The Faculty of Engineering and Science
Aalborg University

Department of Computer Science

TITLE: Developing Mobile Location Based Services for Tourists

PERIOD:
01-02-2004 – 04-06-2004

GROUP:
Yan Zhao
Jasmin Ćatović
Vedran Alikalfić

SUPERVISOR:
Christian S. Jensen

NUMBER OF COPIES: 8

NUMBER OF PAGES: 96

ABSTRACT:

With advances of the mobile Internet, location based services (LBSs) attracted attention of many people. This report presents description of a software system that supports rapid development of integrated, high quality LBSs for tourists in urban areas. We describe system architecture that divides system in several parts. One of the parts consists of reusable components that are used to make up LBSs. Reuse of the components allows integration and rapid development of LBSs. We also emphasize that LBSs should provide correct and up to date information. As proof of concept, we combine components into mobile LBSs for tourists.

Preface

This report is the result of a DAT6 and KDE4 project unit at the Department of Computer Sciences at Aalborg University. It has been written by group E4-112. The names of the authors are stated below. The report is written within area of Database and Programming Technologies with focus on developing a location based service. We want to use this opportunity to thank Euman/AS, Jevgenij Gagach, Kristian V. B. Andersen, Michael Cheng, Stardas Pakalnis, Linas Bukauskas, Aalborg Tourist Bureau, Xuegang Huang, Wladyslaw Andrzej Pietraszek, Kort & Matrikelstyrelsen and Aalborg Municipality for all the help and resources provided to us throughout the semester. We also want to thank our supervisor, Christian S. Jensen, for all guidelines and comments he provided.

Aalborg, 11th June 2004

_____ _____

Yan Zhao Jasmin Ćatović

_____

Vedran Alikalfić

Contents

Chapter 1

Introduction

The Information revolution occurred at the beginning of 1950s when digital computers found application in science, military, and government. This is considered to be the first wave of Information Technology. The revolution continued at a slow pace until the 1980s when personal computers became available to the general public. This was the second wave of information technology. At the beginning of the 1990s, the information revolution occurred once more. It was the Internet that attracted the attention of many people, made transfer of information faster, and made computers in general more attractive to the public. Recent advances in wireless and mobile technology are beginning of a new wave: the mobile Internet. Advances in the mobile Internet stimulate the development of many services for mobile users. This report deals with a particular type of services that can be delivered via the mobile Internet, namely location based services.

1.1 Location Based Services

A location based service (LBS) is an application that is responsive to the user's location. The response is some content associated with the location. An LBS does not necessarily need to be delivered via the mobile Internet. LBSs can be delivered via the Internet, Krak is an example of such a service [19], or they can run as a stand-alone application, a car navigation systems for example [1]. Regardless of means of delivery, it is the user's location that is relevant to LBSs since the location determines what content is delivered to the user.

Mobilein [21] categorize LBSs into four categories: location sensitive billing, emergency services, tracking, and location–based information. They say that one can establish different zones and charge users of services at different rate for different zones. They also mention that this type of LBS is useful in conjunction with other services, e.g., prepaid wireless. Users of emergency services could be pinpointed immediately after dialing an emergency phone number. An application implementing such a service can make use of content. For example, a fire department, after receiving a call, may check available entrances into the building set on fire, where the closest hydrants are and similar. User tracking

is another application of LBS. This is an application that computes the whereabouts of users. One application of tracking is in mobile commerce. If one wants to offer a product to a user, the one would need to know that a user is within a reasonable range from the store that is offering the product to the user, i.e., tracking is needed to know where the users are located. Location–based information in another type of LBS. Mobilein points out that a way to develop a highly personalized LBS is to enable them to be location sensitive. To do so, one needs user tracking and content which is associated with locations.

Concept of "push" and "pull" are applicable to variety of disciplines among which is the Information Technology. "Push" and "pull" define a relationship between a consumer and provider. In our context, the consumer is the user of a LBS and the provider is the LBS. Then, LBSs implemented as "pull" technologies serve the content to a user whenever the user requests the content [14]. On the contrary, the "push" technologies deliver the content to the user without user's involvement; most often, the content is delivered when a certain condition is satisfied.

This report gives a description of a software system that implements LBSs that integrate user tracking and location–based information. The LBSs combine pull and push technologies, and are delivered via mobile internet.

1.2 Goals and Intentions

The main goal of this project is to develop a software system that supports rapid development of integrated, high–quality location based services for tourists in urban areas.

By rapid development of LBS, we mean that we provide components that can be used as nuts and bolts for development of a LBS. The choice of possible components reflects on a set of possible queries that a tourist can ask in an urban area. Examples of such queries are: what are the town's attractions, what are the attractions in my vicinity, where are the attractions, are the attractions available, and so on. Since a tourist in an urban area travels in a road network, many of the components will reflect on the queries that a traveller in a road network can ask. This implies that the system may serve as the basis for a wider range of LBSs for users in road networks. Examples of such queries are: where is the user with respect to the road network, what is the nearest attraction, where is the nearest attraction of a particular type, how does one get from a point in a road network to a specific attraction, and so on. In addition, the system provides components that allow client independence. The system architecture separates the components from services and therefore allows reusability.

By integrated location based services, we mean that services are developed on the top of same database, user interface has the same look and feel, and that LBSs reuse functionality provided by the components. The last implies that the services do not contradict each other. To understand what we mean when we say contradict, consider a case where a user, who is not moving, queries for the closest point of interest. The system replies with a point of interest. Then, the user asks for the path to the closest point of interest, but the system this time provides a path to a different point of interest than point of interest from the

first query.

A high–quality LBS provides accurate results. For example, consider LBS that provides shortest route from one point in a road network to a desired point od interest. If user requested the service, the LBS would need to compute the nearest restaurant in terms of travelling time. To compute shortest travelling time, the LBS needs to be aware of road distance and speed limit. Then, the service would need to consider what transportation means the user is using in order to give the correct route, for example a truck may not be allowed in some parts of road network. Finally, the LBS would need to know what restaurants are open at the time of the request because navigating a user to a point of interest that does not satisfy user's needs is not a correct result. This simple example illustrates what we mean when we say a high–quality LBS. In order to provide accurate results, input data to a LBS must be up to date; hence, the system provides a sophisticated data value chain that maintains the data in the database up to date.

This report does not only present the system for developing LBSs, it also describes a set of LBSs for tourists in urban areas. In order to develop these LBSs, one needs to consider the way tourists carry out tourism. We consider an empirical study that addresses this concern [9] where the authors identify the main tourist problems and solutions to the problems available to tourists. The set of LBSs that we describe proposes new mobile solutions to these problems. We refer to the set of LBSs as The Mobile Tourist Guide.

In this report we consider:

- System design — we show the separation of data, components, and services.
- System implementation — we describe components and present implementation details for some of the components.
- We perform experiments to find best parameter values for the component that answers the query "where is the user with respect to road network"
- Implementation of tourist LBSs as proof of concept — we implement a set of LBSs through various compositions of the implemented components.

Before we leave this section, we need to note that our work focuses on the technical aspects of the system mentioned above. There is a range of issues, such as useability, business prosperity etc., that we do not consider.

1.3 The Report Overview

This report is structured in the following way. Chapter 2 explains tourists' behavior. We describe the structure of tourist visit, tourists' problem and solutions, and we propose a set of mobile solutions to the problems in the form of LBSs. Chapter 3 describes system architecture. We implemented client server architecture to separate presentation from logic. We also describe how to structure the application logic in order to obtain modularity which in turn provides

reusability and hence rapid development of LBS. Chapter 4 describes the data tier. Here we present the data model used to implement the database, justify our modeling decisions, and present the data value chain. Chapter 5 describes client application. Chapter 6 describes the components and services. There, we describe logics of individual components and how we combined the components into the LBSs. Chapter 7 describes how the data on the client side changes as the user moves in a road network. Chapter 8 describes a mechanism used to position a user in a road network. Finally, Chapter 9 presents conclusion.

Chapter 2

Tourists' Problems and Solutions

Researchers who studied tourism usually concentrated on the effect of tourism on society and economy. Unlike others, Brown and Chalmeres [9] concentrate on the way tourists organize their activities, use the tourists' literature and the tourists' tools (such as maps and public transportation time tables), and find solutions to the problems they face during a visit. We consider these problems and solutions and create a set of new, mobile solutions to the problems.

There is number of projects that are implementing LBSs for tourists [3] [4] [2], but apparently non of them considers tourist behaviors in the way Brown and Chalmeres do.

In this chapter we first present a formal way of structuring tourists' visits, most common problems that tourists face and tourists' solutions to the problems. Then we present the solutions that The Mobile Tourist Guide provides.

2.1 The Structure of a Tourist's Visit

Most tourists follow the same pattern when visiting a destination. Tourists first, in one way or the other, examine the available destinations before deciding where to go. After deciding where to go, tourists make plans on what to do after the arrival at the destination. During the visit, tourists try to explore the destination and to enjoy the visit. They take photos, record home videos and buy souvenirs that will remind them of the interesting places they visited.

The described pattern can be decomposed into three activities:

- pre–visit,
- visit,
- post–visit.

Activity of planning a trip in advance and and getting familiar with the destination is called *pre–visit*. The pre–visit is an important part of a tourist experience

for two reasons. First, pre–visit has a practical purpose since the tourists get familiar with the place they are about to visit, get ideas about what to do during the visit and make rough plans. Second, becoming familiar with an exciting place one is about to visit is satisfying and joyful.

Time spent at the destination is called *visit*. The visit is the core activity of the tourist experience. During the visit a tourist enjoys a destination and caries out the usual tourist activities.

Being a tourist and visiting an interesting place is joyful; however, many tourists like to remember the visit long after the visit took place. The activity of remembering a visit is called *post–visit*. Post–visit is an important part of a tourist experience since it extends excitement and joy of the visit.

We decompose tourists' experience of visiting a destination into pre–visit, visit and post–visit because the problems that tourists face and the solution to the problems are specific for each phase. In the next section, we state the problems and currently available solutions. The section is based on empirical study presented by Brown and Chalmeres [9].

2.2 Tourists' Problems and Tourists' Solutions

One of problems that tourists face is to find a way of how to carry out the pre–visit. The pre–visit is usually carried out in three ways: reading the tourist literature, copying plans provided by others, or the combination of the first two. All of the methods have weaknesses. For example, tourist literature may be out of date and usually does not provide means for organizing a trip. When trying to copy plans of others, tourists are often limited to a small number of opinions about particular attractions related to the destination. If a tourist that is pre–visiting a destination has different taste than the tourist that is giving the advice, then the tourist that is pre–visiting may be mislead.

During a visit, most tourists have to deal with three problems: what to do, how to do it and when to do it.

Tourists have to decide what to do during pre–visit and visit. They often decide what to do in advance, during the pre–visit, because they have to filter out a large number of attractions. They have to balance attractions of different sites, budget, constraints such as distances between sites and public transportation time tables, and most importantly time allocated for the visit. However, Simon et al. state that tourists are satisfied with rough plans to allow flexibility during the visit [24]. Here, flexibility is needed so that tourists can adjust their plans during the visit.

For example, we can consider a two–day visit to a city. A tourist decides to go sight–seeing on the first day of the visit and to visit a museum the second day. Also, the tourist has in mind to see several monuments the first day. But while visiting one of the monuments, the tourist found out that the monument was on a hill with a beautiful view of the city. Moreover, tourist found out that there is a restaurant close by and decided to stay and enjoy the view while having lunch. However, since the tourist lost some time, the tourist had to change plan and give up visiting of one of the monuments. On the second day of visit, the

tourist found out that the museum is very large and that it is impossible to cover the entire museum for the allocated time. In both cases the tourist had to alter the plan. Therefore, problem of what to visit remains even if plans are made ahead. Tourists also keep plans rough because exploration of unknown and uncertainty is part of the thrill.

Once a tourist decides what to do, the tourist has to know how to do it. Tourists must be aware of behavioral norms of the places they are visiting since they differ from place to place. Also, tourists are often exploited in stores, restaurants and similar. They often fail to notice this problem and consequently find themselves in awkward situations.

After a tourist chooses activities and gets an idea about how to do it, the tourist must decide when to do the chosen activities during the visit. This implies that the tourist has to know where the chosen activities take place so the activity hours and public transportation time tables can be coordinated.

The two most common tools that tourists use to decide what, how and when to do are the guidebooks and friends' advices. The guidebook is often a large source of information about the available activities, maps and photos. Some tourists hear about an exciting trip their friends completed and want to experience the excitement themselves; consequently, some tourists make rough copies of their friends' plans.

The weakness of both tools, a friend or a guidebook, is that the information provided by them can be out of date and maybe not be very practical to use during the visit. For example, use of a guidebook during a visit may be time consuming because a tourist has to filter out a large amount of information. Even when a tourist finds a suitable attraction, the guidebook often does not provide up to date opening hours of the attraction or its availability at that very moment.

Traditional ways of preparing for a post–visit are: taking photos, taking videos and buying souvenirs during a visit.

2.3 The Mobile Tourist Guide's Solutions to the Tourist Problems

The Mobile Tourist Guide is a set of location based services (LBS) that make trip planning easier, visits more efficient and enjoyable, and trips recollection better. Before we justify this claim, we need to explain what we mean by "making a visit efficient".

It is important to realize that tourists are not always looking for most convenient solutions to the problems presented in the previous section because problem solving is a part of the enjoyment that tourism provides. When designing The Mobile Tourist Guide, we put the tourist enjoyment in the heart of each LBS. The Mobile Tourist Guide often offers partial solutions to the problem and leaves some room for tourist to make decisions. For example, The Object Finder does not return a single point of interest but a set of them and leaves a tourist to decide which point of interest is most interesting to visit. This service makes tourists more efficient by narrowing down choice of the point of interest that

are, for example, within reasonable distance but still retains the mystery and excitement that tourism offers by allowing the tourist to chose the point of interest. More about this service will be presented later in Section 6.2.3. Another way of making a tourist more efficient is, for example, insuring that tourists do not waste time by travelling between attractions during a visit. Following sections justify the claim made at the beginning of this section.

2.3.1 Pre–visit

As already explained, pre–visit is the activity of getting to know the destination and planning the trip. We extend means of pre–visit with following features: The Electronic Mobile Guidebook, Theme–Tours and Tourist–to–Tourist. Each feature is a LBS integrated into The Mobile Tourist Guide.

The Electronic Mobile Guidebook

The Electronic Mobile Guidebook is a digital version of the traditional guidebook. The traditional guidebook is subject of revision and update few times a year. Most of tourist offices print relevant tourist literature once a year. In a dynamic environment, such as a popular tourist destination, tourist business changes often and therefore information about it changes often as well. It is obvious that one update is not enough to keep the information up to date. Another, weakness of the printed literature is that different tourist association in the same city print information in different brochures which decentralizes the source of information which in turn makes it more inconvenient for a tourist to find interesting attractions.

The Electronic Mobile Guidebook provides means of accessing centralized, up to date tourist information from anywhere. The information presented to tourists via The Electronic Mobile Guidebook is retrieved from a database. This concept makes it easier to update the guidebook since the guidebook does not have to be reprinted but the database needs to be updated only.

Theme–Tour

Traditional way of advertising tourist destinations, via guidebooks and similar literature, presents what attractions can be experienced at a particular destination. However, the traditional literature often leaves out details on how and when to experience the attractions. To remedy this weakness, The Mobile Tourist Guide provides means of planing parts of a trip to a particular destination in a way that proposes plans. The suggested plans include an advice on what, when and how to do. This feature of The Mobile Tourist Guide is called Theme–Tour.

Theme–Tour is a predefined set of points of interest, a time table, a path through a transportation infrastructure and description. This feature is different from the guidebook in a way that it organizes a set of point of interest into a trip that can be completed for a certain time period. The points of interest are an advice on what to do. The time period in which a theme–tour can be accomplished is defined in the time table as tour length and earliest time the tour can start.

This allows tourists to know when the theme–tour can be accomplished. As the set of points of interest suggests what attractions to visit and the time table suggests when to do a theme–tour, the path and the description suggest how to accomplish the theme–tour. The path describes a subset of the transportation infrastructure along which a tourist should travel in order to complete the trip in time. A potential tourist may have to travel parts of the way on foot, and parts using public transportation or a car which is described in the path. If the tourist needs to use public transportation, then bus numbers and departure times are provided and if a tourist has to travel in a car, the parking areas are provided as well. The description is used to provide tourist additional information, e.g., information about cultural differences.

Tourist–to–Tourist

Tourist–to–Tourist is a similar but different feature than the Theme–Tour. It is similar in the way that organizes points of interest into a trip and proposes the trip to tourists as a plan. But, it is different from the Theme–Tour in the way that the proposal is made by a fellow tourist and that the plan is less detailed. Each Tourist–to–Tourist plan is recorded by a tourist while the tourist was visiting. The plan is composed of a set of points of interest, a path and the start time of the tour. Here, the path is simply a subset of a transportation network. From the point of time when a tourist indicates that the Tourist–to–Tourist feature is "ON", The Mobile Tourist Guide starts recording tourist's movement through the transportation infrastructure. Once the user indicates that the Tourist–to–Tourist feature is "OFF", the system stops recording the tourist's movement. The sequence of coordinates recorded in the meantime is the path. The time table is composed of two entities: the start time and the end time. The start time provided by Tourist–to–Tourist is the time when the feature was switched on. Similarly, the end time is the time when the feature was switched off. The set of points of interest are chosen by the tourist. To add a point of interest to the set, a tourist has to click on it sometime between start and end time.

The three LBSs, The Electronic Mobile Guidebook, Theme–Tours and Tourist–to–Tourist, make trip planning easier. They provide tourists up to date information about a destination via The Electronic Mobile Guidebook. Tourists are also able to obtain advice from the local tourist bureaus and fellow tourists via Theme–Tours and Tourist–to–Tourist respectively. Getting an advice from a tourist office is valuable because this advice comes from the experts on the local tourism. As Brown and Chalmeres point out, tourism has collaborative nature; tourists trust to each other and it is important to exchange opinions. The Mobile Tourist Guide allows tourists to exchange opinions via Tourist–to–Tourist feature. In a way, Theme–Tours and Tourist–to–Tourist are also means for trip planning since they allow tourists to copy plans of other tourists.

Theme–Tour and Tourist–to–Tourist are presented to tourists in a similar way as Remember your Visit service, see Figure 2.5.

2.3.2 Visit

As mentioned above, the three problems that tourists encounter are: what attraction to visit, when and how to visit them. The Mobile Tourist Guide provides four LBSs that help tourists solve the problems. The four LBS are: The Object Finder, The Explorer, The Advertiser and The Electronic Mobile Guidebook. The Object Finder and The Electronic Mobile Guidebook are implemented as pull technologies while The Explorer and The Advertiser are implemented as mix of push and pull technologies.

Following the maxim that problem solving is part of the excitement, Object Finder, The Explorer and The Advertiser act as a guide rather than a provider of complete most convenient solutions to the tourists. They help tourists to narrow down the number of available attractions to an apprehensible number of choices and in this way help them decide what to do. If a tourist pursues one of the given choices, the system implicitly or explicitly gives user hints on how and when to carry out the activity (how and when to visit a museum for example). The number of choices is reduced via application of different filters.

The information presented by a system like The Mobile Tourist Guide should be applied three filters: time, space and profile [16]. Application of the filters is a means of offering tourists attractions that are accessible at the time of the enquiry, in the tourist's vicinity and relevant to the user. Technical details of the filters' application are covered in later chapters. Through following sections we describe The Electronic Mobile Guidebook, The Explorer and The Advertiser, and The Object Finder. An example of accessing these services on the client is illustrated in Figure 2.1.

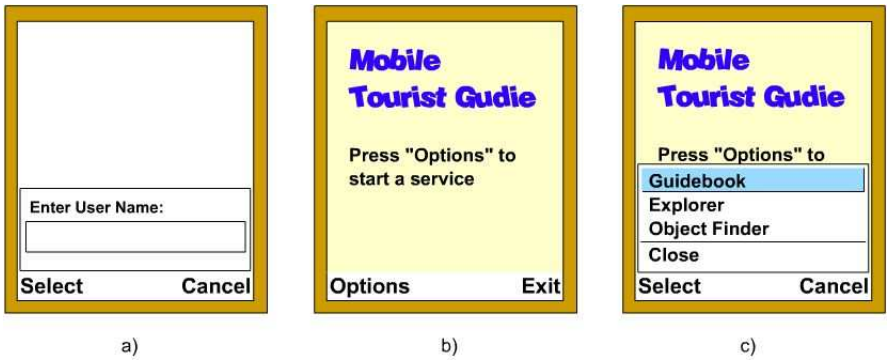


Figure 2.1: Starting The Mobile Tourist Guide

After having started the application, the log-in screen is presented to the user. Upon inserting the user id the application switches to welcome screen. As shown in Figure 2.1, by choosing "Options" a list of main services pops up. In the figure there are three of them — Mobile Guidebook, Explorer and Object Finder.

The Electronic Mobile Guidebook

Most tourists use the tourist guidebooks throughout visit. Consequently, The Electronic Mobile Guidebook is suitable as a tool used during the visit. The only difference between the versions used during pre–visit and visit is the means of delivery of the information. A tourist can search the The Electronic Mobile Guidebook via a web browser or a mobile client (which in our case is a cell phone). If the tourist, during the visit needs information from the guidebook, then the tourist can retrieve information on the mobile client anywhere any time. Tourist can see the same information in a web browser and a mobile client but the presentation of the information is slightly different. An example of the guidebook service on the client is illustrated in Figure 2.2

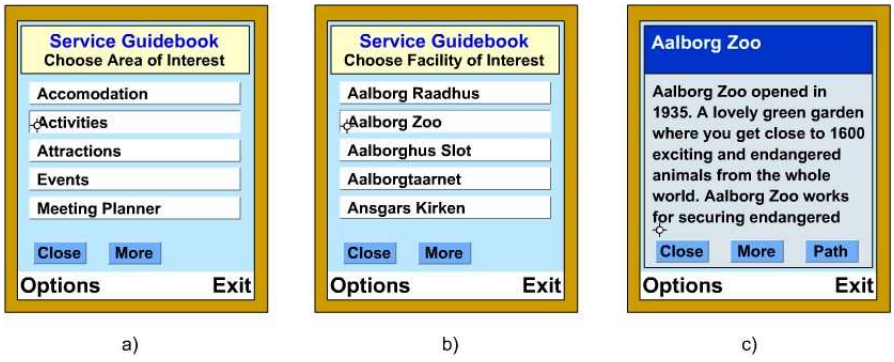


Figure 2.2: Guidebook Service

By selecting the guidebook service, the list of all the available facility areas is presented to the user, as shown in Figure 2.2.a. White items constituting the list of areas are buttons. In the bottom of the screen there are two additional buttons — "Close" and "More". "Close" will close the current active window and return the user to the start screen, whereas "More" indicates that there are more items in the list and by clicking on it, brings the following items to the display.

By clicking on one of the area items, the user gets the list of facility types within the selected area. Selecting the type of interest will get the user to the screen similar to Figure 2.2.b which is the specific facilities that are within selected type. Finally by clicking on a facility ("Aalborg Zoo", 2.2.b) the user gets information about selected facility, 2.2.c. At this stage "Path" button is available as well. By clicking on it, the user requests traversable path between current location and the selected facility.

In order for user to conceive the path and facility location correctly, this service requires a map as a background. Hence, the application switches to "Explorer" service, Figure 2.3.c.

The Explorer and The Advertiser

The Explorer and The Advertiser are two different LBS that are designed to work together. Unlike The Electronic Mobile Guidebook, The Explorer and The

Advertiser can be push technologies that do not need to interact with tourists in order to work.

The main objective of the The Explorer and The Advertiser is to allow tourists to be informed of what the city can offer without taking any input from the user. Both LBSs extract the relevant information to a particular tourist by means of time, space and profile filters. A tourist can see the relevant information on the client's screen which is divided in two parts.

One part of the screen is used by The Explorer. In this part, tourists can see the relevant points of interest in their vicinity. The other part of the screen is used by the Advertiser. A point of interest may advertise; for example, a restaurant may advertise daily special offers. With each advertisement, the Mobile Tourist Guide associates a geographical zone. If a tourist enters a zone associated with an advertisement, then The Advertiser displays title of the advertisement in its part of the screen if the advertisement is relevant to the tourist. From this perspective, The Explorer and The Advertiser are push technologies.

Another objective of the two LBSs is to present detailed information about a point of interest if a tourist pursues this option. The detailed information is presented in the same way as in the case of The Object Finder. An example of these services running on the client is illustrated in Figure 2.3.

Figure 2.3: Explorer Service

The Explorer service uses user's profile in order to display only a certain group (type) of points of interest. In Figure 2.3.a the user has placed the mouse icon above one of the displayed POI locations — blue icon in the mid–left part of the screen. This action displays the name of the POI. If the user wants to know more about the POI, one clicks on the circle, which displays further information about the POI as already seen in figure 2.2.c. Otherwise the name of the POI vanishes when user moves the mouse icon away from the POI.

