WEB BASED GEOGRAPHICAL INFORMATION SYSTEMS FOR MIDDLE EAST TECHNICAL UNIVERSITY CAMPUS

A THESIS SUBMITTED TO THE
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÖKÇE TÜRKMENDAĞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
GEODETIC AND GEOGRAPHICAL INFORMATION TECHNOLOGIES

MAY 2009

Approval of the thesis:

WEB BASED GEOGRAPHICAL INFORMATION SYSTEMS FOR METU CAMPUS

submitted by **GÖKÇE TÜRKMENDAĞ** in partial fulfillment of the requirements for the degree of **Master of Science of Geodetic and Geographical Information Technologies Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Assoc. Prof. Dr. Mahmut Onur Karslıoğlu _____
Head of Department, **Geodetic and Geographical Information Technologies**

Assoc. Prof. Dr. Nurünnisa Usul
Supervisor, **Civil Engineering Dept., METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Şebnem H. Düzgün _____
Mining Engineering Dept., METU

Assoc. Prof. Dr. Nurünnisa Usul _____
Civil Engineering Dept., METU

Prof. Dr. Volkan Atalay _____
Computer Engineering Dept., METU

Prof. Dr. Tanju Mehmetoğlu _____
Petroleum and Natural Gas Engineering Dept., METU

Assoc. Prof. Dr. İrem Dikmen Toker _____
Civil Engineering Dept., METU

Date: 01 / 06 / 2009

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : **Gökçe Türkmendağ**

Signature :

ABSTRACT

WEB BASED GEOGRAPHICAL INFORMATION SYSTEMS FOR MIDDLE EAST TECHNICAL UNIVERSITY CAMPUS

Türkmendağ, Gökçe
M.S., Department of Geodetic and Geographic Information Technologies
Supervisor : Assoc. Prof. Dr. Nurünnisa Usul

May 2009, 102 pages

Middle East Technical University (METU) campus has such an extensive area that reaching the necessary information which affects campus life, such as the locations of the buildings, classrooms, computer labs, and etc. may be very difficult for anyone who does not know the campus well, and even for a student, personnel or a graduate who had a long time in the campus. An interactive campus map and a database structure related to this map which can be accessed by multiple types of users on the Internet can display this information with its geographical locations, and will reduce the "difficulty for reaching information" widely.

For this purpose, data of METU were collected from various sources, edited, organized, and inserted into data tables. An interactive campus map displaying the locations of the physical structures and facilities in the campus was created in Scalable Vector Grapics (SVG) standard, and published on the Internet. By JavaScript functions, the map can be browsed with map navigation tools, including zoom in, zoom out, move and information buttons, and layers control. There is a search section on the user interface, which allows users make queries to find building and classroom names, and list the buildings and facilities according to their usage and category types. Data are stored in PostgreSQL database, transmitted

through PHP scripts, and can be edited by authorized users through the specialized web interfaces. Lastly, web-based implementation of the application is entirely based on open-source standards.

Keywords : Web, GIS, SVG, METU, Open Source

ÖZ

ORTA DOĞU TEKNİK ÜNİVERSİTESİ KAMPUSU İÇİN WEB TABANLI COĞRAFİ BİLGİ SİSTEMLERİ

Türkmendağ, Gökçe
Yüksek Lisans, Jeodezi ve Coğrafi Bilgi Teknolojileri Bölümü
Tez Yöneticisi : Doç. Dr. Nurünnisa Usul

Mayıs 2009, 102 sayfa

Orta Doğu Teknik Üniversitesi (ODTÜ) yerleşkesi çok geniş bir alana sahiptir. Bu kadar büyük bir alanda yerleşkeyi pek tanımayan birinin, hatta yerleşkede uzun zaman geçirmiş bir öğrenci, personel ya da mezunun binaların yerleri, derslikler, bilgisayar laboratuvarları, v.b. gibi yerleşke yaşamını etkileyen pek çok bilgiye hızlıca ulaşması oldukça zor olabilir. Internet ortamında yayınlanan interaktif bir harita ve bu haritaya bağlı bir veri tabanı yapısıyla bu tip bilgiler coğrafi konumlarıyla beraber birden fazla kullanıcı tipine sunulabilir ve “bilgiye ulaşma zorluğu” büyük ölçüde hafifletilebilir.

