unit, a logical unit and a storage unit for data. To the presentation unit, some kind of user interface is often attached. This enables a (human) user to interact and to demand changes in the presentation of the material. The logical unit works as a link between the storage unit and the presentation. It receives responses from the presentation unit and it requests data from the storage unit. Here, data are processed so that a presentation can be accomplished. (Peng and Tsou 2003)

It is not always obvious which part of a large system acts like a server, and which does not. In fact, many computer programs (e.g. Microsoft Access) have the client / server model structure - with a presentation interface, a logical element and a data storage unit - and everything is physically located on the same computer. However, since most computers today are connected to a network, these three units are often located at different computers. One advantage with this is that the data only need to be stored at one place (that is on a server), whereas several users through client applications can access these data. The logical work is either carried out on the client (which is then called a *heavy client*) or on the server. In the latter case, the client is referred to as a *thin client.* Figure 2.9 shows the three elements without defining which constitutes the client or the server.

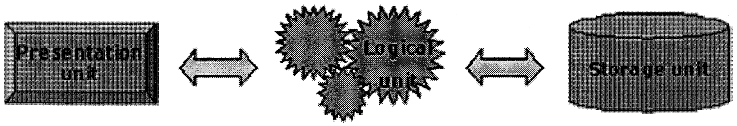


Figure 2.9: The three elements of the client / server model.

3 Maps and GIS

This chapter presents some terms that are used later on in this study. It begins with describing properties of a map and map formats. After that it presents GIS and the impacts on GIS from the Internet.

3.1 Map properties

This section presents map properties that have to be known in order to be able to fully interpret a map. These are *projection, coordinate systems, scale* and *geographical extent.* For a map producer it is important to choose the most suitable properties, which depend on the application of the map.

A map is supposed to represent parts of the real world. Since the earth is a sphere (or, to be more precise, an ellipsoid) the representation on a plane paper or a screen cannot be exact. There are different kinds of projections with different properties. Depending on projection, a map can represent angles correctly, represent areas with correct size or represent the distances of a number of lines correctly.

The next important property is the coordinate system. There are several coordinate systems, both local and global systems. Often, there is a separate horizontal system and a separate vertical (height) system. Depending on which system is used, the coordinates can be presented in degrees (with minutes and seconds) or in Cartesian coordinates; x, y and z.

The scale indicates the relationship in size towards the real world. The geographical extent could be given as the coordinates of the left-lowest and right-highest corners in a map. Thus, this property indicates the area that is covered in a map.

3.2 Map objects

Special objects in the real world are often symbolized in a similar way in different maps. For example, a lake is often symbolized as a blue-filled area, a road with a coloured line and towns on a small-scale map as point objects. Text and icons are often used to add important information to the map. This study will use the following terms when describing digital maps: *Features* in a map constitute single objects such as a building or a road. Buildings and roads are two *feature types*. Digital maps are often rendered from different *layers*, where each layer contains features from one or more feature types. The layers show the separation from original paper maps. The digital map user can often select only the layers he or she is interested in.

3.3 Map formats

For digital maps there are two categories of map formats. These are raster and vector formats. A raster format is an image whereas a vector format constitutes coordinates of lines and points.

In raster format, a value for each pixel is stored. If a basic map only consists of four feature types, e.g. gardens, open fields, buildings and roads, each pixel has the value of one of these types (see figure 3.1). The same map in vector format could e.g. set the background to constitute gardens. Then there is a set of tables. One table constitutes the node coordinates of a line. A column indicates if the line is a limit of an open field or a building or if it is a road. In the former case, the whole open field or urban area polygon is found in another table, which contains references to the constituting lines. A vector format does not become an image until a program renders the lines in the tables and fills the polygons with specified colours (see figure 3.2).

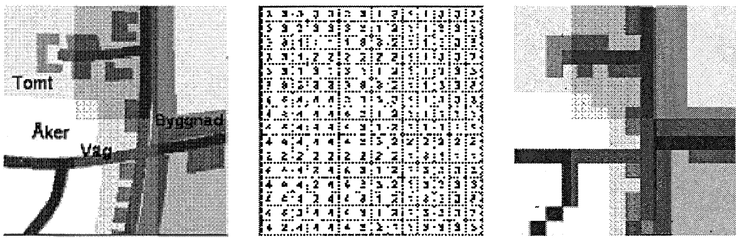


Figure 3.1: A map in raster format. Each pixel stores the value of the covering feature type or feature.

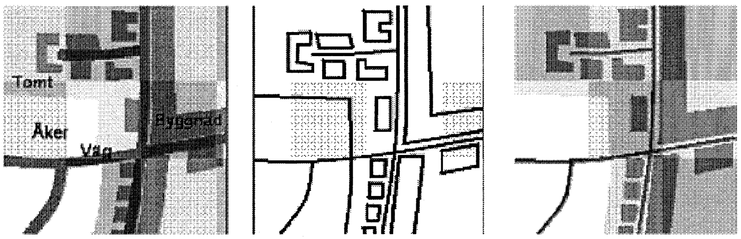


Figure 3.2: A map in vector format. Only the coordinates of the constituting nodes in a line are stored. Links to other tables determine if the line constitutes a side in a polygon of a special feature.

3.3.1 Features of raster formats

One advantage with raster formats is that they are easy to handle. Many programs support several raster formats. A map in the commonly used GIF format is easily displayed in a web browser or inserted in a text editor program such as Microsoft Word. Thus, the raster format (and especially a well-known format) reaches a wider audience more easily.

An advantage compared to the vector format in GIS applications is that raster formats is ideal for logical overlay analysis. It is easy to implement operations to overlay features from two images with conditions through the use of arithmetical operation (as long as the two images have the same map properties).

An often mentioned disadvantage with raster formats is probably the resolution. When a zooming-in operation is performed several times, the pixels become large and the image vague. Another disadvantage is the file size. Generally a raster map demands a larger file size since each pixel is stored. The larger garden area in the example above describes this when it is compared to the vector format. It should be mentioned that several raster formats have (built in) compressing algorithms that decrease the file size.

3.3.2 Features of vector formats

Unlike raster formats, vector formats depend on programs that render a map according to the specifications in the tables. This is a disadvantage because the format cannot be easily distributed since fewer programs can handle the format. Also when zooming and panning operations are performed on a vector formatted map, a new map must be rendered (or fetched from a cache memory). Common vector formats are SVG (briefly mentioned in section 2.3.4.) and *shape* file. SVG is an XML application describing geographical data. The shape-file is its own format, developed by ESRI. It is used or at least understood by almost any GIS programme.

An advantage with the vector format is that it does not depend on the resolution. A thin line between two nodes remains thin independent of the zooming factor.

The features of vector formats could also lead to a smaller file size (as in the example given above). This is, however, not the case if the map contains a large number of features and if these features are not generalised. An example: A building rendered as a rectangle demands less size than if all its 16 corners constitute nodes in the polygon that symbolizes the building.

3.4 GIS fundamentals

A common definition of a *geographical information system* (GIS) is a computerized information system, for management and analysis of geographic data (Eklundh et al. 1999). There are three main tasks for a GIS:

Input and output of data. The system has to understand several data types (e.g. different file types). Thus it can utilize data from a variety of data providers.

Data management. The data storage follows a certain structure that enables easy access and selection of data, as well as updating and deleting operations on the data.

Presentation. The final result of an analysis could be presented in different ways, e.g. by map images, diagrams or tables.

GIS serves several fields of application. Therefore the use of GIS is spread among several professions and fields of science.

3.5 Development towards Internet GIS

Desktop GIS is a program that resides on a personal computer. Since the use of personal computers increased, Desktop GIS became the traditional GIS utility. Today, Desktop GIS still serves many applications. However, with the impact from Internet technologies, the physical location of the system is not bound to one location.

According to Peng and Tsou (2001), the Internet affects traditional GIS in three major areas.

GIS data access: It is easy to acquire data from different data providers and it is easy to keep the data up-to-date.

Spatial information dissemination: Spatial information can be made accessible to a much wider audience. The general public can access and analyze geographical information from their browsers.

GIS modelling/processing: Also in this field, the Internet allows new means. Web services can be created, where modelling and processing are executed on the server-side, through the interaction of a user. Furthermore, the Internet enables faster access to new GIS processing components for advanced GIS users.

3.6 Examples of Browser GIS

A map image in an HTML document does not constitute a GIS. As described in chapter two, HTML only serves the means to present texts and images. It is not possible to do enquiries and analyses in such a document unless other techniques are linked to it.

A browser GIS is created when query- and selection operations on a map are enabled. This constitutes management and analysis of data. Below follows three approaches to form GIS functionalities in a browser (Peng and Tsou 2001):

1) Through the use of easy (java-) scripts in the document shown in the browser, some level of interaction between the user and the system can be achieved. However, the scripts running in the browser are restricted to carry out easier operations. The major advantage is that scripts are simple and can be used by any user as long as he or she has an Internet connection and a standard browser.
2) With plug-in programs to a browser, the level of interaction is dramatically increased. The disadvantage is that it requires the user to download and install an extra program, which both takes time and demands that the user has control over the computer, e.g. it is his or her personal computer and not a public one.
3) Much of the functionality is handled by the server-side. This approach enables a big level of interaction, because more advanced operations are implemented on the server-side. Though, a disadvantage is that often the client (the browser) and the server must have a stable connection, since many calls can be made between them. The client transfers parameters using CGI (see section 2.1.2). An example of this is a web page that contains a WMS client, which connects to a WMS server (see chapter 4).

4 Standardization of Internet GIS

This chapter introduces two standards developed by the OGC. These standards provide interfaces handling geographic data on the Internet. They are developed to serve as an open alternative in contrast to GIS vendors' own interfaces.

4.1 Open Geospatial Consortium (OGC)

OGC is an organisation consisting of over 200 active members from universities, research institutions and commercial organisations that works on developing standards concerning geographic information (OGC 2005). OGC works closely together with the International Standarization Organisation (ISO). Some of the standards developed by OGC have been included in the ISO specifications. One example of a standard that has been adopted by the two organizations is GML, *Geography Markup language*, which is briefly mentioned in section 2.3.4. Standards developed by OGC are available free of charge.

The organization has developed both WMS and WFS. WMS handles maps on the Internet and WFS handles geographical data. Using terms presented in chapter 3, WMS is mostly raster formatted whereas WFS is vector formatted.

4.2 Web Map Service (WMS)

4.2.1 Introduction

The Web Map Service is a standard developed by the OGC. The current version is 1.3. This text is based on the document OGC (2004). The standard enables transfer of maps on the Internet. It is based on a tight connection between a server and a client; both implementing and following the WMS interface. Since it is recognized as a worldwide standard, WMS plays an important role in the development of different map services on the Internet. It is also an important free alternative to vendor specific formats. There are today several WMS applications developed. Many of them have been developed as open source projects.

A WMS session consists of several requests from a client and, to each request, a response action from the server. There are three different requests, *GetCapabilities, GetMap* and *GetFeatureInfo*. The first two are mandatory for all WMS servers to support (so called *Basic WMS*), whereas the latter one is optional. A WMS that also supports the *GetFeatureInfo*-request is called a *Queryable WMS.*

Figure 4.1 shows a WMS session. Such a session begins with a *GetCapabilities* request, followed by one or more *GetMap* requests. The third possible request, *GetFeatureInfo*, is also shown in the figure. See sections 4.2.2 – 4.2.4.

