Student thesis series INES nr 236

Automatic Geospatial Web Service Composition for Developing a Routing System

Author: Arash Amiri

2012

Department of

Physical Geography and Ecosystems Science

Lund University

Arash Amiri. Automatic Geospatial Web Service Composition for Developing a Routing System.

Master degree thesis, 30 credits in Geomatics.

Supervisor: Dr. Ali Mansourian

Department of Physical Geography and Ecosystems Science, Lund University.

Abstract

Routing or path finding is one of the most popular services, which is widely used by citizens and tourists, around the world. These systems, which are generally developed based on WebGIS provide users with the capability of introducing a start and an end point and then to observe the optimum (e.g. shortest) path between these two points. However these systems can only perform analysis for specific areas for which the data is available in the WebGIS and also can only work based on the specific routing algorithm(s), by which the system is designed. To increase the flexibility of the system, an alternative solution could be the availability of a Web-based routing system that works based on Distributed Geospatial Information System (DGIS) and Automatic Service Composition techniques. An aim of this research is to design and develop a system for routing based on automatic service composition.

Although, many efforts have been done to develop path finding algorithms for vehicle navigation, less attention has been paid on developing flexible path finding algorithms for pedestrians. So another aim of this research is to develop such an algorithm.

In this research, a routing algorithm for pedestrians was developed, implemented and tested based on OGC's WPS standard. This was conducted, as part of the European Union funded HaptiMap project. Also, general architecture of a routing system that works based on automatic Web service composition was designed and implemented. The results of the test and the evaluation of the system shows:

- The applicability and the flexibility of the proposed routing algorithm for pedestrians.
- The flexibility and the efficiency of routing based on automatic service composition.

Keywords: Automatic Web Service Composition, Routing in open spaces, WFS, WPS, GML, Syntactic Interoperability.

Acknowledgements

I would like to express my gratitude to my supervisor Dr. Ali Mansourian for his excellent guidance and sympathetic support throughout this research. His valuable advices were the biggest help for me during the 10 months I spent on doing this research. Also I would like to appreciate his detailed and instructive comments on this report very much.

Contents

1. Introduction

A Geographical Information System (GIS) is a system designed to capture, store, manipulate, analyze, manage, and present all types of geographical data (Foote & Lynch, 2000).Web GIS enables the communication of all components of GIS to happen through the web, enabling diverse data, analysis algorithms, users and visualization techniques that may be hosted at any location on the Web (Avraam, 2010).

Now a day, Web GIS provides citizens with the capability of using GIS and spatial planning techniques for daily activities. It is generally easy to communicate with WebGISs through user-friendly interfaces, which are designed for specific purposes such as map navigation, spatial search or analysis, e.g. finding the shortest path. However, while using WebGIS applications, users are generally limited to the functionalities and the data that are supported/provided by a WebGIS system. In other words, if a user's request is beyond the scope of the data and the functionalities of a WebGIS, the system is incapable of responding to that request and hence the user would get no result, although the required data and functionalities may be still available somewhere on Internet. Shifting from client-server WebGIS architecture to Distributed Geospatial Information Services (DGIS) architecture may be a solution to offer more proper GIS services to citizens. In this case, automatic service composition would be the central technique for achieving the aim. Service composition is the area of science which looks for proper processes and tools to chain Web services for fulfilling user's request (Yue et al., 2007). These services can interoperate together to fulfill the user's request. But the main challenge is still finding solutions on how to chain these services in a proper way and by which tools.

Routing or path finding is one of the most popular services, which is widely used by citizens and tourists, around the world. These systems, which are generally developed based on WebGIS, provide users with the capability of introducing a start and an end point and then to observe the optimum (e.g. shortest) path between these two points. However, as discussed earlier, these systems can only perform analysis for specific areas for which the data is available in the system. An alternative is availability of a service which is more flexible and can do routing on any given geographical boundary. This can be done through finding proper Web services and chaining them in a proper order.

On the other hand algorithms of path finding are each designed for a specific purpose. In general they can be categorized into algorithms designed for vehicles and algorithms designed for pedestrians. The rapid popularity of navigation systems in cars has caused the algorithms of second kind to be neglected. In most applications the path which is offered by the system is more proper for vehicles and special limitations, preferences and capabilities of pedestrians are not considered. The simplest example could be possibility of making shortcuts when walking in a city. Pedestrians are not restricted to streets and can make shortcuts through parks, yards, pathways and open spaces while traveling.

1.1 Aims

This thesis has two general aims:

a) Development of an algorithm for planning path for pedestrians which can be used to guide users in a city.

b) Using automatic Web service composition for routing.

1.2 A typical scenario

Suppose a user asks for the shortest path between a given arbitrary start point to another arbitrary end point in a city. Instead of connecting to a WebGIS -which is limited regarding data and functionality- a system based on automatic Web service composition can exist to which the user can connect and introduce the desired start and end points as well as routing algorithm. Then the system

- searches on the internet to find available data services and processing services (to satisfy routing),
- selects the most proper data and processing services,
- composes these services, automatically, to perform the routing analysis and to generate the result, and finally,
- shows the result to the user.

Such a system may be more flexible than a client-server WebGIS.

1.3 Thesis structure

This thesis is organized in 6 chapters. After the present chapter, which is an introductory, in chapter 2 path finding algorithms are reviewed according to their functionalities. Pros and cons of each method is discussed and some algorithms are described with more details. Chapter 3 describes the concept of Web service composition and reviews the most important standards and interfaces for Web service composition in geospatial community.

In chapter 4 the proposed path finding algorithm and how the algorithm was implemented is described. In chapter 5 the system architecture, which is used for automatic Web service composition is described as well as the implementation of the system. Finally the thesis ends with conclusion and future directions in chapter 6.

2. Path planning algorithms

There are a variety of path planning algorithms based on the purpose or application of the planning. The optimum path maybe based on the shortest distance or the fastest travel time. The path found for a vehicle may go through highways which makes it longer regarding the total length but makes the traveling time shorter. Most drivers prefer this comparing to waiting in long traffic lines in downtown streets. Other factors might be important in other fields. E.g. in robotics, each time a robot wants to make a turn it has to stop and it consumes considerable effort to make it run again. Here the number of turns becomes important and it is tried to offer a path with the least number of turns possible for the robot. To guide the traffic to desired directions in transportation science different costs could be defined for roads which give each road some weight that makes the road more proper or less proper for the final path. Some algorithms are proper for vehicles while other may be better for pedestrians. The concern of this thesis is planning path for pedestrians and for simplicity the focus is on finding the shortest path.

Shortest path algorithms can be divided into two main categories: Algorithms that rely on network structure and algorithms for open spaces. We will now discuss each of them in details.

2.1. Algorithms that rely on network structure

These algorithms are mostly used for vehicle navigation systems. They are based on a graph (network) consisting of nodes and edges/links (Figure 2.1). These algorithms are a main research question in the field of Geographic Information Systems. Some well-known methods are Dijkstra’s algorithm (Dengfeng & Dengrong, 2001), Floyd’s algorithm (Garcia et al., 2007) and A-star (Hart et al., 1968).

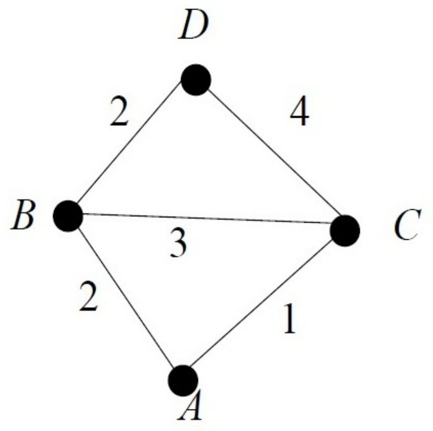


Figure 2.1 A network consisting of four nodes (A, B, C, D) and four edges. Each edge has a distance or weight (Harrie, 2010).

2.1.1 Dijkstra’s algorithm

Dijkstra’s is one of the well-known algorithms for determining the shortest path, proposed by Edsger Dijkstra in 1959. The algorithm finds the shortest path between two specific nodes as well as the shortest path between a node and all other nodes in the graph.

Considering a graph consisting of edges and nodes and a source node as inputs, a brief description of the algorithm is as following (Harrie, 2010):

- set the distance to every node on the graph with infinity (nodes with infinite distances from source are not ‘*visited’*).
- start iteration
- select a ‘*current node’* which is the closest unvisited node to the source (for the first iteration the current node is the source node and the distance to it will be zero)
- determine the sum of the distances between an unvisited node (which is directly connected to current node) and the distance determined for the current node, if it is less than the distance determined for unvisited node, update the distance for the unvisited node
- mark the current node as visited and select the unvisited node with lowest distance (from the source) as the current node
- do iteration until all nodes are marked as visited

The shortest path tree is constructed edge by edge and at each step one new edge is added to the tree. Dijkstra’s algorithm does not search along a possible path direction but rather in all directions (Harrie, 2010). Different weights can be defined for edges which can be length, traveling time or any other thing based on the purpose of application.

With a network consisting of n nodes, Dijkstra’s algorithm has a computational complexity of O (n^2).

2.1.2 Floyd’s algorithm

Similar to the Dijkstra’s the input of Floyd’s algorithm is a weighted graph but unlike the Dijkstra’s, the weights could be both positive and negative.

Considering nodes of a graph numbered through 1 to N and a function shortestPath(i, j, k) that returns the shortest possible path from node i to node j using vertices only from the set {1,2,...,k} as intermediate points along the way, the following recursive formula is the core functionality of the algorithm (Floyd, 1962):

shortestPath(i,j,k) = min(shortestPath(i,j,k-1), shortestPath(i,k,k-1)+shortestPath(k,j,k-1))

And the base case is:

shortestPath(i,j,0) = w(i,j)

Where w (i,j) is the weight of the edge between vertices i and j

Floyd's algorithm has a computational complexity of O (n^3), where n is the number of nodes in the network.

2.1.3 A-star algorithm

Hart et al. (1986) first described the algorithm in 1968. It is an extension of Dijkstra's algorithm. Dijkstra's algorithm's computational complexity (O (n^2)) makes it inappropriate for finding the shortest rout over long distances (e.g. between Stockholm and Copenhagen). Using heuristics the problem of high computational complexity in Dijkstra's is tried to be alleviated in A-star algorithm.

All the algorithms described above use a network structure for finding the shortest path which makes them limited regarding two aspects: First the source and destination should be a part of the network (in practice this means the user should be on one of the networks nodes). Second the path is limited to the edges of the network (while in practice it may be possible to make shortcuts in e.g. open spaces). This makes the algorithms inappropriate for pedestrians.

2.2 Algorithms for open spaces

These algorithms are not restricted to edges and nodes of a network. The user can start from anywhere (not always a node) and make shortcuts through open spaces. These algorithms are research questions in the field of Computational Geometry. They are designed for pedestrians who have more freedom in choosing the path and in some other applications such as finding optimized path for robots.

Most of the methods use the triangulation of a polygon (Lee & Preparata's, 1984; Guibas et al., 1986; Reif & Storer, 1985). These algorithms first make a triangulation of the open space polygon and then compute the shortest path in the resulted network. However the result is not the actual shortest path and is an estimation of the real path.

Pan et al. (2009) have designed an algorithm by creating an internal network inside the open space by linking all the pairs of vertices inside the polygon and the start point and end point. A shortest path is then found by using Dijkstra's algorithm. The algorithm is much more precise comparing to those algorithms which use triangulation. Yet the study does not get detailed about polygon holes (like buildings inside an open space) in the algorithm description.

All the efforts for developing a shortest path algorithm could be placed in one of the categories described in chapters 2.1 and 2.2. This thesis presents a combined algorithm which benefits from methods mentioned from both groups. Some methods are slightly modified to enhance the usability.

3. Web Service Composition

Service composition is the process of creating the service chain though composing a collection of interoperable Web services (Yue et al., 2007). Interoperability is the capability to exchange information, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units (Percivall, 2002). There are two levels of interoperability: Syntactic Interoperability and Semantic Interoperability (Percivall, 2002). The former deals with data encoding and software interfaces for data exchange. The latter deals with semantic definition and description of spatial data and Web services.

Syntactic interoperability is achieved by different standards and interfaces. In geospatial community the most popular standards are defined by Open Geospatial Consortium (OGC) together with ISO. OGC is an international industry consortium aimed to develop interface standards for making interoperable geo-services on Web (OGC, 2012). In geospatial community most of the standards are developed by OGC and are used more than any other similar standards.

The semantic interoperability is achieved by describing data and web services semantically, using ontologies. There are a variety of languages for developing ontologies such as RDF (W3C, 2012b), OWL (W3C, 2012a) and WSML (W3C, 2012c).

3.1 OGC standards

OGC has developed a variety of encoding and interface standards to satisfy syntactic interoperability among geospatial web services.