Figure 2.3.b shows the Advertiser service in action. Advertiser is a sub–service that in this case is used by the Explorer. The Advertiser does not require any direct input from the user on the client since it is a push based service. The displaying of the information on the screen is provoked by user physically entering the geographical ad–zone of the POI colored in red. The red header gives short information about the ad, and three buttons described in the context of "Guidebook" are available again. If user takes no action, the short ad is

displayed only for a certain period of time (e.g. 30 seconds). Before this period expires, the user can always hide the ad by pressing "Close". "More" button gives more information on the POI while path makes a requests for the traversable path to the POI. Once the push–ad is hidden, it will not be displayed again.

As already mention, Figure 2.3.c shows the situation where user has requested a path to one of the POIs. A small red dot in the upper right corner of the screen indicates that the user has activated the tour recording service. This is also a sub–service available from the "Options" menu of all other services. Until this service is stopped, client cashes the traversed UTM coordinates and the POIs for which the user has shown interest (by clicking on them). The cashed data is periodically reported to the server and stored in the database.

The Object Finder

The Object Finder is a LBS that allow tourists to search for a particular point of interest type, restaurants for example. When a tourist enquires a particular point of interest type, there are two obvious solutions: the Object Finder can respond with all points of interest or a single point of interest of the particular type. If the Object Finder responds with a single point of interest, then a certain criteria such as distance or customer ratings can be used to select the point of interest. Neither of the two solutions is suitable in the context of The Mobile Tourist Guide. If the Object Finder responds with all points of interest, then the tourist would be overwhelmed with information, the client performance would deteriorate and displaying a large number of points of interest on a small client's screen would not be visually acceptable. Instead, the Object Finder first extracts a subset of points of interest by applying the filters to all points of interest and then displays where on the map the extracted points of interest as well as the tourist's position are. Now tourist can have an idea where the points of interest are with respect to the tourist's position. Tourist can also get further information about a displayed point of interest. The further information includes four pieces of information:

- an approximation of the distance to the point of the interest,
- description of point of interest which may include advices on how to reach the destination (by bus, by train, on foot...) and business hours
- other attractions in the vicinity of the point of interest under consideration.
- an option to get the shortest path through the transportation infrastructure.

The Object Finder solves the problem of what to do by offering points of interest that are available at a given moment and relevant to tourists. The description of the attraction, provided in the further information can give tourists insights that can help tourists decide whether it is interesting to visit the attraction or not. Information about other attractions in vicinity of the attraction gives tourists ideas about what can be done in that part of the city. The further information also gives useful hints to tourists that can help solve problems of how and when

to do an activity. Tourist can make decision when to visit the attraction based on approximation of the distance and business hours. The proposed ways to reach point of interest help solve problem on how to visit. The Object Finder can easily extend the further information to point out behavioral norms and similar.

2.3.3 Post–visit

The Mobile Tourist Guide provides two tools to post–visit a visited destination: Remember Your Visit and Photo Repository.

Photo Repository

The Photo Repository is a feature that allows tourists to store a large number of photos during the visit. In Chapter 3, we propose an architecture for developing location based services. The architecture proposes use of a cell phone as a client. Tourists can use camera that is integrated with the phone to record photos. After a photo is recorded, the tourist can upload the photo. Later, the tourist can view and download the photos via web browser, see Figure 2.4.

Figure 2.4: An Instance of Photo Repository

Remember your Visit

Remember Your Visit is a feature that allows a tourist to recollect a visit in a comprehensive way. Using this feature a tourist can view the traveled routes

through the town, attractions visited and photographs taken, see Figure 2.5.

The figure illustrates a map, a path that tourist recorded, points of interest that the tourist thought were interesting (circles on the map) and square icons referencing the taken photos. In this particular case the user has clicked on the left most point of interest icon bringing the textual description and the image related to the point of interest to the right part of the screen. In case the user clicks on one of the icons referencing personal photos, the photo will be displayed.

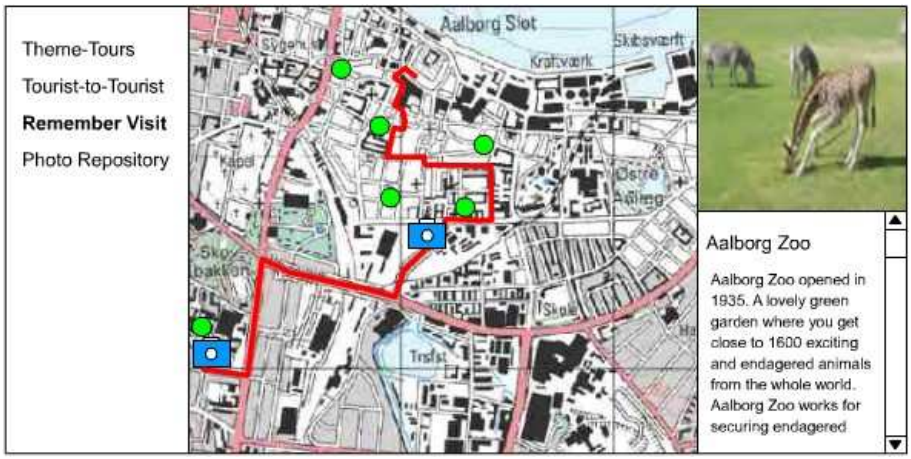


Figure 2.5: An Instance of Remember your Visit

Chapter 3

System Architecture Overview

There are various different types of location based service applications — e.g. Krak.dk's [19] route planning, where users location at the time of service request is not relevant. In such a system the user approximately knows the location that one will be visiting in future and by simulating the location, the user can retrieve information about location surroundings before even getting there.

Another example of location based service applications are car navigational systems such as Mobile Danmark [23]. This particular product uses a GPS receiver in order to understand user's geographical location. Based on the location and pre–selected destination the system can guide the user through the road networks and also draw attention to the user to surrounding points of interest. However, the problem with such a line of products is that the data is provided on a digital media at the time of purchase, and once installed on the client it cannot be updated frequently.

In Chapter 2 we have seen examples of how users can access some provided location based services with the system that we have developed. They use mobile clients which according to their location, are capable of getting up to date information about their surroundings. In order for the retrieved data to be up to date there have to be some sources providing and maintaining the data independently of the client. Hence, there have to be some stationary servers providing such services for the clients. In Chapter 4 we cover the details on the provided content and talk about the data value chain that makes our system provide up to date data.

Such location based service applications have to have physically separated multi–tiered architecture. Figure 3.1 illustrates the proposed application framework for location based service applications providing up to date content. The architecture from the figure also makes a clear separation between presentation of the content and application logic. Following sections describe general aspects of the framework.

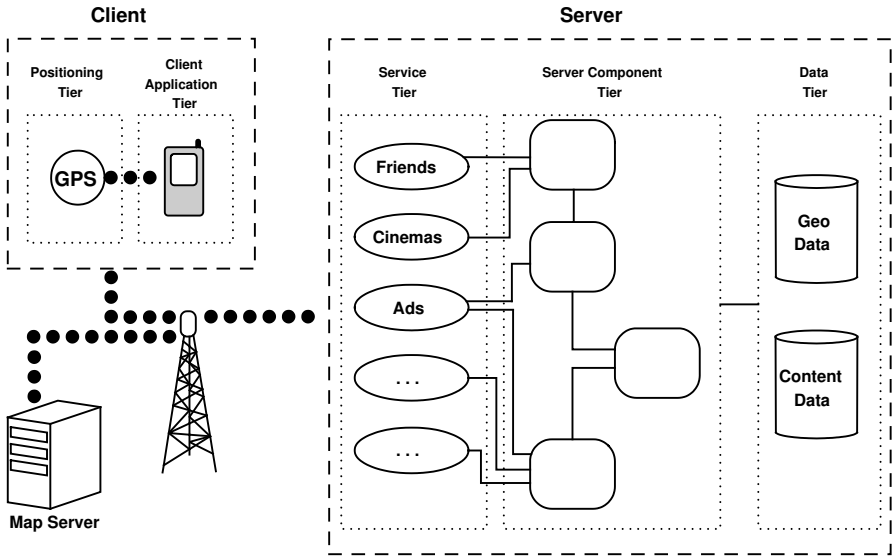


Figure 3.1: Application Framework

3.1 Client Tier

Cellular phones are most prominent personal digital mobile devices today. The latest mobile technology enables development of simple applications for mobile devices so it was natural to choose a cell phone as client hardware — more specifically a Nokia 3650 cell phone. The Nokia 3650 is equipped with General Packet Radio Service (GPRS) which enables users to access the Internet over mobile networks [15]. Since the phone does not have an integrated GPS receiver, it is essential that every client is equipped with an external device. Mobile phones with integrated GPS receivers are expected to be soon available on the market (e.g. 3GEO system from 3G, Motorola i88s, etc.). Positioning technology will be a common feature in a couple of years leaving many potential users of LBS [7].

Nokia 3650 is one of many mobile devices running on Symbian OS 6.1 operating system. Series 60 System Development Kit enables application development on such devices in C++ programming language. In order for similar devices to run the mentioned operating system they have to meet certain hardware standards. The developed applications are therefore not dependent on one specific device but rather a larger group.

The implementation of our client services is based on the SVG browser which was earlier developed at AAU. It provides easily accessible APIs for manipulation of GPRS and Bluetooth [6] which are used for establishing connection to the mobile internet and opening a communication channel with other Bluetooth devices — GPS receivers.

Furthermore and more importantly, BitFlash Mobile Player is integrated in the browser making it compliant with the SVG Basic standard. Scalable

Vector Graphics format (SVG) is a modularized language for describing two–dimensional vector and mixed vector/raster graphics in XML format [22]. The format was primarily developed for web graphics which are data–driven (not static images) and can easily be made interactive and personalized. This means that we can practically send any content to the browser as long as it is encoded in SVG format. An additional bonus of using the SVG is that it is implied that we will be using XML format for data exchange between clients and serrvers. SVG specifications ensure that we are provided a parser and just have to stick to its standards.

The features provided by the SVG browser simplify the development and therefore reduce probability of making mistakes in the application development. As we mentioned at beginning of this section, clients will be exchanging data with a server or multiple servers while a service is in use. Having agreed on the standard of the exchange format, our work comes down to what to do with the data and when to do it on the client side. Since what the client gets is SVG format, we can manipulate the data through SVG DOM functions then return the result to the client screen, or send it back to the server. Document Object Model (DOM) is a platform and language neutral interface allowing programs to access and update the structure and content of documents — in other words DOM makes documents dynamic [25].

In a system as the one presented in Figure 3.1, in order for client to deliver location based services to users there are certain general tasks that it has to be capable of accomplishing.

- It must be able to communicate with the server over wireless network
- For most of the services client must know it's position at the time of the request.
- It must understand services provided by the server
- Enable interaction from user's side
- Most of provided services will be using maps as background images. These maps have to be downloaded with respect to a user's location.

The first criteria is met by client being able to make HTTP requests over GPRS. Second criteria is met by client being able to communicate with external GPS device over Bluetooth. The third criteria requires that server and client understand each other, hence there must be a communication standard. In our case third and fourth criteria are mostly covered by the SVG technology since it is up to the server to provide services in SVG standard so that client can understand them. As illustrated in Figure 3.1 the map–images will primarily be retrieved over GPRS. There will not be a need to adapt image format to the application since SVG technology is capable of encapsulating most image standards.

Details on the implementation of the client application are covered in Chapter 5

3.2 Server Tier

As already mentioned at the beginning of this chapter, in order for clients to receive high quality and up to date data based on their geographical location, a client–server architecture is essential. The server works like a big brain being able to answer all the questions that clients ask. It has access to different kinds of data related with certain geographical locations, which is provided by the data tier. Before answering clients' requests it retrieves relevant data only and puts it into a context by processing it.

The web server is running on Apache Tomcat servlet container [5] which uses the official Java Servlet and JavaServer Pages technologies. The developed server application works exclusively on Java Servlets whose main purpose is to provide dynamic content over the web. With the help of the extensive APIs a developer has easy access to management of loading, unloading, resource sharing and dealing with security issues. Since all the server logic is written in Java it is also cross–platform.

From Figure 3.1 we can see that the server logic is divided in two layers — service layer and server component layer. Server components do the data processing which is needed in order to provide the services (e.g. finding a path between two geographical locations, encoding of data according to the standard implemented on the client, etc.). At the very bottom of the components there is a database interface that executes queries that are needed in order for other components to retrieve the relevant data.

The service layer implements the services as objects. They are an extension of HttpServlet managing HTTP requests from clients. The three purposes of service objects are listed below.

- The information provided by the client is filtered out in order to understand what it is exactly that the client is requesting
- The underlying server components are accessed for the purpose of data processing
- Response is sent to the client

By separating services and components into two layers the system achieves better modularity since the purpose of the components is to provide functionalities. They are easily exchanged or their functionalities extended while new components are easier integrated. Services are there to integrate the functionalities provided by the components and to control the data flow. The architecture makes it also easier for developer to think of new services and integrate them into the existing system. Implementation details on the services and server components are covered in Chapter 6.

3.3 Data Tier

A location based service (LBS) is an application that is responsive to the user's location. The response is some content or information which was derived from

the content. From definition of the location based services, it is obvious why geographical and content data are needed. In our context, the user is a person equipped with a mobile phone and user's location is a position in a road network or a point in two dimensional space at which the user is placed at a given moment. This implies that geographical data about road network is essential with respect to our system. When we say content, we mean speed limit, road type, restaurants, stores, cinemas, medical clinics and similar road characteristics and points of interest. If one needs to personalize an LBS, then data about users of the LBS is needed.

In order to implement personalized LBSs, the following data is needed:

- geographical data
- content data
- user data

Since the system provides the infrastructure for development of high quality LBS, the system must be supplied with up to date data. We implement a sophisticated data value chain that maintains the data in a way that the data is always up to date Chapter 4.

Chapter 4

The Data Tier

In this chapter we describe the data chain that provides up to date data to the system. Following describes the source data, the data model used to implement the database and an overview of the data value chain.

4.1 The Data Sources

The data stored in the database originates from four sources: Aalborg municipality, Aalborg Tourist Office [8], the users of The Mobile Tourist Guide and advertisers.

The Aalborg municipality provided information about geometry of the road network in Aalborg. The road network is described as a set of poly lines where a poly line is a sequence of coordinates in two dimensional space. Unfortunately, the only data provided to us is the data that describes the geometry. The data that describe number of lanes, speed limit, allowed turns and similar were not available to us.

The Aalborg Tourist Office provided information about the attractions in Aalborg. The data includes names, descriptions, addresses, types and similar information about the attractions. The national tourist organization maintains a database which holds information about points of interest from all of Denmark and distributes the data to local tourist offices in XML format periodically. Periodical data distribution is done as the way of update. The Aalborg Tourist Office obtains data in this way as well.

The Mobile Tourist Guide is a personalized service which implies that it needs information about its users. We have implemented a system component that enables users to configure The Mobile Tourist Guide to serve them the best. The user's preferences are obtained from a user and stored in the database

We have explained earlier that The Mobile Tourist Guide has capability to advertise. The data needed to describe ads can be obtained from the parties who want to advertise.

4.2 The Data Model

The data model that we present in this section is an extension of the data model presented in the previous semester [26]. Then, we presented a data model that deals mainly with modeling of the road network, content and integration of the two. Natural questions that rise here are: how to model a road network, associate content with a road network and how to define movements through a road network in a database. These questions have been addressed by several people[10] [17]. All of the referenced work addresses the issues mentioned above from road or transportation management perspective. However, many of these ideas are suitable, at least in our opinion, in the context of LBSs. We have incorporated some ideas from [10] and [17] into a data model that is used in our project. The data model that we use is simpler than models in [10] and [17]. The reason for that is that many attributes needed for road and transportation management are not needed in our context. Also, several representation of a road network, e.g., KM-post representation or cartographic representation, are not needed. This report extends the data model to include information about user, user profiles, Theme-Tour data, and Tourist-to-Tourist. Following gives an overview of the data model.

4.2.1 Overview

Data model used in the project consists of two main parts: internal and external. Figure 4.1 illustrates this concept. The internal part is illustrated to the left in the figure. This is a set of tables that describe the road network, points of interest, integration of points of interest with the road network, users, user profiles, advertisements, Theme-Tours and Tourist-to-Tourist. The external part is illustrated in the middle of the figure. It consists of a set of materialized views derived from the internal part and a set of views of the internal part. All of the LBSs, to the right in Figure 4.1, should access only the external part.

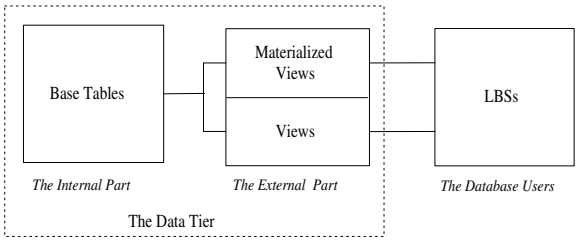


Figure 4.1: Data Model Overview.

The internal part is organized in a way that allows simple maintenance, i.e. to provide a simple way of updating the existing data and extending the data model. The external part is organized in the a way that simplifies software development and improves program execution efficiency.

First we will explain the internal and external parts separately, and then we will justify the separation.

4.2.2 The Internal Part

The internal part was partly developed in the previous semester. Then, we presented segment representation of a road network, integration of the content in a road network, and road network geometry.

Idea of segment representation of a road network originates from [10]. The internal representation of a road network is described as a collection of segments where segment represents a road section in such a way that segments are made as long as possible while they preserve topology. Segment representation also includes information about connections that intersect with the segments. Figure 4.2 illustrates the schema of the segment representation.

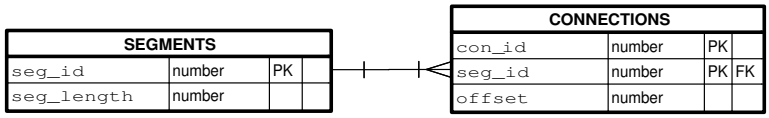


Figure 4.2: Internal Representation of a Road Network Proposed by Referenced Source.

The idea of segment representation was adopted into the data model used in the project. We used data provided by Aalborg municipality to create segment representation. Each segment represents a chain of one or more poly lines where one poly line can be part of a single segment. From above, one can infer that segments represent linear subsets of the road network. Also, segments were created to include as many poly lines as possible. Segment representation in our data model strongly resembles the original one. Figure 4.3 illustrate database schema of the segment representation.

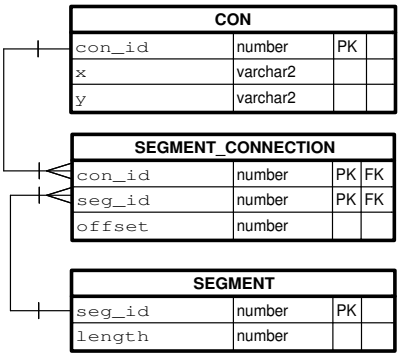


Figure 4.3: Segment Representation of a Road Network.

Tables `SEGMENT` are identical in both models. Attribute `seg_id` is a primary key and it is used to identify a segment. Information about length of segments is stored in `length`. Table `SEGMENT_CONNECTION` from Figure 4.3 is similar to table `CONNECTIONS` from Figure 4.2. Their attributes describe the fact that a connection, described by attribute `con_id`, intersects a segment, described

by attribute `seg_id`, at some distance from the beginning of the segment, described by attribute `offset`. Here, `con_id, seg_id` together make up a primary key. Attribute `seg_id` is a foreign key and it references `SEGMENT`. The difference between table `SEGMENT_CONNECTION` from our model and `CONNECTION` from the original model is that table `SEGMENT_CONNECTION` references table `CON` via `con_id`. Table `CON` is an extra table in our model. This table is needed if one needs to save more information about a connection than ID. Attributes X and Y are coordinates of a connection in UTM format. Information about connection coordinates is redundant since it could be computed by querying `SEGMENT_GEOMETRY` table (`SEGMENT_GEOMETRY` will be described later). Because some applications that we implemented use coordinates of the connections, we store the coordinates in `CONNECTION` table for fast retrieval. Attribute `con_id` is primary key. Table 4.1 shows the data in tables `SEGMENT`, `SEGMENT_CON` and `CON` with respect to the road network illustrated in Figure 4.4.

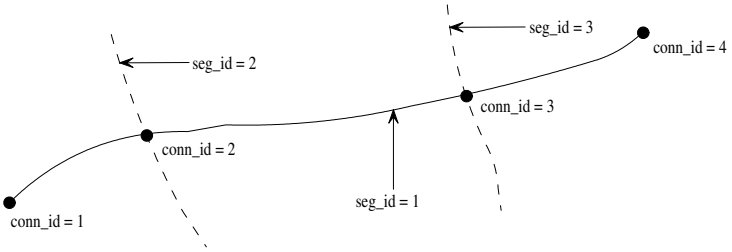


Figure 4.4: A Simple Road Network

seg_id	length
1	400

a)

con_id	seg_id	offset
1	1	0
2	1	70
3	1	280
4	1	400
2	2	122
3	3	320

b)

con_id	x	y
1	559561	6319444
2	559639	6319763
3	559641	6319825
4	559689	6319723

c)

Table 4.1: Data in Table `SEGMENT`(a), `SEGMENT_CON` (b), and `CON` Describibg the Road Network and Content from Figure 4.4.

Now that we explained the segment representation of a road network, we go on and explain how to relate content to this representation.

As already mentioned, content is an important part of the data–tier in a LBS. Content describes transportation infrastructure and points of interests. Here we explain how we integrated content into the data model. We distinguish between two types of content: content that can be associated with a point in a road network and content that can be associated with a stretch of the road network. Kenneth et al. [17] call these point and linear events respectively. We adopt this terminology from this point on. Figure 4.5 illustrates schema that models integration of content with segments.

Table `CONTENT` holds information about a particular content. As Figure 4.5 indicates, our model supports multiple content tables. The two attributes in the table are the core attributes that each table must have. Depending on

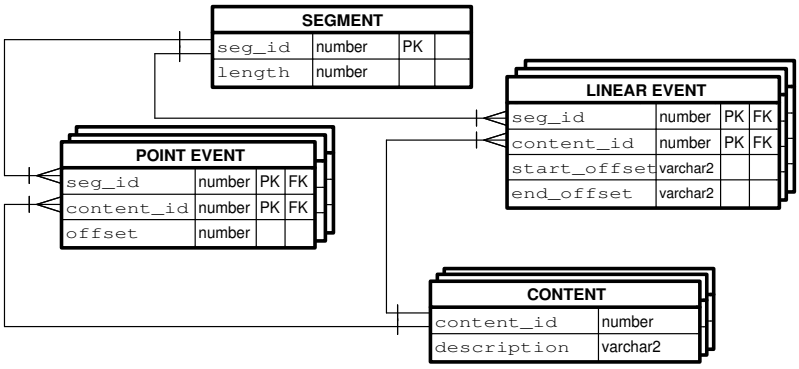


Figure 4.5: Content Integration

type of content, `CONTENT` may have more then the core attributes. For example a restaurant can be modelled as `content_id, description, address, name, business_hours`. Table `RESTAURANT`, Table 4.2, is an example of a table that holds point events. Attribute `content_id` is primary key of `RESTAURANT`, `description` may in few sentence describe a restaurant. Meaning of attributes `name, address, business_hours` is obvious. An example of a linear event is a natural attraction that can be seen along road. Table `SITE`, Table 4.3, can hold information about these attractions. Attribute `type` describes whether, for example, a river, mountain peak or valley can be seen. Some of them may be interesting only during a certain part of year and attributes `season_from` and `season_to` describe this. Attribute `name` stores the name of a particular attraction. Reader should note that attribute `content_id` is unique within the same type of content.

content_id	**description**	**address**	**name**	**business_hours**
1	Pizza & Grill	23	Bella Italia	8–23

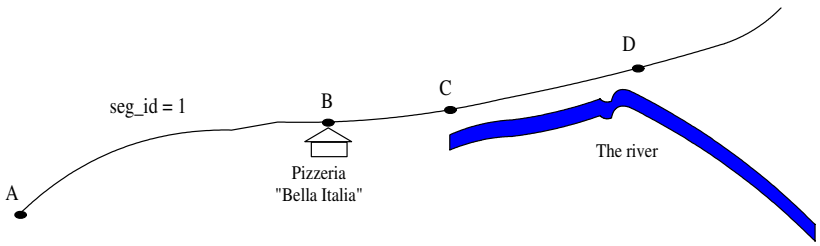
Table 4.2: Instance of Tables: `RESTAURANT`

content_id	**description**	**type**	**season_from**	**season_to**
1	An artificial river.	River	January 1^{st}	March 1^{st}

Table 4.3: Instance of Tables: `SITE`

Tables `POINT_EVENT` and `LINEAR_EVENT` are used to position content on the segments. For each `CONTENT` table there is a `POINT_EVENT` or a `LINEAR_EVENT` table. Both tables have same core attributes: `seg_id, content_id` which together constitute primary key. `seg_id` is a foreign key and references table `SEGMENT`. `content_id` is also a foreign key and references table `CONTENT`. A point event is associated with a segment by a single point expressed as offset from the beginning of the segment. Attribute `offset` in table `POINT_EVENT` describes this. A linear event begins at some offset from the beginning of a segment, attribute `offset_from` in the table `LINEAR_EVENT`, and ends at some offset, attribute `offset_to`. Tables `SEG_RESTAURANT` and `SEG_SITE`, Table 4.4, position restau-

rants and natural attractions with respect to the road and content illustrated in Figure 4.6.