Bu amaçla, ODTÜ kampusune ait veriler çeşitli kaynaklardan toplanarak düzeltildi, organize edildi ve veri tablolarına taşındı. Yapıların ve hizmetlerin yerlerini gösteren interaktif bir kampus haritası Scalable Vector Grapics (SVG) formatında hazırlandı ve Internet’te yayınlandı. Haritayı, JavaScript fonksiyonları ile hazırlanan araçlarla büyütmek, küçültmek, hareket ettirmek, bilgi almak ve katmanları kontrol etmek mümkündür. Kullanıcı arayüzündeki arama bölümü ile bina ve sınıf isimleri sorgulanabilir, ayrıca binalar ve sundukları olanaklar, kullanım amaçlarına göre gruplandırılarak listelenebilir. Veriler PostgreSQL veritabanında tutulmakta, PHP

betikleri ile taşınmakta ve yetkili kullanıcılar tarafından değiştirilebilmektedir. Sonuç olarak, uygulamanın web tarafı tamamen açık kaynaklı standartlarla hazırlanmıştır.

Anahtar Kelimeler : Web, CBS, SVG, ODTÜ, Açık Kaynak

To My Mother

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my supervisor Dr. Nurünnisa Usul for her remarkable patience, guidance, suggestions and evaluation during the preparation of this thesis.

I am grateful to my friend Sevgi Korkmaz for her assistance and support.

I want to thank my colleagues Cihan Yıldırım Yücel for giving me the initial idea of this study and her further suggestions, and Yasemin Saatçioğlu Oran for helping me with the graphical design.

Cengiz Acartürk and Özge Alaçam are gratefully acknowledged for their supports and assistance about the eye tracking study.

Data support of Naim Cem Güllüoğlu is also remembered.

Lastly, I would like to thank my mother for her support, assistance and encouragement throughout my study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Definition of the Problem

Universities are the most important sources of knowledge and technology. For that reason, it is expected that any kind of information that a university provides can be reached easily and quickly. Internet is a very efficient environment for distrubuting its resources if a university has its own information system, which may contain library information services, e-journals, e-thesis, lecture notes, campus phonebook, and so on. By the help of such a system, the university can give better services to larger populations.

Some universities with very large areas may need additional information systems, since it can be hard to find some specific points or destinations when walking around the campus. Middle East Technical University (METU) is one of them, which is almost like a city with its both population of over 30000 people and total area of 4500 hectares. In such an extensive area, reaching the necessary information about the places of some facilities, buildings and other physical structures may be very difficult for anyone who does not know the campus well, for instance a freshman who has just been admitted to the METU, and even for a personnel or a graduate who has spent a long time in the campus.

There are several user groups which visit the campus for different purposes. The main group is composed of students, faculty, and administrative staff, who have a right to use most of the buildings and facilities that METU provide. Academic and administrative buildings, rooms of the staff, health services, library, computer laboratories, wireless network areas, and classrooms are some of them, and sometimes their places must be found in a very short time.

METU Technopolis staff also visit the campus quite often. Besides, there is a large number of people using the campus for other purposes and with different frequencies. Every year, a lot of examinations of some organizations take place, whose applicants are specially interested in finding their exam places. Also, participants of some activities in Cultural and Conventional Center, graduates, families of the students, and all other kind of visitors come to the campus occasionally.

Food and drinking places, banks, ATM machines, transportation points such as bus and minibus stops, mailing services, exhibition and concert halls, shops, tailors, stationaries, hairdressers, markets and parking spaces are the other important facilities that affect the campus life, and can be used by everyone visiting the campus.

All those entities are spread on that extensive area within hundreds of buildings, which make their locations difficult to be found, especially in urgent situations. For example, a student may miss an examination or a class if s/he cannot find the classroom on time, or a visitor cannot park her/his car if s/he does not know the visitors' parking space. There is a variety of needs with different importance levels. But the main idea is, everyone wants to know how to reach their destinations without losing time when looking for them.

In addition to looking for specific features or facilities, people may be curious about the services or buildings which they do not know, or want to get some information about some structures that they are interested in. A prospective student may need to browse the campus without visiting it physically, or someone may want to learn what kind of shops that the shopping center has.

1.2. Statement of the Problem

Maps have always been great instruments when looking for a destination target, especially when someone is lost. It is an easy solution to distribute hard copy maps or publish static maps on the Internet, which show the buildings, roads and other features. However, this solution may not be enough as they cannot carry detailed information, and the users may not know where to look in them to find their target. For example, a student may want to find the place of a classroom without knowing

the name of the building in which it is located. Even if s/he knows the name of the building, still it can be difficult to find the location of the building on the map.

Besides, a map should be as simple as possible. If all entities of an area or detailed information about the features are shown on the same map, it will result a complicated view.