The main encoding standards are Geography Markup Language (GML) and KML (Portele, 2007). Examples of interface standards are Web Map Service (Beaujardiere, 2006), Web Feature Service (Vretanos, 2005), Web Coverage Service (Whiteside & Evans, 2008) and Web Processing Service (Schut, 2007). Each of these services is defined for special type of data or process. WFS defines the standard interface for Web services providing feature (vector) data. It returns the requested feature in GML format. WCS supports coverage (raster) data and the output is in pictorial format (JPEG/PNG etc.). WPS is in charge of the calculations on Web. One can download the data from the data-provider services such as WFS or WCS and pass the results to a WPS for further analysis. The WMS at last can be used for rendering the map in picture formats such as PNG or JPEG.

In this research GML, WFS, WPS are used. So these services will be explained in more details.

3.1.1 WFS

WFS defines standards for publishing feature (vector) data on Web (Vretanos, 2005). In a WFS three operations must be defined:

GetCapabilities: Describes which feature types the WFS can service and what operations are supported on each feature type.

DescribeFeatureType: Describes the structure of any feature type the WFS can service.

GetFeature: Is used to retrieve feature instances.

A basic WFS would implement the operations mentioned above. But there are more optional operations such as *LockFeature*, *Transaction* etc. (Vretanos, 2005).

There are two methods to encode the WFS requests. One can use XML and HTTP POST protocol (Vretanos, 2005). The alternative is using keyword-value pairs (KVP) to encode the various parameters of a request using HTTP GET.

Figure 3.1 is a simplified protocol diagram illustrating the messages that might be exchanged between a client application and a WFS during typical request and responses.

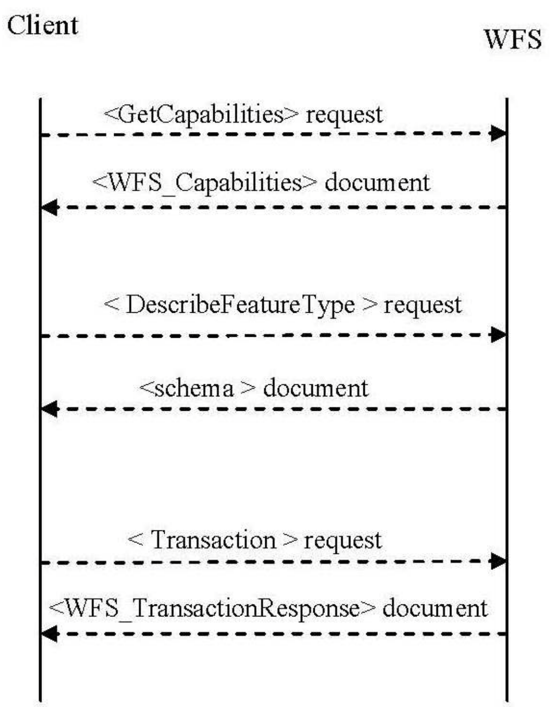

Client
WFS
<GetCapabilities> request
<WFS_Capabilities> document
< DescribeFeatureType > request
<schema > document
< Transaction > request
<WFS_TransactionResponse> document

Figure 3.1 Messages that might be exchanged between a client application and a WFS (Vretanos, 2005).

3.1.2 WPS

WPS defines a standardized interface that facilitates the publishing of geospatial processes, and the discovery of and binding to those processes by clients (Schut, 2007). A WPS may perform any algorithm or calculations or modeling on spatially referenced data. These processes performed by WPS can be as simple as a simple spatial reference system transformation or as complicated as a global climate change model (Schut, 2007). WPS is targeted at processing both vector and raster data.

There are three mandatory operations which all WPS servers should follow (Schut, 2007):

GetCapabilities: By this the user can request and receive back service metadata document (XML) which describes the abilities of the specific server implementation. Each of the processes offered by WPS are included in this document with descriptions about them.

DescribeProcess: Provides the user with detailed information about the processes that can be run on the server. The user can request and receive back a document with the process information including the inputs required, their allowable formats, and the outputs that can be produced.

Execute: This operation allows a client to run a specific process by providing specified inputs and receive back the output.

WPS requests can be encoded using both GET and POST in the same way as WFS request.

3.1.3 GML

Geography Markup Language is an XML grammar written in XML Schema for the description of application schemas as well as the transport and storage of geographic information (Portele, 2007). Data sets containing points, lines and polygons can be defined using GML schema by users and developers. Clients and servers with interfaces that implement the OpenGIS Web Feature Service Interface Standard read and write GML data (Portele, 2007). Many applications use GML as a standard format for the input and output of the system for the purpose of interoperability.

3.2 Semantic description

For semantic interoperability the contents and meanings of the data should be explored in depth. This can be achieved by using ontologies (Yue et al., 2007).

Ontology is a formal, explicit specification of a conceptualization that provides a common vocabulary for a knowledge domain and defines the meaning of the terms and the relations between them (Gruber, 1993).

The World Wide Web Consortium (W3C) recommends Web Ontology Language (OWL) as the standard Web ontology language. Other standards are WSML and RDF (W3C, 2012).

3.3 Web Service Composition techniques

There are different techniques for service composition, from different perspectives. In this section two perspectives including ‘Procedure based’ and ‘Service description based’ are investigated.

3.3.1 Procedure based

Considering procedure management there are two general patterns of Web service composition (Mansourian et al., 2011): *Centralized* and *Cascaded*. In Centralized patterns a central component is responsible for invocation of the services that are being chained. Therefore no information about other services is sent to a specific service. In Cascaded approach each service connects directly to the other service(s) to invoke the required input parameters/processes or to send the output (Mansourian et al., 2011). The final result is presented to the user by the last service. Figure 3.2 demonstrates the difference in architecture of two service chains. One is based on Centralized (a) method and the other on Cascaded method (b).

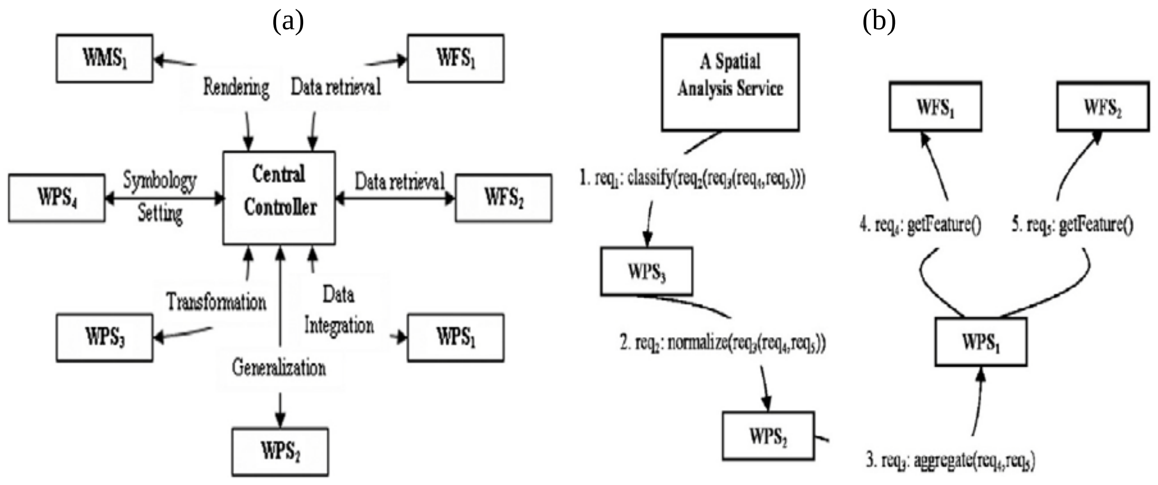


Figure 3.2 Centralized (a) vs. Cascaded (b) patterns in service composition (Mansourian et al., 2011).

3.3.2 Service description based

The other approach for categorizing service composition techniques is by considering service description. Before a service is selected for the chaining process in an automatic or semiautomatic way the service should be described. There are two general ways to describe a service. One ways is using Web Service Definition Language (WSDL). In geospatial community OGC *GetCapabilities* operation can be used which guarantees syntactic interoperability. The other way is to use semantics for describing the services.

Many researchers have based their studies on developing systems that fulfill user's request by chaining different Web services using OGC standards (Stollberg & Zipf, 2007; Friis-Christensen et al., 2009; Kiehle et al., 2007). Some others have brought semantic concepts into chaining process of geospatial Web services (Zhang et al., 2010; Yue et al., 2007; Li et al., 2010). There are also studies bringing both syntax and semantic aspects into considerations (Lemmens et al., 2007).

In general service composition in OGC can be done in a number of ways (Schut, 2007):

1. A BPEL engine can be used to orchestrate a service chain that includes one or more OGC processes (e.g. WPS).

2. A WPS process can be designed to call a sequence of Web services including other WPS processes, thus acting as the service chaining engine.

3. Simple service chains can be encoded as part of the execute query. Such cascading service chains can be executed even via the GET interface.

In this thesis the third approach is used for implementation. Considering the use case in which feature data should be downloaded and loaded into database (chapter 5.2), the third approach seems the best fit. The data needs to be downloaded only once and also it prevents double work. From the procedure based point of view, the Cascaded approach is implemented.

4. Proposed shortest path algorithm and implementation

As highlighted earlier is chapter 2, the shortest path algorithms can be divided into two main categories. For algorithms that rely on network structure, road network topology is required. To create road network topologies, we convert the real road network into an aggregation of nodes and links. Then the shortest path is determined based on this network. However there are some places where the user can make shortcuts. These are called *open spaces* in this thesis. Considering a pedestrian who is traveling between a start point and an end point, he/she might have to pass through some fixed pathways without the option of choosing a shortcut (these are the edges of the network). But there may be some open spaces where the user can skip one or several edges and go directly from a point to another between which there is no edge. The algorithm for finding the shortest path to pass through the open spaces belongs to the second category of the shortest path algorithms (see chapter 2). Algorithms of the first and the second category can be merged to make a global algorithm for routing. Still there are some points which should be considered before designing the algorithm:

The first problem relates to the location of start/end point. We can simply categorize the areas of a city into a network, non-open spaces, and open spaces. The network represents the streets of the city. It consists of edges (roads) and nodes (intersections). The network is a set of lines and points but the open spaces and non-open spaces are polygons. Once a user asks for the shortest path, if the start (or end) point is on the network (on an edge or on a node of the network), then any of the algorithms for network (e.g. Dijkstra) can be used. Note that the algorithms for network find the path from a node to another node so if the start (or end) point is somewhere on an edge (and not exactly on a node) the simplest solution is to guide the user to the nearest node (and for the end point to nearest node to the end) and then finding the shortest path between these two nodes.

If the start (or end) point is not on the network then it is either in a non-open space or in an open space. Being in a non-open space getting the nearest node to start (or end) point could be a solution again. But a major issue arises when the start (or end) point lies in an open space. Here getting the nearest node is not always the best answer. Each open space can be represented by a polygon. This polygon may be convex (Figure 4.1.a, b, c) or concave (Figure 4.1.d). Some obstacles may also exist inside the interior region of the polygon (e.g. buildings in the campus). These are holes or islands of the polygon. The user usually reaches the polygon from one entrance (e.g. a street leading to an open space). In Figure 4.1.a the nearest node to destination point P is "*i*". Suppose the user reaches to open space from point e. Then it is not necessary to walk around the open space and the user can make a shortcut directly from e to P. Figure 4.1.b is the same situation but the existence of two holes prevents the visibility of point P until the user reaches to e´. Figure 4.1.c demonstrates another situation which if the user wants to abide by network links, he/she has to travel much longer and can simply make a shortcut from point e to destination point P. The invisibility of destination point from the entrance point e is not always because of holes and e.g. may be due to a concave polygon (Figure 4.1.d). Here the user should get to node e´ first and then should not continue to node “i” and can directly make a shortcut from e´ to P.