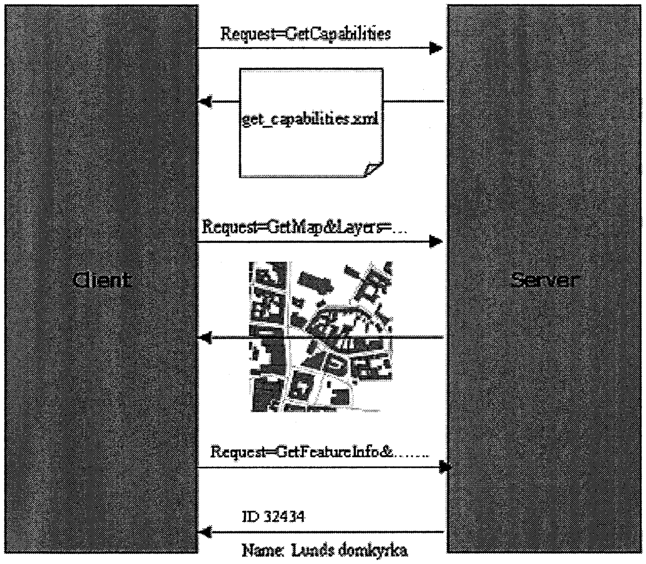


Figure 4.1: Operations during a WMS session

A request to a server is easily accomplished by a normal HTTP-GET request, followed by a query string, containing the following parameters: *service* - defines that it is a WMS service which is requested (the server could support other services as well, e.g. WFS); *version* - states the used WMS version; *request* - states the type of the request.

4.2.2 GetCapabilities

The purpose of a *GetCapabilities* request is to provide the client with information about what data the server can provide and what operations it supports.

```
http://www.some.server.com/wms?
      SERVICE=WMS&
      VERSION=1.1.1&
      REQUEST=GetCapabilities
```

Figure 4.2 A GetCapabilities request. Normally, the whole request is on one single line. It is here separated into three lines for the readability. Furthermore, the parameter names are written in upper case for clarity reasons. WMS makes no difference between lower- and upper-case in parameter names, however it makes a difference in parameter values, e.g. 'WMS' does not equal 'wms'.

There are also other, optional, parameters that could be used in this request. For example, with the *updatesequence* parameter, the client can compare the datum status of the requested document with the datum for the same latest requested document. In this way, the client can avoid parsing the same document twice.

The response from the server is an XML document. This document contains metadata formatted as the specific XML schema specified by the OGC. Below is a short summary of how the document is constructed. The document starts with some elements that present general information about the service and the data provided. After that the actual operations that are supported by the server, the output formats and the URL prefix for each operation are introduced.

The rest of the document deals with describing the properties of and attributes to layers. All the geographic information contents of the service are ordered into layers. They could either be listed one by one, or several child layers could be nestled inside a parent layer (*layer-nestling*). This latter case could often be recommended when the server contains data over different areas, where layers covering the same area are all children to the same parent layer. The parent layer contains properties that are inherited by all children. Examples of layer properties are the reference system (SRS) and geographical extent (BoundingBox).

A style is the property of a layer that tells the WMS server how to render a layer on a map. A layer can provide several possible styles. Furthermore, a layer can have zero or more of six attributes. The default value of all of these attributes is “0”. For example, if the attribute *querable* is “1” (true) the server supports the *GetFeatureInfo* operation.

4.2.3 GetMap

The *GetMap* operation returns a map. Nine parameters are mandatory in a *GetMap* request to a WMS server. In figure 4.1 the session begins with the *GetCapabilities* request. Upon receiving the XML-document, the client parses it in order to enable a user to choose the values of these parameters. If the client requests a parameter with an invalid value, rules decide what action the server may take. Most often, it will respond with a service exception.

Through the *layers* parameter, the client requests what layers the map will present. The order of the layers in the parameter list is then the order in which the actual map layers are rendered. Thus, the leftmost layer in the parameter will be the bottom layer in the map. Furthermore, each layer could be presented in one or more styles. The *styles* parameter serves for this purpose, and is organized in the same manner as the layers parameter.

The *SRS* (Spatial Reference System) parameter states the wanted reference system. The value is a code provided by the European Petrol Serving Group (EPSG 2005). For example, EPSG:32633 corresponds to coordinates given in UTM Zone 33 (north) projected on the WGS-84 ellipsoid. The *Bbox* (bounding box) parameter enables the client to request the area it is interested in. The values of this latter parameter are found in a list, where the minimum X and Y values are followed by the maximum X and Y values. *BBox* values must of course be included in the range of the SRS of the request.

The *format* parameter in the request decides in what format the map will be rendered. Several WMS servers provide a wide range of possible formats; GIF, PNG, JPEG, BMP, SVG and so on. The two parameters *width* and *height* state the pixel size of the requested map. If the size does not correspond to the *BBox* values, e.g. if the *Bbox* is a square and *width* and *height* together make a rectangle, the map will be stretched. This is rarely an advantage and therefore it is recommended (by the OGC) that the client should be implemented in a way that minimizes the risk of requesting distorted maps.

```
http://www.some.server.com/wms?
      SERVICE=WMS&
      VERSION=1.1.1&
      REQUEST=GetMap&
      LAYERS=buildings,roads,railways,lakes,woods&
      STYLES=normal,special,normal,normal,greenwood&
      SRS=EPSG:32633&
      BBOX=385857,6174780,386236,6175159&
      WIDTH=300&
      HEIGHT=300&
      FORMAT=image/svg+xml
```

Figure 4.3: Mandatory parameters in a GetMap request.

There are also some optional parameters that could be useful in a *GetMap* request. If the chosen format supports transparency, e.g. the gif-format, the *transparent* parameter could be used to define whether or not the background is to be transparent or not. The *bgcolor* parameter states the colour of the background, and is generally described in the hexadecimal encoding of an RGB value, e.g. 0xFFFFFF for white, which also is the default value. The default value of the *exceptions* parameter is “XML”, even if this parameter is absent from the request. When this parameter is used in a request, it defines another value, e.g. ‘text’. Finally, there are two other optional parameters. The *time* parameter is to be used if data for a specific time can be requested (e.g. a weather map) and the *elevation* parameter if data for a specific elevation can be requested (e.g. ozone concentration at different heights in the atmosphere).

Request parameter	**Mandatory/ Optional**	**Description**
VERSION=1.3.0	M	Request version
REQUEST=GetMap	M	Request name
LAYER=layer_list	M	Comma-separated list of one or more map layers
STYLE=style_list	M	Comma-separated list of one rendering style per requested layer.
SRS=namespace:identifier	M	Spatial reference system (often in EPSG)
BBOX=minx,miny,maxx,maxy	M	Bounding box corners
WIDTH=output_width	M	Width in pixels of map picture
HEIGHT=output_height	M	Height in pixels of map picture
FORMAT=output_format	M	Output format of map
TRANSPARENT=true|false	O	Background transparency of map
BGCOLOR=color_value	O	Hexadecimal red-green-blue colour value for the background colour.
EXCEPTIONS=exceptions_format	O	The format in which exceptions are to be reported by the WMS
TIME=time	O	Time value of desired layer
ELEVATION=elevation	O	Elevation of desired layer.
Other vendor-specific parameters	O	A vendor can implement own parameters. These must however always be optional, since a client does not have to understand them, yet still be able to request a map from the server.

Figure 4.4: Possible request parameters to a GetMap request. (From OGC 2004)

The response to a *GetMap* request is of course a map in the specified format. This map contains the requested spatial data and has the requested properties. If the request is not correct or if the server can not process the request correctly, an error output is sent to the client.

4.2.4 GetFeatureInfo

The *GetFeatureInfo* request is optional. When being used, it is only supported for layers with the attribute *queryable*=1 (true). If the server does not support this service, it will be mentioned in the *GetCapabilities* document and respond with a service exception upon such a request.

The purpose of this service is to provide the client with additional information about features in the map returned by the previous *GetMap* request. To receive this information, the *GetFeatureInfo* request includes most of the parameters from the *GetMap* request along with the layer(s) it wishes to query, the desired output format and the position of the user's choice (I, J). For example, the user might want to have extra information about a specific building and clicks on it. Then, with a database query based on the user's coordinates, the server can provide the client with the additional information about that building (if there is any).

```
http://www.some.server.com/wms?
      VERSION=1.1.1&
      REQUEST=GetFeatureInfo&
      LAYERS=buildings,roads,railways,lakes,woods&
      STYLES=normal,special,normal,normal,greenwood&
      SRS=EPSG:23033&
      BBOX=385857,6174780,386236,6175159&
      WIDTH=300&
      HEIGHT=300&
      QUERY_LAYERS=buildings&
      INFO_FORMAT=text/plain&
      I=385600&
      J=6175000
```

Figure 4.5: An example of a GetFeatureInfo request.

4.3 Web Feature Service (WFS)

4.3.1 Structure

The Web Feature Service is to be seen as a compliment to WMS. WFS communicates more at the level of layers (*feature types*) and specific geographic objects (*features*), whereas WMS mostly operates on an overview level (a whole map). A server response in WFS is an XML document. Most of them must use GML notations. Section 4.3.2 is based on the document OGC (2002).

4.3.2 Operations

There are five defined operations for WFS. These are *GetCapabilities, DescribeFeatureType, GetFeature, Transaction* and *LockFeature.* A *Basic WFS* only needs to support the first three mentioned operations.

WFS supports both the HTTP *post* and *get* methods. The requests (apart from the *GetCapabilities*) use XML notation. That is why it is strongly recommended to use the post method. A WFS request in a URL (get method) could be far too long. An even greater problem is that special characters like '<' and backspace often cause an invalid URL.

GetCapabilities
This operation is the first request a WFS client makes to a server. The server responds with an XML document, in which the server describes the feature types it can service. It is also mentioned which operations are supported for each feature type. It is constructed in the same way as the *GetCapabilities* request for WMS.

DescribeFeatureType
This operation is used when the client needs information about the properties of one or more specific feature types and which queries it can process on the feature types. This information is returned as an XML schema. The schema defines how the service expects feature instances to be encoded, both for output and input.

GetFeature
After paring a schema describing some feature types, the client can specify one or more features and also do queries on these features through the *GetFeature* request. It contains a *query* element, which defines what features to query and - if a *filter* element exists within the *query* element - how to constrain the query. The *filter* element uses another standard from the OGC, *Filter Encoding*, which is defined XML syntax for spatial, logical and other operators on features.

Transaction
This operation enables the client to modify features; that is to create, update and delete geographic features. A response message, in form the of an XML document, is to follow, in which the server indicates the termination status of the operation.

LockFeature
LockFeature is used to prevent other users from accessing data during a transaction operation. It is not a mandatory operation for a Transaction WFS, but due to practical reason, it is often implemented.

There is a difference between a *Basic WFS* and a *Transaction WFS*. The former only allow read-only operations (*GetCapabilities*, *DescribeFeatureType* and *GetFeature*), whereas the latter also allows writing operations (*Transaction* and *LockFeature*).

4.4 Combining WMS and WFS (Distributed GIS)

Section 4.2 gave a somewhat detailed, yet still basic overview of WMS, whereas section 4.3 summarized the WFS more briefly. The main reason is that WMS will be used in the practical part of this study.

Both WMS and WFS are machine-to-machine interfaces. The WMS serves maps and the WFS serves features in a map. This distinct separation clearly states how the two standards should be used together. Seen from the outside (the user-side), a map is the desired result most of the time. Thus, the user uses a WMS client to request a map from a service. This map generating service could then use a combination of WFS and WMS to get a desired map. This is the concept of distributed GIS.

Many of the WMS server applications available today support requests to other WMS and WFS servers. This offers a wide variety of possibilities to render maps.

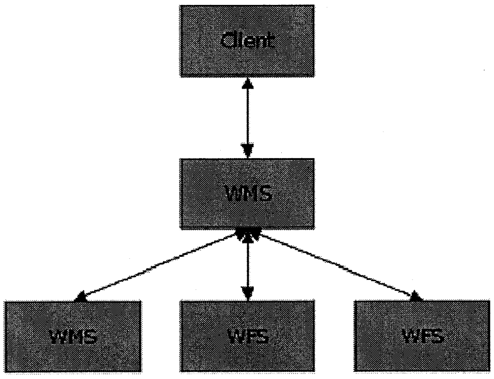


Figure 4.6: A normal configuration for a map rendering service. A client requests a map from a WMS server, which acts as a client to fetch additional data from another WMS server or WFS servers.