An interactive map which does not only show the physical structures of the campus, but also contains detailed informatıon about these entities and some services that the university provides can make campus life easier, since the necessary information will be delivered easily. Geographical Information Systems (GIS) provides a great opportunity to create intelligent maps that serve both spatial and attribute data about the geographical structures referenced to their physical locations. The answers of the basic “where“ or “what“ questions can be reached quickly by making queries with search strings or it is possible to browse the map with turning on the layers that are being interested. Scale, visual properties and position of the map can be altered with some map tools.

If the map is published on the Internet, a large number of audience can take the advantage of using this service. Rapid development of information technologies has been directly affecting the development of GIS. Nowadays, with the expansion of Internet based technologies, GIS can be combined with world wide web, and share geographical information with general public from a centralized location with improved visualization and usability capabilities, and high performance results. By the help of the web-based GIS, it is possible to access geographical data from anywhere in the world with only a simple web browser and an Internet connection, without need for learning and purchasing expensive GIS softwares which are usually very complex and difficult to use.

1.3. Objective of the Study

The objective of this study is to create an interactive map of METU campus that displays the locations of the physical structures and facilities in the campus, such as roads, buildings, parking lots, eating and shopping places, banks, and etc. The map should be published on the Internet to reach multiple communities at any levels of computing experience, everywhere in the world. It must be detailed, fast, user

friendly and easy to use. The users will make queries through the search interface to find the locations of their targets easily. They will also be able to browse the map with several navigation tools, and get information about the features that they are interested in.

It is important that such a system should be updated frequently, since new structures will be built and services can be changed. Some authorized users can access the database and make these alterations with their METU user accounts. For that reason, the system should be hosted with METU servers.

1.4. Methodology

In order to build that system, graphical data of METU were obtained from previous studies, then edited and processed by GIS tools. Non-graphic data were collected from various sources in METU, processed, and referenced to their graphical locations. Map graphics were also stored in data tables after being converted to Scalable Vector Graphics (SVG) format, which is an Extensible Markup Language (XML) standard to create two-dimensional graphics. A user friendly web page was prepared, which displays the main map, map navigation tools, layer controls and search capabilities, and published on the Internet. Additional user interfaces were created for authorized users to edit the map data.

The system was easily integrated with METU systems since SVG is an XML specification, and a map server was not needed. Data are stored in PostgreSQL database, and data transmission is provided by PHP. Hence, web-based implementation of this study is completely an open-source system.

CHAPTER 2

WEB-BASED GIS APPLICATIONS

2.1. Internet GIS Technologies

Since the importance and advantages of web-based Geographical Information Systems have been widely understood, there has been a remarkable growth in use of these systems with the latest advances in Internet technologies. Internet is a considerable data source and it makes GIS data distributed through a common interface in a centralized location and accessed all over the world by anyone who has just an Internet connection and a simple web browser, without need for purchasing and installing any complex and expensive GIS software. Therefore, sharing, manipulating, collecting and updating the geographic information became easier.

In desktop GIS, user interface, data storage and processing elements are normally present on a single machine, but these elements are spread across several machines in web-based GIS. That is the main difference between web-based GIS and desktop GIS [Green and Bossomaier, 2002]. On the other hand, web-based GIS has many common functionalities with desktop GIS such as zooming, panning, viewing full content, measuring distances, buffering and querying. Geographic data can be edited easily if the system architecture is suitable.

Xerox PARC Map Viewer was the first web-based map-building program [Green and Bossomaier, 2002], and developed in June 1993 by Steve Putz at Xerox Corporation's Palo Alto Research Center as an experiment in interactive web mapping. Map Viewer was able to accept requests from the users, create new maps and HTML documents from the geographic databases according to these demands, and publish them on the Internet [Putz, 1994]. After that innovation, the extent of

maps and geographic data served on the Internet started to increase [Su, et al., 2000].

Web-based GIS are used in various areas nowadays. Local governments, environmental and educational institutions, resource managements, businesses and land information systems are some examples of the application areas that use these systems [Aydın, 2006, Aydınoğlu and Yomralıoğlu, 2002]. Also, there are many software companies that are responsible for production, development, installation, support and marketing of some software products used for publishing maps on the Internet.

After the developement of both Internet and GIS technologies, several different techniques have been improved and different programming languages are used to implement web-based mapping applications. The most common approaches for web-based GIS techniques are the client-side and server-side strategies. Client is defined as the software component that can invoke an operation from the server [OGC Inc., 2006] and can be accepted as the web browser of the user's computer which displays the map interface. When browsing and querying the map, the client sends requests to the server-side, where data storage and all of the processing functions take place.

Each side can be categorized as "thin" or "fat (thick)", in terms of the workload and the number of the softwares installed to perform the task. "Thin" refers to the minimal number of softwares while "fat" refers to relatively higher number of softwares [Abel, et al., 1998].