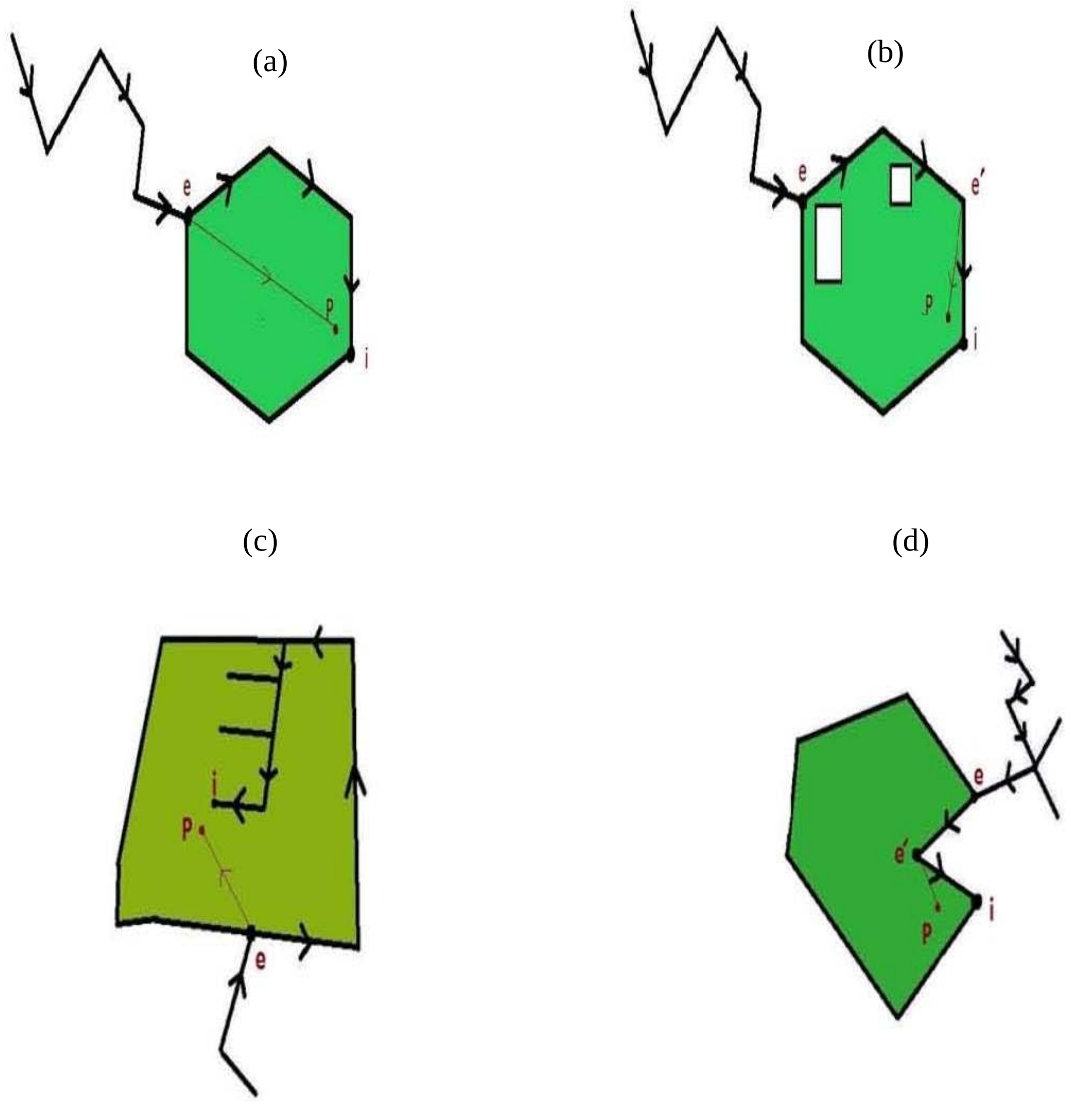


Figure 4.1 Examples of open spaces

Therefore there should be an algorithm for finding the correct node to which the user should be guided. In some cases the first point to which the user should be guided is not a node but can be a vertex of a polygon hole. In Figure 4.1.b the user can start from node e, turn around the big hole and walk through the middle of the polygon toward destination point P.

The second problem is how to tackle the problem of open spaces. To be able to guide the user in open spaces some ‘virtual’ network should be created on the open space. As described in chapter 2, the algorithms for open spaces generally use triangulation of a polygon for creating this virtual network. This triangulation is not the best approach because it is not precise. The algorithm is expected to give the actual shortest path.

In next section the algorithm which presents solutions for the problems mentioned above will be presented.

4.1 Designed shortest path algorithm

Figure 4.2.a shows an area in city of Lund, Sweden. Figure 4.2.b is the network representation of the same area. Streets are edges of the network, intersections are nodes and green polygons are the open spaces. Given a start point S and an end point E, what is the shortest path between S and E? Below is a two phase algorithm, proposed as solution.

Phase one: Shortest path on network

a) Find the nearest node to start point/end point. Let (i) be nearest node to start and (j) the nearest node to end (Figure 4.2.b).

b) Find a shortest path between (i) and (j) using Dijkstra’s algorithm. We regard the path found as the first outline. This outline represents the main direction of our shortest path.

c) Check S and E

If neither the start point nor the end point is in any open space the algorithm is done and the resulted outline is the final shortest path.

Otherwise if any of start point or end point or both are in an open space find A or B or both as described below:

d) Find A (in case S is in an open space) and B (in case E is in an open space).

Where A is: Going from S toward E on the path found on step b (first outline), A is the last node which falls in the same open space as S.

Where B is: Going from E toward S on the path found on step b (first outline), B is the last node which falls in the same open space as E.

e) Find the shortest path between S and A (we denote this as $\overline{SA}$) and/or E and B (we denote this as $\overline{SB}$) based on the algorithm described in Phase two.

f) Based on different conditions, described below, the final shortest path is determined as:

- If start point is in an open space but end point is not: Final shortest path is $\overline{SA}$ and path from A to E based on outline.
- If start point is in an open space and end point is as well: Final shortest path is $\overline{SA}$ and from A to B based on outline and $\overline{SB}$.
- If start point is not in an open space but end point is: Final shortest path is the path found between S to B based on outline and $\overline{SB}$.

Phase two: Shortest path in open space

This part is based on the method proposed by Pan et al. (2007). However slight modifications are applied to make it more practical for the implementation part. Given the start point S and/or end point E which are in open spaces and nodes A and/or B found in Phase one (Figure 4.2.c):

a) Connect the start point S and/or end point E to each vertex of the polygon and each vertex of the holes of the polygon. If the connecting line is completely within the polygon (i.e. it is visible from S/E), then take it as a link, as shown in Figure 4.2.c, otherwise ignore the connecting line.

b) Execute step ‘a’ for the nodes A and/or B.

c) Connect each pair of vertices of the polygon and holes of the polygon. If the connecting line is in the polygon (i.e. pairs have visibility), then take it as a link, as shown in Figure 4.2.c, otherwise ignore the connecting line.

d) Find a shortest path in the resulted road network using Dijkstra’s algorithm between S and A and/or E and B.

Note: Not all pairs of points are connected to each other in Figure 4.2.c to make it more legible.

The Final shortest path after doing two phases above is shown in Figure 4.2.d.

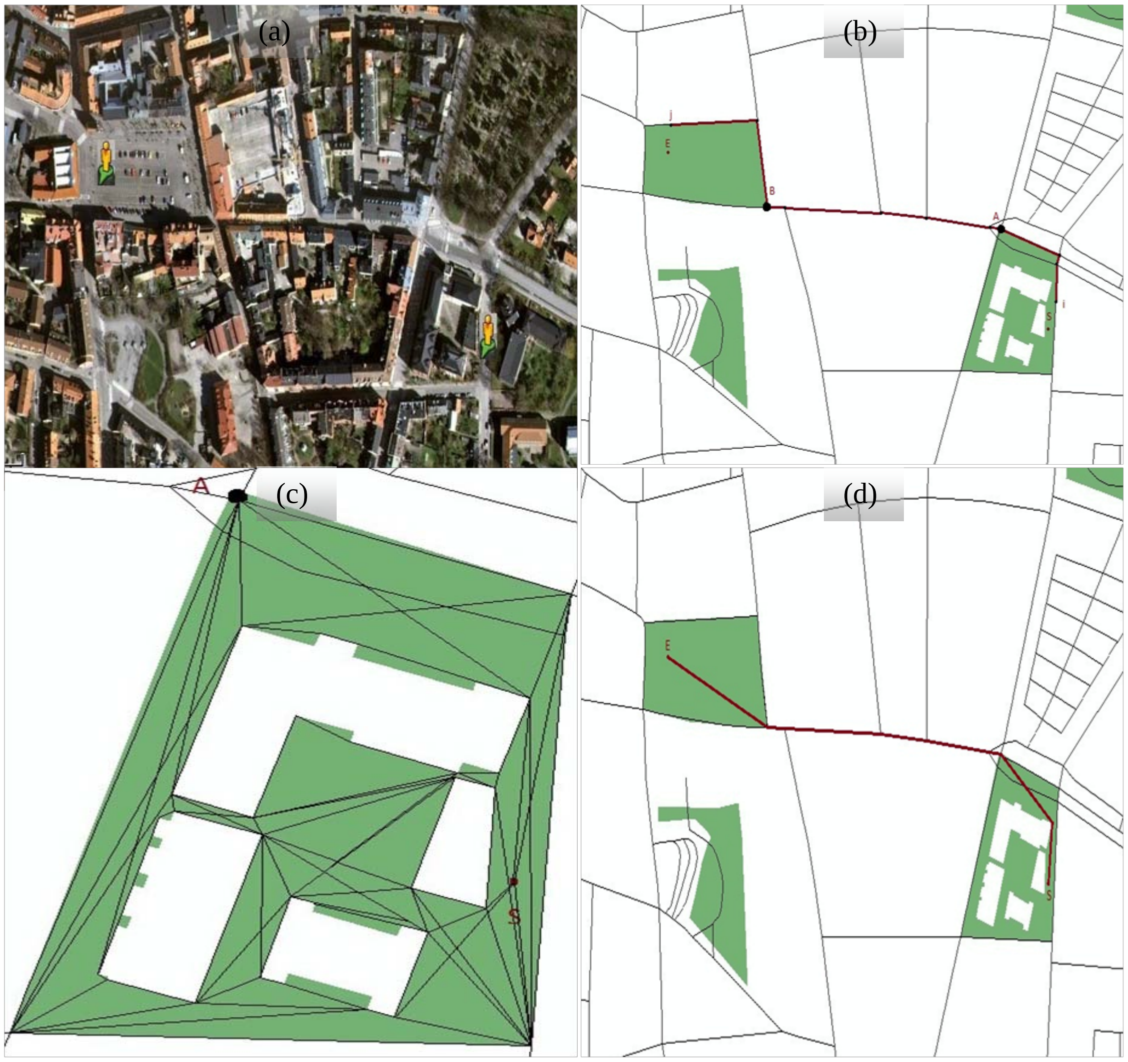


Figure 4.2 (a) Example of start and end point in a city, (b) Network representation of the streets in picture a, (c) Data preparation for algorithm (d) Final shortest path

4.2 Implementation of the algorithm

The algorithm described in previous chapter was implemented using Java programming language (servlets), *PostGIS*, *pgRouting*. *PostGIS* adds support for geographic objects to the PostgreSQL object-relational database. In effect, PostGIS "spatially enables" the PostgreSQL server (PostGIS, 2012). *pgRouting* is another component which extends the PostGIS / PostgreSQL geospatial database to provide geospatial routing functionality (pgRouting, 2012).

Servlets are Java programs that run on a Web server (Liang, 2007). Java servlets are capable of communicating with database and send and receive request. At the same time servlets can receive request from a client application. For this study an HTML client was used which sends the data inputs (coordinates of the start and end point) to the Java program via HTTP GET protocol and waits for the result. The result (XML document) returned from the servlet can be rendered in different interfaces as described later in chapter 5.

The feature data (streets and open spaces) for the city of Lund was downloaded from Open Street Map. This data was then loaded into PostGIS tables using shp2pgsql GUI (PostGIS, 2012). Then some processes were done on street data to make it ready for routing. These processes known as making *Topology* should be done using pgRouting functions. Considering the street data are stored in a table named roads listing 4.1 shows the SQLs which can be executed respectively for making Topology.

```
ALTER TABLE roads ADD COLUMN "source" integer;
ALTER TABLE roads ADD COLUMN "target" integer;
SELECT assign_vertex_id('roads', 0.00001, 'geom', 'gid');
CREATE INDEX source_idx ON roads("source");
CREATE INDEX target_idx ON roads("target");
```

Listing 4.1 SQLs for making Topology in PostGIS using pgRouting functions.

Once the making *Topology* processes is done, the data is ready to make shortest path queries by using pgRouting *shortest_path* function (Listing 4.2).

```
Select * From shortest_path('SELECT gid as id, source::integer, target::integer,
length::double precision as cost FROM roads', 234, 1875, false, false);
```

Listing 4.2 Instance of SQL for finding shortest path using pgRouting *shortest_path* function

The shortest_path function above returns a result-set which is the shortest path from node with ID: 234 to node with ID: 1875 in database using Dijkstra. Yet this function does nothing about issues related to open spaces as described in chapter 4.1. This problem was tackled by coding the proposed algorithm in Java. Wherever Dijkstra's was needed the shortest_path function of pgRouting was used. In other cases java codes were written in a way that all the steps describe in chapter 4.1 were followed for finding the final shortest path.
Some examples of the spatial queries used to fulfill the algorithm proposed in chapter 4.1 are as following:

```
Select * From (Select  case ST_Covers(poly.geom,Point) When false THEN 0 WHEN true
THEN 1 END As results, poly.ID From openareas As poly,
ST_SetSRID(ST_Point("+xStartPoint+", "+yStartPoint+") ,3021) As Point) As temp;
```

Listing 4.3 Instance of SQL for finding shortest path using pgRouting *shortest_path* function

Listing 4.3 shows the SQL which use point-in-polygon relation to check whether the start point falls in an open space or not. If the point falls in the polygon it returns 1 otherwise it returns 0. Note that a geometry is created from the start point coordinates and it is attributed a spatial reference system (the same SRS in which the open-spaces are stored in database which in our case is 3021 denoting RT-90 gon v).

```
Select * From ST_NumInteriorRings((SELECT geom From ST_Dump((Select geom
From openareas  Where id="+TheContainerPolyofStartPoint+"))));
```

Listing 4.4 SQL for querying number of interior rings in a polygon

The nested SQL above returns the number of interior rings in a polygon (open space). The polygon is defined by an ID in database.

```
Select ST_Intersects(ST_SetSRID(ST_MakeLine(ST_MakePoint("+xStartPoint+" ,"+yStartPoint+"),
ST_MakePoint("+X_A+","+Y_A+")),3021),ST_MakePolygon((Select (ST_InteriorRingN((SELECT geom
From ST_Dump((Select geom  From openareas Where id="+TheContainerPolyofStartPoint+"))), "+q+"))))
```

Listing 4.5 SQL for checking intersection of a line and holes of a polygon

The SQL showed in listing 4.5 is like drawing a line between start point and point A (see Figure 4.2.c), and checks whether an interior ring (hole) with a specific id intersects with this line or not. If it does it will be regarded as an interfering interior ring and will be kept for next step (chapter 4.1). In the code only the vertices of the interfering interior rings were used for doing routing as described in chapter 4.1.This will reduce computational complexity to a lot extent.