Segerstedt (2005) shows four prototypes of distributed GIS. The study uses a simple WMS client to request different configurations of WMS and WFS services. In the first prototype the client connects directly to different WMS servers and tries to overlay maps from each one of them. In the second prototype, the client connects to a WMS server that connects to other WMS servers (remote WMS) to fetch the requested data. The same approach is used in the third prototype, but here the WMS server connects to WFS servers and therefore it requests data using GML notations. The last prototype in the study shows that it is possible to control the appearance of maps using the third prototype's configuration combined with SLD (styled layer description) documents.

The results of the tests differ. Not all WMS support remote requests and it is difficulties to get a map with the same properties from different services. The result is often that the maps cannot be combined. But as long as the client connects to "correct" services this kind of distributed GIS works satisfactory.

5 Text and Icon Placement

A good map contains text and symbols (icons) that are placed carefully next to or around the objects they represent. However, if the text and icons are not placed in this way, it could be hard to identify to which object the icon or text belongs. It could also be hard to interpret the map if text and icons are the only things visible. This chapter starts with a brief introduction to cartographic theory concerning text and icon placement matters. Thereafter, methods for icon and text placement on real-time maps are summarized.

5.1 Cartographic theory

The text-icon placement problem is one of many constraints in cartographic science. The preferred positions of different objects' icons and descriptive texts are often in conflict. However, a map without informative text or icons is often hard to understand. Thus, the descriptive text is of great importance on a map. But the text must be clearly presented and readable. At the same time, it must not overlap or hide any important cartographic information.

The approach to solve the text and icon labelling problem in the paper map design was to let rules guide the decision. However, the cartographer's subjective ideas and his or her experience had the final decision. In this way, an experienced cartographer could create a really good-looking map, even though the presumptions of the presentation of different cartographic data were tough. With computer-generated maps, this final human evaluation is of course not possible. But to improve the quality of digital maps, they can be tested by an automated evaluation, where the quality of the map is expressed by a number, calculated after a more or less complicated formula.

As mentioned, the placement of the text (and icon) in relation to the object it represents is important. There are in this case three sorts of objects; point, line or area objects. For a point object, e.g. a city, the preferred position is intuitively just to the right above the point, but it could be placed anywhere around the point (following a decreasing scale of preferences). For a line object, e.g. a river, the position of the text should be placed alongside, and preferably above, the line. Finally, for an area, e.g. a county, the descriptive text should cover the whole area. This is often done with wide spaces between the letters. (Robinson et al. 1995)

5.2 A real-time approach

This chapter aims at summarizing the theory for icon and text placement presented in Harrie et al. (2004), Zhang and Harrie (2005a) and Zhang and Harrie (2005b). The algorithms from these articles have been implemented as part of the GiMoDig project (Gimodig 2005) at the GIS-centre, and this study has modified some of its code (see chapter 8).

The concept of real-time mapping means that a map is created until it is requested. In this case there are no ready-made maps. This also means that a map service (e.g. a routing service with GPS) never knows in advance which maps it will request. Through the GPS-coordinates it can then request a map covering the area around the user's position. The requested maps are generated at a server and directly sent back to the user. The concept of real-time indicates that something should be carried out at once. Thus the user should not have to wait more than a few seconds until the map is presented. Here, big requirements are put on both the map rendering process and the data transfer process.

The methods presented below are developed in order to be fast and to render good-looking maps. However, they have some prerequisites that state that if the rendering process exceeds an acceptable time limit, a map with lower quality will be accepted.

5.2.1 Icon placement

The placement of an icon is a classical cartographic problem. The approach used here is to find the least disturbing position of an icon. To computerize this problem, the following variables must first be defined:

- *original position* of the icon. This could mean the centre coordinate of its corresponding feature, but it might as well be defined in another way.

- *size* of the icon (squares and rectangles are allowed).
- *search distance*. This value states the permitted movement of the icon from its original position.
- *resolution of search space*. This could be the distance between two neighbouring search positions.
- *cartographic objects*. The objects that will be labelled and other objects that will be presented on a map.
- *importance/weight*. Each cartographic type of objects has an importance or weight value. This value states the disturbance of when an icon overlaps an object of this type.

The algorithm works as follows: A grid is placed so that its centre covers the original position. The size of the grid corresponds to twice the search distance. The pixel size of the grid is calculated as an integer fraction of the icon size. Then, the disturbance value for each pixel is calculated. This value is the sum of the weight of the cartographic objects that reside in the pixel. As a last step, the total sum for each possible position is calculated by summing up the weights of each pixel covered. The positions with the lowest values are the least disturbing positions and are called candidate positions. The final position for the icon is chosen among these candidate positions.

5.2.2 Text placement

For text placement (also called *text labelling*) a different approach is used. Also, as mentioned above in section 5.1, different cartographic objects represented by a point, a line, an area, etc., need slightly different methods to decide where the label should be placed. The implemented algorithms concern point and line labelling.

The central concept here is the *range box*. The range box constitutes an area in which the text itself could be placed. If within this box cartographic objects are intersecting, the range box will be reduced so that no conflicting object is overlapped. This part is called *conflict detection*. If this leads to a box of a size that cannot include the text label, another box must be created and the same tests will be carried out on this other box.

For point labelling, four range boxes are created around the point (top, bottom, left and right). Here, conflict detection is carried out on all four range boxes and they are reduced if conflicts occur. Then, following the cartographic rules of where the text should be placed, the best possible position is chosen. First, the top box is examined if its size is big enough for the label to reside within it. If not, the right box is examined, followed by the bottom one and at last the left one.

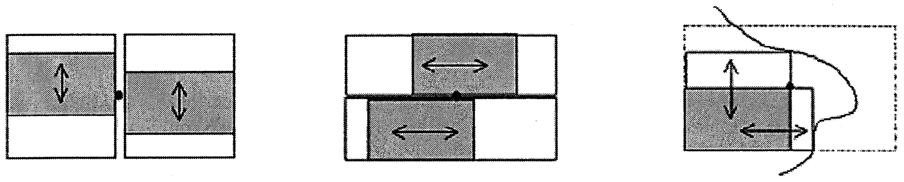


Figure 5.1: The four range boxes around a point. The grey areas correspond to the text label. In the last figure, the only possible positions are shown. (From Zhang and Harrie 2005a.)

As for line labelling, the approach is more flexible. Infinite candidate positions for a label could exist. However, the best position is chosen in a quick and effective way. First, the line is generalised through the Douglas-Peucker algorithm. From this generalised line, all the segments long enough for label placement are stored separately. A range box is created at the longest segment. Conflict detection is carried out in the same way as for point labelling. If the range box is still big enough for the label, the label's position is decided. If the range box is too small, the same procedure is carried out on the second longest segment and so on.

The final result of the text labelling is a set of candidate positions for each text label.

5.2.3 Text and icon placement simultaneously

To take both text and icon placement into account at the same time, an objective function is introduced. It could be written as:

$$f\,(\textit{label positions}) = w_1 * \sum \textit{placement properties} + w_2 * \sum \textit{cartographic disturbance} + w_3 * \sum \textit{label overlap}$$

where w_i constitute the weight of each sum. The significations of the terms are recognized from the previous sections.

It is important to first have a search space for the icons and text labels. The search space constitutes a reduced number of candidate positions, each enabling a possible position for the icon/text. However, many of the candidate positions could overlap. Therefore, the selection of the search space is of importance. Zhang and Harrie (2005b) and Harrie et al. (2005) discuss two ways of selecting the search space; either a stratified or a random selection is chosen. A stratified selection aims at obtaining evenly distributed positions.

After search spaces for icons and text are computed, an optimization process is performed, based on the objective function. The optimization is performed through the use of simulated annealing. This is an iterative improvement algorithm, which could be illustrated as a search process on a surface of a landscape, and the lowest spot would correspond to the best solution. In this specific problem, a "landscape value" corresponds to the sums of the terms of the objective function above. The simulated annealing procedure moves around in the landscape towards a position with a lower value. There is of course a risk that the found solution is a local minimum and a better solution could be found elsewhere. This is partly avoided by the means of incorporating a probabilistic element in the procedure. The more of the landscape computed, the less the probability that a better solution is to be found elsewhere. Because there is a risk that some icons and/or texts overlap each other, as a last step, these are simply removed.

To summarize:

1) (A reduced number of) candidate positions for icon placement are computed (as described in 5.2.1).
2) (A reduced number of) candidate positions for text placement are computed (as described in 5.2.2).
3) A combinatorial optimization is performed on the candidate positions. Simulated annealing is used to find the minimum of an objective function, where label overlap, cartographic disturbance and placement properties act as parameters. This step results in a proposed position for each icon and text label and this solution is either accepted or the procedure continues.
4) If the combinatorial optimization could not prevent overlaps from occur, the overlapping texts or icons are simply removed in this last step.

Examples of text and icon placement are given in section 9.3.

Part II

-

Practical part

6 Introduction to the practical part of the study

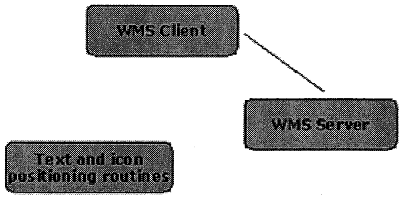


Figure 6.1 The practical part of the study

The practical part of this study has concerned three different parts.

WMS client (chapter 7)
A smaller project within this study was to combine two existing WMS clients. The two clients were developed by Lennartsson (2004) and Segerstedt (2005). The project was carried out on the spare time of the two mentioned authors, whereas the author of this study used time within this study. Due to lack of time, a final result of this project can not be presented within this study; therefore chapter 7 only describes the progress so far.

Chapter 7 also features a brief description of a mobile WMS client (Bozkurt 2005). This description is placed in the practical part because this study provided data to this client (chapter 8), and discussions were continuously held about possible improvements of the WMS service a well as general discussions about WMS as a means for mobile applications.

WMS server (chapter 8)
In the course of this study, two WMS servers were set up. The purpose of these has been to provide data to clients and in the future they can hopefully serve as a base in an integrated WMS service (see below). Sony-Ericsson uses one of these servers for their WMS client.

Program for text and icon positioning (chapter 9)
The main part of this study consists of modifying the code that implements the text and icon positioning algorithms described in chapter 5. Some improvements of the structure, usability and documentation of the code were made.

Integration of text and icon positioning routines in WMS services (chapter 10)
The three parts presented in chapter 7, 8 and 9 should be seen from a greater perspective. What this study has not accomplished is to integrate the text and icon positioning routines with a WMS service. This service should be suitable for clients using small displays. Chapter 10 describes a possible model of how this service could be structured (also see figure 6.2).

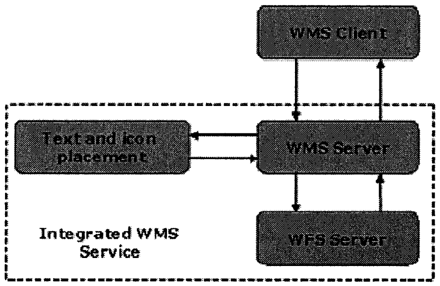


Figure 6.2. Model of possible structure of the different parts carried out in this study.

7 WMS Clients

This chapter presents a project within this study where two existing WMS clients were combined in order to create an improved client. It also describes a WMS client for a mobile phone, developed by Sony-Ericsson. The mobile client used data from a WMS-server set up within this study (see chapter 8).

7.1 An advanced web client

7.1.1 Background

The purpose of this smaller project was to combine the advantages of the two different clients developed by Lennartsson (2004) and Segerstedt (2005).

Lennartsson's client consists of a graphical user interface (GUI), where the user can choose and perform operations such as zooming, panning and measuring distances. There is also a tool to enable queries with the *GetFeatureInfo* method. The client does not support the *GetCapabilities* method (compare to figure 4.1). Thus, the user has to add the first *GetMap* request manually. This is done in the controlling script file. Additional *GetMap* requests depend on the new bounding box, which is created based on the properties of a zooming or a panning function. The main disadvantage of the client is that a user is restricted to one server and the properties of the first, raw coded *GetMap* request.