Location	Offset
A	0m
B	180m
C	220m
D	320m

Figure 4.6: A Road and Points of Interest Associated with it

seg_id	con_id	offset
1	1	180m

a)

seg_id	con_id	start_offset	end_offset
1	1	220	320

b)

Table 4.4: Instance of Tables: `SEG_RESTAURANT` (a) and `SEG_SITE` (b)

Schema that models geometry representation is illustrated in Figure 4.7. Geometry of the road network was preserved during segment creation. As several poly lines were combined into segment, the coordinate sets of the corresponding poly lines were combined into a single coordinate set representing the segment. Attributes `seg_id` and `corder` constitute primary key. `seg_id` is foreign key and it references table `SEGMENT`. `corder` preserves the order of the coordinates. Attributes `x` and `y` represent a point in two–dimensional space. Attribute offset describes distance of a coordinate with respect to the beginning of the segment.

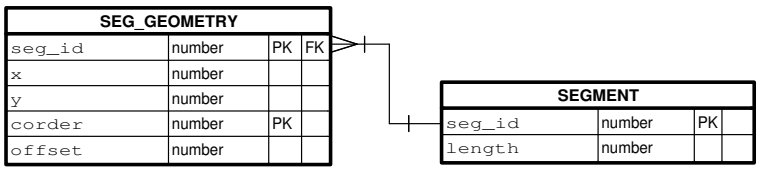


Figure 4.7: Geometry Representation of the Road Network.

So far we presented the part of the model that was developed in the previous semester. Following describes an extension to it. The extension models point of interest, user, user profiles, advertisements, Theme–Tour, Tourist–to–Tourist. The reader should note that the model above did not have to change in order to add the extension. Moreover, this in a way demonstrates extensibility of the internal part.

As mentioned above, the data about points of interest is obtained from Aalborg Tourist Bureau. So far we described how to integrate the content (and therefore point of interest) and we gave hypothetical examples of point of interests. Now we show the schema that models points of interests received from the tourist bureau. All of the points of interest are integrated into the road network as point events. Figure 4.8 illustrates database schema that models the points of interest and their integration into the road network.

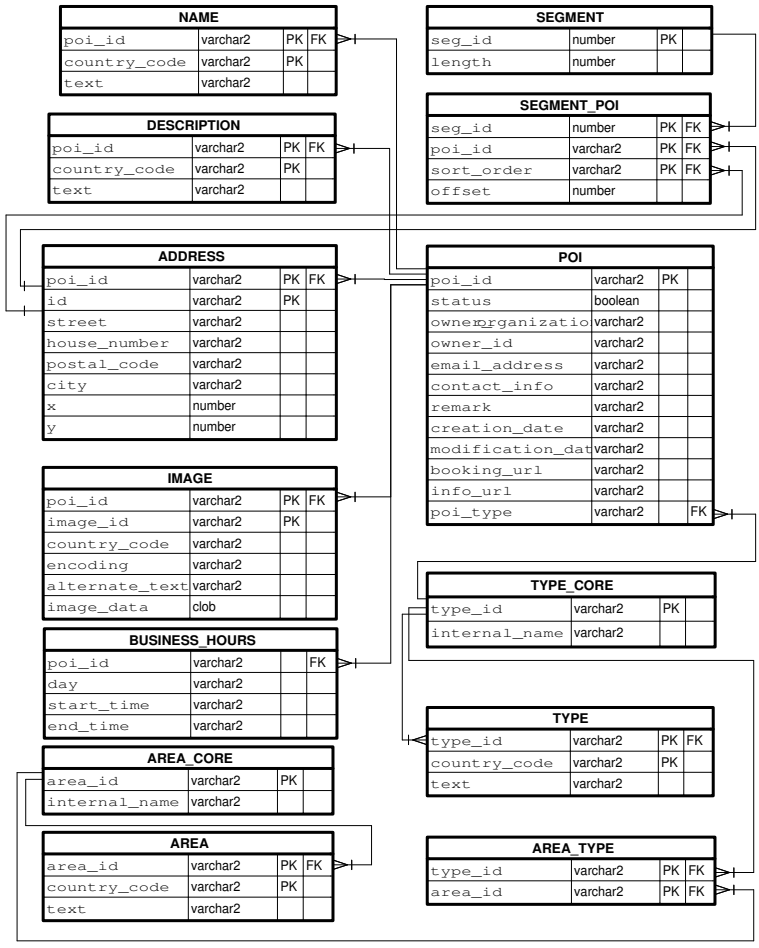


Figure 4.8: Integration of the Tourist Data

Attributes in table `POI` describe a point of interest. The data model proposed earlier proposes different entities for different types of points of interest. Here

however, we model all points of interest as the same entity because only the attributes that are common to all points of interest were available to us. The attribute `poi_id` is a string of characters that uniquely identifies a point of interest and is primary key of `POI`. Point of interest is available if the attribute `status` is set to TRUE. The LBSs can use this information to either respond to LBS's users with this content or not. `owner_organization` and `owner_id` are foreign keys to the remote databases from which the point of interest originates. If it is possible to make a contact with a point of interest, then the contact information can be retrieved from the attributes `contact_info` and `email_address`. The date of creation and the last modification are recorded in the attributes `creation_date` and `modification_date`.

Attributes `inf_url` and `booking_url` store URLs at which further information about the points of interest can be obtained or booking can be carried out if applicable. If the owner of a point of interest needs to make an extra remark, the attribute `remark` can store this information. Finally, the attribute `poi_type` describes type to which a point of interest belongs. This attribute is a foreign key and references table `TYPE_CORE`.

Attributes in table `TYPE_CORE` describe types of points of interest. A point of interest can be restaurant, hotel, monument or something else of sixty two available types. Attribute `type_id` describes string of characters which uniquely identifies a type and is primary key of `TYPE_CORE`. Each type has a name which is described by the attribute `internal_name`.

An area is a set of similar types. For example, `area` Transportation includes four types: Car Hire, Coach Hire, Ship Connection and Airports. Attributes of table `AREA_CORE` describe an area. Each area has id and name described by `area_id` and `internal_name` respectively. `area_id` is the primary key.

Attributes in table `AREA_TYPE` describe which types make up an area. Attribute `area_id` is a foreign key that references table `AREA_CORE` while `type_id` specifies what type belongs in the area. `type_id` references table `TYPE_CORE`.

The Mobile Tourist Guide is a multilingual application. Data in tables `NAME`, `DESCRIPTION`, `ADDRESS`, `IMAGE`, `AREA` and `TYPE` are stored in different languages. This fact had an impact on the data model as illustrated in Figure 4.8. The data model is designed to trade off space for time. Consider table `NAME`. Attributes in the table `NAME` describe name of a point of interest in different languages. Attribute `poi_id` is foreign key that references table `POI`. Attribute `text` stores the name of a point of interest in the language described by `country_code`. The tourist office translates the information in two foreign languages: German and English. An alternative design for the table `NAME` is illustrated in Figure 4.9. The two designs preserve same information.

Using this design in Figure 4.9 the name is stored in one record for a single point of interest and using design in Figure 4.8 the name is stored in up to three records. However, design in the in Figure 4.8 allows an LBS to store the active language in a variable and then use the variable in the "where" clause when retrieving the data, see Table 4.5.b. If the design in Figure 4.9 was implemented, then the logic would become more complex and time demanding, see Table 4.5.a. The logic would have to check which language is active and then, according to the result, retrieve information from the appropriate attribute. Attributes in table `DESCRIPTION`, `TYPE` and `AREA` describe a point of interest, type and area

respectively in the similar way the attributes in the table NAME describe the name of the a point of interest.

NAME			
poi_id	number		FK
textDK	varchar2		
textDE	varchar2		
textUK	varchar2		

Figure 4.9: An Alternative Design for the Table NAME

```
if language = 'DK' then SELECT textDK FROM NAME
WHERE poi_id = poi;
else if language = 'UK' then SELECT textUK FROM NAME
WHERE poi_id = poi;
else language = 'DE' then SELECT textDE FROM NAME
WHERE poi_id = poi;
```

a) The Space Efficient Implementation.

```
SELECT text FROM NAME WHERE poi_id = poi AND country_code =
language
```

b) The Time Efficient Implementation.

Table 4.5: Logic Needed to Retrieve Multilingual Information

Table IMAGE holds information about the images of a point of interest. A point of interest may have more than one image. Attribute id uniquely identifies an image of a single point of interest. poi_id is foreign key that references POI. If an image is displayed in a web browser, its alternate text can be retrieved from attribute alternate_text. Attribute image_data holds encoded image data. Information about encoding format is stored in the attribute encoding. Sometimes, text on the images are translated into different languages. Similarly as above, the information about the language is stored in attribute country_code. Primary key of IMAGE is combination of id, poi_id and country_code

Table ADDRESS holds information about addresses of a points of interest. A point of interest can have more then one address. The attribute id is used to identify different addresses of a point of interest. The meaning of rest of the attributes is intuitive.

Finally, each point of interest is projected on a segment, distance between the beginning of the segment and the point of projection was calculated and stored in the table SEG_POI as a point event.

The Mobile Tourist Guide is a set of personalized LBS. This fact had an impact on the data model. The data model had to be extended to include information about user and user's preferences. Ethical issues and user information security issues are beyond scope of this report. We have designed the system to function with the minimum information about user but the design is flexible to include more information about the users. Figure 4.10, illustrates data model for the users, user profiling and advertising.

The Mobile Tourist Guide requires a user to provide a password and optionally a name which can be used by the system to display messages such as "John, there is a 18th century monument in you vicinity". The name provided by user

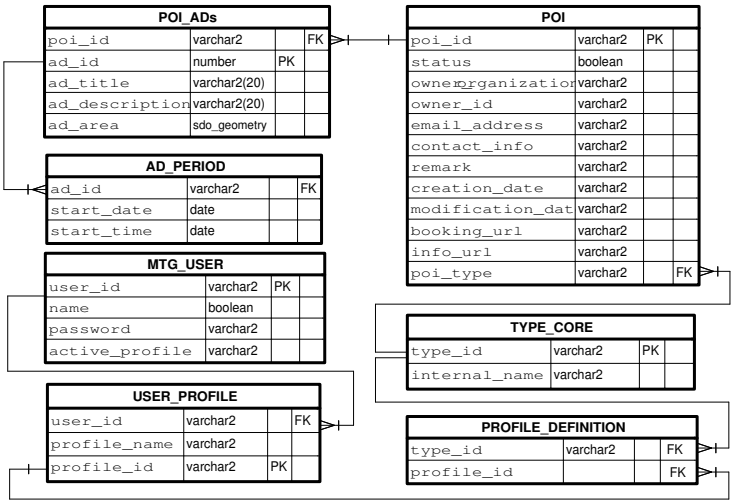


Figure 4.10: Data Model for User, User profile and Advertisements

does not have to be user's real name. The information about the user is stored in the `MTG_USER` (MTG stands for Mobile Tourist Guide). Attribute `mtg_id` stores an integer that uniquely identifies a user. This integer is assigned by the system when user creates a profile. Password and name of preference are stored in attributes `name` and `password` respectively. Finally, the user's active profile is stored in attribute `active_profile`.

A user can create one or more profiles. Attributes `profile_id` and `profile_name` in table `USER_PROFILE` describe unique number that identifies a profile and a name given to a profile by the user during profile creation. Attribute `user_id` is foreign key and references table `MTG_USER`. Profile consists of a set of types of point of interest which are specified by the user at the time of creation. The types are stored in the table `PROFILE_DEFINITION`. Attribute `type_id` specifies which type belongs to the profile stored in the attribute `profile_id`.

As we said earlier, The Mobile Tourist Guide can advertise attractions, restaurants, cafes, special offers at department stores and similar. This functionality is achieved in the form of push–ads. Tables `POI_ADs` and `AD_PERIOD` hold data needed to implement the functionality. Attributes `title` and `ad_description` store information that is shown to the user. When user is made aware of the vicinity of a point of interest, the user is shown text stored in the attribute `title`. If user chooses to view details of the point of interest, text stored in attribute `ad_description` is retrieved. Every ad can be uniquely identified by a number stored in attribute `ad_id`. Attribute `poi_id` is a foreign key and references table `POI`. User in not shown all ads stored in `POI_ADs` but only a subset. Subset is determined by filtering content of `POI_ADs`. One of the filters is user location. User is shown an ad only if the user is within ad's active area. The active area of an ad is stored in attribute `ad_area` which is of type `sdo_gemetry` capable of expressing most of two–dimensional shapes. Attributes in `AD_PERIOD` specify the periods during which an ad is active. Database type `DATE` is capable

of storing year, month, day and time in one table column. Therefore, a period can be modeled with two columns of `DATE` type. Attributes `start_date` and `end_date` specify the start and of a period respectively. An ad can have multiple periods.

Figure 4.11 illustrates data model used to implement Theme–Tour. Table `THEME_TOUR` holds some information about a Theme–Tour. The Theme–Tour time table is stored in attributes `earliest_time` that indicates earliest time the Theme–Tour can start and `tt_time` that indicates the time needed to complete the theme–tour. Theme–Tour description is stored in the attribute `description`. Tourists identify a theme–tour by its name stored in the attribute `name` and the system, internally, identifies a theme–tour by a unique number stored in the attribute `tt_id`. As explained in Chapter 2, Theme–Tour is pre–visit tool that tourists can use to obtain advice on what to do. The advice is a set of points of interest and is stored in the table `TT_POI`. Attribute `tt_id` describes that a point of interest, stored in `poi_id`, belongs to a particular theme–tour. The points of interest are visited in an order which is preserved in the attribute `sequence_number`. Attributes`poi_id` and `tt_id` are foreign keys and they reference table `POI` and `THEME_TOUR` respectively. A theme–tour contains a path that is stored in table `TT_PATH`. A path is sequence of coordinates. Attributes `X` and `Y` are coordinates in UTM format. Attribute `corder` preserves the order of the coordinates that form a path. All records in the table which have the same value for attribute `tt_id` describe path of the same theme–tour. Attribute `tt_id` is a foreign key and it references table `THEME_TOUR`.

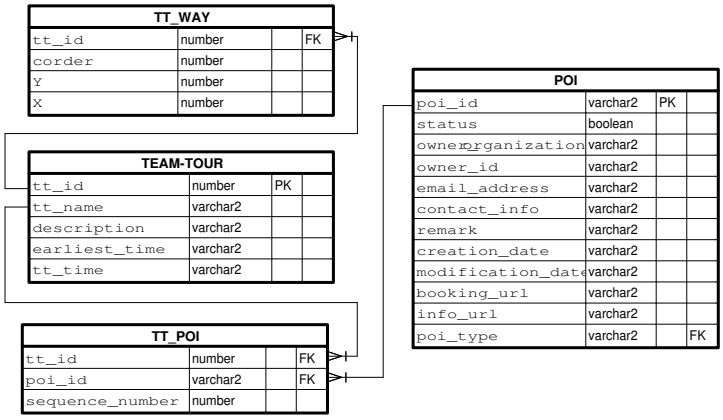


Figure 4.11: Data Model for Theme–Tour

Figure 4.12 illustrates data model used to implement Tourist–to–Tourist. Attributes in tables `T2T_PATH` and `T2T_POI` have identical names and meanings as the attributes in tables `TT_PATH` and `TT_POI`. Attributes `t2t_id` and `t2t_name` in the table `TOURIST2TOURIST` have same meaning as the attributes `tt_id` and `tt_name` in the table `THEME_TOUR`. The time when a tourist starts recording a Tourist–to–Tourist is stored in the attributes `start_time` and the time when the recording is stopped is stored in the attribute `end_time`. Table `T2T_USER` relates the users, attribute `mgt_user`, with Tourist–to–Tourist they recorded and attribute `t2t_id`.

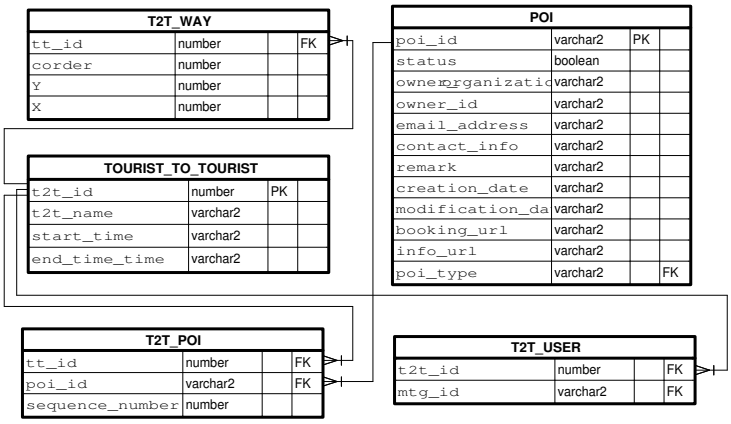


Figure 4.12: Data Model for Tourist–to–Tourist

4.2.3 The External Part

As we mentioned earlier, the external part is composed of a set of materialized views and a set of views of the internal part. In this section we describe the internal part but before that, we first describe the difference between the database tables, views and materialized views.

Tables, Views and Materialized Views

Generally speaking, a database is collection of related data that holds facts about a Universe of Discourse. Universe of Discourse is composed of entities and relationships between the entities. Entities are represented by records which are set of attributes that describe an entity. Records that describe same entity are combined into tables. Relationships between entities are preserved via common attributes in records that describe different entities. Relational-database theory, roughly speaking, states rules how to compose attributes into records and how to preserve the relationships. When a database is designed according to the rules of the theory, all facts from the Course of Universe are stored in the database once. Motivation behind this is to have consistent and easy to update database. The tables that store information about a Course of Universe in the described way are called base tables. Two database tabes are shown in Table 4.6.a, table `EMPLOYEE`, and Table.4.6.b, table `DEPARTMENT`. A record in table `EMPLOYEE` holds employee number, name, surname and department in which the employee works in attributes `emp_num`, `name`, `surname` and `dep_num` respectively. A record in table `DEPARTMENT` holds information about department number, and city, street and number where the department is located in attributes `dep_num`, `city`, `street` and `number` respectively. Fact that an employee works in a department is captured by including attribute `dep_num` in the `EMPLOYEE` record.

Now that we have database tables and records in them, we can retrieve information from database via SQL queries. Typically, there is more than one database

emp_num	name	surname	dep_num
1	John	Smith	001
2	Steve	McDonald	001
3	Mike	Corleone	002
4	Dave	Jonson	003

a)

dep_num	city	street	number
001	Toronto	Oulette Ave	11123
002	Edmonton	Tecumseh Road	4691
003	Vancouver	Huron Road	123

b)

Table 4.6: Tables `EMPLOYEE` and `DEPARTMENT`, a and b Respectively.

user and different users may be interested in different subsets of database. Even though base tables can be queried directly, access to information in a database is, most often, given through views. A view is a named SQL expression that provides a user subset of data that is relevant to the particular user. Main motivation for this is security and user friendliness. By providing only relevant subset of data to a user, the user queries become simpler and user can not get hold of information that is not meant for the user. For example, assume that accounting in our imaginary company is contracted out. Moreover, accounting for different departments are handled by different accounting firms. To provide only relevant subset of data to the accountants, we can supply three different views (see Table 4.7) to three different accounting firms. Accountants that do accounting for department with ID of 1 do not have to query table `EMPLOYEE` for employees with `emp_num` of 1 but they query view `EMP_DEP_1` for all employees. `EMP_DEP_1` is not a table but, as pointed out, an SQL expression. Every time a query is executed against a view, the SQL expression is executed as a subquery of the query. However, result of an SQL expression is a table and on a high level of abstraction, a view can be thought of as a table.

`EMP_DEP_1` := CREATE VIEW AS SELECT * FROM `EMPLOYEE` WHERE `emp_num` = 1
`EMP_DEP_2` := CREATE VIEW AS SELECT * FROM `EMPLOYEE` WHERE `emp_num` = 2
`EMP_DEP_3` := CREATE VIEW AS SELECT * FROM `EMPLOYEE` WHERE `emp_num` = 3

Table 4.7: Different Views of Table `EMPLOYEE`

Sometimes, regardless of database design, queries are so complex that performance of the queries is poor. To overcome this situation, we can materialize views. To do so, we physically store result of a an SQL expression that defines a view. This way, the SQL expression is not computed each time a materialized view is queried. Consider the query from Table 4.8.

```
SELECT name, surname, city, street, number

FROM   EMPLOYEE, DEPARTMENT

WHERE EMPLOYEE.dep_num = DEPARTMENT.dep_num
```

Table 4.8: Example of a SQL Query Used to Generate a View

Now suppose that this query was executed very often. Since the query involves a join operation, it may be a bottleneck in the system. One way to eliminate this bottleneck is to physically store the result of the query, i.e. to provide a

materialized view. Table 4.9 shows data that is physically stored.

name	surname	city	street	number
John	Smith	Toronto	Oulette Ave	11123
Steve	McDonald	Toronto	Oulette Ave	11123
Mike	Corleone	Edmonton	Tecumseh Road	4691
Dave	Jonson	Vancouver	Huron Road	123

Table 4.9: Materialized View for the View from Table 4.8

However, one needs to be careful because by eliminating one bottleneck, one can create another one. Materialized views are not appropriate in a volatile system since updates to base tables have to be propagated to the materialized views. As we describe the external part, we will point out what we modeled as views and what as materialized views and justify the modeling decision.

The Graph Representation of a Road Network

The information about a road network is stored in segment representation tables and content tables that describe properties of the road network. Here, we present the graph representation of a road network which is usually used to describe how a user can move throughout the road network. In our model, the graph representation is a materialized view derived from the segment representation and the tables that store the relevant content.

If the graph representation models, for example, how a car can move through a road network, then the relevant content must include information about the intersections, allowed turns, speed limit and similar. As already pointed out, we do not have any content that describes the road network. The only information we have about the road network is the geometry that we used to create the segment representation. However, the The Mobile Tourist Guide can show closest path through the transportation infrastructure from a given point in the transportation infrastructure to a particular point of interest and therefore we needed to have a graph representation.

The graph representation used in this project is derived from the segment representation. The nodes in the graph are intersections and the edges are the roads between the intersections. Since we do not have data that describe where the intersections are, we assumed that intersections are:

- ends of a segment,
- the points at which two or more lines that are geometrical representation of segments intersect.

Figure 4.13 illustrates schema that models the graph. Attribute `edge_id`, in table `RN_GRAPH` is a number that uniquely identifies an edge and it is primary key. Here, an edge is modeled as an ordered pair (`node_from`, `node_to`)

Attributes `node_from` and `node_to` are foreign keys that reference table `INTERSECTION`. Data stored in `node_from` and `node_to` provides information about direction of movement; a user can move from the intersection stored in

node_from to the intersection stored in node_to. Table INTERSECTION has one attribute, intersection_id. It is a number that uniquely identifies an intersection. Table SEG_INTERSECTION relates the intersections, and therefore the graph representation, to the segment representation. Attribute length describes the length of the edge.

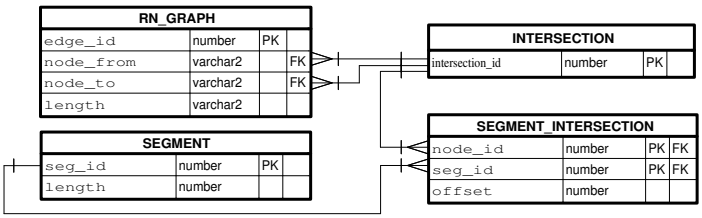


Figure 4.13: Graph Representation

A road network is an entity that changes over time: roads are modified, removed or added. However, data about roads does not change often in the magnitude that would make the database volatile; therefore, it is appropriate to model the graph representation as a materialized view.

The Available Points of Interest

Information about the points of interest is scattered across several tables: POI, NAME, DESCRIPTION, ADDRESS and TYPE which have to be joined in order to retrieve comprehensive information about a point of interest. We provided a materialized view AVAILABLE_POI to store comprehensive information about the available points of interest (point of interest is available if the attribute value for status in table POI is set to "TRUE"). The data stored in AVAILABLE_POI is data that is result of the query from Table 4.10.

Schema of AVAILABLE_POI is illustrated in the Figure 4.14. The meaning of the attributes is identical to the meaning of corresponding attributes in the base tables described in Section 4.2.2.

Points of interest are integrated with the road network via SEG_POI table, i.e. they are integrated with the segment representation. We provide a materialized view, GRAPH_POI, that integrates the points of interest with the graph representation. Figure 4.14 illustrates schema that models GRAPH_POI. Attribute poi_id identifies a point of interest. Each point of interest is associated with an edge which is described by edge_id. Attribute offset indicates how far from the beginning of the edge the point of interest is placed.