After the interior interrupting rings (holes) are distinguished the below SQL was used to detect the visible vertices of these holes:

```
Select ST_Covers((Select geom From openareas Where gid="+TheContainerPolyofStartPoint+")
,(Select ST_MakeLine((Select ST_PointN(ST_InteriorRingN((SELECT geom From ST_Dump
((Select geom  from openareas Where id="+TheContainerPolyofStartPoint+"))),
"+ThisInterruptingInteriorRingtoStart+"), "+Vertex num i+")) ,
ST_SetSRID(ST_MakePoint("+xStartPoint+" ,"+yStartPoint+" ),3021) )))
```

Listing 4.6 SQL for knowing the visible vertices of holes of a polygon from a point

This SQL checks whether the line drawn between start point and the vertex of interior ring with ID i, falls completely in the same open space where the start point is. It returns true or false.

Then between each of these nodes and the point A/B Dijkstra's shortest path algorithm was performed. The node which returns the shortest total path (sum of length of path from node of the ring to A/B and the direct distance from node to start/end point) was chosen and shortest path found was the final path given between start/end point and A/B (chapter 4.1).

Note that all the routing process should be performed in projected reference system appropriate for the area in which the user asks for shortest path. This could be a part of data inputs. However the inputs of the system (which are the start and end point coordinates) were in WGS84. Therefore the below SQL was used for SRS conversions during calculations:

```
Select * From ST_X(ST_Transform(ST_SetSRID(ST_Point("+xrt90+", "+yrt90+"), 3021), 4326))
 As a, ST_Y(ST_Transform(ST_SetSRID(ST_Point("+xrt90+", "+yrt90+"), 3021), 4326)) As b
```

Listing 4.7 SQL for converting X,Y coordinates from EPSG: 3021 (RT 90 2.5 gon v) to EPSG: 4326 (WGS 1984) spatial reference systems.

4.3 Algorithm test

The system was run for different cases and the results were evaluated visually. The result showed that the algorithm is successfully implemented and works fine. Two examples of the runs are illustrated in Figure 4.3 on top of Google images. In both cases the path found goes through open spaces. In Figure 4.3.a if the user was to abide by streets he/she would have to turn around the square (dotted lines) while in reality one can easily walk through the open space and make a shortcut. The algorithm perfectly finds the shortest path possible by eliminating unnecessary paths. In Figure 4.3.b both the start and points lie within open areas. The figure shows how the algorithm acts with "open eyes" in open spaces. If the shortest path algorithm was only based on the nearest node in the network, the dotted lines would be replaced with the shortcuts and would make the path much longer and in some cases even confusing for the user to follow.

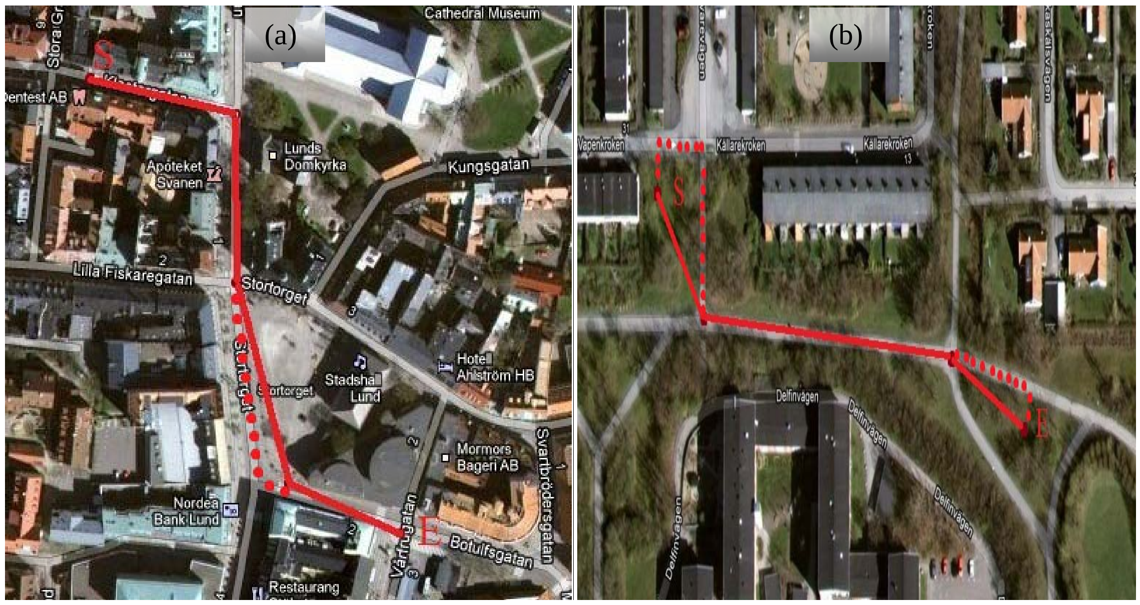


Figure 4.3 Results of running the proposed shortest path algorithm with the system developed for two cases in the city of Lund, Sweden. In both figures S denotes the start point and E denotes the end point. The solid red lines are the shortest path found and the dotted lines show the case if the algorithm would not consider open spaces.

4.4 Discussion

The proposed shortest path algorithm has many advantages over common algorithms. It is designed for pedestrians and therefore is more applicable for some specific purposes such as guiding tourists in a city for sightseeing or for helping people with disabilities like those with visually impaired or elderly people.

Using Dijkstra’s algorithm different weights can be introduced for each edge of the network (streets). In this research only the lengths of the edges were considered for the purpose of implementation to obtain the shortest path regarding the distance. But e.g. if the algorithm is going to be used for guiding tourists in a city, the streets in historical areas can be given more weights so the users have the chance to visit these places while getting to their destination.

In the system developed for implementing the algorithm the data for the city of Lund, Sweden was used. The data covered an area of ca. *4400 ha* and the network representing the city streets and intersections was consisted of ca. *51000* links (edges) and ca. *44000* nodes. The response time of the system, however, was quite low (a couple of seconds for any arbitrary start/end point) which seems quite reasonable.

The algorithm may have open spaces feature data as well as the street data as the inputs. Street data for a city may be available through many resources. Many open source/wiki based data providers on Web are available for downloading streets data of the cities. But there is no general definition of open spaces in a city. Therefore the data for open spaces may not be directly available. The automatic extraction of open spaces is currently difficult (or almost impossible).One approach is to create a data frame and remove all the non-open space areas from this layer manually. With this technique there is always a risk that areas considered as open spaces are not really open spaces on earth. Some yards, parks, squares grasses and other grounds that citizens may walk through, may not be even "legally" walkable. There are many legal, conventional and technical aspects to be taken into account before preparing the data for open spaces.

The system developed for implementing the algorithm can easily be published on the Web and can communicate with other applications via standard interfaces. It can be used in many applications such as smart phones and/or applets. This has many advantages comparing to using global routing systems. Although many companies are providing routing functionalities over the Web now but these services may not be useful in small scales. For example the data for pedestrian paths or local streets are not updated fast enough in their data bases. In many cases having a precise and detailed routing system is highly desired for local sectors and municipalities. In such cases using systems similar to our system is beneficial. The algorithm and implementation of this thesis was being used in a European Union funded project belonging to Haptimap which is an EU project aiming at making maps and location based services more accessible by using several senses like vision, hearing, and, particularly, touch (Haptimap, 2012).

The developed routing system can also be used as a Web service and be implemented in a Web service composition with other services as described in chapter 5.

5. System architecture and implementation

5.1 System architecture for composition process

Figure 5.1 shows the system architecture used for the composition process. The architecture is based on the selected approaches for service composition described in chapter 3. It consists of a Geoportal, several WFSs and several WPSs as shown in Figure 5.1.

Geoportal is the central component which has the responsibility of getting user's request, discovering available Web services for providing data and functionality to fulfill the request, translating the user's request to the format acceptable by Web services and chaining these services in a reasonable order. It has four subcomponents: A user interface, service finder, a composer and a registry.

Each WFS corresponds to street and open spaces data of a city. The WPSs are Web services that perform required processes for finding the shortest path. Differences between WPSs may be because of the algorithms for path finding or the data format they accept.

Once the user asks for the shortest path between two desired points, this request is transferred to Service Finder (Figure 5.1.). The Service Finder looks for candidates which can be chained to fulfill the user's request. OGC services define themselves via *GetCapabilities* operation (Schut, 2007). The service finder sends *GetCapabilities* request to all WFSs and WPSs which are registered within Service Registry. After receiving the capabilities documents it checks whether the WFS contain layers for street and open spaces and if the layers cover the area requested by user (regarding BBOX). Finding a WFS satisfying both these conditions the service finder looks for proper WPS. A WPS which accepts the possible output format of the selected WFS and supports the processes needed for path finding would be chosen. This information of the proper WFS and WPS are sent to the composer. The composer generates *Execute* operation request (Schut, 2007) according to acceptable format by WPS and sends this to the selected WPS. This request is a nested request which contains the location for accessing feature-level data (chapter 5).

Once the WPS receives the request different components are distinguished from that. First a URL to the Web address where the street data can be accessed. Second a URL to open spaces. Third the coordinates for start point and end point. The feature data will be downloaded from the selected WFS (via GetFeature request) and will be processed furtherer and made ready for routing. Once the WPS finishes the process the result (shortest path between input coordinates) is returned to the composer and from there to the user interface to be rendered. The whole processes mentioned above are performed automatically.

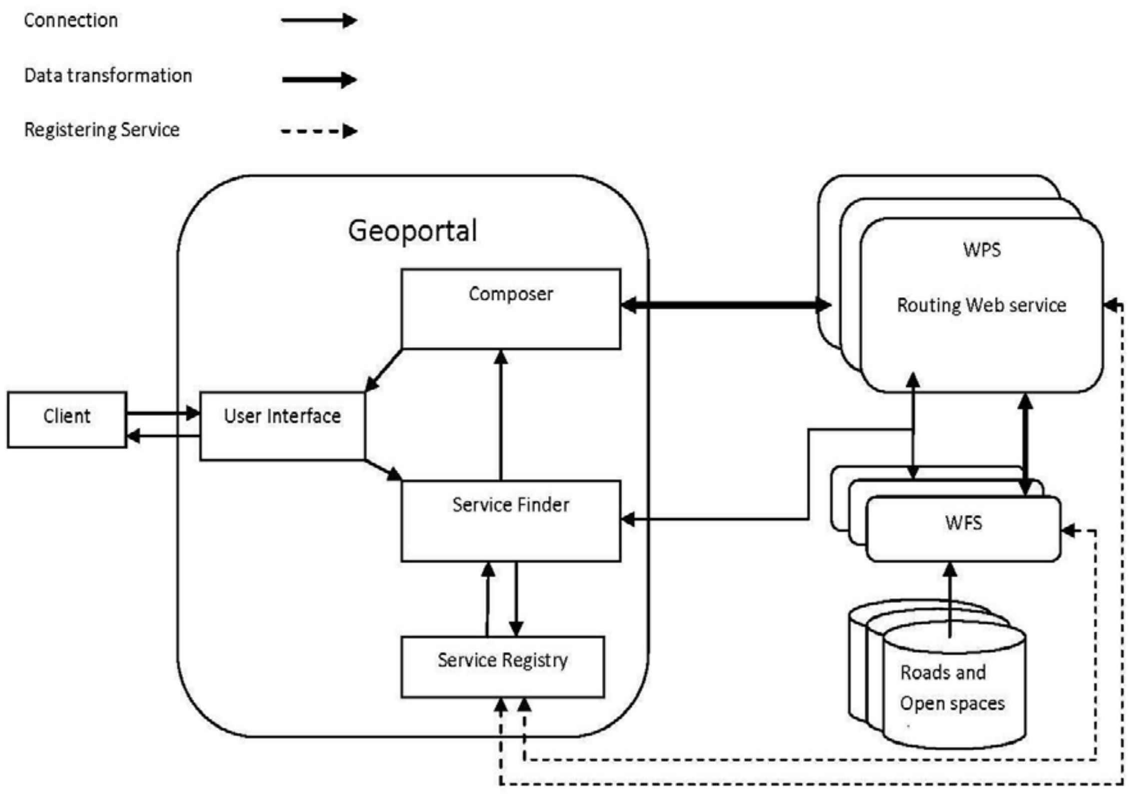


Figure 5.1 System architecture

5.2 System implementation

A prototype system was implemented based on the designed architecture using software and tools shown in Figure 5.2.