On the contrary, Segerstedt's client enables the user to choose between different WMS servers, stored in an XML file. From a drop-down menu, one of the available servers is chosen and a *GetCapabilities* request is immediately performed. The client extracts the available layers and enables the user to compose a map at the layer level. A disadvantage is that the user has to know the bounding box, since these values are not extracted from the *GetCapabilities* document. This client's *GetMap* method uses the parameters chosen by the user. While the map is displayed it is possible to hide layers. Panning or zooming functions as well as the *GetFeatureInfo* method are not supported.

An improved client based on these models should therefore consist of a combination of Lennartsson's GUI and Segerstedt's *GetCapabilities* handling part.

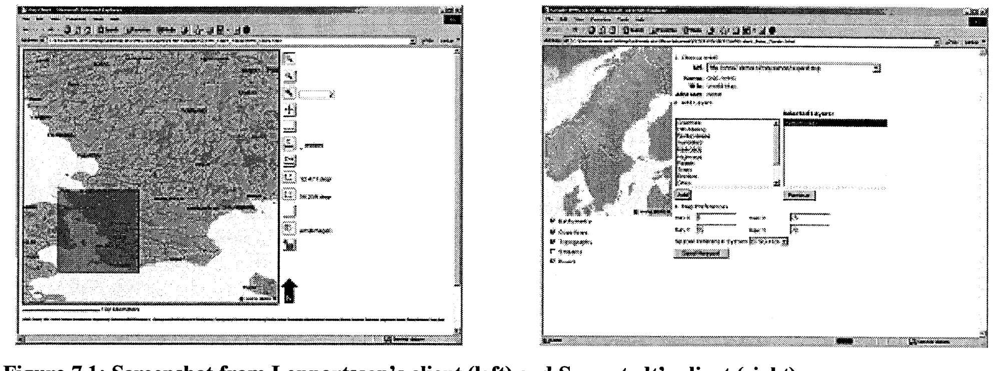


Figure 7.1: Screenshot from Lennartsson's client (left) and Segerstedt's client (right).

Therefore, a (spare time) project was therefore initialized where the two developers and the author of this study took part. The chosen solution to improve the client was to create an XML file serving as a parameter object between the two parts and a script file that provided both update and get methods on this file.

Technically, these two clients are quite different. The Segerstedt client renders a layer in the map for each chosen layer, whereas the image element in Lennartsson's client does not consist of separable layers, but an image object.

7.1.2 Structure of new client

The structure of the client is shown in figure 7.2. The figure shows three horizontal levels. It also shows that the client is vertically separated into two parts, where the left one origins from Lennartsson's client and the right one from Segerstedt's.

The presentation level consists of three HTML files. The index file consists of two frames, where the left frame contains a map image, a tool panel and selectors of which layers are enabled (compare to figure 7.1, left). The right frame contains a form, similar in appearance to the Segerstedt client.

The logical level is situated in the middle of the figure. The two top script files control the corresponding HTML file. They communicate to each other through the third script file, which gets values from and sets values in a context document, an XML file found on the bottom level.

The context document contains the current configuration shown in the map, e.g. the actual values for the bounding box and chosen layers. Apart from the context document, the storage level also features an XML document where URLs to the *GetCapabilities* request at different servers are found.

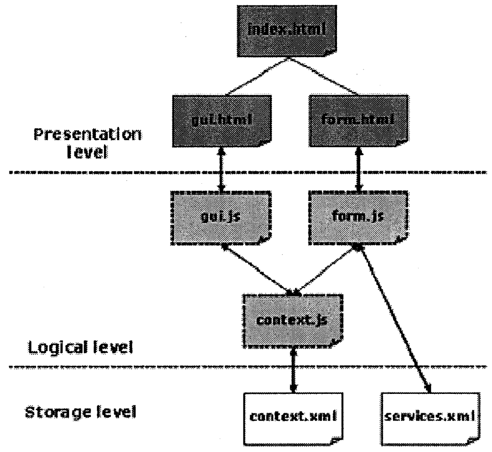


Figure 7.2: Proposed client structure

To summarize the proposed structure, this is how it should work: First, the user selects a WMS service and information from the *GetCapabilities* request is shown in the client. The user now chooses layers and a special bounding box as a first step in a *GetMap* request. These properties are then set in *context.xml.* The GUI part starts to request these layers and their bounding box from *context.xml.* The map is rendered and the *GetMap* request is terminated. The user is now free to request another map, either by performing zooming or panning actions on the map, or using a completely different request (perhaps from another service). In both cases, the top script files have to request *context.js* to update the properties in the *context.xml* file. This is done before a new map is rendered. In this way *context.xml* always keeps the current configuration.

7.1.3 Status of project

The project is not terminated. Initial work was carried out during some days in October 2004. The *context* document and script file were created (the author of this study implemented the *context* script file). Some additional work has been carried out within the *gui* script file, but after that not much has been done. The reason for this is scarcity of time for the team members.

7.2 Sony-Ericsson WMS client

This part aims to describe an implementation of a WMS client in a mobile phone. It was developed within a master thesis study (Bozkurt 2005) at Sony-Ericsson/Lund Institute of Technology.

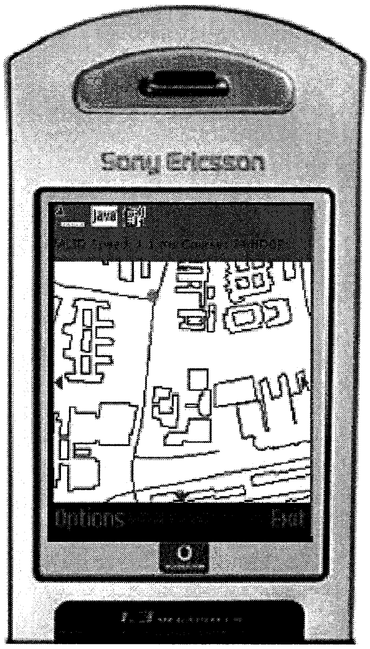


Figure 7.3 Sony-Ericsson WMS client in action. (From Bozkurt 2005)

7.2.1 Background

The purpose of Bozkurt's thesis was to evaluate WMS as a means for map services on a mobile phone. The theoretical part discusses the advantages and disadvantages of the standard. The practical part is a developing process, where the final goal is a WMS client with integrated GPS positioning.

Bozkurt's study was carried out at the same time as this study. Continuously, specific wishes about data provided by the server as well as general point of views about WMS were discussed.

It should be mentioned that this is probably the first implementation ever of a mobile phone WMS client.

7.2.2 Properties of client

This part briefly presents the properties of the client and some of the problems that were discussed. For more detailed information see Bozkurt (2005).

The client features a selection tool where layers are enabled or disabled. Furthermore, panning and zooming functions are implemented. Also, a SVG-player, implemented by Sony-Ericsson is used to render the maps. This format is chosen to put more control on the map rendering process in the client. A mobile map application should include the user's own position. Therefore, a GPS receiver is connected to the client. As long as the requested map from the WMS server uses the same coordinate system as the GPS, the client adds the position of the user on the map. The client also enables automatic *GetMap*-requests when the user's position approaches the map border. In this way, the user is always guaranteed to have a map that includes his or her position.

7.2.3 Special remarks

In the process it was discovered that the size of the requested maps from the WMS server often was too large. For example, if the coordinates are given in large float numbers, the result is a larger file. This is a clear disadvantage for mobile applications of WMS, since both the bandwidth and the computational capacity often are smaller compared to an Internet WMS client. Therefore, the smaller capacity of a mobile client should be taken into account by a server. A suggestion is that the server response includes an extra parameter stating the size of the file (see Bozkurt 2005).

When integrating the GPS with this client, the need for maps in the same coordinate system was discovered. This demonstrates a problem of WMS. Even though it is a stable standard, sometimes a server and a client have to be developed in accordance with each other for a special service, e.g. mobile maps with a GPS position.

Due to the small display of a mobile phone, the map should be free from disturbing symbols and cartographic objects should rather be generalised than displayed in all their detail. Therefore a mobile map is not really good until an icon and text placing routine (described in chapter 9), as well as generalising routines, are integrated within the WMS server. For more interesting remarks on the limitation of WMS for mobile application see Bozkurt (2005) and Bozkurt et al. (2005).

8 WMS Servers

This chapter deals with the experiences of two implementations of WMS servers

8.1 Introduction

Several implementations of WMS servers exist on the market. Many of them are developed as open code projects and are available for free (for private use). Other studies, e.g. Lennartsson (2004), show that from a client perspective, the majority of WMS servers works fine and conform well to the standard. However, if a problem exists, it is often in the *GetFeatureInfo* method, where supported formats (annotated in the *GetCapabilities* document) cannot be resolved upon request.

In this study, two WMS servers from two different vendors were set up. The servers have provided Sony-Ericsson with some test data. It is possible that these two in the future will act as a cascading service (see chapter 10), where the icon and text placement routines are integrated.

Below, a description of the two servers used within this study follows. The configuration procedures are briefly described as well as some of the problems encountered while setting up the servers.

8.2 Geoserver

Geoserver is an implementation of the Web Feature Service and has an integrated WMS server in it. It is a free software, developed by a number of persons with financial support from *The Open Planning Project* (Geoserver 2004). *Geoserver* offers two ways of storage of data. It is possible to store data either as *shape* files (see section 3.3.2) or in databases. Four types of databases are supported, among them *PostGIS*, which was used within this study. *PostGIS* is an extension to the *PostgreSQL* relational database, which expands the database with tools for queries and other operations on geographical data (PostGIS 2004).

8.2.1 Configuration

Geoserver in this study was set up on a Linux machine by Roger Groth at the GIS-centre. It is run under a *Tomcat* servlet container. Data are stored in a *PostgreSQL* database. (See figure 8.1.)

In order to store new data in the database, a program, *shape2pgsql*, in combination with scripts written by Groth were used. This program dumps *shape* files (the original dataset) into the database and the scripts convert them into *PostGIS* format. This is annotated with the converter symbol in figure 8.1. The database was physically located on the same machine.

Geoserver includes a web administration tool. It consists of forms that enable changes of the descriptive parts of the WFS and WMS *GetCapabilties* documents. These documents are then automatically generated. The administration tool also has forms where data are added. Thus, a new layer could easily be added through a drop-down menu that contains layers represented in the database. Thereafter parameters could easily be changed, e.g. *style*, *title*, *SRS* and *BoundingBox*.

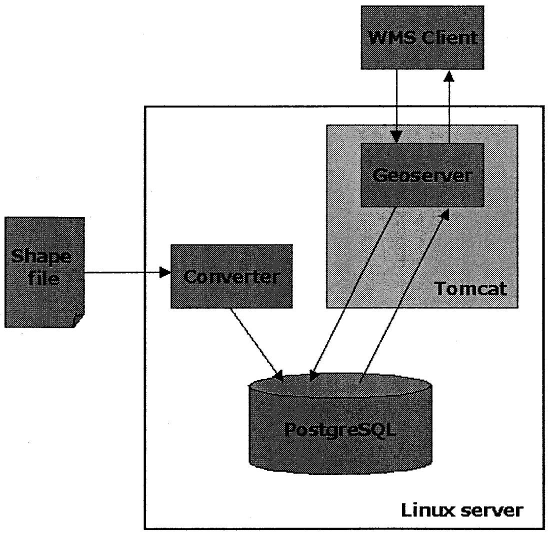


Figure 8.1: Structure of configuration

8.2.2 Encountered problems

It was discovered that the *Java Advanced Imaging* (JAI) package did not work. This lead to a situation where the *GetCapabilities* document declared that the WMS supported several image formats (JPEG, TIFF, PNG, etc.), but a *GetMap* request with another format than SVG always returned errors. The reason for this might be that the path to the JAI package is not correct (some different placements of the JAI package were tried unsuccesfully). However, since neither Sony Ericsson nor the GIS-centre used any other formats than the SVG (see section 3.3.2), this was not further investigated.