Point of interest data is not dynamic. Names, descriptions, addresses, images, contact information, booking url and similar information about points of interest do not change very often. Attribute status, which indicate availability of a point of interest, allow dynamic change of availability. Fact that information about availability may change at any time may seem a potential source of volatility. But, even if points of interest were changing availability often, the effort to update AVAILABLE_POI is not great. When a point of interest becomes unavailable, the records in AVAILABLE_POI associated with that point of inter-

```
SELECT POI.uuid, POI.owner_organization, POI.owner_id,
       POI.email_address, POI.contact_info,
       POI.remark, POI.creation_date,
       POI.modification_date, POI.booking_url,
       POI.info_url, TYPE.text type, PRODUCT_NAME.text
       name, PRODUCT_DESCRIPTION.text description,
       ADDRESS.sortorder, ADDRESS.place, ADDRESS.road,
       ADDRESS.housenumber, ADDRESS.subcity,
       ADDRESS.floor, ADDRESS.postalcode,
       ADDRESS.city, ADDRESS.country, ADDRESS.x,
       ADDRESS.y, IMAGE.image_data, IMAGE.image_id,
       IMAGE.alternat_text

FROM   POI, NAME, DESCRIPTION, ADDRESS, TYPE, IMAGE

WHERE POI.uuid = NAME.uuid AND POI.uuid =
       DESCRIPTION.uuid AND POI.uuid = ADDRESS.uuid
       AND NAME.countrycode = DESCRIPTION.countrycode
       AND POI.type_id = TYPE.type_id AND
       TYPE.countrycode = NAME.counterycode AND
       POI.statues = TRUE
```

Table 4.10: Logic Needed to Retrieve Multilingual Information

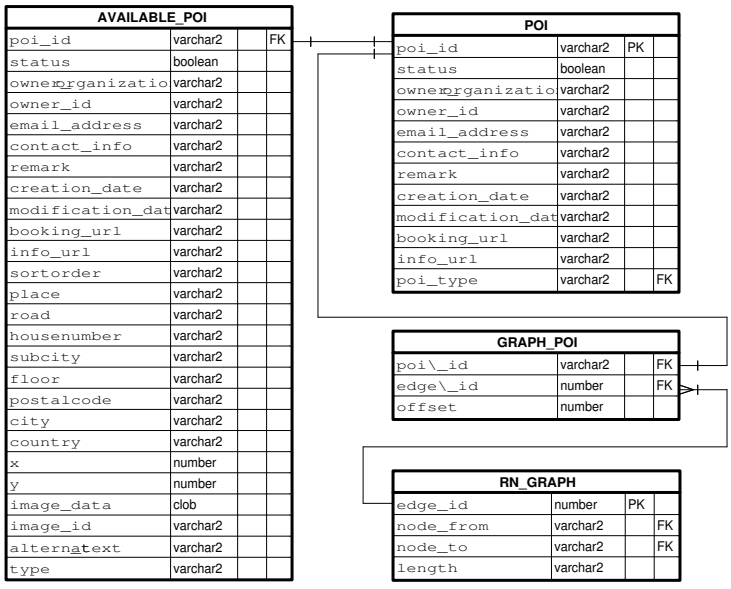


Figure 4.14: Query for The Materialized View `AVAILABLE_POI`

est are simply deleted. If a point of interest becomes available, then the update procedure does not need to recompute the entire join presented in Table 4.10. The update procedure would be restricted only to the points od interest that

changed status from unavailable to available, i.e. the update specifies which points of interest to update in the "where" clause. Consequently, the performance of the update can be tuned up via indices. The information about the points of interest changes periodically, once a day; therefore, the materialized view needs to be updated only once a day which is acceptable in our context. All of the facts above indicate that modeling `AVAILABLE_POI` as a materialized view is an appropriate choice. Since it is appropriate to model `AVAILABLE_POI` and `RN_GRAPH` as materialized view, the same is true for `GRAPH_POI`.

Time and Profile Filters

Earlier we explained that The Mobile Tourist Guide uses three filters (time, space and profile) to deal with large number of points of interests. Here we describe time and profile filters.

User profile is a set of point of interest types. As we explained, a user can have more than one profile. However, a user can have only one active profile. The profile filter is a set of point of interest type in the active profile. Table 4.11 shows the query needed to compute the profile filter for a particular user. The profile filter indicates what kind of points of interest the user in interested to see at a given point in time. The profile filter is a view named `PROFILE`. Tables that `PROFILE` uses are volatile since user can switch between profiles and change definition of individual profiles as pleased. Therefore, it is appropriate to model `PROFILE` as a view.

```
SELECT MTG_USER.id u_id, PROFILE_AREA.p_type p_type
FROM   MTG_USER, PROFILE_AREA

WHERE PROFILE_AREA.mg_user = MTG_USER.id AND
      PROFILE_AREA.id = MTG_USER.active_profile
```

Table 4.11: Query That Retrieves the Profile Filter

The time filter can be applied to points of interest and advertisements. If the time filter is applied to points of interest, then the time filter is a set of attribute values for attribute `BUSINESS_HOURS.poi_id`. In this case, the time filter is a view named `ACCESSIBLE_POI`, see Table 4.12, that computes identification number of all all points of interest that are accessible at a given point in time. If the time filter is applied to the advertisements, then the time filter is a set of attribute values for attribute `AD_PERIOD.ad_id`. In this case, the time filter is view named `ACCESSIBLE_ADS`, see Table 4.13, that computes identification number of each advertisement that is associated with a point of interest that is accessible at a given point in time. If `ACCESSIBLE_POI` and `ACCESSIBLE_ADS` were modeled as materialized views, they would have to be updated very often (for example every five minutes) in order to store up to date information; this justifies the decision to model them as views.

SELECT poi_id **FROM** BUSSINESS_HOURS **WHERE** day = *today* AND start_time $\leq$ *sysTime* AND *sysTime* $\leq$ end_time

Table 4.12: Query That Retrieves the Time Filter Applied to Points of Interest.

SELECT ad_id **FROM** AD_PERIOD **WHERE** start_date $\leq$ *sysDate* and *sysDate* $\leq$ end_date

Table 4.13: Query That Retrieves the Time Filter Applied to Advertisements

Active Points of Interest and Active Ads

Here we present two views that provide data about comprehensive data about accessible points of interests, ACTIVE_POI, and advertisements, ACTIVE_ADVERTISMENTS, to the LBSs. ACTIVE_POI provides comprehensive information about accessible points of interests that are in the active profile a particular user. Table 4.14 presents the view. ACTIVE_ADVERTISMENTS provides comprehensive information about points of interest that advertise and the associated advertisement.

SELECT u_id, uuid, owner_organization, owner_id, email_address, contact_info, remark, creation_date, modification_date, booking_url, info_url, type, name, description, sortorder, place, road, housenumber, subcity, floor, postalcode, city, country, x, y, image_data, image_id, alternat_text, **FROM** AVAILABLE_POI, PROFILE, ACCESSIBLE_POI **WHERE** AVAILABLE_POI.uuid in (SELECT poi_id from ACCESSIBLE_POI) AND AVAILABLE_POI.type=PROFILE.p_type

Table 4.14: View ACTIVE_POI.

4.2.4 The Justification

As explained, the data model consists of two parts: the internal and the external part. The internal part is set of tables where all facts are stored in a way that database maintenance is simple. The external part is a set of views and materialized views of the internal part. Views and materialized views are provided to

```
SELECT ad_area, ad_title, ad_description, u_id, uuid,
       owner_organization, owner_id, email_address,
       contact_info, remark, creation_date,
       modification_date, booking_url, info_url,
       type, name, description, sortorder, place,
       road, housenumber, subcity, floor, postalcode,
       city, country, x, y, image_data, image_id,
       alternat_text,

FROM   AVAILABLE_POI, PROFILE, ACCESSIBLE_ADS,
       ACCESSIBLE_POI

WHERE  AVAILABLE_POI.uuid in (SELECT poi_id from
       ACCESSIBLE_POI) AND POI_ADs.ad_id in
       (SELECT ad_id from ACCESSIBLE_ADS) AND
       AVAILABLE_POI.type=PROFILE.p_type AND
       AVAILABLE_POI.uuid=POI_ADs.poi_id
```

Table 4.15: View ACTIVE_ADS.

support simpler software development and more efficient query processing. In this section, we justify these claims.

The Internal Part

The internal part of the model supports simple update with respect to content and position of content in a road network. In order to add to content, only two insert operation would need to be performed: insert a record in Table CONTENT and insert a record in Table POINT_EVENT or LINEAR_EVENT. Similar steps are needed to delete content. Changes to a road network are somewhat more complicated but road networks do not change often and most of the changes are not significant. It is important to realize that most information about road network is stored as content and that changes to segment representation are made only when geometry of the road network changes. We distinguish between two types of changes of segment representation: adding a new road and reshaping of the existing roads. Adding a new road would not cause a change in the existing CONTENT tables and only new tuples would need to be inserted to describe the new segment. Modifying an existing segments is not hard either. There are two things that need to be done here. First we need to modify tuples in table SEGMENT. Depending on whether segments are joined or divided into several segments, tuples are deleted or added in table SEGMENT respectively. Also, CONTENT tables need to be updated which requires resetting seg_id and offset attributes in the POINT_EVENT and LINEAR_EVENT tables. Reader should realize that all of the steps needed to modify segment representation require simple computation and single scans through the relevant database tables.

The internal part of the model is easy to extend. To add a new category of content, one simply needs to add the new content entity to the model and a new relationship between the content and the segment representation which will not cause changes to the existing model. If the new content needs to be described

with more then one entity, then the entities describing it will be related to that content and possibly to other content; but, they will not cause changes to the existing model.

The External Part

As already mention, an LBS uses data that describes road network and content. Development of an LBS on the top of the internal part has following weaknesses.

First, depending on purpose of an LBS, each LBS needs a subset of data from the base tables. Consequently, developer would be exposed to the data that does not concern the LBS. Second, the relevant subset of data may be also large and hard to comprehend. For example, in order to compute allowed turns in a road network, developer would need to query table `SEGMENT` and different `CONTENT` tables. Moreover, the result of the queries would need to be processed in order to compute desirable result.

One way to avoid these pitfalls is to provide different materialized views to different LBSs. Each materialized view is a different representation of the road network or content that summarizes necessary data about a road network or content from a particular LBS perspective. It is also important that a particular materialized view is created in a way that relevant queries become simple and efficient. We demonstrate this concept by an example. Consider the query that retrieves content associated with an edge in the graph representation of a road network. This type of query is relevant to the nearest neighbor and the shortest path procedures. First we explain what queries are needed to retrieve this information if the information is retrieved directly from the internal part; then, we explain what queries are needed to retrieve the information if the information is retrieved from a materialized view.

Suppose that a user is traveling on an edge `e`. Then to retrieve content associated with the edge `e`, we need to go through three step. The first step is to associate the edge with the segment. The query in Figure 4.15.a does this. The second step is to compute offsets of the two nodes associated with the edge. The queries in Figure 4.15.b do this. At last, the query in Figure 4.15.c retrieves content associated with the edge. If we were to retrieve the same information from the materialized view `RN_POI`, then the only query needed is the query in Figure 4.15.d. From this example one can see merits of the external part.

The graph representation of the road network has has same effect as the `RN_POI`. For example, to compute allowed turns at an intersection, one would have to query the segment representation of the road network and different content tables that store data about traffic directions, traffic signs and similar.

In Section 4.2.2 we described schema that models points of interest and in Section 4.2.3 we described materialized view that pre-joins the information about the point of interests in a single table `AVAILABLE_POI`. If a LBS queries the internal part to retrieve the information, then join in the Table 4.10 needs to be executed every time the query is executed. But, if the LBS queries `AVAILABLE_POI`, then the join is executed once a day because the information about points of interest changes once a day.

The external part is not composed only of materialized views. If the informa-

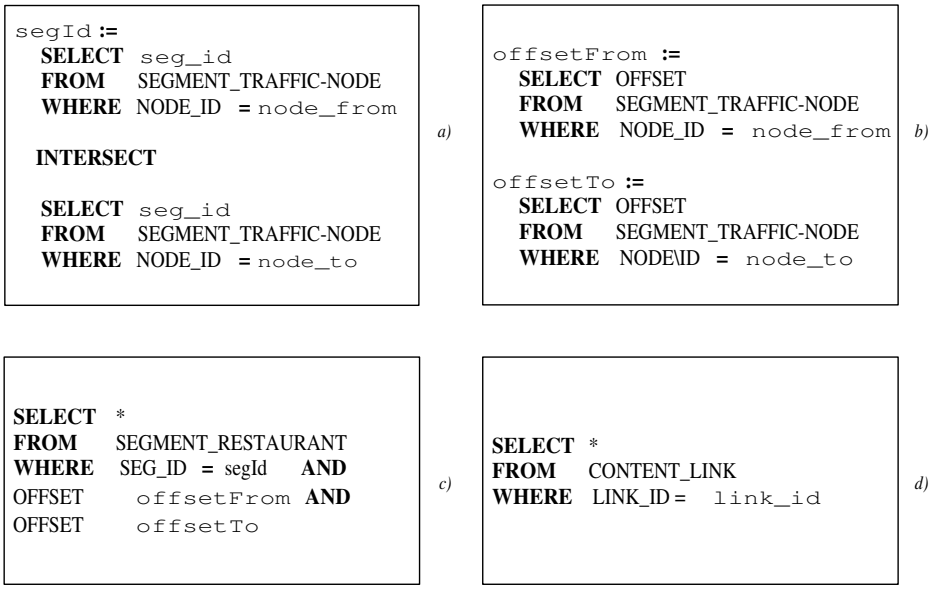


Figure 4.15: Query Simplification

tion in a table in the internal part is volatile, then the queries that access data from this table should not query materialized views but only views. This is the case with the temporal data and profile management. Earlier, we presented two views that use time and profile filters to provide the currently accessible advertisements (`ACTIVE_ADS`) and currently accessible points of interest (`ACTIVE_POI`) that are in the user's profile. These views do not improve query efficiency but rather provide a friendly interface to a complex queries and therefore simplify software development.

4.3 The Value Chain

So far we presented the source data, the format in which the data are stored and the format in which the data are delivered separately. In this section we present the entire data value chain. The links in the data chain are either data or procedures that keep the data up to date. The first link in the data chain is the source data and the last link is the external part that serves data to a particular LBS in the format most suitable for the particular LBS. The links in the middle are the internal part and two sets of procedures that keep the data up to date. Following describe the value chain in two steps:

- transformation of the source data to the internal part, Section 4.3.1
- transformation of the internal part to the external part and delivery of the data to the location based services, Section 4.3.2.

4.3.1 The Internal Part Maintenance

As presented in Section 4.1, the system takes input from different sources. The source data is the first link in the value chain, see Figure 4.16. The second link in the chain is a set of procedures that take as input the source data, process it and store it in the database tables in the third link. The database tables in the third link form the internal part presented earlier. The second link in the chain is capable of generating and updating the third link.

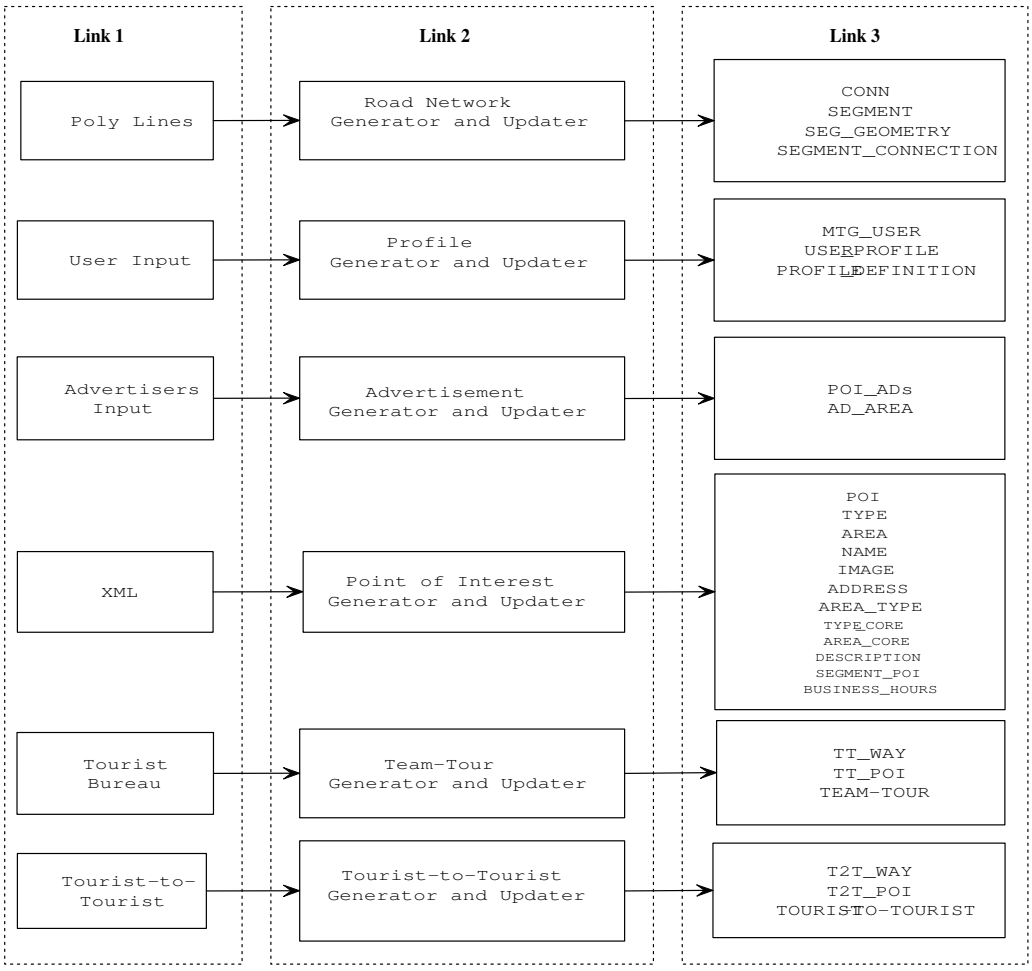


Figure 4.16: Value Chain: Internal Maintenance

4.3.2 The External Part Maintenance

The external part is the fifth link in the chain. As described earlier, the external part is a as set of views and materialized view that are derived form the internal part and maintained through SQL queries and sophisticated procedures, see

Figure 4.17. The queries and the procedures form the fourth link in the chain. Second and and fourth links of the chain are the mechanism that keep the data up to date.

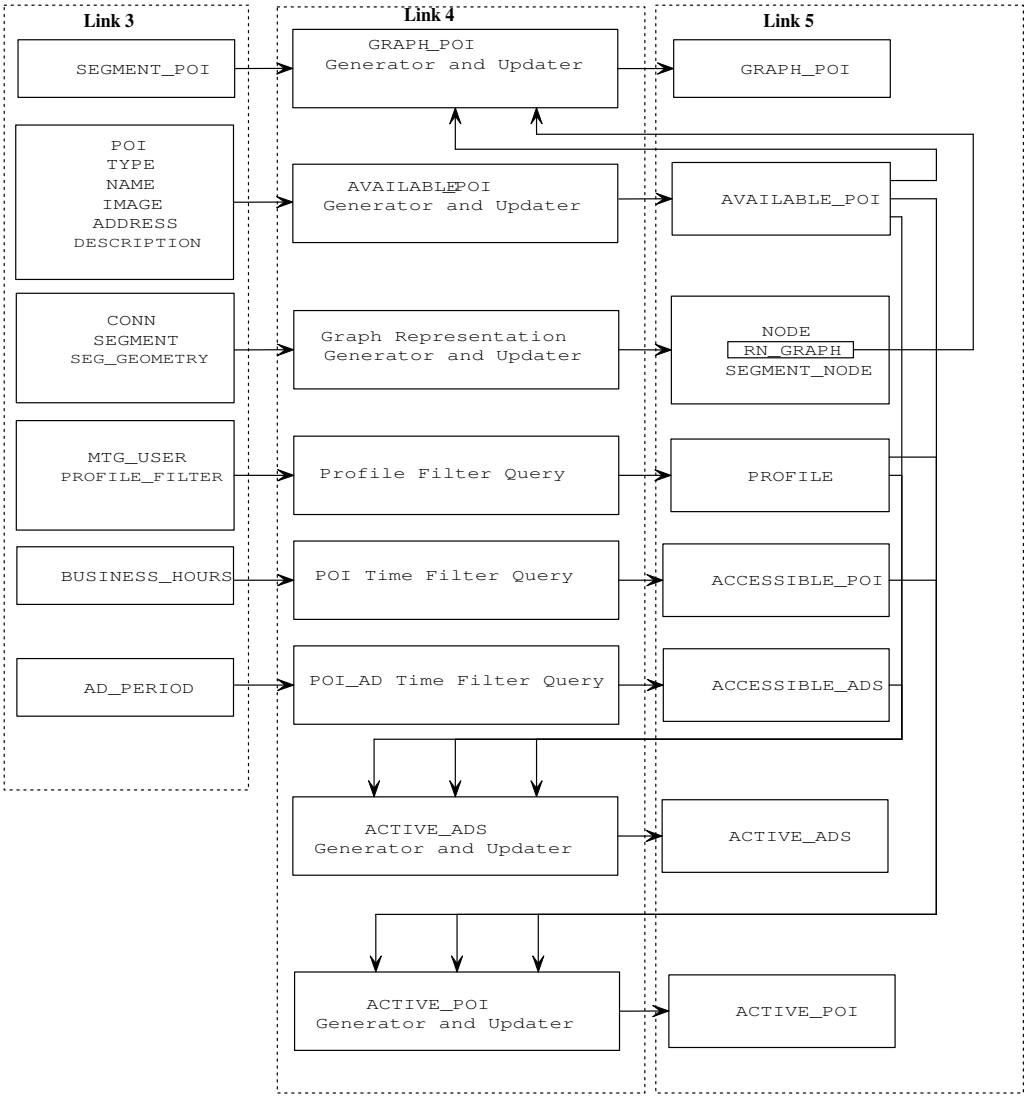


Figure 4.17: Value Chain: External Maintenance

As explained earlier, the LBSs mainly access the external part. Figure 4.18 illustrates the external part, LBSs and the relationship between them.

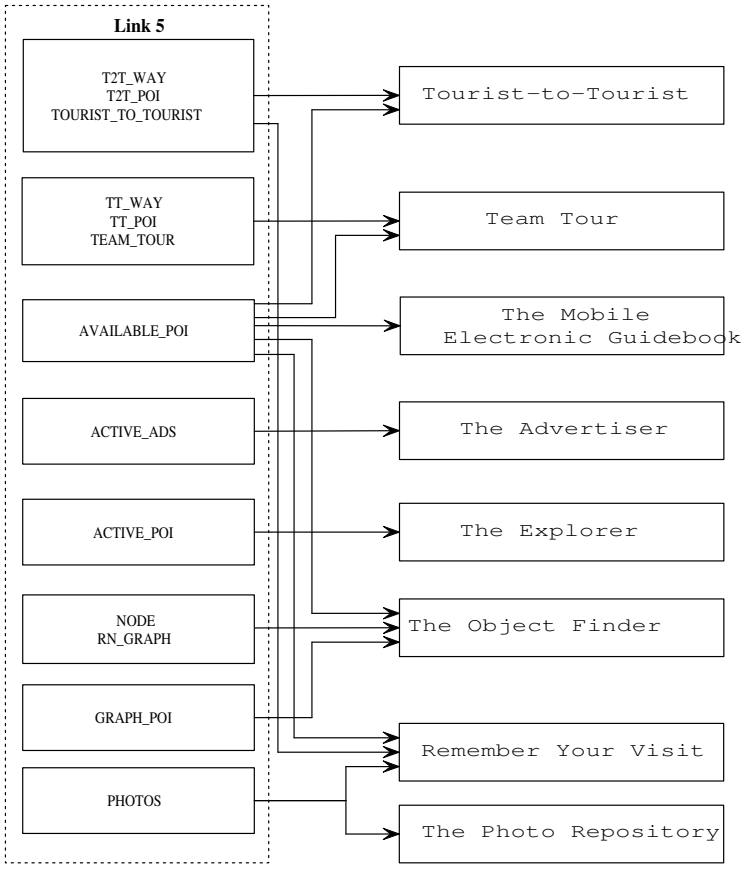


Figure 4.18: Value Chain: Data Delivery

Chapter 5

Client Application

In the project report from previous semester we described several general functionalities of a client since our prototype was in a very early stage of development [26]. Client's purposes were mainly to successfully communicate with a server. The data that was exchanged were current user's position which client had to report to the server. This was followed by server sending a scalable vector graphics (SVG) page to the client. The SVG page contained some information on the available services as well as references on how to access them (web links). According to user's geographical location at the time of service request, it was server's task to compute how client should retrieve a map from the web service provided by Kort og Matrikelstyrelsen and generate a link that was included in the SVG page.

We have continued working with the SVG browser on the client and extended the functionalities to accommodate the needs described in Chapter 2. Following sections describe the client architecture and functionalities.

5.1 Client Architecture

Figure 5.1 illustrates an extract of the implemented classes in addition to the SVG browser. SVG browser is using BitFlash Player technology to parse and display SVG pages. Their software development kit provides interfaces for manipulation of XML based language (SVG DOM). Furthermore the SVG browser provides interfaces for communication with a GPS receiver over a bluetooth network and connection to mobile Internet — GPRS. These functionalities provide an architecture that is extendible with desired services.

The extract of the implemented classes from Figure 5.1 shows that we have implemented all our services as one extension module to the SVG browser. Our services, such as the digital guide book, tour recorder, etc. could easily be implemented separately. However we found it much easier to implement all our services as extensions to the general service from our previous work [26], since it provided basics for the functionalities that all other services are based upon — map updating, SVG page updating and reporting to the server.