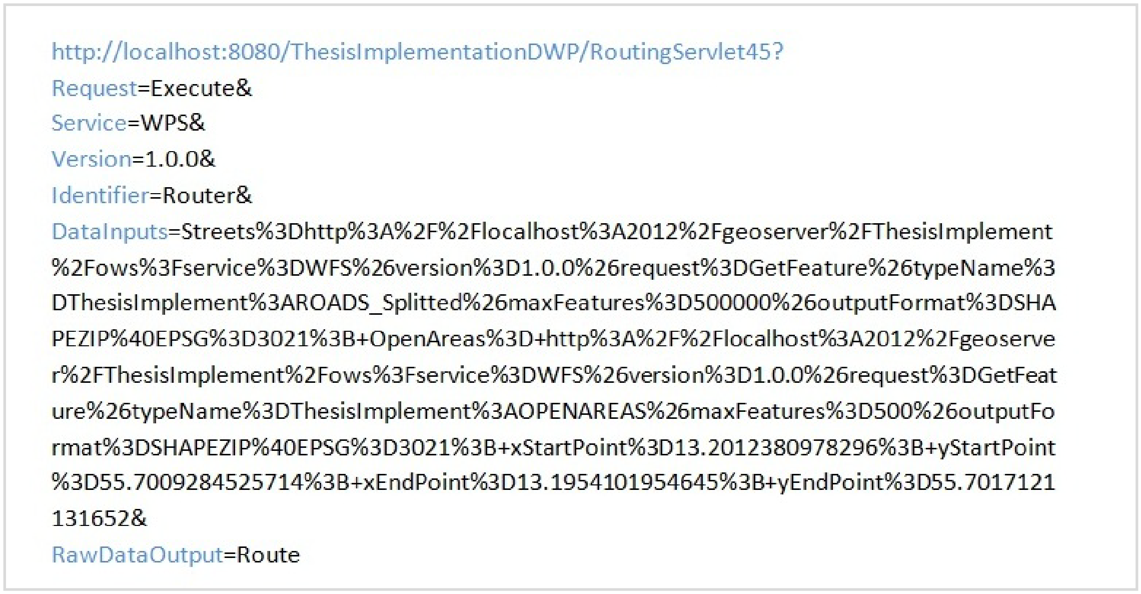
A Geoportal was developed by using Java programming language. User interface was an HTML client which sends the user request to the java program via HTTP GET protocol. At the end it is capable of receiving result from composer and rendering it on top of Google layers using java scripts and Google API. Other components of the Geoportal were programmed in Java.

For publishing the data two WFSs were considered. ArcGIS Server 10 (ArcGIS Server, 2012) which supports OGC WFS was used to publish street and open spaces data for the city of Malmo, Sweden. The ArcGIS Server is capable of giving feature data in various formats. The most famous one is GML. Another prototype WFS was created by using GeoServer (Geoserver, 2012). Streets and open spaces data for the city of Lund, Sweden were published with WFS support via GeoServer. Geoserver gives out feature data in various formats including GML and ESRI Shape files.

Two prototype WPSs were developed using *Java servlets*, *PostGIS*, *pgRouting*. One WPS is capable of receiving feature data in GML and the other in ESRI Shape files formats.

The WPS *Execute* request (when using GET method) should be encoded before being sent to the server. Inputs can be both features and scalars (Schut, 2007). The feature inputs are in fact addresses to the location where features can be accessed. In this case the features (streets and open spaces) can be reached from WFS (Geoserver or ArcGIS Server). Therefore the WFS *GetFeature* request will be passed as feature input for the WPS. In our system the routing request which is sent to the WPS has six inputs, two features (streets and open spaces data sets) and four scalars (latitudes and longitudes for start and end points).

Listing 5.1 shows a KVP encoded URL which is a sample WPS *Execute* request generated by composer.



```
http://localhost:8080/ThesisImplementationDWP/RoutingServlet45?
Request=Execute&
Service=WPS&
Version=1.0.0&
Identifier=Router&
DataInputs=Streets%3Dhttp%3A%2F%2Flocalhost%3A2012%2Fgeoserver%2FThesisImplement
%2Fows%3Fservice%3DWFS%26version%3D1.0.0%26request%3DGetFeature%26typeName%3
DThesisImplement%3AROADS_Splitted%26maxFeatures%3D500000%26outputFormat%3DSHA
PEZIP%40EPSG%3D3021%3B+OpenAreas%3D+http%3A%2F%2Flocalhost%3A2012%2Fgeoserve
r%2FThesisImplement%2Fows%3Fservice%3DWFS%26version%3D1.0.0%26request%3DGetFeat
ure%26typeName%3DThesisImplement%3AOPENAREAS%26maxFeatures%3D500%26outputFo
rmat%3DSHAPEZIP%40EPSG%3D3021%3B+xStartPoint%3D13.2012380978296%3B+yStartPoint
%3D55.7009284525714%3B+xEndPoint%3D13.1954101954645%3B+yEndPoint%3D55.7017121
131652&
RawDataOutput=Route
```

Listing 5.1 Sample WPS *Execute* request

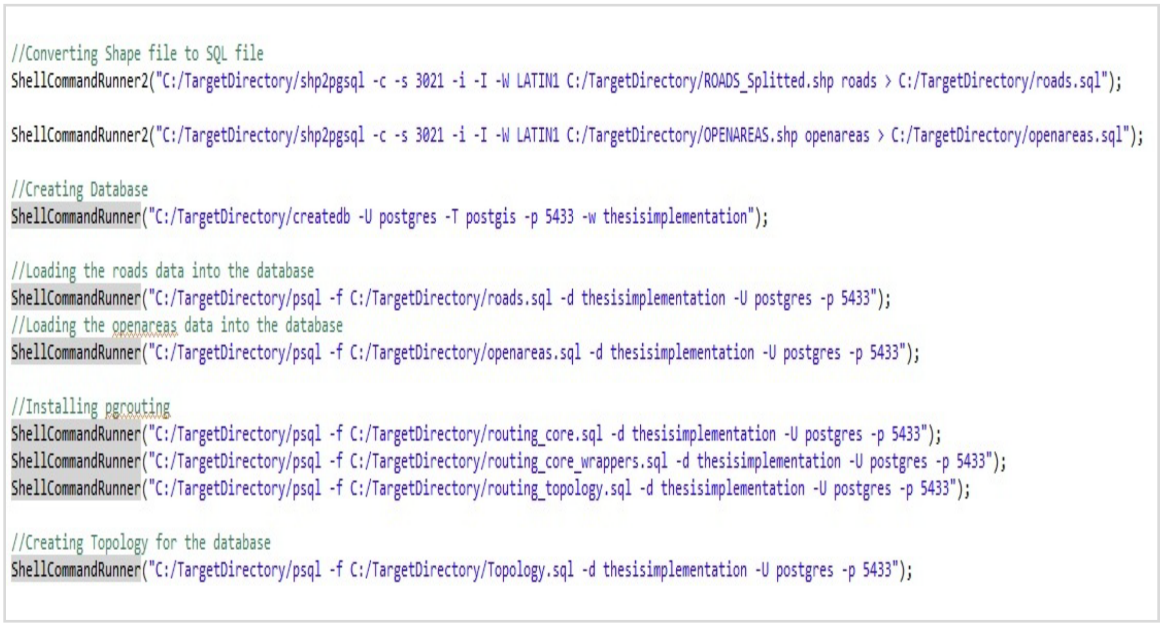
Once the WPS receives the above request, it distinguishes the DataInputs argument. The java URLEncoder class was used to decode the value of the argument. In this case the decoded value of the DataInputs is shown in Listing 5.2.

```
Streets=http://localhost:2012/geoserver/ThesisImplement/ows?service=WFS&version=1.0.0&r
equest=GetFeature&typeName=ThesisImplement:ROADS_Splitted&maxFeatures=500000&out
putFormat=SHAPE-ZIP@EPSG=3021;
OpenAreas=http://localhost:2012/geoserver/ThesisImplement/ows?service=WFS&version=1.0.
0&request=GetFeature&typeName=ThesisImplement:OPENAREAS&maxFeatures=500&outputF
ormat=SHAPE-ZIP@EPSG=3021;
xStartPoint=13.2012380978296;
yStartPoint=55.7009284525714;
xEndPoint=13.1954101954645;
yEndPoint=55.7017121131652
```

Listing 5.2 DataInputs for WPS *Execute* request

After the feature data is downloaded from the WFS it is stored as files (Shape files or GML) on the local directory of WPS's server. The Geoserver generates a SHAPE.ZIP file which should be unzipped first. It may also give GML according to the request. The ArcGIS server gives GML. The next step is loading these data into the PostGIS tables.

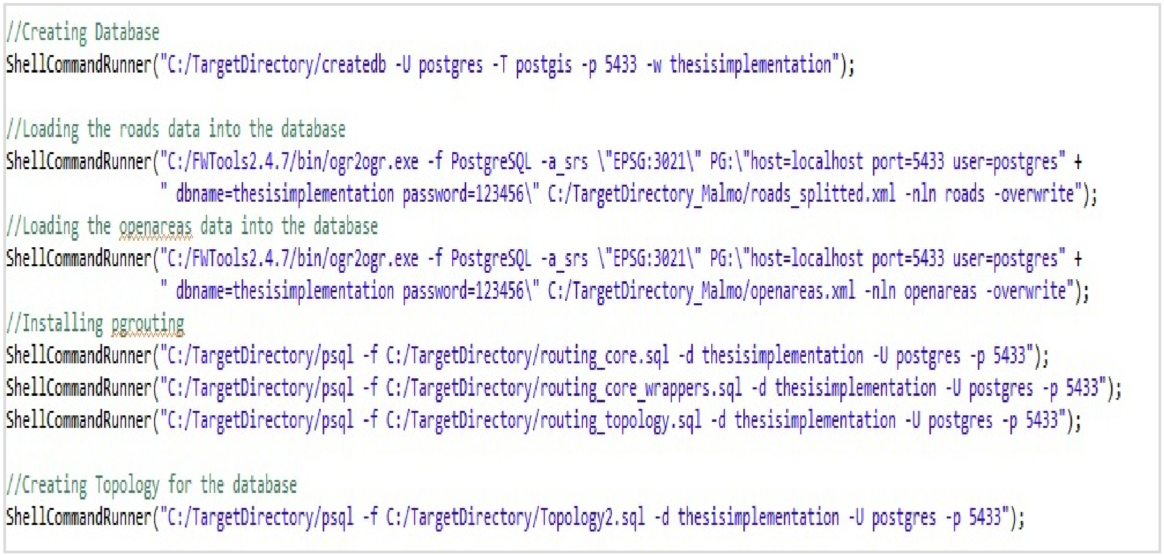
If the data is shape files the ZIP files are first unzipped by using Java. The *shp2pgsql* GUI can no longer be used since everything should be done automatically. Instead the command line version of the *shp2pgsql* together with Java were used to make the process automatic for loading the shape files into the data base. Listing 5.3 shows the Java codes for doing this. The ShellCommandRunner method sends the input strings to *Windows Command Prompt* for execution. The *shp2pgsql* converts shape files to SQL files. Then the *creatdb* was used which is a command line executable program using for creating database in PostgreSQL. For loading the SQL file into the database the *psql* command line program was used. *psql* is another command line program for running SQL files. The pgRouting is then installed using installing SQL files. At last the SQLs for making Topology are executed and the data is ready for routing (Listing 5.3).



```
//Converting Shape file to SQL file
ShellCommandRunner2("C:/TargetDirectory/shp2pgsql -c -s 3021 -i -I -W LATIN1 C:/TargetDirectory/ROADS_Splitted.shp roads > C:/TargetDirectory/roads.sql");

ShellCommandRunner2("C:/TargetDirectory/shp2pgsql -c -s 3021 -i -I -W LATIN1 C:/TargetDirectory/OPENAREAS.shp openareas > C:/TargetDirectory/openareas.sql");

//Creating Database
ShellCommandRunner("C:/TargetDirectory/createdb -U postgres -T postgis -p 5433 -w thesisimplementation");

//Loading the roads data into the database
ShellCommandRunner("C:/TargetDirectory/psql -f C:/TargetDirectory/roads.sql -d thesisimplementation -U postgres -p 5433");
//Loading the openareas data into the database
ShellCommandRunner("C:/TargetDirectory/psql -f C:/TargetDirectory/openareas.sql -d thesisimplementation -U postgres -p 5433");

//Installing pgrouting
ShellCommandRunner("C:/TargetDirectory/psql -f C:/TargetDirectory/routing_core.sql -d thesisimplementation -U postgres -p 5433");
ShellCommandRunner("C:/TargetDirectory/psql -f C:/TargetDirectory/routing_core_wrappers.sql -d thesisimplementation -U postgres -p 5433");
ShellCommandRunner("C:/TargetDirectory/psql -f C:/TargetDirectory/routing_topology.sql -d thesisimplementation -U postgres -p 5433");

//Creating Topology for the database
ShellCommandRunner("C:/TargetDirectory/psql -f C:/TargetDirectory/Topology.sql -d thesisimplementation -U postgres -p 5433");
```