Another problem with *Geoserver* is that it only supported the same standard style for each layer. (This will be altered in forthcoming versions; the version used in this study was 1.2.0.) A map containing several layers did not turn out well, because it was impossible to discern e.g. a road from a railway or a building from a water surface. Oddly enough, it was possible to modify a layer's style name in the administrator tool, but a new style sheet (SLD document) could not be uploaded. This way, the administrator could let the *GetCapabilities* document show that a layer could be presented in different styles. However, only the normal style would result in a map, the other requests were answered by an exception message, stating that the server does not recognize the desired style.

8.3 Deegree

Deegree is also a free, open-source project implementing OGC standards such as WMS and WFS. It is created by the Department of Geography at the *University of Bonn* (Germany) and the company *lat/lon* (also in Bonn).

Deegree runs under a java servlet container. It requests data from local stores (*shape* files or databases) but could also be configured to fetch data from remote WMS/WFS servers. In this study the *deegree WMS* (demo version 1.1.2) was installed on a *Windows 2003* server under an *Apache Tomcat* servlet (version 4.1.31). The java version used was Java Runtime Environment, version 1.4.2. Data were stored locally as *shape* files.

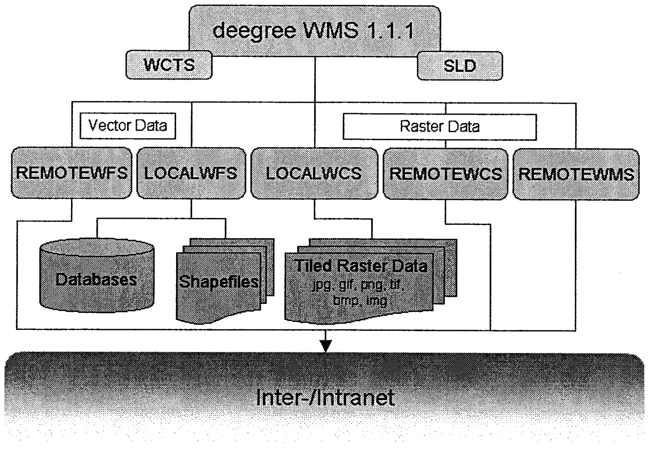


Figure 8.2: Architecture of Deegree WMS. The configuration in this study corresponds to the LocalWFS branch with shape files. (From Deegree 2004.)

8.3.1 Configuration

In order to get the *DeegreeWMS* up and running, *Tomcat* must be configured to look for *Deegree*. This is done in the **server.xml** file in *Tomcat's config* folder. In the *Deegree* file structure, the **web.xml** file has to be slightly modified. This is the file that is needed by the servlet.

In this study *Deegre WMS* has been configured to fetch data from a *localWFS* datastore within the file structure of the program. To accomplish this, firstly the shape files were added under the data folder. Then, the following three xml-files were modified.

Sample_wms_capabilities.xml (in a non-demo version only called wms_capabilities.xml). This is the document that is returned by *Deegree* upon a *GetCapabilities* request. Every new layer must therefore be manually added in this document.

LOCALWFS_config.xml Here all layers (*FeatureTypes*) are mentioned with their data store type. The *responsible class* element tells what data sort the layer is (e.g. a shape file) and its *configURL* attribute points to a file where it is configured.

This configuration file is in this case **demo_config.xml**. It states where the data store (here: the shape file) is found physically.

8.3.2 Encountered problems within this study

Deegree does not always understand the value '*image/svg+xml*' to the FORMAT parameter in a *GetMap* request. If that problem occurs, the notation *'image/svg%2Bxml'* must be used. Here the plus sign is replaced by a percentage sign followed by its ASCII-code in hexadecimal form (base 16). If the client is not aware of this minor problem, it will get an error message saying that the desired format is unsupported, something that does not conform to neither the *GetCapabilities* document nor the reality. Worth mentioning is that this procedure has to be performed in other cases as well. Segerstedt (2005) shows that several signs have to be substituted with the corresponding ASCII-code, if the optional SLD or SLD_BODY parameters are used in a *GetMap* request.

9 Program for Text and Icon Placement

The implementation of text and icon placement algorithms was carried out in a java environment. This chapter describes the developing environment, the structure and workflow of the program and the contributions of this study.

9.1 Programming Developing Environment

The text and icon placement algorithms described in chapter 5 are written in java. The base consists of the *java runtime environment* package (version 1.4.1) and open-source packages for geometries and relations between geometries (see below). The code has been written, compiled and executed with the java developing program *Eclipse Platform* from IBM.

9.1.1 Program structure

The over-all structure is shown in figure 9.1. In this figure, three main packages are identified; GiMoDig, JUMP and JTS. Other packages are represented by the grey box marked "other java packages". These are e.g. the java standard package and sub packages that are of less interest in this presentation.

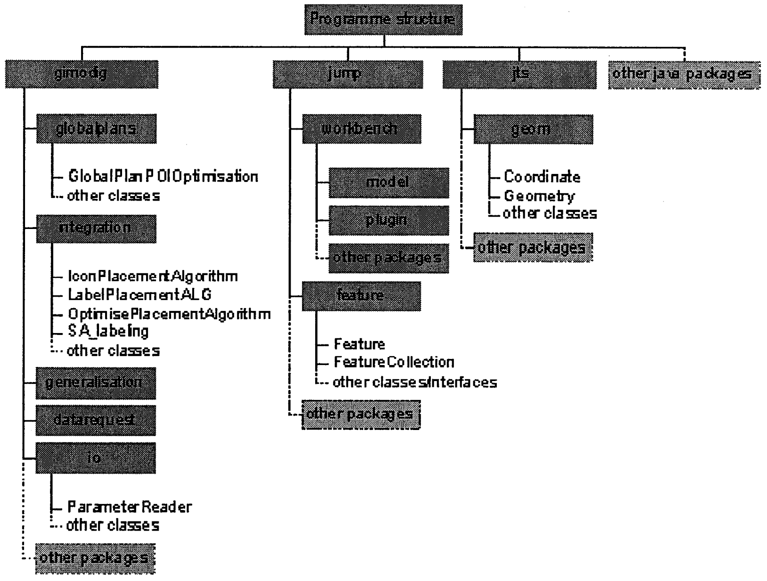


Figure 9.1: Package structure

JUMP package

The algorithms are developed in a special java environment named JUMP, which stands for Unified Mapping Platform. JUMP, developed by Vivid Solutions, contains a graphical user interface, GML parsers and classes for features and collection of features. Features are then used as layers in the graphical representations. (Vivid Solutions 2004b)

The JUMP package consists of several main packages of which two will be mentioned here. The *Feature* package consists of two interfaces, *Feature* and *FeatureCollection.* A dozen of classes implement these interfaces and together they constitute the layer of abstraction between the JTS geometries (see below) and layers in the graphical user interface. This GUI is hereafter called 'JUMP Workbench' or just 'workbench'. The *Workbench* package contains classes that create this interface (shown in figure 9.3). Through the *Workbench plugin* package, other packages can be connected to the workbench. The GiMoDig operations, such as icon and text placement optimisations are in this way integrated into the JUMP workbench.

JTS package
The JTS package contains classes that implement a robust description of geometries and spatial operations (Vivid Solutions 2004a). Classes commonly used by the GiMoDig package are *Geometry* and its subclasses *GeometryCollection, Point, Polygon* and *LineString*. The most common topological and geometrical methods are implemented on these classes.

GiMoDig package
In the scope of this study, the GiMoDig package is the most interesting. Its sub packages and classes have been developed within the GiMoDig project, which is briefly described in sections 1.1 and 11.3. For the concern of this study the classes handling icon and text placement were investigated and some of them were modified. A few additional classes were also created.

GlobalPlan classes control the workflow. The classes in the *datarequest* package request data from pre-defined WFS services. In the *io* package, classes read and parse requested data. In the *integration* package, classes are found that execute a text or icon placing algorithm or that contain special help methods for the algorithm classes.

Generalisation algorithms are found in the *generalisation* package. An example of an implemented generalisation is the Douglas-Peucker algorithm for the generalisation of lines. For example, the label-placing algorithm, when operating on lines, uses this algorithm to obtain a simplified line.

9.1.2 Program Workflow

The *GlobalPlan* package constitutes the overall function of the algorithms. A *GlobalPlan* class is called from the workbench. Once this is done, the *GlobalPlan* class controls the workflow in its *run* method (figure 9.2). A call is made to *DataRequest* to fetch data; thereafter, *io* is used to parse this data into the program and the special routine (e.g. icon placement) is handled with a call to the integration package. The workflow is in detailed described for the *GlobalPlanPOIOptimisation* class, see section 9.2.1.

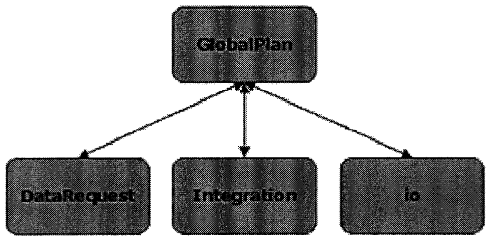


Figure 9.2: The *GlobalPlan* class controls the workflow with the interacting classes (here represented by whole packages).

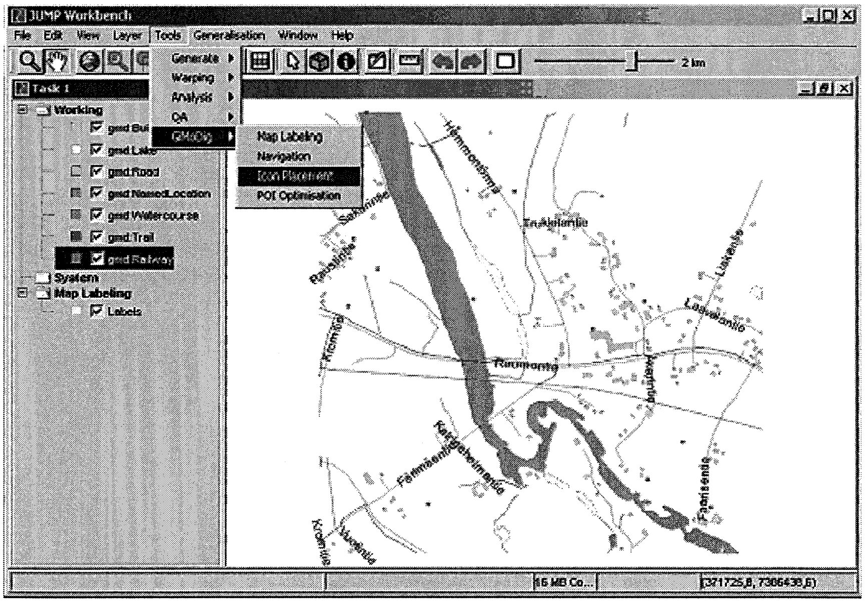


Figure 9.3: Screenshot from the JUMP Workbench. Upon starting the workbench, calls can be made to the GiMoDig operations. The user chooses an option in the drop-down menu, and this then makes a call to the corresponding GlobalPlan class.

9.1.3 Initial status

The icon and text placement algorithms were fully implemented and functional. However, due to time limitations of the project, the code showed some deficiencies.

- Several variables were raw coded. This was the case for the variables controlling the fetching of data, as well as for the variables controlling the preferences of the algorithms. Therefore, the testing and evaluation of an algorithm was difficult, because all the small changes had to be performed at the right spot in the code.
- Some classes suffered from absence of clear descriptive comments in the code.
- The structure of the classes was not fully logical.
- No external documentation was available.