Following sections describe classes that provide general functionalities of the

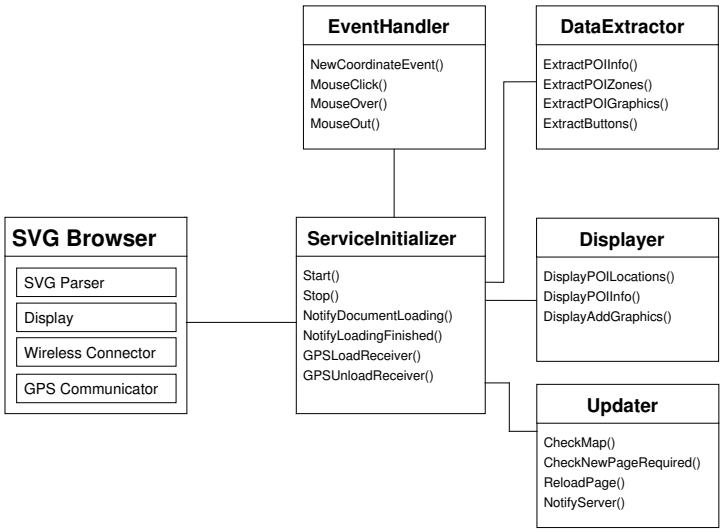


Figure 5.1: Class Diagram Representing Client Application

implemented extension to the SVG browser.

5.1.1 ServiceInitializer

As the name implies `ServiceInitializer` class is used to initialize the services. Its purpose is to extend several interfaces provided by the SVG browser that are needed by the rest of the client architecture. It also controls the application flow by making calls to the functions of the underlying classes.

When SVG browser application is started it recognizes the extended services and prompts the user to start a service (illustrated earlier in Figure 2.2). After user chooses a service `Start()` function is called and `ServiceInitializer` takes over control of the browser. The GPS receiver is initialized as well.

Every time the GPS receiver reports arrival of a new coordinate, a `NewCoordinateEvent()` is generated in the application. The event is handled by the `EventHandler` class which pulls out the values of the retrieved GPS coordinate. These values are sent to the `Updater` class which finds out that no previous coordinate exists — hence a new page has to be requested from the server. Finally `ReloadPage()` collects the data that describes the current state of the client and sends a HTTP request to the server with appropriate parameters, so the server knows that new user is active and requesting services. At the initialization stage of the client state data are user's id and geographical location.

When `ReloadPage()` function is called, a new page (start page) is requested and before an answer arrives from the server (in form of a service or that the server is down etc.) the application has nothing to do, so it is stalled for a period of time. This means that the control is returned to the SVG browser — more exactly the module for GPRS connection. When a server sends an answer in form of an SVG

page, once it passes the SVG parser, `ServiceInitializer` gains the control of the application again since it extends the interfaces `NotifyDocumentLoading()` and `NotifyLoadingFinished()`.

In case of no reply from the server a timeout occurs in the module for GPRS communication which will returns control to the `SeriveInitializer` by generating an SVG error document — `NotifyLoadingFinished()` is activated.

When server answers with a document and it is completely retrieved by the client, it is validated, parsed and SVG graphics displayed by the parser and display module of the SVG browser. While this is taking place `ServiceInitializer` perceives it as if the document was still loading — hence it is still stalled. After the document is validated, `NotifyLoadingFinished()` is invoked and control returned to the service. In case of document being invalid, an SVG error page is generated and returned to the service. Following sections describe how client components handle data in valid SVG pages.

5.1.2 Updater

`Updater` class has two purposes — to check whether an update of data or the map is required and if so to initiate a connection to the internet in order to retrieve new data. Map retrieval is covered in Section 5.2.

`CheckNewPageRequired()` function implements the part of the algorithmic strategy related to the client that is discussed in Chapter 7. One of the objects that `DataExtractor` extracts from the SVG document is the update threshold which was decided by the continuous update method based on the density of the points of interest, Chapter 7. Whenever points of interest are present in the SVG page this check for update will be initiated on every coordinate event.

There are two ways of communicating with a server. Client can make reports to the server. For instance, when in tour recording mode (see Figure 2.3) where user's movement history is recorded by the server, client periodically reports geographical locations to the server and facilities that were of user's interest. On these occasions the client does not receive an answer from the server. The reporting is implemented in the `NotifyServer()` function which sends all the GPS related data that is cached in the client's main memory for the purpose of server's recording of it.

The other way of communicating is by sending a request to the server in which case the server responds with an SVG page. This can be initiated by one of the functions that check for necessary update, by user requesting a path to one of the provided points of interest or by user requesting a new service.

`NotifyServer()` and `ReloadPage()` functions are extensions to `DataRetriever` interface provided by the SVG browser's module for Internet connection.

5.1.3 DataExtractor

All the data that is used by our services has to bypass the SVG parser, since SVG parser will go through a page, find all graphical elements and display them. The data needed by the implemented services must not be modified in any sense by the SVG browser in order to be delivered to the classes for data processing.

Once the data is passed to the classes, we can use SVG DOM API module which is provided by the BitFlash Player to find the data that server has sent to the client. SVG DOM builds up a DOM tree which is an internal representation of data in memory that is structured as a logical tree. The root of the tree is *Document* and the DOM specification is mostly organized around the descendants of the *Document* — namely *Node*s. Three functions are mostly used to traverse the nodes of the DOM tree and to extract the values of interest — `getFirstChild()`, `getNextChild()` and `getNodeValue()`.

DOM trees are very well known from XML technology which is widely accepted; mostly used as a standard for data exchange between applications from different vendors or applications developed for different platforms — applications using different business rules for data storage. The only way for such applications to understand each other's data is by parsing each other's XML documents. Hence by encoding data in SVG format and bypassing the SVG parser we are still using the XML technology which means that our system is not strictly dependent on the SVG browser. Many of the implemented rules for data exchange, which are the basics for the working services, are reusable in case of other types of clients. We know that hardware is rapidly getting stronger thus exchanged. Our platform leaves the field open for future development.

`DataExtractor` class is mostly consisting of helper functions that traverse the DOM tree, find elements of interest and extract the data from them or store pointers to them for quicker access. In Section 6.1.4 we look at an example of the SVG page that server returns to a client. Functions provided by the `DataExtractor` go into such fields of an SVG page.

`ExtractPOIInfo()` finds title and information of points of interest. The data is used for displaying so only pointers to the graphical representation are passed to `Displayer` class.

The purpose of `ExtractPOIZones()` is to find zones of delivered push–ads. In Section 6.1.4, Figure 6.3 we look at an example of an SVG page being sent to the client. Geographical zones related to the points of interest are encoded within `geo-locations` element. `ExtractPOIZones()` traverses the zones as a linked list and checks whether user is currently in one of the ad–zones from the list. In case that user is not in a zone it extracts the distance to the nearest zone and remembers user's position. These values are used as a threshold for when to perform the next check. On every coordinate event user's geo–location will be checked against these two values. When user's distance to the remembered location is at least the distance to the closest zone from the perspective of the remembered location, the zone list is traversed again. In this way we minimize the computational overhead that will otherwise occur on every coordinate event. Flowchart in Figure 5.2 illustrates the idea behind the algorithm.

`ExtractPOIGraphics()` looks for the graphical representation of POIs. Just as POI info data, once pointers for these are found they are sent to `Displayer`. Furthermore mouse events are assigned to these elements (e.g. if a user clicks on a representation of a point of interest, name and information associated to the POI will pop up on the screen; Figure 2.3).

`ExtractButtons()` looks for the extra button graphics included in the SVG pages and assigns mouse listeners on their content making it interactive.

5.2 Retrieving and Handling Maps

We had four different ideas on how maps could be used by the client. We have experimented on three of them and implemented two.

- Include a link to KMS server into SVG pages that client would visit in order to get a map that will cover the screen.
- Upload maps covering areas of interest on the client and as user moves through the map covered areas maps are retrieved from the secondary memory.
- Map caching algorithm that converts user's surroundings in a grid system and keeps maps of visited areas. As user moves through the grid, corresponding maps are downloaded and displayed. In case user goes back to previously visited area local map pieces covering client's screen are retrieved from the secondary memory.
- User enters a route into a system. Route is converted into grids and required maps downloaded before even reaching the locations. As the user moves along the route corresponding maps are retrieved from secondary memory.

5.2.1 Temporary Maps of Screen Size

Ideally the task of map retrieval would be client's alone, however the implementation of the top most method from above is in our case carried out on both server and client, due to SVG browser's limitations which are covered in Section 5.3; more specifically SVG browser being unable to make hard modifications on the retrieved page and reload it. This means that a link to an image, either local or remote, cannot be modified by the client and then processed.

In order for the method to work, client and server have to agree on three parameters — size of the image (resolution), scale (in meters per pixel) and user's geographical location at the time of request. When client requests a page with a map, the server will calculate correct link request for the KMS server according to the specifications [18] and include it in the SVG page that is sent back to the client. Client will parse the link and download the content. Examples of a link and corresponding image are given in Figures 5.3 and 5.4.

In order to present user motion in this mode, movement of user–icon has to be animated and all other objects on the screen fixed. The method is fairly simple to implement. The client holds on to the coordinate that was used at the map request time and by knowing the scale of the image we can calculate the top left map coordinate and translate the image and user–icon into the screen coordinate system.

In the examples from Figures 5.3 and 5.4 the scale is 8 m/pixel. The *bbox* argument in the link is the bounding box of the image requested from the KMS server. First two values are `x` and `y` coordinates of the lower left corner of the image and the other two are x and y of the upper right corner in UTM format. We can get the scale by subtracting the x coordinates and dividing them with

the width of the image. Hence, $(560645 - 559269)/172 = 8$. Similarly knowing the scale, user's position is easily translated into the screen coordinates.

```
userX_Screen = (userX_UTM - lowLeftX) / scale

userY_Screen = (upRightY - userY_UTM) / scale
```

On every coordinate event we calculate user's position with respect to the screen's coordinate system. We keep the of track whether a user has reached the edge of the screen and if so update the map. The drawback of this method is that it is only suitable for pedestrians. Any faster moving user would reach the edge of the screen very quickly. This situation can be remedied by having maps of smaller scale, however the maps would be less detailed and only main roads would be included. Since Mobile Internet technology is still relatively new on the market and slow, faster moving users would spend more time on downloading the maps rather than viewing them. As computer technology is growing stronger and faster then ever before, so this might not even be an issue in a near future.

5.2.2 Fixed Local Maps

Maps demand most on–line time since once the map retrieval method is chosen, they will most likely be of fixed size and user's are very likely to exit their covering area unless the maps are very big. It is also a service that will be used most of the time on the client.

In order to minimize the on–line time we still wanted to provide a solution such that maps would not need to be requested very often. Since our content data is covering the city of Aalborg we designed a solution that will work for a specific area of Aalborg. However, any other location which is of relevance to service providers can be handled just the same. Maps of Aalborg are uploaded on the client and used whenever applicable. In order to bypass the limitations of the SVG browser, like in previous method server and client have to agree on map dimensions and the area they are covering.

Simple solution for this method would be to have a single map that would cover entire city. We discus the main and secondary memory issues of the client in Section 5.3. By looking at Figure 5.5 it seems that it would not be a problem to have one jpeg image of scale 8 m/px to cover 64 km^2 since it will only occupy 181 kb of main memory. The problem is that all images are converted to bitmap format before being displayed on the screen. A resolution of 600 x 600 pixels causes already serious reductions in performance.

We strived to have approximately 800 kb of free main memory and chose to cover approximately 50 km^2 of Aalborg that are mostly covered with some content data. This resulted in having four images of 500 x 500 px with scale of 8 m/px taking approximately 730 kb in bitmap format.

To animate user movement in this mode, we chose to do it in the opposite way as was described in the previous section. We fix the user–icon in the center of the screen and translate movement of map and POIs with respect to user's

position.

Working with one image at a time requires that images overlap. If we chose to cover the entire screen in every possible situation we would have very little unique space for each map, since we simply could not have made maps bigger because of the memory restrictions. We chose overlaps of 88 px in x axes or 704 m while y is 94 px or 752 m. The equation for overlaps is given in Figure 5.6.

Figure 5.7 illustrates 4 maps with overlapping areas. Maps' boundaries are colored in 4 colors red, blue, green and purple. Black line between overlapping areas is the map zone delimiter for each map. Small rectangle with the dot in the center is the client screen and white area is the visible map area. When user crosses the delimiting line a request for new map is sent to the server. Between 25% and 50% of the screen will be covered with map on line crossing. The worst case occurs when user reaches crossing point of the delimiting lines.

5.2.3 Dynamic Map Caching in Grid Representation

In case of many local maps, method from Section 5.2.2 is space inefficient, since there is a lot of overlapping areas. Better solution is to have connected maps filling out the grid space like in previous method. Animation in this method works just like in the previous one. User icon is fixed while maps and POIs are indented.

The method works by checking whether a map has been downloaded previously, corresponding to user's location, and if so loaded into memory. Milestones as for how maps should be downloaded have to be determined. For instance a map should be 2000 m x 2000 m and its top left coordinate dividable by 2000. So if user's coordinate is (559957, 6319466) a check is made whether map named 558000_6320000.jpg exists in the local repository and if not such map is downloaded and stored under the same name.

Next step is to cover the entire screen, and as we have seen in previous method it is not possible with a single map with no sharing areas. The idea is to have a map peace slightly bigger than the screen area. In this case single map can cover the entire screen whenever user exits the map area, next map peace is loaded into memory as well. Correct map piece is found by checking locations of the four screen corners. They are translated to UTM coordinate system and compared with map–grid rules. Figure 5.8 illustrates the idea.

This method gives only sense when it is server independent, since its purpose is to minimize the connection to the internet. Otherwise whenever user reaches a map boundary the server would be consolidated and we are back to the very first method. It is also essential that the developer has complete control of the memory resident objects in order to load and unload correct maps at any time.

5.3 Limitations of SVG Browser

SVG browser is still at a prototype stage, so even though it provided an easier approach for our implementing of the services on the client, there are still couple of lacks regarding its functionalities.

At the current stage the only way of reading a server response is through SVG pages. This fact is a big deficiency with respect to the size of data that is being exchanged. Every time a new page is downloaded it is also automatically displayed by the BitFlash's SVG Player. This means that the old page has to be discarded, hence all the graphics are lost. So every time server has to generate an SVG page, it has to include all the graphics that the client was displaying a moment before requesting an update.

An SVG page must have a reference to a local image in order to display it. All the references to local images are automatically processed on page load, hence even if images are not visible (e.g. maps covering areas outside current view) they will be loaded into main memory of the client. We could not find a way to bypass the tight implementation of page loading in order to free up the main memory and load only parts of data that might be needed at the moment. Hence a map caching algorithm is not implementable without dramatically decreasing the performance of the client.

Main and secondary memory of the client are one and same thing. In the next version of SVG browser access to the extended memory would be desired. We have noticed that performance of the SVG browser dramatically decreases as the size of available main memory drops. There are approximately 3 Mb of free memory when the browser is installed and when the application is run it occupies approximately 1.5 Mb of main memory. When the amount of main memory drops down to 600 kb, the client response time becomes very slow.

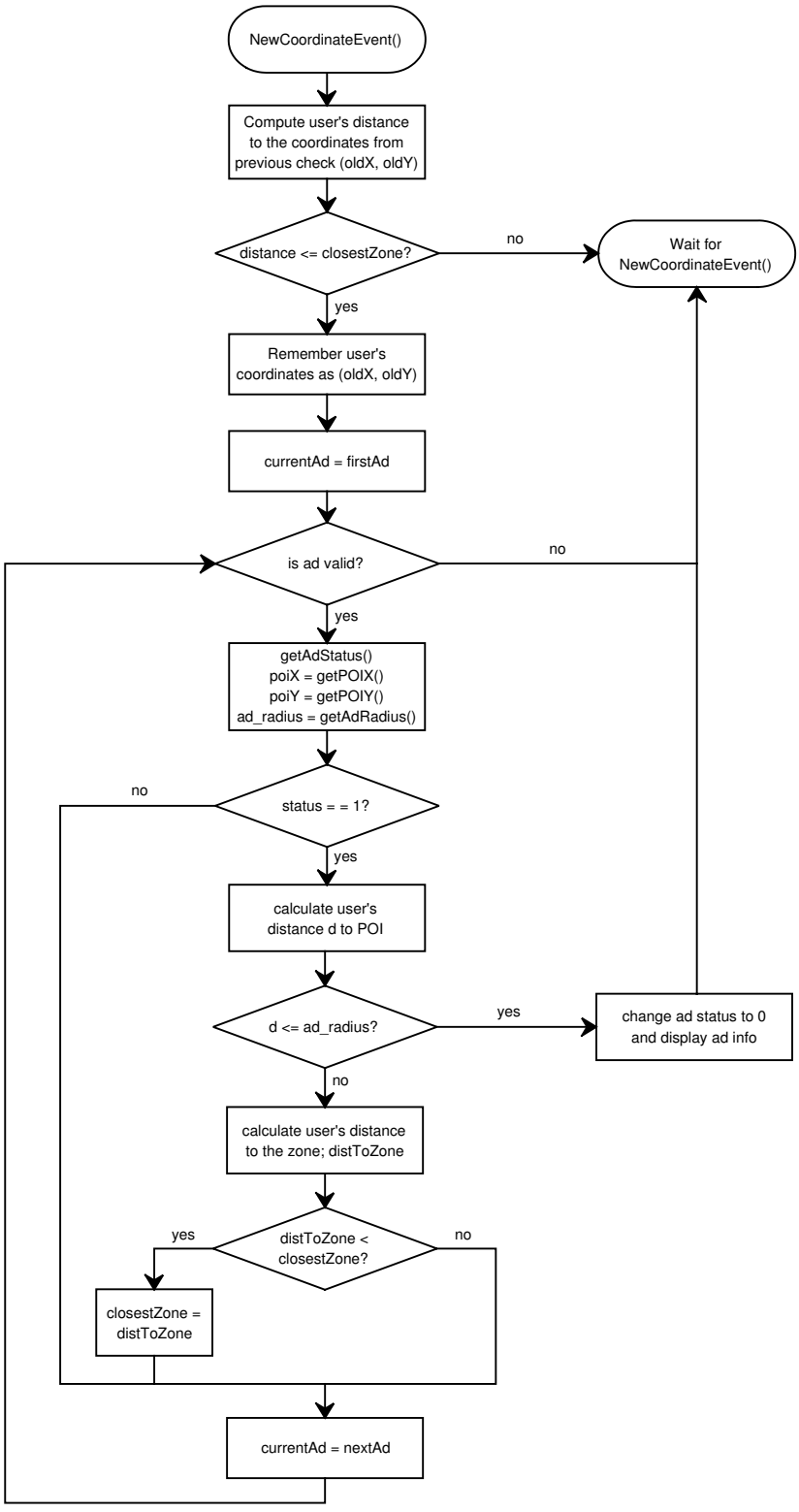


Figure 5.2: Flowchart of ExtractPOIZone() Function

```
<svg>

   ...
   <image id="bg_map" x="2" y="2" width="172" height="184"
      xlink:href="services/aalborg_guide/srv.jpg"/>

   <text id="bg_map_url" visibility="hidden">
      http://kortforsyning.kms.dk/service?
      wmtver=1&request=map&servicename=D_50&srs=EPSG:32632&
      bbox=559269.0,6318730.0,560645.0,6320202.0
      &width=172&height=184&exceptions=INIMAGE
      &format=JPEG&jpegquality=20
   </text>
   ...

</svg>
```

Figure 5.3: Example of a Link to KMS Server Included in the SVG Page

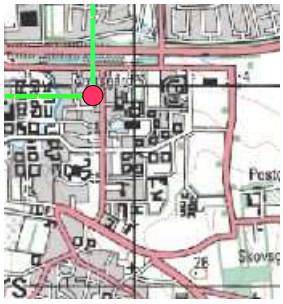


Figure 5.4: Image Corresponding to the Link from the Example of Figure 5.3

px/side	125	250	375	500	625	750	875	1000
JPEG kb	4	13	29	52	79	112	148	181
BMP kb	46	184	414	733	1146	1650	2246	2930
km^2	1	4	9	16	25	36	49	64

Figure 5.5: Relation between Image Resolution and Size in Bitmap and Jpeg Format with Image Scale of 8 m/px

```
overlapX_UTM = (screenWidth / 2) * scale

overlapY_UTM = (screenHeight / 2) * scale
```

Figure 5.6: Equations for Calculation of Minimal Map Overlap in UTM Format

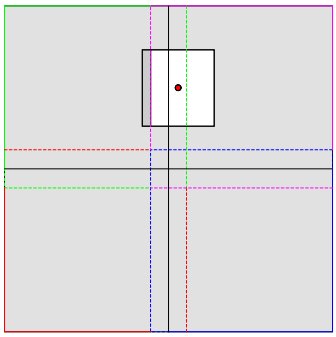


Figure 5.7: Overlapping Images

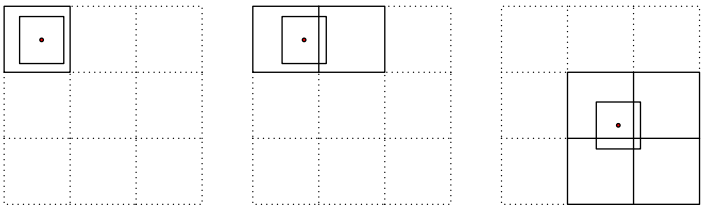


Figure 5.8: Dynamic Map Caching

Chapter 6

Server Components and Services

As mentioned earlier in Chapter 3, our server architecture is divided in three tiers — data, component and service tier. We have already explained the data tier in Chapter 4. In this chapter we cover the details of the component tier and service tier.

6.1 Components

The purpose of the components is to provide functionalities that can be reused to compose different location based services. The choice of the components reflects on the queries that a tourist can ask in a road network. Examples of these queries were given in Chapter 1. Reusability of the components allows rapid development and integration of location based services. Following sections describe the components providing the functionalities of the server logic.

6.1.1 Map Matching

As already mentioned, user's location is essential in context of LBSs. The system we developed in this project serves the users in a road network. Therefore, the system must be aware of the users location with respect to the road network. In our case, the user's location is reported by a GPS receiver. Ideally, the location provided by GPS could be used to locate where is the user in the road network. However, there are two obstacles to carry this out: inaccuracy of GPS receivers and inaccuracy of road networks. To deal with this problem, we developed a `Map Matching` component that takes GPS data as input and outputs the user's position in the road network via public function `mapMatch(GPS)` that uses map matching techniques. Since this is one of essential components in the system, we took in consideration many factors that have influence on the result of this functionality and developed a sophisticate map matching techniques that provide accurate results. Chapter 8 presents the details about the map matching.

6.1.2 Space Filter

As already mentioned, we use time, profile and space filter to determine a relevant subset of data for individual users. The time and profile filters are described in Chapter 4. Here, we describe the basic functionality of the space filter.

The space filter determines what points of interest are within a geographical window where the user is placed in the center of the window. Here we present three public function in component `Space Filter` that provide the functionality needed to determine the geographical window and the points of interest within it. Later in Chapter 7, we describe the techniques used in more details. The three public functions are: `getActive()`, `getAvailable()` and `getPOIAlongPath()`.

`getActive()` is used in services where users' profile plays a role. The function applies the space filter to advertisements and points of interest in views `ACTIVE_ADS` and `ACTIVE_POI`. The data from `ACTIVE_ADS` and `ACTIVE_POI` are retrieved trough functions `getActiveAds()` and `getActivePoi()`, from Database Interface component, described in Section 6.1.8, respectively.

`getAvailable()` is used in services where users' profile does not play any role. The function applies space filter to the point of interest in materialized view `AVAILABLE_POI` through function `getAvailablePoi()` from the `Database Interface` component.

`getPOIAlongPath` function is used to retrieves points of interest and advertisements along the path in the services that use a path.

6.1.3 Shortest Path

Component `Shortest Path` provides functionality that computes the closest path between two points in a road network. In the case of The Mobile Tourist Guide, the two points are the point at which the user is placed and the point at which the point of interest that the user requested is placed. This functionality is provided through the public function `getShortestPath()`. Figure 6.1 illustrates the component.

To compute the shortest path, the component uses the graph representation of a road network, materialized views `RN_GRAPH` and `GRAPH_POI` that associate points of interest with the graph representation. Two functions in the `Database Interface`, `getOutgoingEdges()` and `exploreEdge()`, provide functionality that retrieves all edges with same attribute value for attribute `node_to` and content associated with an edge respectively. The figure also illustrates the private functions used internally by the component. `getShortestPath()` takes as input point at which the user is placed and `poi_id` of the point of interest, and outputs the shortest path between them.

Before we describe how the `getShortestPath()` functions, we define `path`, `distance` and `path_record`. The `path` describes how a user can reach from one point to the other through the road network and is a sequence of edges where an edge is a row of `RN_GRAPH`. Path also satisfies the following property: edge_i`.node_from` is the same value as edge_{i-1}`.node.to`, where $1 < i \leq n$ and n number of edges in the path. The distance is sum all lengths of links in a path. `Path_record` is record with two fields: `path` and `distance`.