Listing 5.3 Java codes sending command lines to windows command prompt for loading the data into the database

If the data is GML, *ogr2ogr* was used to make the process of loading the data into database automatic. Listing 5.4 shows how the ShellCommandRunner Java method sends command lines to windows command prompt for invoking *ogr2ogr*.



```
//Creating Database
ShellCommandRunner("C:/TargetDirectory/createdb -U postgres -T postgis -p 5433 -w thesisimplementation");

//Loading the roads data into the database
ShellCommandRunner("C:/FWTools2.4.7/bin/ogr2ogr.exe -f PostgreSQL -a_srs \"EPSG:3021\" PG:\"host=localhost port=5433 user=postgres" +
                   " dbname=thesisimplementation password=123456\" C:/TargetDirectory_Malmo/roads_splitted.xml -nln roads -overwrite");
//Loading the openareas data into the database
ShellCommandRunner("C:/FWTools2.4.7/bin/ogr2ogr.exe -f PostgreSQL -a_srs \"EPSG:3021\" PG:\"host=localhost port=5433 user=postgres" +
                   " dbname=thesisimplementation password=123456\" C:/TargetDirectory_Malmo/openareas.xml -nln openareas -overwrite");
//Installing pgrouting
ShellCommandRunner("C:/TargetDirectory/psql -f C:/TargetDirectory/routing_core.sql -d thesisimplementation -U postgres -p 5433");
ShellCommandRunner("C:/TargetDirectory/psql -f C:/TargetDirectory/routing_core_wrappers.sql -d thesisimplementation -U postgres -p 5433");
ShellCommandRunner("C:/TargetDirectory/psql -f C:/TargetDirectory/routing_topology.sql -d thesisimplementation -U postgres -p 5433");

//Creating Topology for the database
ShellCommandRunner("C:/TargetDirectory/psql -f C:/TargetDirectory/Topology2.sql -d thesisimplementation -U postgres -p 5433");
```

Listing 5.4 Java codes sending command lines to windows command prompt for loading the data into the database

The WPSs use the algorithm and implementation described in chapter 4 for finding the final shortest path. The result is in form of GML and at the end will be passed to composer and from there to UI for rendering.

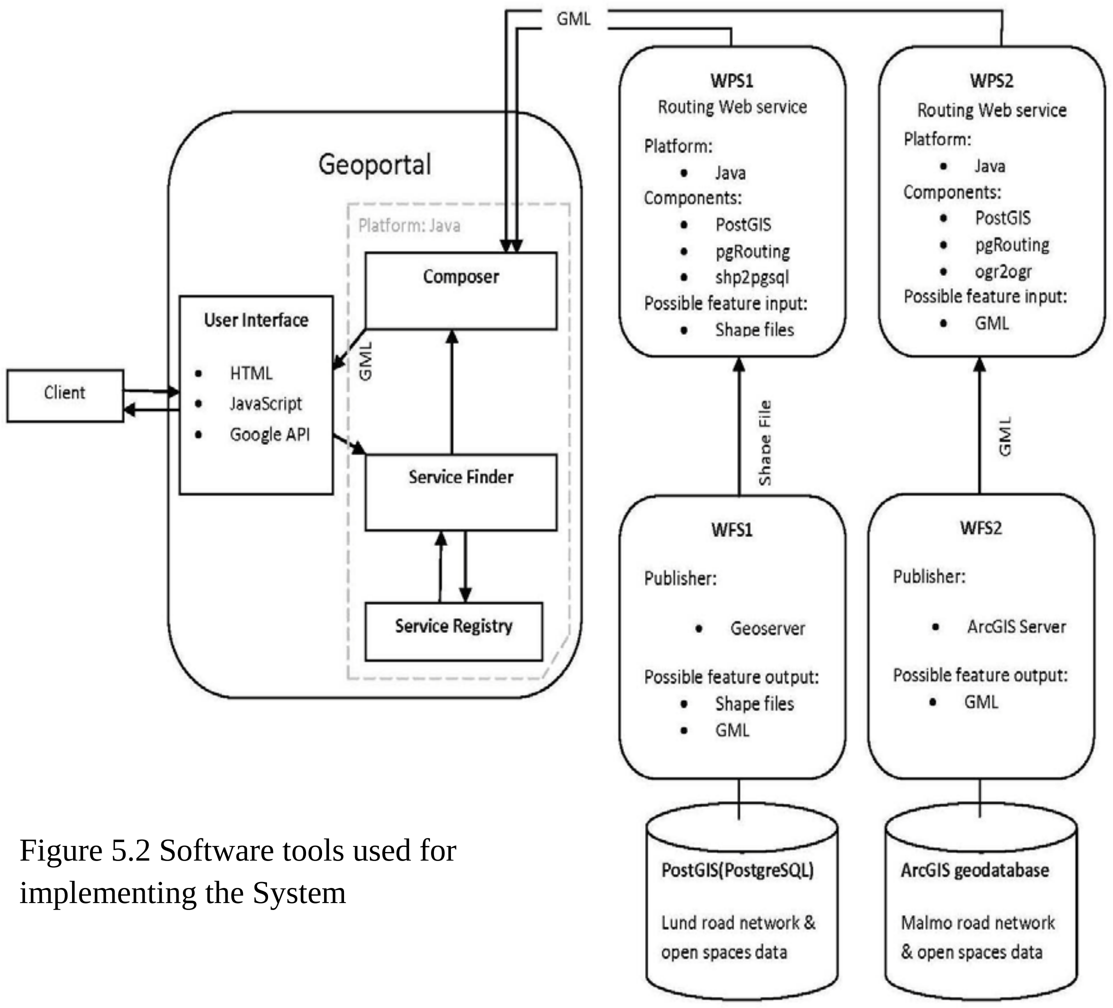


Figure 5.2 Software tools used for implementing the System

Figure 5.3 is a flowchart demonstrating the workflow of the system. As shown in Figure 5.3, depending on the area on which the user asks for the shortest path, several WFSs and WPSs might be found. E.g. if a WFS containing feature data needed for routing is found and it gives only GML as output (like ArcGIS server) the system looks for a proper WPS which accepts GML. The same process happens for output of Shape files. But what if the WFS is capable of giving both GML and Shape files (like Geoserver)? In this case the system algorithm was designed to prefer Shape files. GMLs are text based files and for this they take more space in memory and consume more time for loading while Shape files are binaries which are considerably smaller in size for the same area.

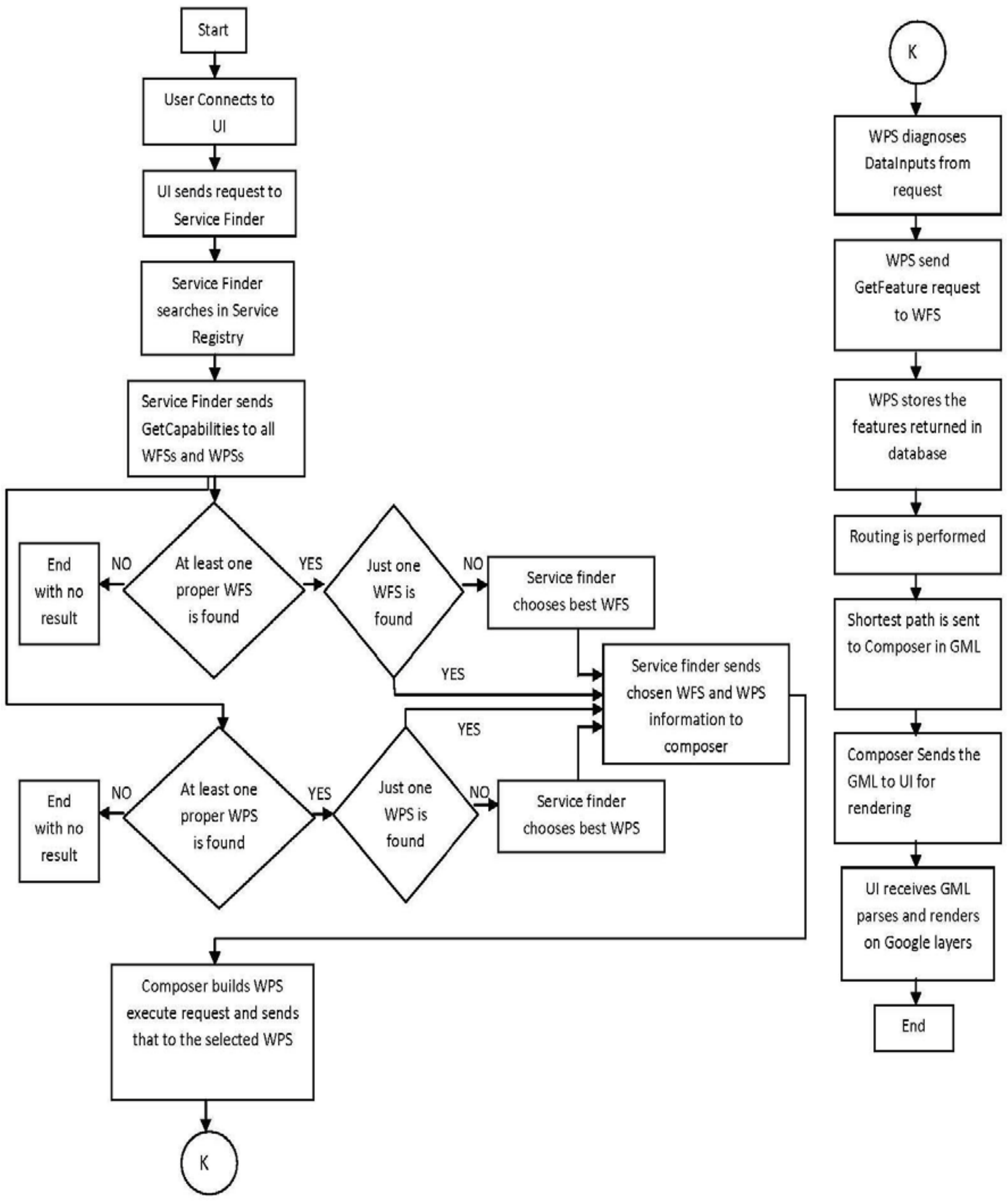

Start
User Connects to UI
UI sends request to Service Finder
Service Finder searches in Service Registry
Service Finder sends GetCapabilities to all WFSs and WPSs
At least one proper WFS is found
NO
End with no result
YES
Just one WFS is found
NO
Service finder chooses best WFS
YES
Service finder sends chosen WFS and WPS information to composer
At least one proper WPS is found
NO
End with no result
YES
Just one WPS is found
YES
NO
Service finder chooses best WPS
Composer builds WPS execute request and sends that to the selected WPS
K
K
WPS diagnoses DataInputs from request
WPS send GetFeature request to WFS
WPS stores the features returned in database
Routing is performed
Shortest path is sent to Composer in GML
Composer Sends the GML to UI for rendering
UI receives GML parses and renders on Google layers
End

Figure 5.3 System work flow

5.3 Case study

The system was tested for two possible scenarios:

A) The user asks for the shortest path between following coordinates (WGS84):