9.1.4 Modifications made within this study

This study has contributed to maintaining the original ideas of the class structure and to eliminate raw coded variables and parameters. As for the effectiveness and the performance of the code, nothing was changed. More specifically, the following was done:

1) A minor "cleaning-up" of the code in the three *GlobalPlan* classes *MapLabeling, IconPlacement* and *POIOptimisation* as well as in some classes in the *integration* package. This means that unnecessary lines in most cases were deleted, the code was made more "airy" in the way the author prefers it and some extra comments were made. However, this cleaning-up was not carried out completely, and there is still some work to do.

2) A new class, *OptimisePlacementAlgorithm* was added to the *integration* package. This class consists of a run method, which constitutes almost the same lines of code as the former *optimisePlacement* method in *GlobalPlanPOIOptimisation*, By doing this, the structure better conforms to the other classes and methods found in both the *gimodig.globalplans* package, as well as in the *gimodig.integration* package.

3) The *ParameterReader* class (in the *gimodig.io* package) was created. This class reads a text file (an example is found in appendice A) and stores the values as variables of a *ParameterReader* object. These can then be accessed from methods taking the *ParameterReader* object as a parameter. Often, the variables are numbers or text strings, but the *ParameterReader* also creates instances of other classes, such as *java.awt.font* and *jts.geom.envelope*.

4) The *ParameterReader* object is initiated and the text file is read in the constructor of a *GlobalPlan* class, thereafter it is passed on as a parameter to supporting classes and methods. Changes are made in classes that use the *ParameterReader*. In a few cases some extra lines of code have been added. This approach was used when a specific class object could not be created within the *ParameterReader*, but one or more variables from the *ParameterReader* were used to create the object, e.g. the labelling methods in *LabelPlacementAlgorithm*.

5) Name changes of methods so that they better conform to their task.

6) The documentation found in section 9.2.

This was not done, but it would be desirable to proceed with:
- commented and cleaned-up code everywhere.
- a complete documentation of the methods.
- evaluation of the algorithms.

9.2 An Implementation of Simultaneous Text and Icon Placement

9.2.1 GlobalPlanPOIOptimisation class

This part will give an overview of the class. The class makes calls to both text- and icon-placing algorithms. At the end, the *optimisePlacement* method finds the optimal placement for icons as well as for texts. This is achieved by a number of calls to different algorithms, found in the *gimodig.integration* package.

This class is an implementation of the theory presented in section 5.2. Figure 9.4 shows how the work is carried out. It starts with finding candidate positions for icons, then candidate positions for texts and ends with an optimisation of the placement. Below the class methods are described.

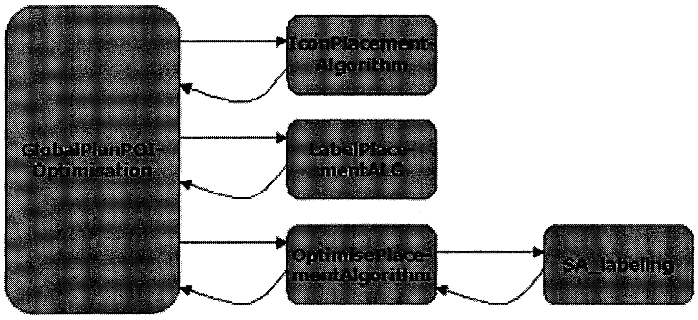


Figure 9.4: Figure of the main calls of the class.

The class consists of the following methods and internal classes:

- an empty constructor

- *run* method: This method executes all tasks so that icons and texts are positioned. It makes calls to all the other methods in the class. First it initiates a *ParameterReader* object and reads user-defined parameters from a file. Then it calls the following methods, which are also found in the file; *setParameters, getCartographicData, placeIcon, placeText, optimisePlacement* and, finally, *addToLayerManager*.

- ***GlobalParameters*** **class:** This class contains variables that are used by a number of methods in this class. Since they otherwise would have been global variables, they are now enclosed in an object that is passed on where these variables are needed. Examples of variables in this class are the vectors that contain candidate positions for texts and icons, special context objects for the algorithms and query strings that are sent to a server.

- ***setParameters*** **method:** Here some of the variables of the *GlobalParameter* class are given new values, some are verified to be cleared.

- ***getCartographicData*** **method** is used to retrieve data from a WFS server. The data is parsed into the Java environment. During the course of this thesis, this method requested data from a local XML file instead of a WFS server. To do this the *getDataFromFile* method is called.

- ***getDataFromFile*** **method:** This method is only used for local tests of the algorithms, when the data are not of any importance. It parses an XML file into the environment.

- ***placeIcon*** **method:** The only thing this method does, is to make a call to *iconPlacementAlgorithm,* which is an implementation of the theory presented in chapter 5.2.1. Candidate positions are chosen for each icon, and these are herafter found in the *GlobalParameter* class.

- ***placeText*** **method:** This method creates an instance of the *LabelPlacementALG* class, which is an implementation of the theory presented in chapter 5.2.2. After this, methods within *LabelPlacementALG* are called. These methods define which feature/layer should be labelled, e.g. roads and governmental buildings. After the algorithm is executed, candidate positions for each text label are stored in the *GlobalParameter* class.

- ***optimisePlacement*** **method** creates an instance of the *optimisePlacementAlgorithm* class. Hereafter, it calls the run method on the created object. The result of this method is that icons and texts are placed at the best possible position. See section 9.2.2.

- ***addToLayerManager*** **method.** With this method, the icons and texts are made visible in JUMP. The method simply adds icons and texts to a *LayerManager* object. The method uses the context variable (*PlugInContext* class) in the internal *GlobalParameters* class to get the right names of the layers.

- ***IconComparator*** **class.** This internal class defines how to compare two icons to one another; that is, to find which of two icons is closest to the original placement.

9.2.2 Workflow of the OptimisePlacementAlgorithm class

This class is an implementation of the approach given by Zhang and Harrie (2005b), summarized in section 5.2.3 in this study. It functions as a control process to the optimization process. The optimization task is implemented in the *SA_Labeling* class, where SA stands for simulated annealing. Also, other classes are requested, which all carry out specific tasks.

This class consists of two methods, a constructor and a *run* method. The constructor is empty whereas the run method takes six parameters. Among them only the *ParameterReader* is not retrieved from the *GlobalParameter* class (see above section 9.2.1). The vectors *theText* and *theIcons* contain all candidate positions obtained in the *placeIcon* and *placeText* methods.

While executing the *run* method, a set of vectors is used as temporary storage units. Finally after repeated calls to other help classes and iterations on these vectors, the optimised placements for text and icons are stored in a *FeatureCollection* object in *GlobalParameters.* Figure 9.5 aims to describe the workflow.

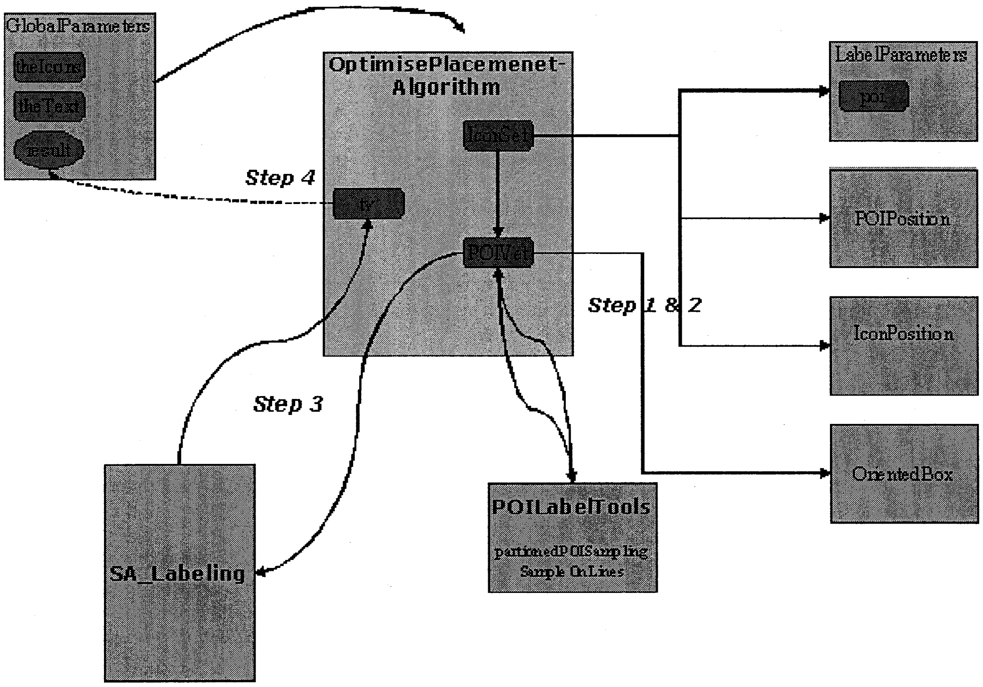


Figure 9.5: Important vectors in OptimisePlacementAlgorithm and calls to other classes.

Upon calling the run method, *theText* and *theIcons* vectors are transferred to the method. These contain candidate positions for ALL icons and text labels. Following the notations in figure 9.5 this is what happens hereafter:

Step 1 – reduces number of candidate positions for icons
iconSet contains all positions of ONE icon. Each of the elements in the *iconSet* vector is stored as an instance of the *LabelParameter* class. This class contains e.g. properties like text, width, height, coordinates (one if the icon symbolizes a point object, two – start and end coordinate - if it symbolizes a line object) and a vector containing the candidate positions. Furthermore, objects of the classes *IconPosition,* and *POIPosition* are used as storage variables for each *iconSet* element.

Each *iconSet* element is stored in the *POIVet* vector. When this has been filled, the candidate positions are reduced through calls to methods within the *POILabelTools* class.

Step 2 – reduces number of candidate positions for texts
A similar procedure is also carried out on the text. *OrientedBox* is used to describe the placement of a non-horizontal text label. With this step the *POIVet* vector then contains both icons and texts.

The used approaches when reducing candidate positions are here *partitionedPOISampling* and *sampleOnLines.* The partitioned sampling method chooses evenly candidate positions in four sections around a point (up left, up right, down left, down right). If there is a lack of points in one of the sections, extra points will be chosen in another section. The *sampleOnLines* calculates the sum of the length between the first and the last line node divided with the desired number of candidate positions. This number guarantees that the reduced candidate positions are evenly spread along the line. In the *POILabelTools* class, there are also other implementations of sampling methods. If another approach (e.g. random reduction) is preferred, that corresponding method can be used instead.

Step 3 – optimizes the placement and deletes overlaps
An instance of the *SA_labeling* class is created. Through the call to the *saVector* method, the whole optimization part is carried out and a vector containing only the optimal placement of all icons and texts is returned.

The theory of the optimization procedure is briefly described in section 5.2.3. A solution is guaranteed by an evaluation function. Firstly, high demands are put on the quality, that is a measure of the mutual placement of icons and text, but with decreasing execution time a solution is accepted even though the quality is not as good. Thus, overlaps might occur between some icons and/or text labels. As a last step, the overlapping icons and text labels are simply removed.

Step 4 – creates icons and labels and adds them to the map
Here additional tasks are carried out. The icons and labels are now stored at their optimal position. Finally, the icons and texts labels are added to a *LayerManager* object and stored in *GlobalParameters* as *result* (a *FeatureCollection* object). This way, texts and icons are visible as layers in the JUMP workbench.

9.3 Test of the program

This part does not evaluate the performance of the algorithms (it is beyond the scope of this study). It gives, however, a good example of how elaboration with parameters creates quite different results.

A test was carried out where the value of a parameter controlling the simulated annealing process was changed. The parameter controls the probabilistic element described in section 5.2.3. The probability for a fast solution decreases the lower this initial value is. Thus, one could expect that a lower value influences the quality of the map negatively due to an early accepted solution.

The program was executed 20 times for each configuration. The first configuration set this parameter for a higher quality on the map, whereas the second configuration featured a lower value. The execution time for the second configuration was almost 20 % faster than the original one. Furthermore the expectation that this resulted in a map of lower quality was intuitively verified by a quick glance on the maps. More conflicts with overlapping labels occurred within the second configuration. But it should be mentioned that, since the simulated annealing depends on a probabilistic element, the variation of both the execution time and the placement of icons and text was very large in both configurations.