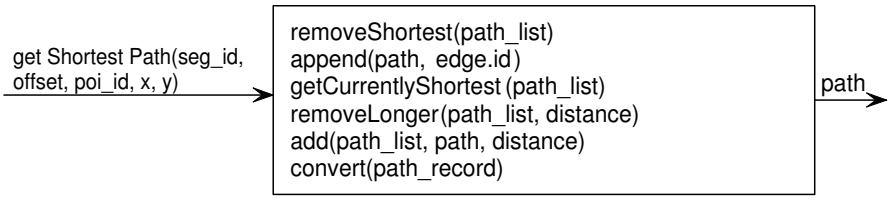


Figure 6.1: `Shortest Path` Component

`getShortestPath()` attempts to reach the point of interest by attempting to explore all paths that can be reached from the ending nodes of the edge that the user is traveling on. The paths are stored in the variable `path_list`. When the point of interest is found, `getShortestPath()` stores this path in the variable `result`. Then, the algorithm attempts to reach the point of interest by some other path that is shorter then the path in `result` found so far. `getShortestPath()` does this by removing all entries in the `path_list` which distance field is greater the the distance field of the `result` and exploring paths that are shorter that the path in `result`. Once the `path_list` is empty, getShortestPath uses `convert` function to transforms the sequence of links into format that the user can understand. For example, `convert` can convert a sequence of links into sequence of coordinates that are displayed on a map, textual description of the path, voiced base description and so on. Table 6.1 shortest path presents the details of the function. At last, `getShortestPath` returns the path in desired format and terminates.

6.1.4 SVG Encoder

The task of the encoder component is to encode data in a format understandable by clients. In our case since we are dealing with SVG technology, this component will encapsulate the data processed by other components into SVG format. SVG format is mainly used to display graphics, however as a standalone technology the standard does not provide, capability for interaction. This means that in order to provide interaction, the graphical and non–graphical data must be reprocessed by the client. Hence, not only must the data comply with the SVG standard in order to be displayed, but an additional data encoding format must be invented so that data can easily be accessed by the client.

This component has therefore two purposes. One is to generate the SVG graphics that will be displayed by the client and another one is to encode the additional data that is processed by the client. Figure 6.2 illustrates the SVGEncoder component.

The component provides a single public function — `generatePage()` — which generates the entire page. Inputs of the component are `ads` and `POIs` objects and user's geographical location as `x` and `y` coordinates in UTM format. Objects `ads` and `POIs` are records of ad and POI data. `ads` consists of short and long ads, while POI data record contains ids, names, descriptions and UTM coordinates of the POIs.

```
getShortesPath(seg_id number, offset number, poi_id varchar2, X number, Y number)

  path_list;              list of path_ records
  result;                 a path_ records
  iterator;               Structure capabale of holding multiples records of RN_ GRAPH
  content;                Structure capable of holding a record of RN_ POI
  edge;                   Structure capable of holding a record of RN_ GRAPH
  shortest;               a path_ records
  found := false;         boolean
  poi_found := false;     boolean


BEGIN

    1. Compute the link on which the user is travelling and store it in edge;

    2. content := exploreEdge(edge)

    3. if poi_id in content then RETURN portion of the edge between the user and the point of
       interest;

    4. add (path_listedge.id, edge.length)

    5. while(NOT found)

        (a) shortest:= getCurrentlyShortest(path_list);
        (b) iterator := getOutgoingEdges(shortest.path.last_edge.node_to);
        (c) removeShortest(path_list);
        (d) while(iterator not empty)
              i. edge := iterator.nextEdge;
             ii. add(path_list, append(shortest, edge.id), shortest.distance + edge.length);
            iii. content := exploreEdge(edge);
             iv. poi_found := poi_id in content;
              v. if(poi_found and result.distance > (shortest.distance + edge.length))
                   A. found := true;
                   B. result := shortest;
                   C. append(result.path, edge.id);
                   D. result.distaance := result.distaance + edge.length;

    6. while(path list is not empty)

        (a) removeLonger(path_list, result.distance);
        (b) shortest:= getCurrentlyShortest(path_list);
        (c) iterator := getOutgoingEdges(shortest.path.last_edge.node_to);
        (d) removeShortest(path_list);
        (e) while(iterator not empty)
              i. edge := iterator.nextEdge;
             ii. add(path_list, append(shortest, edge.id), shortest.distance + edge.length);
            iii. content := exploreEdge(edge);
             iv. poi_found := poi_id in content;
              v. if(poi_found and result.distance > (shortest.distance + edge.length))
                   A. found := true;
                   B. result := shortest;
                   C. append(result.path, edge.id);
                   D. result.distaance := result.distaance + edge.length;

    7. return convert(result)

END getShortesPath;
```

Table 6.1: `getShortestPath()` Function that Computes the Shortest Path between Two Points in the Road Network

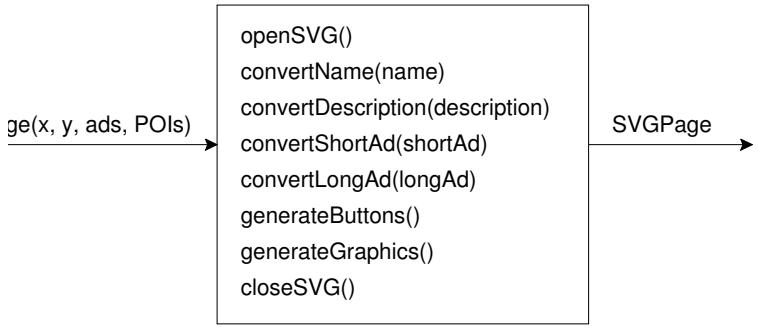


Figure 6.2: SVGEncoder Component

An example of an extract of a generated SVG page is illustrated in Figure 6.3. In the figure on can see that the data consists of three main elements. The top most element `background-graphics` is generated by the `generateGraphics()` function of the `SVGEncoder`. Second element `poi-info` contains various information about points of interest such as names, descriptions, ads etc. Finally the third element `geo-locations` contains geographical information about the included POIs. The data is generated by combinations of calls to the `convert` functions.

Considering the structure of the document it would be more logical to also include the geographical information together with the information data (e.g. a subelement of the `poi-info` element), however in our case the geographical data will extensively be traversed by the client (comparisons of distances, conversions to client's screen coordinate system, etc.) which was the reason for separation of this data. This way clients can access the data much faster by traversing it as a linked list and finally reprocess it.

The component is exclusively adapted to our system where clients are running on the SVG browser, however similar component is necessary for the systems of architecture similar to the one presented in Chapter 3 since the data needs to be encoded in the standard that the client is using. As we will see in Section 6.2 this component is run after all the data has been processed by the call to the only public function provided, making the component easily exchangeable in the system.

6.1.5 Map Handler

Purpose of this component is to make a map reference and encode it into SVG format which is the reason that the component is only used by the `SVGEncoder`.

The references can either point to a local maps on the client or as a HTTP request they can point to remote locations. In Section 5.2 we have covered the map issues and also, because of some of the limitations of the SVG browser, described why it is important to have this component on the server.

The component provides a single public function — `getMapReference()` — that takes `x` and `y` coordinates as input and accordingly determines whether user is

```
<svg>
   <g id="background-grapchics">...

   <g id="poi-info" transform="translate(7,8)">
      <g id="info-159">
         <text id="title" x="22" y="15" fill="white"
               font-weight="bold" font-size="16">Aalborg Zoo</text>
         <g id="info" transform="translate(12,0)" fill="black" font-size="12">
            <g id="p1">
               <text y="34">Aalborg Zoo opened in</text>
               <text y="48">1935. A lovely green garden</text>
               <text y="62">where you get close to 1600</text>
               <text y="76">exciting and endagered</text>
               ...
            </g>
            <g id="p2">
               ...
            </g>
         </g>
         <g id="push-adds" fill="black" font-size="12">
            <g id="short-add" fill="white">
               <text x="2" y="13">Aalborg Zoo, 30% off</text>
               <text x="2" y="27">rebate for students...</text>
            </g>
            <g ig="long-add">
               ...
            </g>
         </g>
      </g>
      <g id="info-3014">
         ...
      </g>
   </g>

   <g id="geo-locations">
      <g>
         <text id="stat">1</text>
         <text id="x">559500</text>
         <text id="y">6319500</text>
         <text id="radius">70</text>
         <text id="geo-id">159</text>
      </g>
      <g>
         <text id="stat">1</text>
         <text id="x">559900</text>
         <text id="y">6319800</text>
         <text id="radius">120</text>
         <text id="geo-id">3014</text>
      </g>
   </g>
</svg>
```

Figure 6.3: Extract Example of a Generated SVG Page

in area covered by a local map, and if not generates the HTTP request.

6.1.6 Path Converter

In this section we present the component that converts the path provided by the `Shortest Path` component in the format in which the result is served to the client. In our context the client understands SVG format. Public function

`convertToSVG` provides the functionality that converts a path to SVG format. If the format that the client understands changes, then new function that converts the path to the new format can be added to the component without modifying anything else.

6.1.7 Logger

The `Logger` component is illustrated in Figure 6.4. Its purpose is to store data about user's movement into the database. It has a single public function `doLogging()` which takes `GPS` object as input. The `GPS` object is a record consisting of seven attributes which describe information about the user — `x` and `y` coordinates at the time of generation of the `GPS` object, `speed` of the user, `direction` of movement, `time` of the object generation, `user_id` and `POI_id`. The last attribute is used in case that user is recording a route in which case the `POI_id` will be id of a POI that user has showed interest for by clicking on its icon representation.

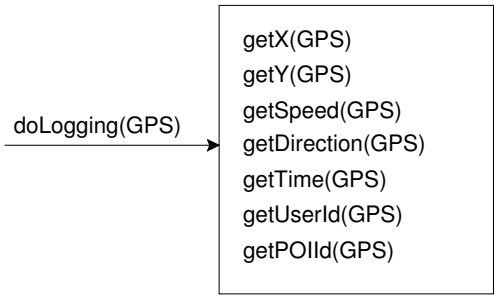


Figure 6.4: Black Box Model of the Logger Component

After decoupling the attributes, `Logger` will make a call to `deposit()` function of the `DatabaseInterface` component which will store the values into the database.

6.1.8 Database Interface

This component provides a set of public functions that retrieves data from the database. The database queries are made available through a component for two reasons: the queries become reusable and queries are easier to maintain. Figure 6.5 illustrates public functions in the `Database Interface` component.

6.2 Services

In Chapter 2 we described the services that make up The Mobile Tourist Guide. In following sections we demonstrate reusability of the components by combining them into the services.

Name	Input	Query	Used by
exploreEdge	edgeID	SELECT * FROM RN_POI WHERE edge_id = edgeID	Shortest Path
getActiveAds	u_id, x, y,n	SELECT DISTINCT ads_id FROM ACTIVE_ADS WHERE user_id=u_id	Space Filter
getActivePOI	u_id, x, y,n	SELECT DISTINCT poi_id FROM ACTIVE_POI WHERE user_id=u_id	Space Filter
getArea/Type	void	SELECT A.text, T.text FROM AREA A, TYPE T, AREA_TYPE AT WHERE AT.type_id = T.type_id AND AT.area_id = A.area_id	Guide Book
getAvailablePOI	x, y,n, input_type	SELECT poi_id FROM AVAILABLE_POI WHERE type = input_type	Space Filter
getAvailablePOI	POI list	SELECT poi_id, name, description, x, y FROM AVAILABLE_POI WHERE poi_id IN (POI list)	Object Finder
getAvailablePOI	type_id	SELECT DISTINCT poi_id, name, description, x, y FROM AVAILABLE_POI WHERE type = type_id	Guide Book
getConnections	segment1, segment2	SELECT a.conid FROM segment_connection a, segment_connection b WHERE a.seg_id!=b.seg_id and a.con_id=b.con_id and a.seg_id=segment1 and b.seg_id=segment2	Map Matching
getOutgoingEdges	nodeTo	SELECT * FROM RN_GRAPH WHERE node_to = nodeTo	Shortest Path
getProjection	x, y, geometry	SELECT PROJECT_PT(geometry,SDO_GEOMETRY(x,y)) FROM SDO_SEGMENT WHERE SEG_GEOMETRY=geometry	Map Matching
getSegment	x, y, geometry	SELECT seg_id, seg_geometry FROM SDO_SEGMENT WHERE SDO_WITHIN_DISTANCE(geometry,SDO_GEOMETRY(x,y))	Map Matching
deposit	GPS list	INSERT INTO LOGDATA VALUES (GPS list)	Logger

Figure 6.5: Summary of the Public Functions in the DatabaseInterface

6.2.1 The Electronic Mobile Guidebook

Figure 6.6 illustrates the sequence diagram of service The Electronic Mobile Guidebook. The service starts at the user's request, see Figure 2.1.c. To start the service, the client makes an HTTP request to start `startGuideBook()` which in turn starts the service. This service presents to the user all of the available points of interest; to do so, the service does not need any arguments. The execution starts with the call to the `Database Interface` component which returns all point of interest areas and types. This is done by `getAreas/Types()`. Then this information is encoded into SVG format by the public function `generatePage()`.

Once the information is in the SVG format, it is returned to the client. The

client then first presents all of the areas to user, Figure 2.2.a. If the user selects one of the areas, the client lists all of the types within the area.

The user can select a certain type to view all points of interest within that type. To do this, the client makes an HTTP request to start `startGuideBook(type)`. Then the service makes a call to the `Database Interface` component which trough `getAvailablePoi()` returns information about all point of interest within this type. The information is encoded into and SVG page via public function `generatePage()` in the `SVGEncoder` component. Once the information is in the SVG format, the client can display it, see Figure 2.2.b. If a user decides to view information about a specific point interest, then the information is presented to the user with the option to view the path to the point of interest, see Figure 2.2.c.

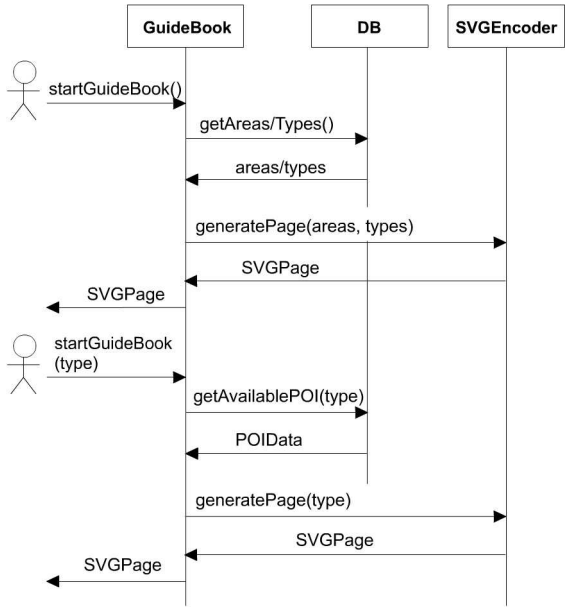


Figure 6.6: Sequence Diagram of The Electronic Mobile Guidebook Service

6.2.2 The Explorer and The Advertiser

The logics of The Explorer and The Advertiser services are illustrated through the sequence diagram from Figure 6.7. The Advertiser service is a sub service of The Explorer, hence depending on it. Examples of these services in use were previously seen in Figure 2.3.

When client initiates this service a HTTP request to start `startExplorer()` is made. The client will pass information about one's geographical location as `GPS` object and the user id — `u_id`.

Explorer service will first make a call to the `SpaceFilter` in order to get a list of surrounding POIs and ads based on the user profile consisting of the facility and ad ids. Since this information is in the database, `SpaceFilter` will make calls

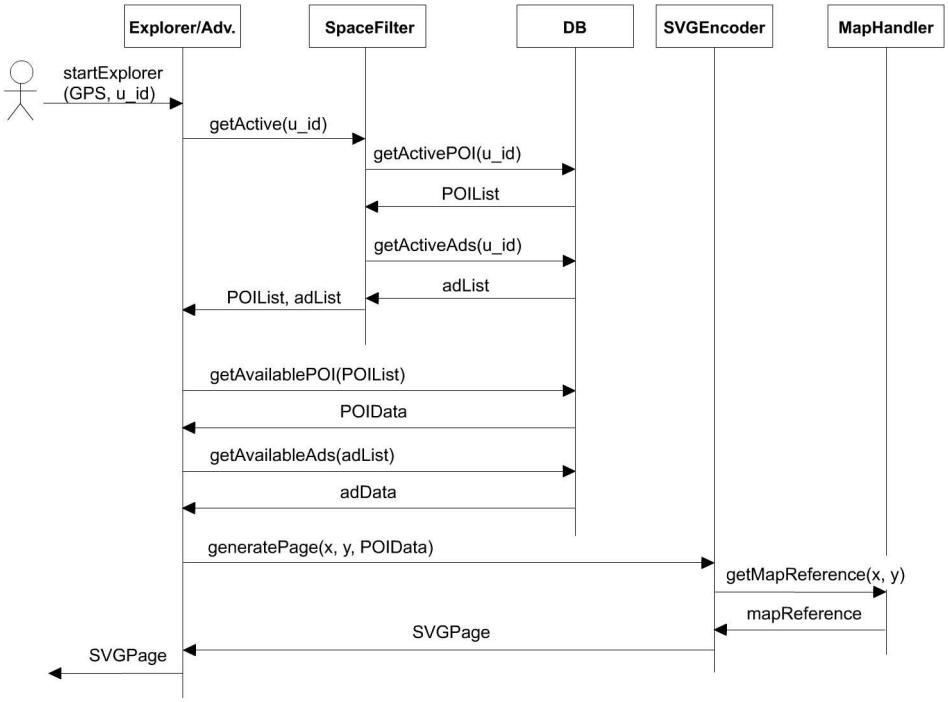


Figure 6.7: Sequence Diagram of The Explorer and The Advertiser Service

to necessary functions in the `Database Interface` component which performs the necessary queries.

Explorer now knows which POIs and ads are relevant to user's location and interest, therefore it can retrieve all the POI and ad data by making a calls to the database through the `Database Interface`. Once it retrieves the necessary data, it is converted in a format understandable by client — in our case by the `SVGEncoder` and finally returned to the client.

6.2.3 The Object Finder

The Object Finder service provides user with all the available types. After selecting a type and a number of facilities of interest, the user will receive `n` requested POIs of the selected type in the surrounding area. The sequence diagram in Figure 6.8 illustrates the idea.

User makes a HTTP request to the service and forwards `GPS`, `type` and `n` parameters which initiates the `startObjectFinder()` function. The service brakes up the `GPS` object and forwards `x` and `y` coordinates altogether with the requested `type` and the requested number of facilities to the `Space Filter`. The `Space Filter` makes a call to the `getAvailablePOI()` function in the `Database Interface` which returns a list of ids of the `n` closest POIs with respect to the user's location. The same list is then used by the service in order to retrieve ad and POI data.

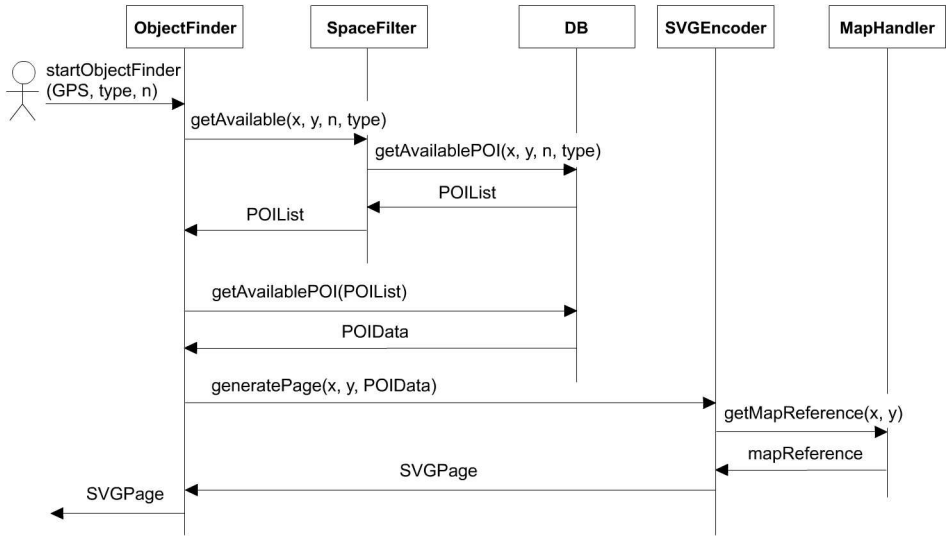


Figure 6.8: Sequence Diagram of The Object Finder Service

With the service having all the data it can finally generate the SVG page with included map reference since the clint is supposed to display a map and highlight the POI locations on it.

6.2.4 Path

`Path` service is used whenever user requests a path to some location. It is implemented as a sub service that can be accessed from other services. An example of `Path` service was given previously in Figure 2.3c. A sequence diagram of the service logic is illustrated in Figure 6.9.

The client initiates the service by making a HTTP call to it, which then starts the `startPath()` function. In order for the service to work, it must know correct positions of the two locations between which the path will be computed. It is therefore essential that client sends user's location and a reference to the destination point. These two arguments are a `GPS` object and the id of the destination POI (`destinationPOI_ID`) which can directly be used to reference a specific POI in the database. Once the service has these values it calls the `mapMatch()` procedure which identifies the user's location with respect to the road network in the database.

The `Map Matching` component makes necessary calls to the `Database Interface` in order to compute the segment corresponding to the user's location and the `projection` of the user in the network. Finally it returns projected `x` and `y` coordinate along with the `offset` and the `segmentID`.

`Path` can now pass these values to the `Shortest Path` component which generates the path between the requested locations and returns it to the service. In order to provide client with some POI data again, the `getPOIAlongPath()` of the `Space Filter` component is called which returns ad and POI data to the

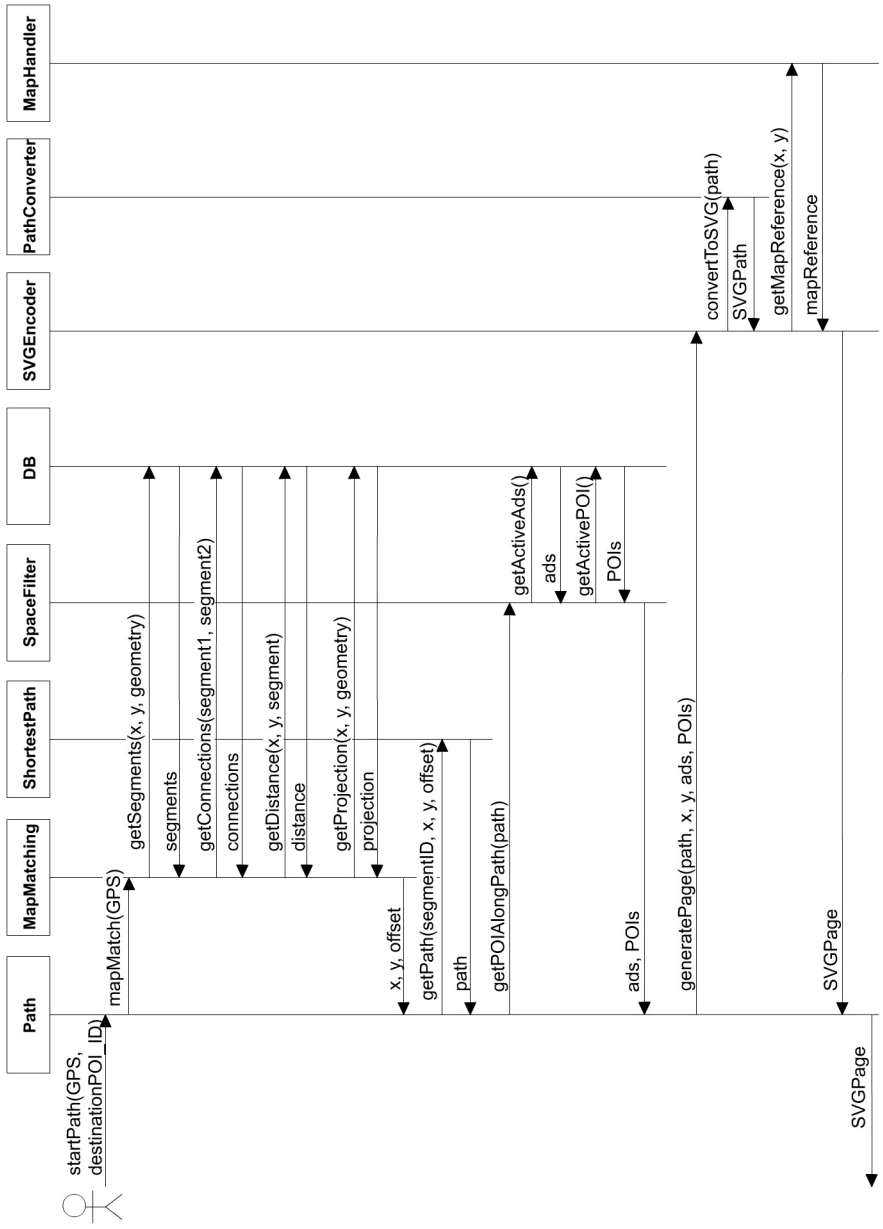


Figure 6.9: Sequence Diagram of The Path Service

service.