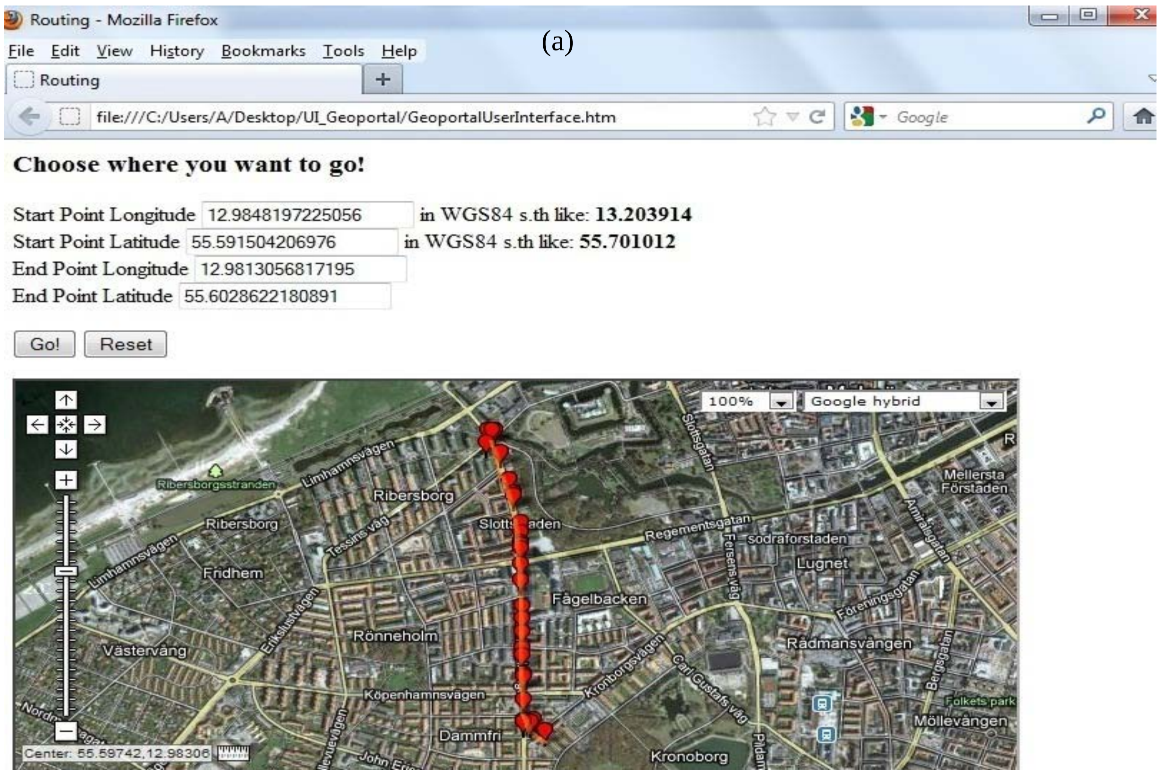
Start Point Latitude: 55.591504206976
Start Point Longitude: 12.9848197225056
End Point Latitude: 55.6028622180891
End Point Longitude: 12.9813056817195

This set of coordinates fall within city of Malmo. The ArcGIS server which contains the data for city of Malmo should be chosen as the proper WFS. It should then be chained to the WPS which accepts GML which is the possible output format of ArcGIS Server. This scenario was tested successfully and the result is shown in Figure 5.4.a.

B) The user wants to travel between (coordinates in WGS84):

Start Point Latitude: 55.7009284525714
Start Point Longitude: 13.2012380978296
End Point Latitude: 55.7017121131652
End Point Longitude: 13.1954101954645

The above boundary is located within municipality of Lund. So, the selected WFS should be Geoserver which contains the feature data for city of Lund. Geoserver can generate GML and Shape files as output. Therefore, two WPSs will be chosen as candidates.
Shape files should be preferred and the WPS which accepts Shape files should be selected for the second part of the chain. This test returned successful result as well (Figure 5.4.b).

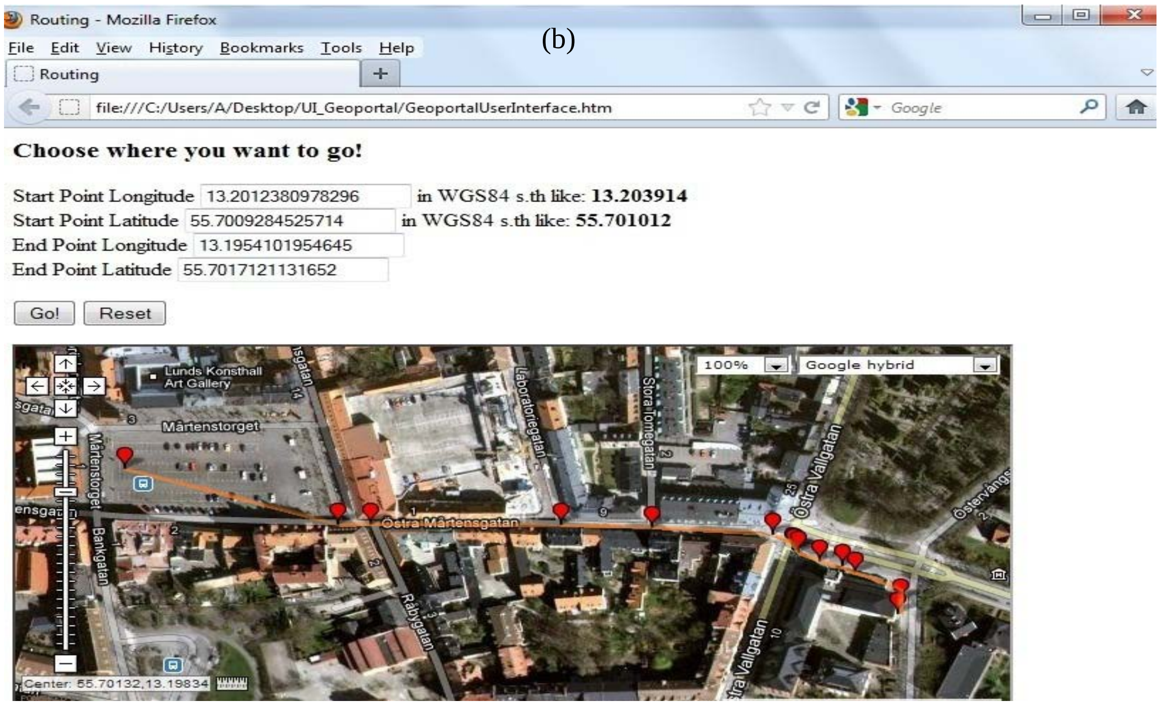

(a)

Routing - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Routing
file:///C:/Users/A/Desktop/UI_Geoportal/GeoportalUserInterface.htm
Google
100%
Google hybrid
Ribersborg
Fridhem
Västervång
Rönneholm
Fågelbacken
Lugnet
Rådmansvången
Möllevången
Kronoborg
Dammfri
Center: 55.59742,12.98306

Choose where you want to go!

Start Point Longitude 12.9848197225056 in WGS84 s.th like: **13.203914**
Start Point Latitude 55.591504206976 in WGS84 s.th like: **55.701012**
End Point Longitude 12.9813056817195
End Point Latitude 55.6028622180891

Go! Reset


(b)

Routing - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Routing
file:///C:/Users/A/Desktop/UI_Geoportal/GeoportalUserInterface.htm
Google
100%
Google hybrid
Lunds Konsthall Art Gallery
Mårtenstorget
Östra Mårtensgatan
Laboratoriegatan
Stora Tomegatan
Östra Vallgatan
Bankgatan
Center: 55.70132,13.19834

Choose where you want to go!

Start Point Longitude 13.2012380978296 in WGS84 s.th like: **13.203914**
Start Point Latitude 55.7009284525714 in WGS84 s.th like: **55.701012**
End Point Longitude 13.1954101954645
End Point Latitude 55.7017121131652

Go! Reset

Figure 5.4 Running the system with two different scenarios (a) For set of coordinates in city of Malmo, (b) For set of coordinates in city of Lund.

5.4 Discussion

The system developed is a representation of automatic Web service composition using OGC standards and interfaces for routing. Although the primary concern is finding the shortest-path the design, philosophy and architecture are general enough to be applicable to the broader community of geospatial Web service composition.

In an automatic service composition all the processes should be done without human interference. The user has no control over what is happening inside the system and everything should be done automatically. A simple problem at some point of the chain can ruin the whole processes. For example for any unexpected reason if the Windows operating system does not accept the command: *”C:/FWTools2.4.7/bin/ogr2ogr.exe”* the data is not loaded into the database and the chain stops working and no matter how well other steps are accomplished, the user gets no result. Although it has been tried to make standard interfaces for programs in computer technology but many applications are still platform dependent. For example, in executable programs such as *ogr2ogr* or *shp2pgsql* the commands for making the same request may be different depending on the operating system of the machine on which the system is working.

For very long distances the system may not be efficient enough regarding the time of response. For queries within common extents of a city (Lund or Malmo in Sweden) the response time of the system is quite reasonable (couple of seconds). But if the user asks for the shortest path from Stockholm to Copenhagen even if the required feature data could be found from a data server, the response time of the system may rise considerably. Also if the features are available as GML this is even worse. The computer used for the purpose of implementing the system in this study was a PC with common configurations up to date. Downloading GML from any of ArcGIS Server or Geoserver WFSs would take much time for a city. Both GML and SQL are text based formats and therefore need more time for loadings or reading/writings and this makes the system not appropriate for queries over long distances. Running the system on server machines can ameliorate the pain and of course in future computers/servers with higher configurations can reduce the computation time.

6. Conclusion and future directions

The purpose of this study was firstly presenting an algorithm for planning path for pedestrians to guide users in a city and secondly developing a system for an automatic Web service composition for routing.

To fulfill the first aim a shortest path algorithm with a special emphasis on open spaces in cities was developed. The algorithm is unique in the way that it takes open spaces data as well as the streets data as inputs and gives the shortest path considering open spaces and shortcuts which can be made through them. Chapter 4 describes the proposed shortest path algorithm and the system which was developed for implementation. The algorithm and the system were tested and showed successful results.

The second aim was achieved by developing a system capable of automatic composition of Web services for routing. Two WFSs and two WPSs were developed and considered as data and processing services, respectively. A geoportal has also been developed to manage the composition processes. The system was run for different scenarios and the results were successfully achieved.

This work demonstrates that using service composition for designing navigation systems is much more beneficial comparing to Web-GISs. The designed system based on service composition and DGIS is more applicable and is not limited to data and functionality of the Web-GIS. Such systems can use all the resources which are available on Web and hence can be much more useful for the users.

In the system described in section 5, the user has no control over the composition process. For instance, when the user asks for the shortest path, the system looks for available data needed for routing on the Web. In this way the quality of the data which is found by the system is not guaranteed. For enhancing the system proposed in this thesis, a component which checks the data quality is recommended. Likewise, during the system run, there could be some points where the system functionality can be expanded regarding e.g. the response time, accuracy, etc. The prioritizing of using Shapefiles over GML files for the WPS input which was described in chapter 5.2 is an example of such situations. One solution could be giving more background information to the user by, for example, providing options in the user interface for having more control on the system work flow.

The system presented in this study is working based on syntactic interoperability of Web services. The services which were contributed in composition processes could define themselves syntactically and it was supposed that there are no semantic heterogeneities in the data and the analysis. However in reality semantic heterogeneity exists between different data services as well as processing services. For example, there may be two processing services, both for routing, but one of them is designed for vehicle's routing while the other only supports pedestrians path finding algorithms. The proper selection of service for each use case can be achieved by semantic description of the services. Or for example there might be two data services both publishing a 'Main Road' layer. However the definition of main road might be different between different data publishers. It could refer to highways or simply city main roads. A semantic description of the data can solve this problem. Future studies are encouraged to consider semantic description of data and services in their work. For example the automatic extraction of open space data which was discussed in chapter 4 can be achieved by defining semantics for proper data and services. Although there have been many researches about using semantics in service composition in last years, but it is still an ongoing research area in geospatial community. The proposed system in this research can be enriched semantically and in this way gain more functionality and also cope with future of Web technology.

References

ArcGIS Server. (2012). ArcGIS for Server, Overview <http://www.esri.com/software/arcgis/arcgisserver/index.html> Accessed 05.12.

Avraam, M. (2010). Geoweb, web mapping and web GIS <http://michalisavraam.org/2009/03/geoweb-web-mapping-and-web-gis/>Accessed 05.12.

Beaujardiere, J. (2006). OpenGIS_ Web map server implementation specification. Open Geospatial Consortium Inc. <http://www.opengeospatial.org/standards/wms>Accessed 03.12.

Dengfeng, Ch.,& Dengrong, Zh. (2001). Algorithm and its application ofN shortest paths problem.International Conferences onInfo-tech and Info-net, 2001. Proceedings. ICII 2001 - Beijing.

Floyd, R.W. (1962).Algorithm 97: Shortest path. Communications of the ACM, Volume 5 Issue 6, June 1962, NY, USA.

Foote, K.E., & Lynch, M. (2000). The Geographer's Craft Project, Department of Geography, The University of Colorado at Boulder.<http://www.colorado.edu/geography/gcraft/notes/intro/intro.html>Accessed05. 2012.

Friis-Christensen, A., Lucchi, R., Lutz, M., & Ostländer, N. (2009): Service chaining architectures for applications implementing distributed geographic information processing. International Journal of Geographical Information Science, 23:5,pp 561-580.

Garcia, M., Lenkiewicz, P., Freire, M.M., & Monteiro, P.P. (2007). On the performance of shortest path routing algorithms for modeling and simulation of static source routed networks – an extension to the Dijkstra algorithm.Second International Conference on Systems and Networks Communications (ICSNC 2007), icsnc, pp.60.

Geoserver. (2012). Geoserver Home Page. <http://geoserver.org/display/GEOS/Welcome> Accessed 05.12.

Gruber, T.R. (1993). A translation approach to portable ontology specification. Knowledge Acquisition 5 (2), 199–220.