	Icon	**Text**	**SA**	**Total**	**Maximum**	**Minimum**	**Quality**
Config 1	1394	423	14418	16235	24065	6900	better
Config 2	1360	403	11513	13292	28591	7220	good

Table 9.1: Time comparison. Figures are given in milliseconds but should just be seen as a measure of comparison since the program ran on a very slow computer.

Table 9.1 shows the difference in time between the two configurations. The first two columns indicate the time for the *placeIcon* and *placeText* methods (see section 9.2.1). They are independent from the configuration and should not differ. (It could be assumed that this value will converge towards a certain value with more executions.) SA stands for the simulated annealing process. The three leftmost columns indicate maximum and minimum total execution time and a subjective estimation of the map quality.

Figure 9.6 below shows a random map from the test. It is impossible to choose a typical map for each configuration since the positioning of icons (squares) and texts (rectangular boxes) vary as much as the time. Some maps rendered with the second configuration resulted in better maps, but all maps taken into account, the first configuration generally resulted with better maps (e.g. less overlaps).

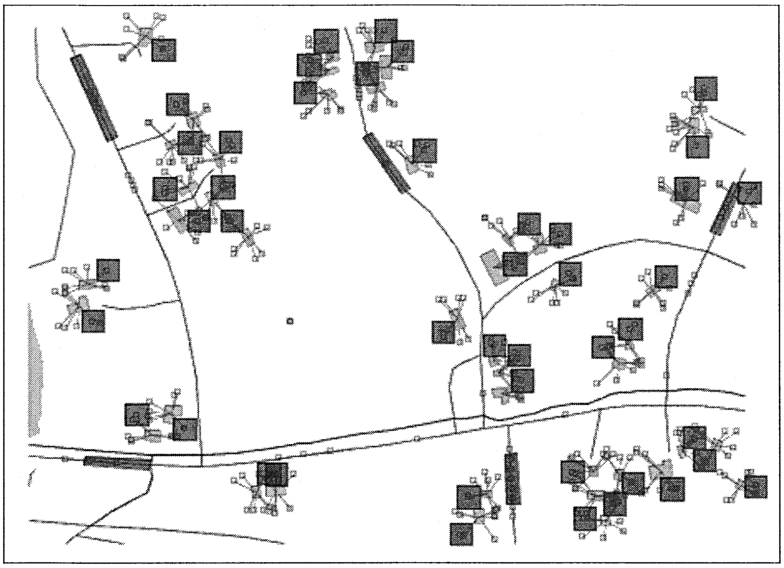


Figure 9.6: A map created within the first configuration. The overlaps are not removed in the figure. The small points constitute the remaining candidate positions. They are attached to the object that they represent through a thin line.

10 Model of Integration

Before beginning this study, expectations were indulged that it would end up with a WMS server with text and icon placement routines. This WMS service should have been made as ideal as possible for a client operating on small displays (such as the *SonyEricsson* client). Unfortunately, these expectations could not be met. Thus, this opens up to others to proceed with this task. Theoretically, this is how it *could* work:

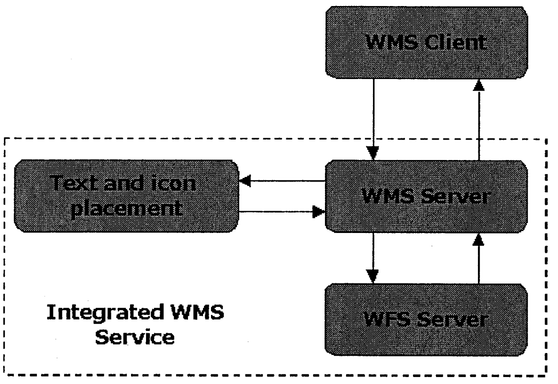


Figure 10.1: Model of an integrated WMS service.

From the client-side in the simplest model, the service functions like a normal standalone WMS. The client requests certain layers and the server responds with such a map. But before that, the WMS fetches data from a WFS server. The WMS is then implemented to send the data to the text and icon placement algorithms, which respond with layers containing icons and texts.

The first step in the integrated WMS service is not complicated. The most available implementations of WMS servers today support communication with a WFS server (see sections 8.2 and 8.3. The trickiest step in this model is the second step, the communication with the WMS server and the text and icon placement service. A tool for this communication must be created. Since, for example, *Deegree* supports JTS, such a tool should not be too hard to implement.

With a tool between the WMS and text and icon placement service the WMS has to construct a detailed request. An example of possible parameters in such a request (that is sent using GML notations) follows.

Label_layers. A list of layers that should be labelled.
Constraint_layers. A list of layers that should not be overlapped with disturbing values for each layer.
Quality parameter. A parameter that states how accurate the placement algorithms should work. This could for example be the parameter that controls the simulated annealing process (see sections 5.2.3. and 9.3).

Of course all the cartographic data has to be read into the text and icon placement module. The input consists of the same GML-strings that the WMS has requested from the WFS.

Part III

-

Final Analysis

11 Discussion

11.1 WMS servers and clients

This study presents two implementations of WMS servers (chapter 8). These are developed as open-coded projects, both directed from organizations that embrace the idea of sharing and improving code with others. Also, WMS-servers are developed by traditional GIS-vendors. This interest in the standard from both non-profit and commercial developers indicates that WMS has a future. WMS servers are stable and conform well to the standard, yet the standard allows different technical solutions, of which one has to be aware (see below).

The WMS clients presented in this study (chapter 7) were all developed within studies like this one. They all show that is easy to implement a client with one specific server in mind. However, more effort has to be put into a project for a WMS client that is fully compatible with all existing servers. Such a client does not have to constitute several packages of code, but at least it needs a well-defined library for error-handling and a flexible parser. This is due to the fact that different servers sometimes use different technical solutions, e.g. layer-nestling (see section 4.2.2). Developing clients for the web does not encounter any larger difficulties. This is, however, not the case with mobile applications (see section 11.2).

But is WMS a common standard for map services on the Internet today? Which standards are its competitors? The answer is that there exist several WMS services on the Internet, but given as a percentage of the total number of map services, this percentage is rather small. When WMS is chosen, it is for minor projects and more on a non-profit basis, whereas other solutions are used for larger, often commercial, projects. This is probably due to the fact that GIS professionals are used to the GIS-vendors' programmes. Therefore, it is natural to continue working with these programmes, when a new service is created. Among these other solutions from GIS vendors, ArcIMS from ESRI is probably the most widely used. But almost every GIS-vendor has its own technical solution for (interactive) map services on the Internet. It is somehow reasonable to believe that the use of WMS in larger applications will increase in the future. Both in Denmark and Norway, there are good examples of public institutions using WMS in map applications. In Sweden, groups with both public and commercial participants are looking into the possibilities of WMS (see e.g. Segerstedt 2005).

11.2 WMS for mobile applications

Throughout the work of this study regular discussions were held with personnel at Sony-Ericsson. These discussions dealt with WMS as a means for maps in mobile phones and the special limitations such mobile applications encounter.

Even though WMS is well defined and constructed, it is often difficult to apply a standard service to which a standard client connects. In the case of mobile phone applications, there are two main constraints compared to ordinary web applications. Below these are described and possible solutions are suggested. For more information on this matter, see Bozkurt (2005) and Bozkurt et al. (2005), where these limitations or constraints are discussed more thoroughly.

The first constraint is the limited bandwidth. WMS does not support any means, such as a special parameter, that tells the client the file size before it is transferred. If a rendered map contains large float number coordinates (in the case of vector formatted maps) the file size can be very large. Another factor that increases the file size is the grade of generalisation. If, for example, a map contains lots of buildings and these are rendered with a line between every corner instead of being generalised to a pre-defined square or rectangle, this has a major increasing impact on the file. A solution for this problem must be taken care of on the server-side. If the service only provides integer coordinates and if it is connected to generalising routines (an enlarged model of the one presented in chapter 10) the file size will decrease. It is perhaps more complicated to guarantee small file sizes to a client. Either a special WMS service to mobile phones guarantees small file sizes from the beginning, or the WMS interface must expand with an optional file size parameter that is defined to serve e.g. mobile phone clients. The latter option would be desirable, since one aim with of developing WMS clients for mobile phones is to enable the use of several servers in different situations and thus not be dependent on one service only.

The second constraint is the small display on a mobile phone. This constraint demands much more of a map than you imagine at first. Today there are quite a few research projects in this area and this study would like to refer to the GiMoDig project and research at the GIS-centre. One thing is nevertheless certain: A small display demands a rigorous selection and reduction of the cartographic objects to be shown. Furthermore, to increase the readability, texts and icons must be used economically, and, when being used, placed in an intuitive position clearly indicating which object they belong to. This is really the continuation of this study: to adjust WMS rendered maps so that they conform to small displays! As for the bandwidth constraint, a solution would be to integrate a WMS service with a combined generalisation and integration application. Another means would be by conforming SLD (styled layer descriptor) documents for small displays. Through the combination of SLD and filtering, a kind of generalisation can be achieved.

Is WMS really an alternative for mobile applications? The development of WMS clients should lead to a greater freedom for mobile phone vendors to choose data distributors. However, today the main actors in mobile phone map services use their own formats. To guarantee and improve these services, it is perhaps better for mobile phone vendors to collaborate with these actors instead of using a new interface. For the mobile phone vendors' point of view in this matter, see the conclusions given in Bozkurt (2005).

11.3 Icon and text placement

The main part of this study concerns icon and text placement. This section aims at putting icon and text placement in a larger perspective. Besides good icon and text placement, generalisation and reduction of objects are of major importance to obtain a map of high quality. The term map rendering algorithms includes all these functionalities as well as the basic (map) image generating methods.

It is no understatement to assert that this field of research will continue to increase in importance. Digital maps have been more accessible the last 10 years due to the wider use of the Internet, but today much more demand is put on the map presentation and interaction. Furthermore, digital maps are today more often rendered in real-time. This indicates the need for all kinds of effective map rendering algorithms.

Several GIS programmes have long featured built-in easy text placement functions. In an ordinary desktop GIS environment such functionalities are rather easy to include. A simple placement algorithm in such an environment benefits from all spatial operations that are already defined in the environment.

Text and icon placement turns out to be more complicated when it comes to Internet GIS applications. This is often due to the fact that a large number of standards regarding storing, rendering and transferring of geographical data and maps exist. If an automated icon and text placement function exists, it is likely to have been developed for a certain environment. An example is the *Deegree WMS* server that has a built-in functionality on text placement, which, although rather simple, improves the map quality.

The GiMoDig project is an attempt to customize a model for these matters (Gimodig 2005). The idea of GiMoDig is a gathered a map rendering service that includes the fetching of data from different data providers. It is an ambitious project, but it will not have an impact on standardizing text and icon placement methods. Having all different formats and services concerning geographical data in mind, this seems hard to accomplish.

12 Conclusions

The aim of this Master of Science thesis has been to study the WMS interface and algorithms for text and icon placement. The title of the study indicates an integration of these algorithms with WMS. This was, however, not accomplished by the study, but it contains a description of a possible integration approach. The study presents WMS servers and clients, as well as algorithms for text and icon placement.

This study has shown that the WMS interface is fully functional. Within the study two different implementations of server software were used, *Geoserver* and *Deegree WMS*. Their functionality was verified both through manually constructed requests in an ordinary web browser and through the clients described in this study. The configuration and the administration both differed between the two servers as well as other differences on technical approaches.

It is possible to develop clients according to the WMS specification. However, there are a number of constraints that have to be dealt with in order to create a flexible client. With a flexible client it is understood that it easily communicates with different servers. Examples of such constraints encountered within this study are different types of layer nestling, reference systems and coordinates.