Finally having all the data the SVG page can be generated by the `SVG Encoder` component which this time also is passed the path representation. Before sending the complete page to the client, the `SVG Encoder` must convert the `path` into SVG representation and also get a reference to a map. These two operations are performed by executing `convertToSVG()` function of the `Path Converter`

and `getMapReference()` function of the `Map Handler`.

Chapter 7

An Update Strategy

The system we developed supports users of LBSs that are moving in a road network. When the user requests a service, the service responds with content. In Chapter 2 we mentioned that a service can not respond with all of the content from the database for the following reasons:

- Displaying all of the content would overwhelm the client's small screen.
- The transfer of large amount information over wireless connection may take long time.
- The client's memory may be overwhelmed.

Instead, the client is presented with a subset of the content determined by time, profile or/and space filters (in this section we refer to the subset as "relevant data"). In the Chapter 4, we demonstrated how to compute profile and time filters and we introduce the space filter in Chapter 6. The space filter determines a geographical window for a particular user and determines what points of interest are within the geographical window. The client is moving along with the user and the client will eventually leave the geographical window. This means that the client needs to request the service again in order to receive data relevant with respect to the client's new geographical position. We implement a strategy that attempts to keep the client up to date with respect to its location.

All components needed to implement the strategy were already presented. Some of the of the components are part of the client (`Updater`) and some are part of the server (`Space Filter`). Following first describes computation of the geographical window in the `Space Filter` and then we present the overall update strategy.

7.1 Computation of the Geographical Window

As already pointed out, `Space Filter` computes the geographical window. The size of the geographical window is determined by two factors: number of points of interest that the client requests and the point of interest density. Figure 7.1.a

illustrates computation of a geographical window. `Space Filter` first calculates initial predetermined geographical window. Then it increases or decreases in stepwise fashion until the number of point of interest requested by the client is reached. If the client requested the service from an area where the density of point of interest is high, then the geographic window will be relatively small. Figure 7.1.a illustrates an example where the density is low and the `Space Filter` needed to expend the geographical window two times before it reached the desired number of points of interest.

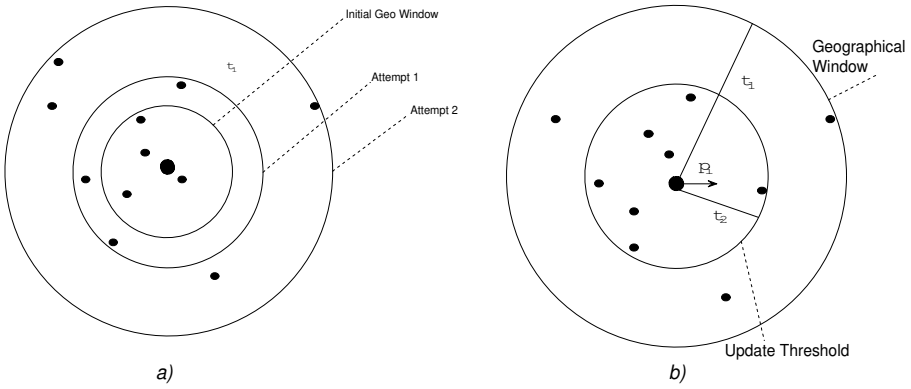


Figure 7.1: Illustrating: Computation of POI Threshold *(a)*, and POI and Update Thresholds *(b)*

7.2 The Update Strategy

After a service replies with content, the client receives, displays the content and monitors user's movement. While the users is moving, the client performs different tasks (records a route if tourist-to-tourist feature is active, displays advertisements and similar). One of the tasks is to check if the data in the geographical window is still relevant to the user. A client could simply wait until the user leaves the geographical window and then update. This strategy does not provides data relevant to the user continually because the data is relevant to the user only if the user is in the center of the window. Instead, the client can decide that the data is irrelevant when the user violates some update threshold. Figure 7.1.b illustrates an update threshold that includes 70% of points of interest. The service provides more relevant data, if the threshold is smaller. Figure 7.2 summarizes the update strategy. First the space filter computes the update threshold in the `Space Filter`. Then the Client uses functions `NewCoordinateEvent()` and `CheckNewPageRequired()` to check whether the update threshold was violated. If the threshold was violated, the client requests the service again. Reader should note that Figure 7.2 shows only subset of overall control flow relevant to the updating strategy.

Before we leave this section, we need that point out that continues update is well beyond scope of this project. However, if one needed to implement any

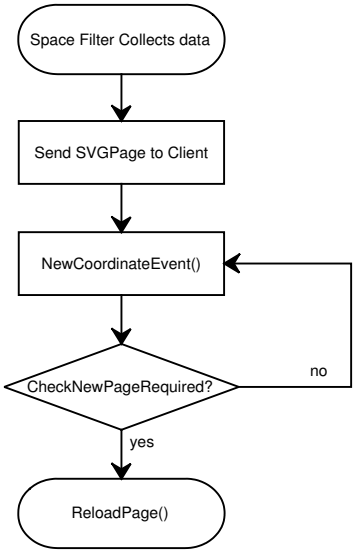


Figure 7.2: Flowchart of the Algorithm for the Update Strategy

other updating strategy different than described above, then due to the system flexibility, this could be done without any changes to the existing system. One would simply needed to add the new functionality in the `Space Filter` and `Updater`, and use it when needed.

Chapter 8

Map Matching

An LBS is an application that must be aware of a user's location. In the road network related services like nearest neighbor, user's location is the position of the user in a digital road network. Since the user's location related information from GPS receiver and the underlying digital road network may not always be accurate enough and the real road network is represented as center lines of roads, we need to reconcile the user's location with its underlying digital road network. This process is called map matching. We use the input from a GPS receiver and the geometries of the digital road network to project the user on the road network.

In our previous work [26], a map matching weighting system was proposed hypothetically. In this semester, we make improvements on weighting factors and we propose dynamic computation window and stop aggregation. The map matching system is proved that it successfully reconciles user's location with the underlying digital road network, eliminates uncertainties and saves computation time in Section 8.2. And the map matching weighting system is expanded as a inherently flexible and versatile solution, which can incorporate various weighting factors and therefor reconcile user's location with digital road network. The map matching solution is designed to be capable of fitting different users e.g. users by bus and users on foot.

8.1 Improvements to Map Matching

8.1.1 Existing Map Matching Algorithms

A number of different map matching algorithms have been proposed for different purposes. Bernstein and Kornhauser [13] propose a map matching algorithm as mapping GPS positions to the closest node, shape point, or arc in a road network. White et al. [11] reviewed some map matching algorithms and suggest that particular attention should be paid to problems that arise at intersections. Quddus et al. [20] propose a weighting system to assess the similarity between the character of the road network and user's location related information. This weighting system works well in their tests. We get the impression from those

existing literatures that the core of map matching procedure is to identify the correct segment from candidate segments.

Comparing with existing map matching algorithms we propose the generic map matching solution based on weighting system proposed by Quddus et al. [20]. The generic map matching solution is designed as an inherently flexible and versatile solution. We can incorporate various weighting factors and they can easily be removed if we are not satisfied with. Meanwhile the generic map matching solution can fit different kind of users e.g. people by car and people on foot.

8.1.2 The Two-Step Map Matching Procedure

A key and common issue of existing map matching algorithms is to identify the correct segment among candidate segments [20]. We propose a two-step algorithm that for every GPS point:

- Find candidate segments for it.
- Identify correct segment among candidates.

The reason we choose segment as the geometry to project user's location is that the segment has been designed to be as long as possible, and this will reduce the number of candidates and therefore reduce processing time.

We observe that when user stops, the GPS receiver continuously outputs the same coordinates. It seems useless to apply map matching algorithm on positions with same GPS information. So we propose to aggregate consecutive points which have same GPS information. The stop aggregation should be taken before the two-step map matching algorithm since it will not affect the map matching accuracy. We expect that stop aggregation will improve map matching efficiency if the user stops frequently for example to cross the intersection.

Find out Candidate Segments

In the first step of finding out candidate segments, each GPS point has a computation window to associate itself with candidate segments. As Figure 8.1 demonstrates, the computation window of a GPS point is a circular window that centers at the user's current location, the GPS point p_i, to retrieve candidate segments. Candidate segments are segments that intersect with the computation window. The radius of circle, r, affects the size of computation window and therefore effects the map matching performance in both accuracy and time. A small r may produce a window having no candidate at all, which leads to inaccurate mapping but costs less time. A big r may produce a window having too many candidates, which helps improve mapping accuracy but requires more time.

Since different roads have different width and the real road network does not always reconcile with the underlying digital road network, we propose a dynamic computation window (DCW) for GPS points, of which the radius r varies in different situations. For most of GPS positions the initial value of radius, which

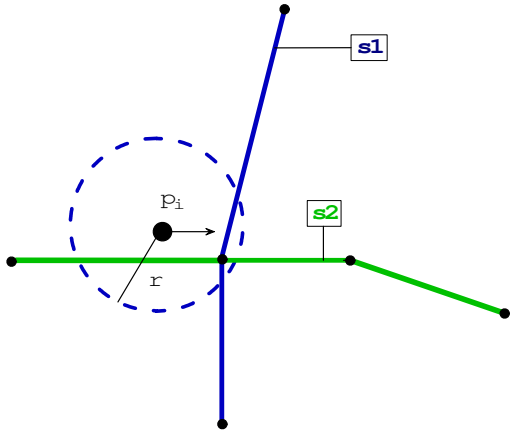


Figure 8.1: Computation Window

is r_0, should be able to find out candidate segments. But in circumstances as user's location is far away from any segments in the road network as Figure 8.2 demonstrates, the initial computation window cannot find out candidate segments, then r_0 increments by 5 meters repeatedly until the computation window intersects with a segment. The reason of extending computation window is to associate the user with at least one candidate segment when one is requiring road network related services like shortest path and is unfortunately moving along a road which does not exist or is not properly represented in the digital road network, and therefore eliminates uncertainties that happen with imprecise digital road network and GPS positioning technology.

Identify Correct Segment among Candidates

In the second step of identifying correct segment among candidates, a weighting system is proposed. We consider various factors in existing map matching algorithms and make improvements on them. The factors are assigned to weighting parameters to control their influence on map matching. They are normalized by their maximum values. The weighted average of weighting factors is calculated for each candidate segment. The selection of correct segment then is based on the weighted average of these factors. We design the weighting system as the candidate segment with the smallest weighted average is chosen as the correct one. The correct segment is used as the one to project user's location.

By investigating the performance of different factors, we can simply discard the factors that do not help improve map matching performance. The criteria is map matching accuracy and time used, which is described in Section 8.2.

8.1.3 Weighting Factors

To improve the performance of map matching, factors in existing map matching algorithms are considered as the influencing factors in the weighting system.

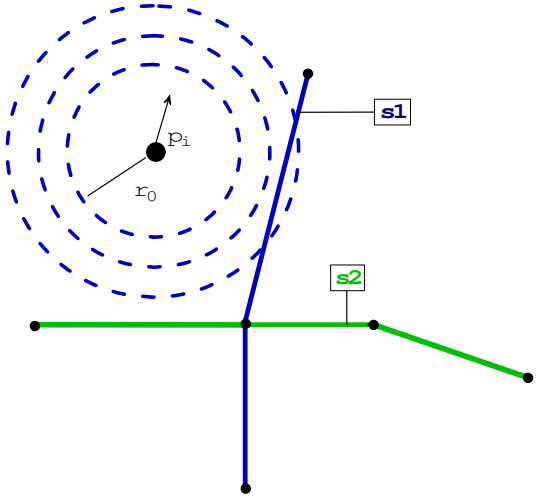


Figure 8.2: Dynamic Computation Window

Effects of those factors are investigated in Section 8.2. Each weighting factor is associated with weighting parameter to control their influence in the map matching process. The weighting factors are described in the following.

Weighting for Perpendicular Distance

The most straightforward method is to simply map the locations obtained from GPS receiver to the closest node, shape point, or arc in a road network [13]. We extend the idea with calculating the perpendicular distance from a GPS point to the appropriate line of a segment, which gives the proximity of the point to the segment. The selection of the appropriate line of a segment is described in Section 8.1.4. As Figure 8.3 demonstrates, d_2 represents the perpendicular distances from GPS point p_i to the appropriate line of segment s_2.

For each candidate segment S_j of current GPS position p_i, the weighting function W_d for perpendicular distance is given as:

$$W_d(R, S_j, P_i) = P_d d_j \tag{8.1}$$

Here

R is the road network

S_j is a candidate segment for p_i

P_i is the GPS information for p_i

P_d is the weighting parameter that controls the influence of W_d

d_j is the perpendicular distance from p_i to the appropriate line of S_j, its maximum value is the radius of DCW

The weighting of the perpendicular distance from a GPS point to the appropriate line of a segment is expected to help associate user with the road one

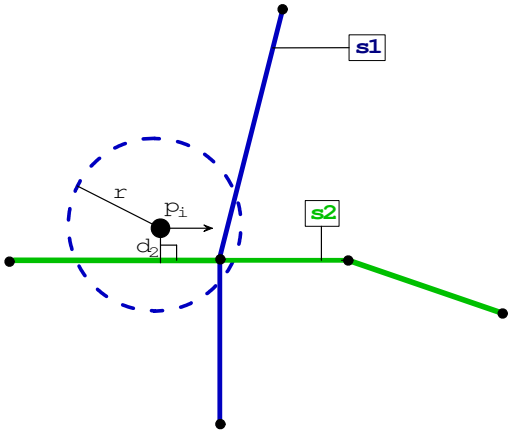


Figure 8.3: Weighting for Perpendicular Distance

is moving along. But for the reason of imprecise road network representation and positioning technology, we are sure that the weighting of the perpendicular distance is not enough to identify correct segment among candidates.

Weighting for Movement Direction

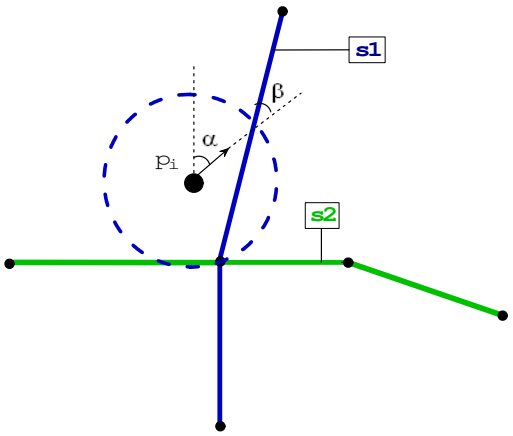


Figure 8.4: Weighting for Movement Direction

As we know, the user's current movement direction is available from GPS receiver. In Figure 8.4, angle α represents user's moving direction at p_i and angle β represents the intersecting angle between the user's movement direction and the appropriate line of candidate segment s_1. Intersecting angle β also gives the proximity of the point to the segment. For each candidate segment S_j of p_i, the weighting function W_m for user's movement direction is given as:

$$W_m(R, S_j, P_i) = P_m\beta \tag{8.2}$$

Here

R is the road network

S_j is a candidate segment for p_i

P_i is the GPS information for p_i

P_m is the weighting parameter that controls the influence of W_m

β ($0 \leq \beta \leq 90$) is the intersecting angle between the movement direction and the corresponding line belonging to S_j

The weighting of movement direction is expected to play a very important role in map matching, because even the user's position deviates from the road network, its movement direction should reconcile with the real road along which the user is moving. But we also realize that user's movement direction will not always follow the direction of the digital road network because of imprecise representation of the road network.

Weighting for Road Network Topology

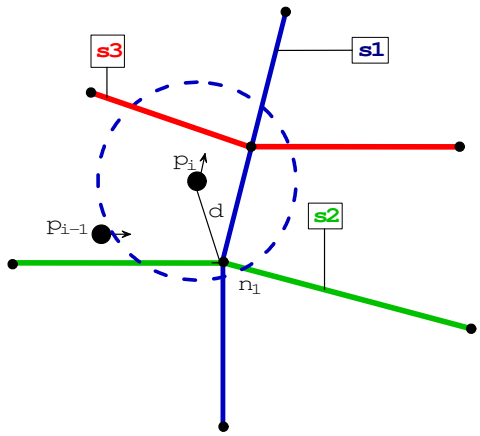


Figure 8.5: Weighting for Road Network Topology

We know that users' movements are constrained by road networks topology such as road type, intersection, turn restriction, and so on. So we propose the weighting for road network topology that reduces or increases the likelihood as a correct segment of each qualified candidate segments. The road network we have at this moment provides connections among segments. So the weighting factor is based on connections of road network. The idea can be widely applied to other topology properties when they are ready.

In Figure 8.5, p_{i-1} is on the road represented by segment s_2 and p_i is on the road represented by segment s_1. We assume that p_{i-1} has successfully chosen s_2 as the correct segment to project itself. And we know that the computation window of p_i intersects with candidate segments s_1, s_2, and s_3.

For each candidate segment of p_i, if it is exactly previous selection s_2 or has connections with s_2 within computation window, for example candidate segment

s_1 has connection n_1 with s_2 within computation window, we propose those candidates should have more chances to be chosen as the correct one. The candidate segments like s_3 that have no connection with s_2 within computation window should have less chances. We require the connections should be within computation window because computation window indicates the deviation of the GPS point from the road network.

For each candidate segment S_j of p_i, the weighting function W_t for road network topology is given below as:

$$W_t(R, S_j, s_{i-1}) = P_t T \tag{8.3}$$

Here

R is the road network

S_j is acandidate segment for p_i

s_{i-1} is the previously selected segment for previous GPS position p_{i-1}

P_t is the weighting parameter that controls the influence of W_t

$$T = \begin{cases} 1 \text{ if } S_j \text{ does not intersect with } s_{i-1} \text{ within computation window} \\ 0 \text{ if } S_j \text{ is } s_{i-1} \text{ or intersects with } s_{i-1} \text{ within computation window} \end{cases}$$

Here for T, we simply assign it 0 or 1 to distinguish two kinds of segments. The effect of the topology information on map matching will be changed with the varying value of P_t.

White et al. use road network topology to reduce the number of candidate segment [11], which means that all candidates that are not connected with previous selection are removed from candidate array. But we use road network topology to give the likelihood as a correct segment for each candidate segments rather than simply remove unqualified candidates. The idea we propose eliminates the errors that may occur when none of the candidates is connected with previous selection.

We realize that the road network topology itself cannot be used alone to tell which candidate segment is the most likely one. It only gives a hint that which candidate segment is the less likely one comparing with other candidates.

The weighting of road network topology also helps correct inaccurate computation window. We observe that some rotaries like Figure 8.6.b are represented as Figure 8.6.a demonstrates. If user is moving along the road in Figure 8.6.b from west to east, then the trajectory will be like Figure 8.6.a demonstrates and segment s_2 should be the correct segment for all positions. We assume position p_{i-1} successfully chooses segment s_2 as correct segment, but position p_i fails to do so because the initial computation window of p_i does not cover s_2 at all.

The scenario is possible to happen because the initial computation window is used to cover correct segment for most of positions not for all and we never know how big the gap between real road network and the digital one is. In such a circumstance initial computation window of p_i violates road network topology i.e. it does not cover even one segment that connects with previous selection in computation window, the initial computation window of p_i will increment by 5

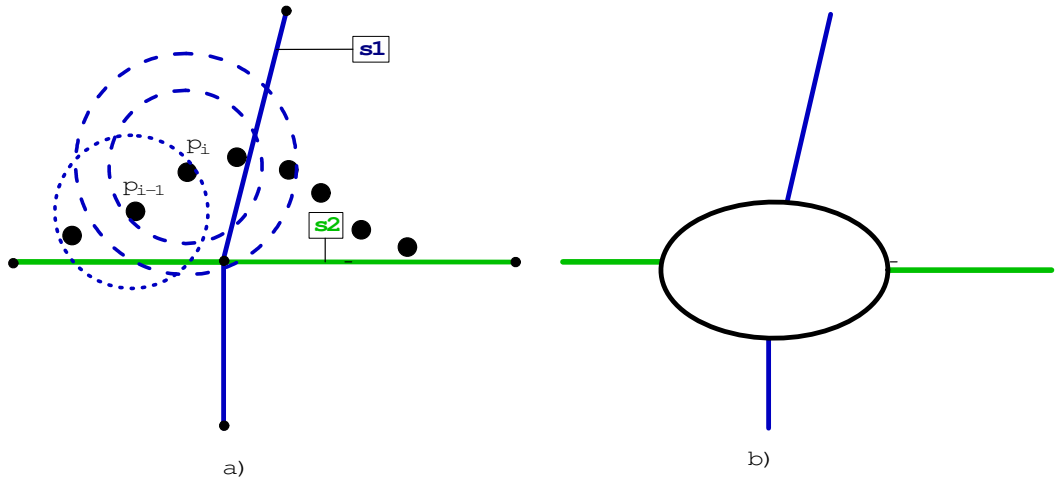


Figure 8.6: Inaccurate Computation Window

meters once only to cover more candidate segments. But if still the computation window fails to cover such a candidate, we will not overplay the expansion and simply assume the user truly violates road network topology even it could be the inaccurate positioning technology.

Now, we have three weights for each segment. The criteria used to select correct segment is determined from the weighted average. The weights are normalized by their maximum values. The total weighting score is calculated as,

$$TW = W_d/m_d + W_m/m_m + W_t/m_t \tag{8.4}$$

The segment with the smallest TW is chosen as the correct one. The maximum value of W_d,m_d, is exactly the size of computation window. m_m is 90 degrees and m_t is 1 as we assume.

8.1.4 Map Matching Steps

Figure 8.7 shows the detailed description of average weighted map matching solution. The generic map matching solution uses the following steps to assign each GPS position the correct segment and project it on that segment. Previous GPS position is stored in order to apply stop aggregation. For each new arrived GPS position:

- Initialize parameters, of which i is the index of candidate segment, *seg* stores the correct segment to project current position on, *average* is the current weighted average of map matching functions, and *tmp* stores the smallest weighted average.
- Check whether the position has the same coordinates with the previous one. If so, then wait for next GPS position, otherwise calculate the computation window and get the number of candidate segments.

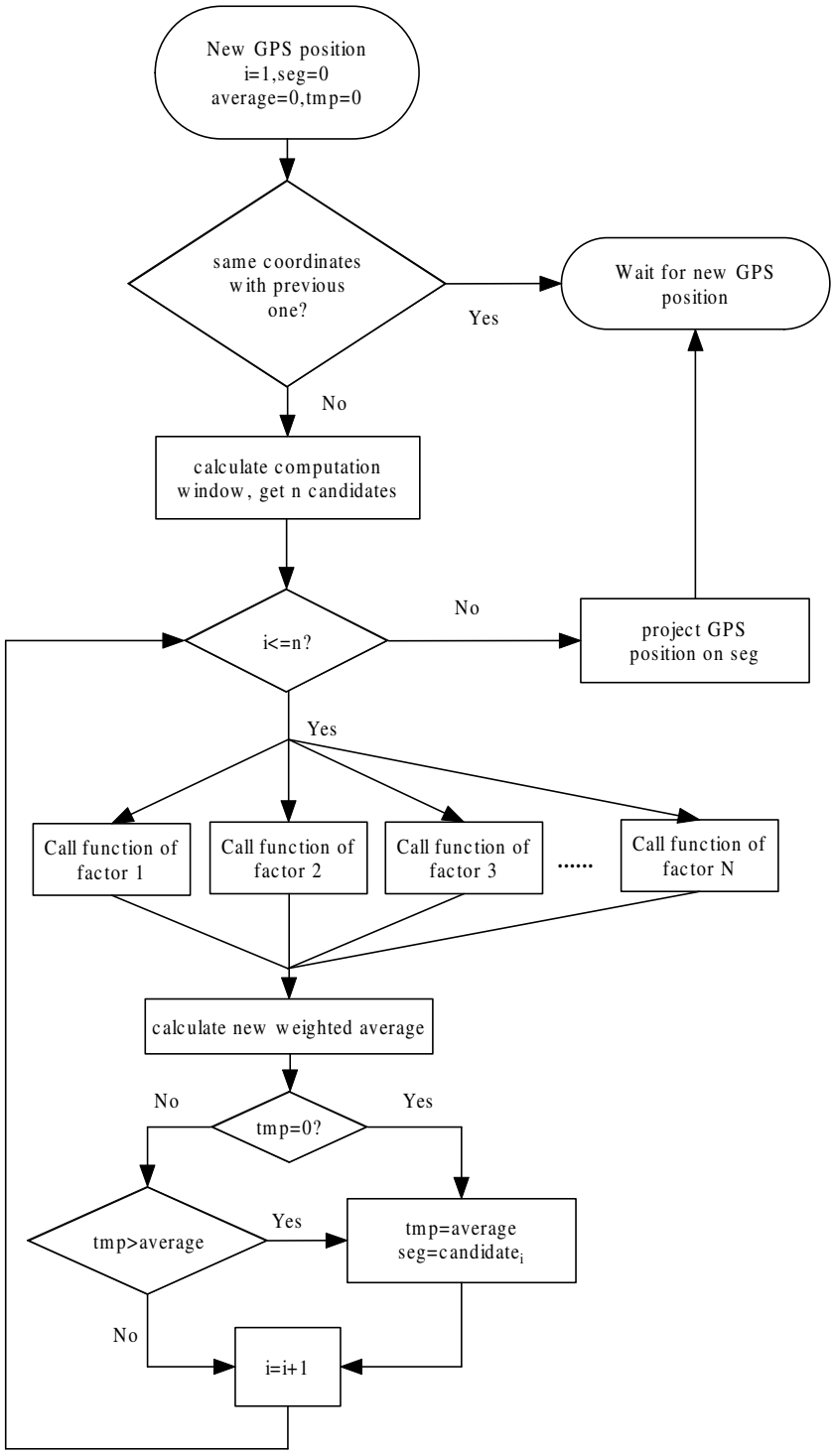


Figure 8.7: Flow Chart of Map Matching Algorithm

- For each candidate segment, apply them functions of different weighting factors as we choose and calculate the normalized weighted average.
- Choose the segment with smallest normalized weighted average as the correct segment and project user's location on it.