Guibas, L., Hershberger, J., Leven, D., Sharir, M.,& Tarjan, R. (1986). Linear time algorithms for visibility and shortest path problems inside simplepolygons. Proceedings of the 2nd ACM Symposium on Computational Geometry, pp.1-13.

Haptimap. (2012). What is HaptiMap? <http://www.haptimap.org/developercompetition.html> Accessed 05.12.

Harrie, L., (2010). Lecture Notes in GIS Algorithms.GIS Centre and Department of Earth and Ecosystem Sciences, Lund University.

Hart, P.E., Nilsson, N.J.,& Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4 4 (2), 100–107.

Kiehle, Ch., Heier, Ch., &Greve, K. (2007).Requirements for Next Generation SpatialData Infrastructures-Standardized Web Based Geoprocessing and Web Service Orchestration.Transactions in GIS , 11(6), 819–834.

Lee, D.T., &Preparata, F. (1984).Euclidean shortest paths in the presence of rectilinear barriers. Networks, 14, pp.393-410.

Lemmens, R., Wytzisk, A., Granell, C., Gould M., &Oosterom, P. (2007). Enhancing Geo-Service Chaining through Deep Service Descriptions. Transactions in GIS, 11(6): 849–871.

Li, W., Yang Ch., & Yang Cho. (2010). An active crawler for discovering geospatial Web services and their distribution pattern – A case study of OGC Web Map Service. International Journal of Geographical Information Science, 24(8), 1127-1147.

Liang, Y.D. (2007).Introduction to Java programming: comprehensive version / 6th ed.Pearson Education, Inc.

Mansourian, A., Omidi E., Toomanian A.,& Harrie L. (2011). Expert system to enhance the functionality of clearinghouse services. Computers, Environment and Urban Systems 35, 159–172.

OGC. (2012). About OGC <http://www.opengeospatial.org/ogc> Accessed 05.12.

Pan, Zh., Yan, L., Winstanley, A.C., Fotheringham, A.S., &Zheng, J. (2009). A 2-D ESPO Algorithm and Its Application In Pedestrian Path Planning Considering Human Behavior .Third International Conference on Multimedia and Ubiquitous Engineering. 2009.Qingdao.

Percivall, G. (Ed.). (2002). The OpenGIS abstract specification, topic 12: OpenGIS service architecture. Version 4.3. OGC 02- 112. Open Geospatial Consortium, Inc., 78pp.

pgRouting.(2012). pgRouting Project <http://www.pgrouting.org/> Accessed 05.12.

Portele, Cl. (2007). OpenGIS Geography Markup Language (GML). Open Geospatial Consortium Inc. Reference number: OGC 07-036. <http://www.opengeospatial.org/standards/gml> Accessed 05.12.

PostGIS. (2012).What is PostGIS? <http://postgis.refractions.net/> Accessed 05.12.

Reif, J., &Storer, J. (1985). Shortest paths in Euclidean space with polyhedral obstacles. Technical Report CS-85-121, Computer Science Department, Brandeis University. Submitted to Journal of the ACM.

Schut, P. (2007). OpenGIS Web processing service. Open Geospatial Consortium Inc. Reference number: OGC 05-007r7. <http://www.opengeospatial.org/standards/wps> Accessed 05.12.

Stollberg, B., &Zipf, A. (2007).OGC Web Processing Service Interface for Web Service Orchestration. Aggregating geo-processing services in a bomb threat scenario.Institute for Spatial Information and Surveying Technology.

Vretanos, P.A. (2005). Web feature service implementation specification. Open Geospatial Consortium (OGC), OGC 04-094. <http://www.opengeospatial.org/standards/wfs> Accessed 05.12.

W3C:World Wide Web Consortium.(2012a). OWL Web Ontology Language Overview <http://www.w3.org/TR/owl-features/>. Accessed 05.12.

W3C:World Wide Web Consortium.(2012b). Resource Description Framework (RDF) <http://www.w3.org/RDF/>. Accessed 05.12.

W3C:World Wide Web Consortium.(2012c). Web Service Modeling Language (WSML) <http://www.w3.org/Submission/WSML/>. Accessed 05.12.

Whiteside, A., Evans, D.J. (2008). Web coverage service implementation standard. Open Geospatial Consortium (OGC), OGC 07-067r5, 2008. <http://www.opengeospatial.org/standards/wcs> Accessed 05.12.

Yue, P., Di, L., Yang, W.,Yu, G., &Zhao, P. (2007). Semantics-based automatic composition of geospatial Web service chains. Computers & Geosciences, 33, 649–665.

Zhang, Ch., Zhaob, T., Li, W., &Osleeb, J.P. (2010). Towards logic-based geospatial feature discovery and integration using Web feature service and geospatial semantic Web. International Journal of Geographical Information Science, 24(6), 903–923.

Institutionen av naturgeografi och ekosystemvetenskap, Lunds Universitet.

Student examensarbete (Seminarieuppsatser). Uppsatserna finns tillgängliga på institutionens geobibliotek, Sölvegatan 12, 223 62 LUND. Serien startade 1985. Hela listan och själva uppsatserna är även tillgängliga på LUP student papers (www.nateko.lu.se/masterthesis) och via Geobiblioteket (www.geobib.lu.se)

The student thesis reports are available at the Geo-Library, Department of Physical Geography, University of Lund, Sölvegatan 12, S-223 62 Lund, Sweden.
Report series started 1985. The complete list and electronic versions are also electronic available at the LUP student papers (www.nateko.lu.se/masterthesis) and through the Geo-library (www.geobib.lu.se)

175 Hongxiao Jin (2010): Drivers of Global Wildfires — Statistical analyses
176 Emma Cederlund (2010): Dalby Söderskog – Den historiska utvecklingen
177 Lina Glad (2010): En förändringsstudie av Ivösjöns strandlinje
178 Erika Filppa (2010): Utsläpp till luft från ballastproduktionen år 2008
179 Karolina Jacobsson (2010):Havsisens avsmältning i Arktis och dess effekter
180 Mattias Spångmyr (2010): Global of effects of albedo change due to urbanization
181 Emmelie Johansson & Towe Andersson (2010): Ekologiskt jordbruk - ett sätt att minska övergödningen och bevara den biologiska mångfalden?
182 Åsa Cornander (2010): Stigande havsnivåer och dess effect på känsligt belägna bosättningar
183 Linda Adamsson (2010): Landskapsekologisk undersökning av ädellövskogen i Östra Vätterbranterna
184 Ylva Persson (2010): Markfuktighetens påverkan på granens tillväxt i Guvarp
185 Boel Hedgren (2010): Den arktiska permafrostens degradering och metangasutsläpp
186 Joakim Lindblad & Johan Lindenbaum (2010): GIS-baserad kartläggning av sambandet mellan pesticidförekomster i grundvatten och markegenskaper
187 Oscar Dagerskog (2010): Baösberggrottan – Historiska tillbakablickar och en lokalklimatologisk undersökning
188 Mikael Månsson (2010): Webbaserad GIS-klient för hantering av geologisk information
189 Lina Eklund (2010): Accessibility to health services in the West Bank, occupied Palestinian Territory.
190 Edvin Eriksson (2010): Kvalitet och osäkerhet i geografisk analys - En studie om kvalitetsaspekter med fokus på osäkerhetsanalys av rumslig prognosmodell för trafikolyckor
191 Elsa Tessaire (2010): Impacts of stressful weather events on forest ecosystems in south Sweden.
192 Xuejing Lei (2010): Assessment of Climate Change Impacts on Cork Oak in Western Mediterranean Regions: A Comparative Analysis of Extreme Indices
193 Radoslaw Guzinski (2010): Comparison of vegetation indices to determine their accuracy in predicting spring phenology of Swedish ecosystems
194 Yasar Arfat (2010): Land Use / Land Cover Change Detection and Quantification — A Case study in Eastern Sudan
195 Ling Bai (2010): Comparison and Validation of Five Global Land Cover Products Over African Continent

196 Raunaq Jahan (2010): Vegetation indices, FAPAR and spatial seasonality analysis of crops in southern Sweden
197 Masoumeh Ghadiri (2010): Potential of Hyperion imagery for simulation of MODIS NDVI and AVHRR-consistent NDVI time series in a semi-arid region
198 Maoela A. Malebajoa (2010): Climate change impacts on crop yields and adaptive measures for agricultural sector in the lowlands of Lesotho
199 Herbert Mbufong Njuabe (2011): Subarctic Peatlands in a Changing Climate: Greenhouse gas response to experimentally increased snow cover
200 Naemi Gunlycke & Anja Tuomaala (2011): Detecting forest degradation in Marakwet district, Kenya, using remote sensing and GIS
201 Nzung Seraphine Ebang (2011): How was the carbon balance of Europe affected by the summer 2003 heat wave? A study based on the use of a Dynamic Global Vegetation Model; LPJ-GUESS
202 Per-Ola Olsson (2011): Cartography in Internet-based view services – methods to improve cartography when geographic data from several sources are combined
203 Kristoffer Mattisson (2011): Modelling noise exposure from roads – a case study in Burlövs municipality
204 Erik Ahlberg (2011): BVOC emissions from a subarctic Mountain birch: Analysis of short-term chamber measurements.
205 Wilbert Timiza (2011): Climate variability and satellite – observed vegetation responses in Tanzania.
206 Louise Svensson (2011): The ethanol industry - impact on land use and biodiversity. A case study of São Paulo State in Brazil.
207 Fredrik Fredén (2011): Impacts of dams on lowland agriculture in the Mekong river catchment.
208 Johanna Hjärpe (2011): Kartläggning av kväve i vatten i LKAB:s verksamhet i Malmberget år 2011 och kvävets betydelse i akvatiska ekosystem ur ett lokalt och ett globalt perspektiv
209 Oskar Löfgren (2011): Increase of tree abundance between 1960 and 2009 in the treeline of Luongastunturi in the northern Swedish Scandes
210 Izabella Rosengren (2011): Land degradation in the Ovitoto region of Namibia: what are the local causes and consequences and how do we avoid them?
211 Irina Popova (2011): Agroforestry och dess påverkan på den biofysiska miljön i Afrika.
212 Emilie Walsund (2011): Food Security and Food Sufficiency in Ethiopia and Eastern Africa.
213 Martin Bernhardson (2011): Jökulhlaups: Their Associated Landforms and Landscape Impacts.
214 Michel Tholin (2011): Weather induced variations in raptor migration; A study of raptor migration during one autumn season in Kazbegi, Georgia, 2010
215 Amelie Lindgren (2011) The Effect of Natural Disturbances on the Carbon Balance of Boreal Forests.
216 Klara Århem (2011): Environmental consequences of the palm oil industry in Malaysia.
217 Ana Maria Yáñez Serrano (2011) Within-Canopy Sesquiterpene Ozonolysis in Amazonia
218 Edward Kashava Kuliwoye (2011) Flood Hazard Assessment by means of Remote Sensing and Spatial analyses in the Cuvelai Basin Case Study

Ohangwena Region –Northern Namibia
219 Julia Olsson (2011) GIS-baserad metod för etablering av centraliserade biogasanläggningar baserad på husdjursgödsel.
220 Florian Sallaba (2011) The potential of support vector machine classification of land use and land cover using seasonality from MODIS satellite data
221 Salem Beyene Ghezahai (2011) Assessing vegetation changes for parts of the Sudan and Chad during 2000-2010 using time series analysis of MODIS-NDVI
222 Bahzad Khaled (2011) Spatial heterogeneity of soil CO2 efflux at ADVEX site Norunda in Sweden
223 Emmy Axelsson (2011) Spatiotemporal variation of carbon stocks and fluxes at a clear-cut area in central Sweden
224 Eduard Mikayelyan (2011) Developing Android Mobile Map Application with Standard Navigation Tools for Pedestrians
225 Johanna Engström (2011) The effect of Northern Hemisphere teleconnections on the hydropower production in southern Sweden
226 Kosemani Bosede Adenike (2011) Deforestation and carbon stocks in Africa
227 Ouattara Adama (2011) Mauritania and Senegal coastal area urbanization, ground water flood risk in Nouakchott and land use/land cover change in Mbour area
228 Andrea Johansson (2011) Fire in Boreal forests
229 Arna Björk Þorsteinsdóttir (2011) Mapping *Lupinus nootkatensis* in Iceland using SPOT 5 images
230 Cléber Domingos Arruda (2011) Developing a Pedestrian Route Network Service (PRNS)
231 Nitin Chaudhary (2011) Evaluation of RCA & RCA GUESS and estimation of vegetation-climate feedbacks over India for present climate
232 Bjarne Munk Lyshede (2012) Diurnal variations in methane flux in a low-arctic fen in Southwest Greenland
233 Zhendong Wu (2012) Dissolved methane dynamics in a subarctic peatland
234 Lars Johansson (2012) Modelling near ground wind speed in urban environments using high-resolution digital surface models and statistical methods
235 Sanna Dufbäck (2012) Öppen dagvattenhantering med grönytefaktorn
236 Arash Amiri (2012)Automatic Geospatial Web Service Composition for Developing a Routing System