The main emphasis in this study was put on work on algorithms for text and icon placement. They are now documented and the programme structure is improved. These algorithms minimize overlaps between text, icons and cartographic objects. At the same time they follow rules, so that the position of a text or an icon is intuitive in relation to the object it represents.

The map rendering process through WMS does not guarantee good text positions. However, some of the implementations of servers support basic kinds of placement methods. The algorithms that are presented in this study would richly enhance this process if they could be connected to a WMS service. This study shows that a bridge between WMS and these algorithms is necessary. Such a bridge does not yet exist.

Literature

Books and articles

Bozkurt, M. 2005: *WMS on a mobile phone*, Master of Science Thesis, Lund Institute of Technology, Sweden

Bozkurt, M., R. Groth, B. Hansson, L. Harrie, P. Ringberg, H. Stigmar, and K. Torpel, 2005: Towards Restricting File Sizes in Web Map Services for Mobile Applications. *ICA Workshop on map generalisation, A Coruña, Spain*

Eklundh, L., W. Arnberg, S. Arnborg, L. Harrie, H. Hauska, L. Olsson, P. Pilesjö, B. Rystedt, and U. Sandgren 1999: *Geografisk informationsbehandling: Metoder och tillämpningar,* Byggforskningsrådet, Stockholm, Sweden, ISBN 91-540-5841-4

Harrie, L., H. Stigmar, T. Koivula and L. Lehto 2004: An Algorithm for Icon Placement on a Real-Time Map. *In Fisher, P. (ed.), Development in Spatial Data Handling*, Springer, pp. 493-507

Harrie, L., Q. Zhang and P. Ringberg 2005: A Case Study of Combined Text and Icon Placement, *XXII International Cartographic Conference, A Coruña, Spain*

Hunter, D., C. Gagle, D. Gibbons, N. Ozu, J. Pinnock, P. Spencer 2001: *Beginning XML*, Wrox Press Ltd, Birmingham, UK

Lennartsson, T. 2004: *Utveckling av en Web Map Service-klient*, Master of Science Thesis, Lantmäteriet, Gävle, Sweden

OGC 2002: *Web Feature Service Implementation Specification*, version 1.0.0. Available at http://www.opengeospatial.org/specs/?page=specs

OGC 2003: *Geography Markup Language Implementation Specification*, version 3.0. Available at http://www.opengeospatial.org/specs/?page=specs

OGC 2004: *Specification for Web Map Service*, version 1.3 Available at http://www.opengeospatial.org/specs/?page=specs

Peng, Z.-R. and M.-H. Tsou 2003: *Internet GIS*, John Wiley & Sons, Inc., Hoboken, N.J., USA

Robinson, A. H., J. L. Morrison, P. C. Muehrcke, A. J. Kimerling and S. C. Gubtill 1995: *Elements of Cartography*, *Sixth edition*, John Wiley & Sons, Inc., Hoboken, N.J., USA

Segerstedt, P. 2005: *Distributed Internet GIS: Prototypes of Cascading Web Map Service (WMS) for applications in Skåne,* Master of Science Thesis, Lund Institute of Technology, Sweden

Zhang, Q, and L. Harrie 2005a: Real-Time Map Labelling for Mobile Applications, *Computers, Environment and Urban Systems*, accepted.

Zhang, Q, and L. Harrie, 2005b: A Real-time Method of Placing Text and Icon Labels Simultaneously, submitted.

W3C 1999: *HyperText Transfer Protocol – HTTP1/1.* Available at http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf

Internet

Deegree 2004: *Installing documentation*: http://deegree.sourceforge.net/inf/wmsdoc/html/index.html accessed 2005-01-17

EPSG 2005: http://www.epsg.org accessed 2005-06-15

Geoserver 2004: *The Geoserver project*: http://geoserver.sourceforge.net/html/index.php accessed 2004-12-20

GiMoDig 2005: http://gimodig.fgi.fi accessed 2005-06-15

ISOC 2003: *A Brief History of the Internet, version 3.32:* http://www.isoc.org/internet/history/brief.shtml accessed 2004-10-05

OGC 2005: http://www.opengeospatial.org accessed 2005-06-15

PostGIS 2004: http://postgis.refractions.net/home.php accessed 2004-10-25

NSCA 2004: http://hoohoo.ncsa.uiuc.edu/cgi/ accessed 2004-09-24

Vivid Solutions 2004a: *Java Topology Suit documentation*: on http://www.vivivdsolutions.com/jts/doc.htm accessed 2004-11-29

Vivid Solutions 2004b: Unified *Mapping Platform documentation:* http://www.vivivdsolutions.com/jump/doc.htm accessed 2004-11-29

W3C 2004a: *Extensible Markup Language (XML) version 1.0*, W3C Recommendation: http://www.w3.org/TR/2004/REC-xml-20040204/ accessed 2004-09-16

W3C 2004b: *HyperText Markup Language:* http://www.w3.org/TR/html401/ accessed 2004-09-15

W3C 2004c: *Extensible Markup Language (XML):* http://www.w3.org/XML/ accessed 2004-09-16

W3C 2004d: *Scalable Vector Graphics:* http://www.w3.org/Graphics/SVG/ accessed 2004-09-17.

Appendix

An input file for ParameterReader

```
/*******************************************************************************************
*
*  FILENAME: parameter.txt
*
*  AUTHOR: Peter Ringberg
*
*   DESCRIPTION: This file enables testing of different parameter values in the GlobalPlan-
classes
*  as well in the algorithm classes (in gimodig.integration library).
*
*  The value of a parameter must be typed correctly.
*
*  After an instance of the class ParameterReader (gimodig.io) has read this file,
*  the value of the parameter is accessed through notation
*    (object of ParameterReader).get(Name of parameter, always starting with an upper-case
letter) ()
*
*   The path to this file has to be stored within the InputFileName class (in gimodig.io
library).
*******************************************************************************************

/*******************************************************************************************
*
* GENERAL PARAMETERS
*
*
*******************************************************************************************

WORK_DIR C:/Program/Eclipse/eclipse-SDK-3.0M6-win32/eclipse/workspace/peter_
OUTPUT_TEMPLATE /templates/out/GiMoDigOutputTemplate.xml
INPUT_TEMPLATE /templates/in


/*******************************************************************************************
*
* DATA REQUEST PARAMETERS
*
*
*******************************************************************************************

WFS_SERVICE http://gimodig.fgi.fi/Topo-WFS/GetFeature
WFS_USERNAME gimodig
WFS_PASSWORD fubkln3-4
WFS_PREFIX <?xml version='1.0' encoding='iso-8859-1'?>


*
*  The following parameters create a query string to send to a WFS server,
*
*  ParameterReader reads maximum nine requested features (NamedLocation-Trail).
*  The two parts correspond to the whole query. These are in ParameterReader
*  concatenated where (_1) ends before sBoundaryBox and (_2) begins after the
*  sBoundaryBox.
*
*  NOTE! The parameter names in this file are example of typical feature names.
*  Any feature from the WFS can be used to any parameter.
*
*  The values of the createWFSQueryString parameter are the names of the desired
*  complete query string (all features that constitute the query).
*  This parameter MUST always begin with 'queryPrefix queryGetPrefix' and end with
*  'queryGetSuffix'. The features are named according to the corresponding parameter name,
*  that is without _1 or _2.
*  [E.g. queryPrefix queryGetPrefix queryRoad queryPark queryGetSuffix]
*
*  The sBoundaryBox is typed like this:
*  xmin,ymin, xmax,ymax
*  Note the comma and space notation! Also note that non-integer values are
*  written with a decimal point, e.g. '1234565.12,234567.434 124444.43,235564.12'
*
```

```
queryPrefix <?xml version='1.0' encoding='iso-8859-1'?>
queryGetPrefix                  <GetFeature                 xmlns='http://www.opengis.net/wfs'
xmlns:ogc='http://www.opengis.net/ogc'                  xmlns:gml='http://www.opengis.net/gml'
outputFormat='GML2' handle='GiMoDigQuery'>

sBoundarybox 370085,6675194 371914,6677205

queryNamedLocation_1         <Query          handle='query00'          typeName='NamedLocation'
version='1.0'><PropertyName>name</PropertyName><PropertyName>position</PropertyName><PropertyN
ame>category</PropertyName><PropertyName>orientation</PropertyName><ogc:Filter
xmlns:gml='http://www.opengis.net/gml'><ogc:BBOX><PropertyName>position</PropertyName><gml:Box
srsName='EPSG:4258'><gml:coordinates>

queryNamedLocation_2 </gml:coordinates></gml:Box></ogc:BBOX></ogc:Filter></Query>

queryRoad_1
queryRoad_2

queryAdministrativeBoundary_1
queryAdministrativeBoundary_2

queryBuiltUpArea_1
queryBuiltUpArea_2

queryBuilding_1
queryBuilding_2

queryLake_1
queryLake_2

queryPark_1
queryPark_2

queryRailway_1
queryRailway_2

queryTrail_1
queryTrail_2

queryGetSuffix </GetFeature>

createWFSQueryString queryPrefix queryGetPrefix queryNamedLocation queryRoad queryGetSuffix


*
*  This line creates an Envelope object which is used ONLY when cartographic data are read
*  from a local file, and not from a WFS server. Gets the extent of the viewport and clips the
dataset.
*  The parameters are separated by space. In order, they constitute:
*  minx maxx miny maxy
*

querybox 372000 373500 7305500 7307000



/*****************************************************************************************
*
* ICON PLACEMENT PARAMETERS
*
******************************************************************************************

DEF_BUFFERT_WIDTH 1.0
DEF_BUFFERT_HEIGHT 1.0
DEF_WIDTH 20
DEF_DISTURBANCE_VALUE_TRESHOLD 0.2
DEF_TEXT_SIZE_DEFAULT 40
DEF_TEXT_SIZE_SETTLEMENT 90
DEF_SEARCH_DISTANCE_GROUND 60
DEF_DISTANCE_LIMIT_POINTS_ON_LINE_SEGMENT 5
DEF_DISTANCE_LIMIT_POINTS_ON_POLYGON 20

* This parameter determines the resolution of the search relative
* to the icon size. Must be an odd number.
```

```
Icon_search_resolution 9


/*********************************************************************************************
*
* TEXT PLACEMENT PARAMETERS
*
**********************************************************************************************

// double

allowedRatio 0.8
scaleDenominator 10000


* These lines create two Font objects from the java.awt font class.
* Their parameters are:
* [name of font] [font.PLAIN] [font size}´]

fontLoc Arial Font.PLAIN 12
fontRoad Arial Font.PLAIN 12


*
*  The constraints_X array contains the NAME of the FeatureCollections
*  (layers) chosed as obstacles for the following methods in LabelPlacementAlg:
*
*  1 labelingLocations()
*  2 labelingRoads()
*
*  The valures are typed in this way:
*    int string string string ...
*  where int corresponds to the number of layers. The amount of the following
*  strings must equal this number.
*  The strings contain two parts; namespace and name of layer, seperated by a colon.
*
*

constraints_1 3 gmd:Road gmd:Railway gmd:Building
constraints_2 3 gmd:NamedLocation gmd:Building gmd:Railway



/*********************************************************************************************
*
* OPTIMISE PLACEMENT FOR TEXT AND ICONS PARAMETERS
*
**********************************************************************************************

*
*   The following variable states how many candidate positions that should be selected to
remain for
*  the test to find the optimal position
*

nrOfReducedCandidatePositions 8


*
* These variables conduct the behaviour of the simulated annealing process. The lower the
temperature is
* from the beginning, the faster a map will be accepted (however probably with more
defiencies).
* [1/Math.log(3)], [0.1*initial-temperature]
*

DEF_INITIAL_TEMPRATURE 2.095903274
DEF_LOW_TEMPERATURE 0.2095903274


//Evaluation value [double - 1.80]

evaluationValue 1.80
```

```
* This will create a weight object. The values are four numbers (double) seperated by a space
* In order, the numbers are the values of:
* preference, disturbance, overlap, deletion

weight 0.001 0.01 1 10
```