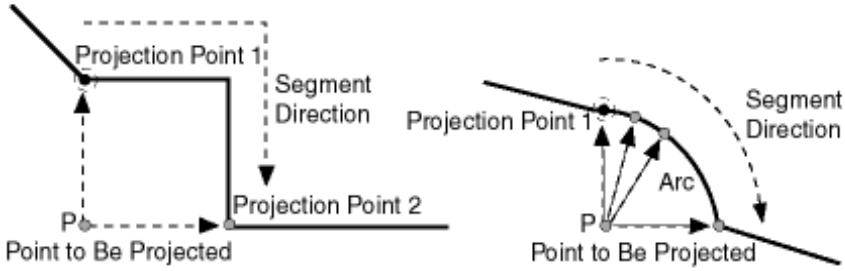


Figure 8.8: Projecting a Point onto a Geometric Segment[12]

The projection of a point along a geometric segment is achieved by Oracle Spatial function `SDO_LRS.PROJECT_PT`. As Figure 8.8 demonstrates, this function returns the projection point of a specified point on the geometric segment with its measure that is the distance from the start point. And if multiple projection points exist, the first projection point is returned. The projected point is used to select correct line and therefore to assess the segment it belongs to.

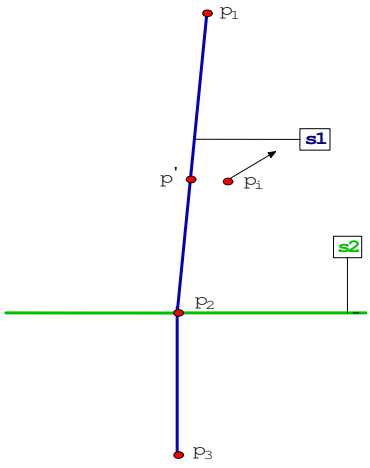


Figure 8.9: Selecting the Correct Line of a Segment

By projecting GPS point to a segment, we know the projection point as well as its measurement m in the segment, which is the distance from the start of segment. In Figure 8.9 the correct line of candidate segment s_1 is the line from

p_1 to p_2 with a start measure smaller than measure of projection point p', which is m, and an end measure bigger than m.

8.2 Experiments and Evaluation of Map Matching

In section 8.1, a generic map matching solution is formulated and the weighting factors are described. In this section, We evaluate the performances of the map matching algorithms, which are implemented in PL/SQL. Experiments are carried out on an Athlon XP 1.2GHz machine with 256 MB RAM running on Microsoft® Windows® XP Professional. A broad range of tuning parameters are considered to improve the map matching accuracy. And the wall clock time are used as one of the performance metric to choose the value of tuning parameters. It is the elapsed time between when a process starts to run and when it is finished. Wall clock time is usually longer than the CPU time because the CPU is doing other things besides running the process such as waiting for disk I/O. But since our LBSs are developed in Windows operating system and it is complicate to calculate CPU time in Windows, we choose wall clock time as the performance metric.

x	y	speed	time	date	direction
559963	6319477	2	14/02/52	12/12/03	191.5

Table 8.1: GPS Sample Data

There are two kinds of experiments. One is carried out on users travelling by buses, another for users on foot. They represent two kinds of users in reality. The experiments are carried out in Aalborg, a city in Denmark. The outputs of GPS receiver are collected with the format in Table 8.1 so that they can be applied to the map matching algorithms.

Table 8.1 shows the some useful GPS output after transformation . As one can see, a GPS receiver provides more information than information about coordinates. The first two columns of Table 8.1 hold `x`, `y` coordinates in UTM format which is the format used in the data model. The third column holds the user's speed measured in knots. One knot is approximately 1.85 kilometers per hour. The forth column contains the time at which the signal is received. The format of time is hour/minute/second. The date column indicates the date on which the signal is received. The format used here is month/day/year. The last column indicates the direction of user's movement. The direction expresses user's movement direction in degrees from North in clockwise.

8.2.1 Experiments for Users on Bus

Routes of bus No. 4 and No. 9 are chosen to for te experiments since they have complex characteristics like suburb, rotaries, and urban areas where we may meet signal interferences. Approximately 2500 GPS positions are collected from GPS receiver. The tuning parameters and their values are summarized in

Table 8.2. The default parameter values are given in bold, which indicates that experiments reach the best performance.

Parameter	Values Used
r	5,10,15,20,**25**,30
P_t	0.5,**1**,5,10,15
P_m	0.5,**1**,5,10,15

Table 8.2: Parameters and Values Used

Computation Window and Weighting for Perpendicular Distance

We begin by investigating how different values of computation window affect the performance of map matching based on perpendicular distance. r is the radius of computation window. It reflects the size of computation window and therefore determines the number of candidate segments. The initial value of r is important because a small computation window may result in a computation window having no or less candidate segments, which in turn requires time consuming expansion of computation window, and a big computation window may require more time to take care of all candidates included.

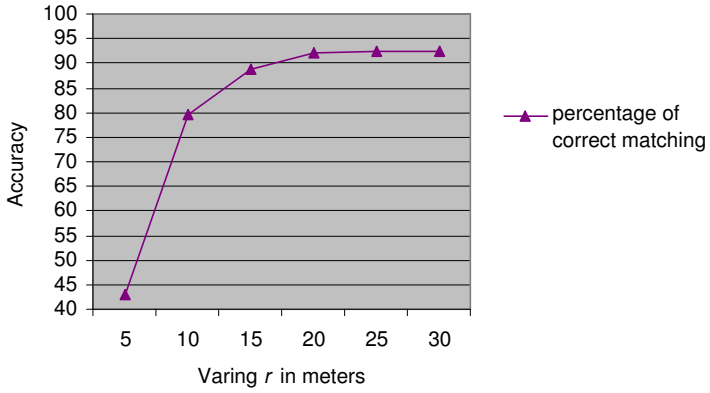


Figure 8.10: Experiments Result of Size of Computation Window

In order to choose the initial value of r, we first evaluate the map matching accuracy of fixed computation window, which means the window will not extend repeatedly to cover more candidate segments when it does not cover even one segment. Figure 8.10 demonstrates accuracy of perpendicular distance with different size of fixed computation windows. The user's movement trajectory and the positioning results are plotted in SVG file as Figure 8.11 demonstrates. The red dots represent user's position from GPS receiver and the blue dots represent the map matching results. By plotting the roads user is moving along, which make up the yellow path, the result of map matching is easy to tell.

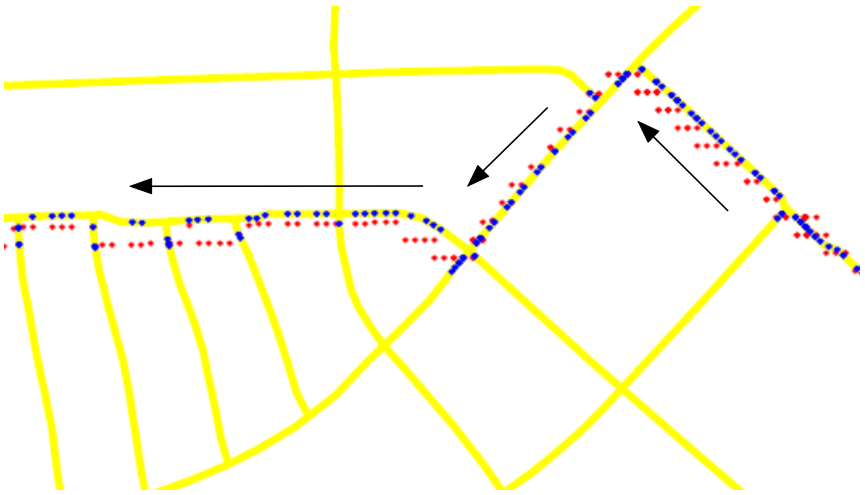


Figure 8.11: GPS Positions and Map Matching Result

Figure 8.10 shows that the accuracy initially does not performs good because a smaller size of computation window produces less candidate segments. With the expansion of computation window, the accuracy increases because it covers more candidate segments and therefore the segments of which user moves along are more likely covered. Map matching procedure performs best at 25 meters and the accuracy becomes unchanged after r is greater than 25 because at this time the computation window has covered enough candidate segments to tell which one is most likely correct.

From Figure 8.10 the initial size of computation window is assigned to 25 meters in radius. When it does not work to find out candidate segments, then the radius increments by 5 meters repeatedly until the computation window intersects with certain segment.

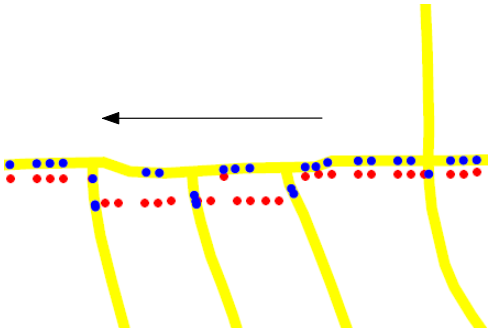


Figure 8.12: Movement Cross Intersection

Note that the map matching accuracy of perpendicular distance is at most

92.53144654 percent. The inaccurate mapping happens as Figure 8.12 demonstrates. When the user is moving across intersections, since user is not moving in the right center of road and the road network is represented as poly lines, some positions are close the other segments rather than the one it is moving along.

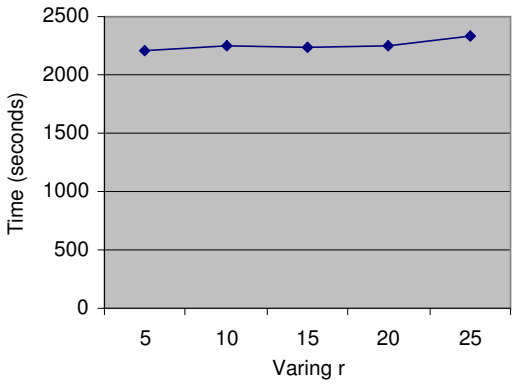


Figure 8.13: Performance of Computation Window

Figure 8.13 shows the performance of map matching in time. We observe that map matching procedure is more expensive in time with the expansion of computation window because more segments intersect with computation window, and therefor more computations are done.

Stop Aggregation

As we know, when people stay in one position for a while, the GPS receiver will produce the same information. From the experiments of map matching procedure with stop aggregation, we note that stop aggregation saves 1050 seconds in average for the data we collected. For the rest of the experiments, we include stop aggregation and select 25 as the initial value of r.

Weighting for Movement Direction

The map matching accuracy of using movement direction alone is 0.94, which is better than using W_d alone. So we conclude that movement direction plays a very important role in map matching procedure as we expect. Because even if the user's position deviates from the road network, the movement direction should reconcile with the direction of the road along which the user is moving.

Figure 8.14 demonstrates the map matching accuracy of movement direction combined with previous experiment results, which means we assign the initial computation window with radius 25 meters and include stop aggregation. To

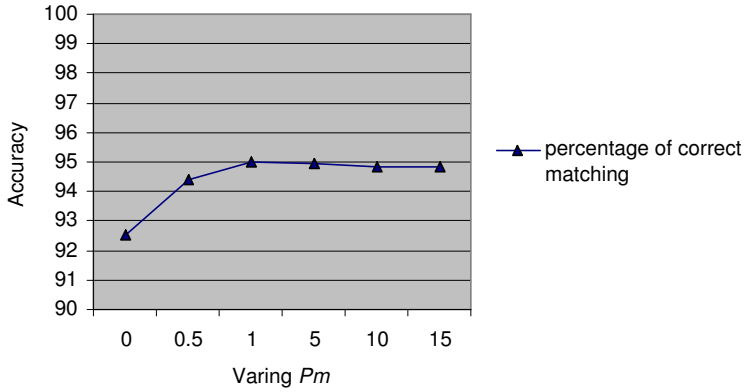


Figure 8.14: Experiments Result of Movement Direction

be simplicity we assign the weighting parameter of perpendicular distance, P_d, to 1. Figure 8.14 shows that map matching performs best when we assign P_m to 1.

Road Network Topology

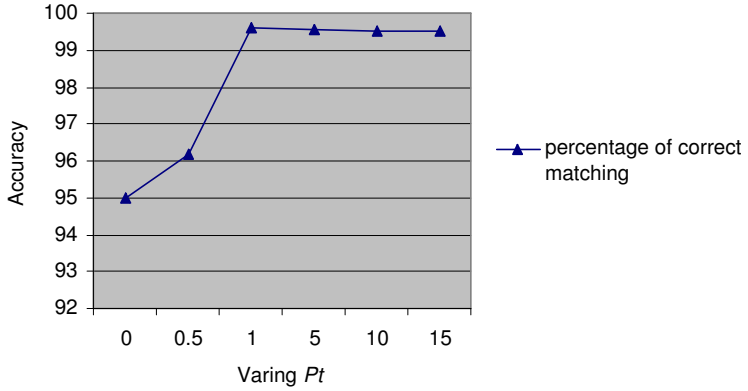


Figure 8.15: Experiments Result of Road Network Topology

We already mention that the road network topology itself cannot be used alone to tell which candidate segment is the most likely one. What we can tell is which candidate segment is the less likely one. Figure 8.15 demonstrates the map matching accuracy of road network topology with previous results. At this moment we only consider intersections in the road network. The map matching

performs very well when we assign the weighting parameter of road network topology P_t to 1. So we conclude that road network topology surely helps locate user in the road network because it reduces the likelihood of unconnected candidate segments as a correct one.

8.2.2 Experiments for Users on Foot

The source data for pedestrians are collected from a user, equipped with a GPS receiver, walk from suburb, through rotaries, to urban areas. Approximately 4000 GPS positions are collected. The tuning parameters and their values are summarized in Table 8.3. The same with experiments for users on bus, the default parameter values are given in bold.

Parameter	Values Used
r	5,10,15,20,25,30,**35**,40
P_t	0.5,**1**,5,10,15

Table 8.3: Parameters and Values Used

Computation Window and Weighting for Perpendicular Distance

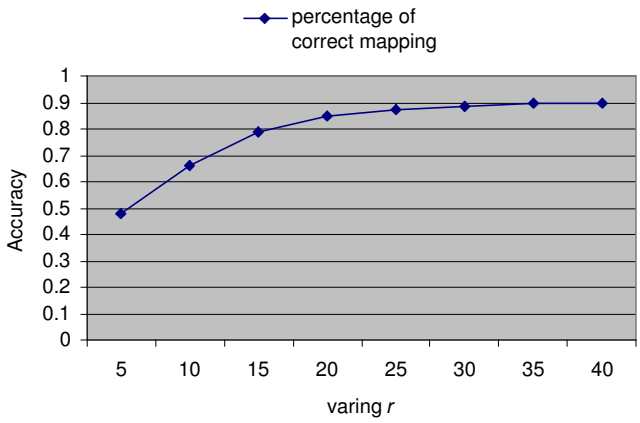


Figure 8.16: Experiments Result of Size of Computation Window

Figure 8.16 shows that map matching procedure performs best at 35 meters and the accuracy becomes unchanged after r is greater than 35. It is reasonable that the best value of r for pedestrian is greater than that of users on bus because pedestrians usually walk on the footways which is the border of roads. Figure 8.17 shows that with the expansion of computation window, the accuracy increases.

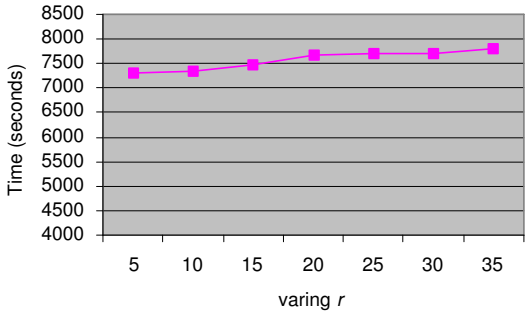


Figure 8.17: Performance of Computation Window

Stop Aggregation

From the experiments of map matching procedure with stop aggregation, we note that stop aggregation saves 1700 seconds in average for the data we collected. It is so because the movement speed of pedestrian is quite slow and that leads to more consecutive same GPS positions. For the rest of the experiments of pedestrians, we include stop aggregation and select 35 as the initial value of r.

Weighting for Movement Direction

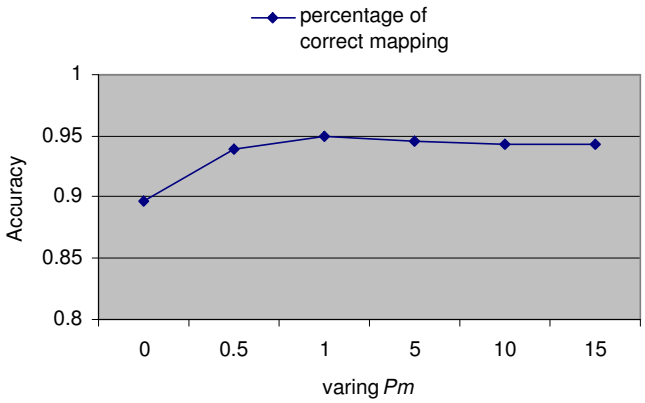


Figure 8.18: Experiments Result of Movement Direction

Figure 8.18 demonstrates the map matching accuracy of movement direction

combined with previous experiment results, which means we assign the initial computation window with radius 35 meters and include stop aggregation and assign the weighting parameter of perpendicular distance, P_d, to 1. Figure 8.18 shows that map matching performs best when we assign P_m to 1.

Road Network Topology

The road network we have are not specific enough to include some alleys. Then we are sure that user is not restricted by current road network topology, and therefor we can not apply road network topology to project user on the road network. The reason is that it may lead to severe mismatching and therefor lead to incorrect LBSs.

8.2.3 Evaluation of Map Matching

From the experiments for users by bus and on foot, we observe that the map matching solution we propose does help project GPS positions on correct segments in the road network and therefore provide accurate LBSs. The normalized weighted system is inherently flexible and versatile. It can incorporate more weighting factors if they are disclosed that they help improve map matching performance. And it can be customized to meet different requirements. For different types of users, we only need to incorporate appropriate factors and assign them the appropriate parameters. For a poor road network representation and poor positioning technology, we need to consider more weighting factors. For a high-accuracy road network and a high-accuracy positioning technology, we can incorporate less weighting factors.

Chapter 9

Conclusion

The report is based on the design of a system that supports rapid development of highly integrative location based services. As we explained there exist various types of location based services on the market, but what makes the documented system stand out is that it supports providing of data of high–quality meaning that users are provided correct and up to date data.

In Chapter 2 we are inspired by a research that studies and discusses behavioral patterns of tourists. Based on the patterns, the study proposes how digital technology can be used as a tool to help tourists in various situations. We let us inspire by the ideas and propose mobile solutions to the presented problems.

Chapter 3 proposes that the system should be based on a client–server architecture that is essential for the delivery of high–quality data. Furthermore, the server architecture is separated in two tiers — services and components. Hereby we separate the data presentation from the internal logic. The components provide functionality while services integrate them in order to provide data to clients. Such a design makes the components reusable and as independent of each other as possible, which in turn makes development of new services and components rapid and highly integrative with the rest of the system. Data processed by the system components is separated in the data tier for easier management.

The data tier is described in Chapter 4. It is based on a data model that is tailored towards location based services. We present two representations — the internal one which is designed for easy database maintenance, and the external one which enables easier querying through views and provides better system performance by being based on materialized views. Furthermore, the chapter describes the value chain which is a mechanism for data maintenance. The value chain ensures that the data in the database is up to date which in turn makes location based services of high–quality.

In Chapter 5 we take a look at the development of client application. Based on the client requirements set in the architecture, we focus on extending the functionalities of the SVG browser by integrating the mentioned client components. Hereby we achieve a client that knows its geographical location, can communicate with a server over wireless network, understands services provided

by the server, enables user interaction and is capable of displaying maps used for putting the content provided by the server into geographical context. We also consider and describe three different ways of handling maps by the client.

As we explained, the server logic is divided in two tiers — components and services. Chapter 6 reflects upon this decision and separately deals with the description of the implemented components and services. As we mentioned in the Introduction, the choice of components reflects on the set of queries. Following is the relationship between components and the queries:

- `Map Matching` — where is user with respect to the road network
- `Space Filter` — what are the attractions in user's vicinity
- `Shortest Path` — what is the shortest path from user's location to certain position with respect to the road network
- `Logger` — what are the locations that user has visited

In addition we presented components that allow client independence.

- `SVG Encoder` which encodes data in a format understandable by the client
- `Path Converter` that converts the shortest path into format understandable by the client
- `Map Handler` which is needed for our application exclusively

Where applicable we give verbal examples of how individual components could be exchanged or their functionalities extended without affecting the rest of the system. Examples of the purposes of individual components were expressed through the five implemented services: The Electronic Mobile Guidebook, The Explorer, The Advertiser, The Object Finder and The Path.

In Chapter 7 we present three issues that have influence on the amount of data sent to clients at one time. Since users are moving they tend to exit the areas covered with the provided data, we present an update strategy that is dealing with these issues.

In Chapter 8 we look more thoroughly into the details of the `Map Matching` component. There are various parameters affecting precision and performance. We propose best parameter–values based on the performed experiments.

Bibliography

[1] CARNAV. http://www.carnav.com/, 2004.

[2] MEXPRESS. http://mexpress.intranet.gr/, 2004.

[3] PALIO. http://www.palio.dii.unisi.it/, 2004.

[4] SCALEX. http://www.scalex.info/, 2004.

[5] The Apache Jakarta Project. http://jakarta.apache.org/tomcat/, 2004.

[6] The Official Bluetooth Website. http://www.bluetooth.com/, 2004.

[7] 3G. 160 Million A-GPS Wireless Phone Sales by 2008. http://www.3g.co.uk/PR/Feb2003/4834.htm, 2003.

[8] Aalborg Tourist & Convention Bureau. Visit Aalborg. http://www.visitaalborg.dk, 2004.

[9] Bary Brown and Mathew Chalmers. Tourisam and Mobile Technology, 2003.

[10] C. Hage, C. S. Jensen, T. B. Pedersen, L. Speicys, I. Timko. Integrated Data Managemant for Mobile Services in the Real World. Technical report, Department of Computer Science, AAU and Euman/AC, 1998.

[11] C.E. White, D. Bernstein, A.L. Kornhauser. Some map matching algorithm for personal navigation assistants. Transportation Research Part C 8,91-108, 2000.

[12] Chuck Murray. Oracle Spatial User's Guide and Reference Release 9.2. Oracle Coporation, 2002.

[13] D. Bernstein, A. Kornhauser. An introduction to map matching for personal navigation assistants. New Jersey TIDE Center, 1996.

[14] Elina Janssen. Gis online - location based services. http://www.geog.uno.edu/ ejanssen/GIS, 2001.

[15] GSM World. GPRS Platform. http://www.gsmworld.com/technology/gprs/intro.shtml, 2000.

[16] Ingemar Eriksson. Working for the Future of the European Tourism, Must finde out.

[17] Kenneth J. Dueker J. Allison Butler. GIS-T Enterprise Data Model with Suggested Implementation Choices. Technical report, Center for Urban Studies School For Urban and Public Affairs Portland State University, 1997.

[18] Kort & Matrikelstyrelsen. Rastertjenesten — WMS Version 1.1.1. http://kortforsyning.kms.dk/partnerportal/Docs/v1.1/Rastertjenesten_1.2.2.pdf, 2004.

[19] Krak.dk. http://www.krak.dk, 2004.

[20] M.A. Quddus, W.Y. Ochieng, Lin Zhao, R.B. Noland. A general map matching algorithm for transport telematics applications. Center for Transport Studies, Dept. of Civil and Environmental Engineering, Imperial College London, 2003.

[21] Mobilin. Locationbased Services. http://www.mobilein.com/location_based_services.htm, 2001-2004.

[22] Ola Andersson, Phil Armstrong et al. Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation. http://www.w3.org/TR/SVG/, 2003.

[23] Route66. Mobile Danmark. http://www.66.com/, 2004.

[24] Simon, H. A. A Behavioural Model of Rational Choice. Quarterly Journal of Economics, vol. 69, no. 3, pp. 99-118., 1955.

[25] W3C DOM WG. Document Object Model (DOM). http://www.w3.org/DOM/, 2004.

[26] Yan Zhao, Jasmin Catovic, Vedran Alikalfic . Developing Location Based Services. http://www.cs.auc.dk/library/cgi-bin/detail.cgi?id=1074074275, 2003.