2.1.1. Client-side

Thin-clients rely on other components which manage data and process the system functions. No additional softwares are needed to be installed other than a simple web browser, such as Internet Explorer, Netscape, Safari or Mozilla Firefox. They send requests to these components, which can either be a map server or a web application server, in order to view the map content on their web browser, in a usually GIF or JPEG image file format.

Some client-side scripting languages can improve interactivity of the static maps on both thin and thick-client. Scripts are executable list of commands created by scripting languages for controlling software applications. JavaScript, which is now specified as ECMAScript, is the scripting language designed for web applications in order to improve user interface. JavaScript code is embedded in HTML files and it provides powerful user interfaces and efficient interactions within these documents.

If more interactivity capabilities on the client-side are desired, thick-client strategy can be considered. Some GIS capabilities can be downloaded and processed locally on user's computer. Therefore, the server is not forced to do most of the work [Foote and Kirvan, 1998]. Since data are previously downloaded on the local computer, there will be no need to send requests to the server repeatedly. On the other hand, user computer should be powerful with a fast network connection and more sophisticated hardware configuration in order to download and process large amounts of data which may cause Internet traffic problems and increase download time.

Applets, plug-ins and ActiveX are some components of the thick-client side. These programs are usually forced to be installed on the user's computer in order to display the map and provide interactivity.

An Applet is a software component designed to run in another application. Java applets, Flash movies and Windows Media Player applets are some examples of these components. An applet comes with the data coming from the server, runs in the memory of the web browser, and it is unloaded when its related web page is closed [Kotzinos and Prastacos, 2001]. Therefore, the applets must be downloaded again and again every time that web site is accessed. Also, it may be time consuming to download an applet, depending on its file size. Java Applet is the application program coded in Java programming language and used to provide interactivity for the web documents. They have platform independency, which is the most advantageous property among the other client-side components.

A plug-in (or add-on, add-in) is a software module for a specific file type and interacts with the web browser to process a related function. Plug-ins are built for specific platforms and should be installed locally in advance. These programs remain permanent in the local computer and not to be installed again like the Java

Applets. Flash, QuickTime, Adobe SVG Viewer and Microsoft Silverlight are some plug-ins that are used for some presentation formats and video display.

An ActiveX control, which is similar to the plug-ins, is a component object model (COM) developed by Microsoft in order to extend browser capabilities. Many Windows applications use ActiveX controls when building their feature sets so that their functionality can be embedded by other applications. However, ActiveX controls can only run in Windows platforms [Mozilla, 2009].

2.1.2. Server-side

Server-side contains the web server, application server, web feature services, and database server.

Web server, which is also known as HTTP server, processes the requests coming from the client and delivers the results as some web documents or appropriate messages back to the remote client. Server-side scripting languages (PHP, Perl, ASP, etc.) help the HTTP server to create dynamic web pages.

Database servers host database services within computers or programs. They store both spatial and non-spatial data, and provide data access through some querying languages, such as Structural Query Language (SQL), XQuery and XPath.

Web Feature Services (WFS) allow data access and manipulation on the geographical features and elements across the web. The features can be created, updated, deleted and queried with these services. Map servers create maps by retrieving the spatial data dynamically, usually by vector to raster conversion, and deliver them in bitmap file formats, which can be viewed on any web browser of the user's computer. ESRI ArcIMS, MapInfo's MapXtreme, Intergraph GeoMedia Web Map, and an open source product Mapserver are some examples for the feature servers. They differ in input and output file formats, in the way of publishing the maps, spatial analyzing capabilities, server platforms, and client-server balance.

An application server is a system software that provides business logic within a distributed network for use by other applications. It hosts services for the

transmission processes and can be accepted as a middleware that manages the connections among the server-side components and the client-side.

Since most processing tasks are handled on this side, the client machine needs minimum requirements for the communication with the server. However, the server may be overloaded when many requests are received simultaneously, which may cause high network traffic. Hence, the server must be very powerful to overcome this workload and prevent time delays during processing.

2.1.3. Client-server architecture

Client-server architecture gives the relationship and distinction between the client and the server sides. Both sides can be accepted as two distinct components that work together for a common goal and called as “tier” [Shea, 2001]. “Two-tier architecture” is a common approach for client-server architectures and consists of “presentation tier” and “data tier”. Presentation tier refers to the client-side where the user interface and final results can be displayed. Information is stored in a file system or a DBMS (Database Management System), and retrieved by “data tier”. In this model, the client makes an HTTP request to the web server with a web browser across the Internet. Then the web server retrieves the information, processes the request and sends it back to the client in order to be viewed in the web browser.

Two-tier architecture may be insufficient for modelling more complex systems like web-based GIS applications. In such cases, computational functions and data can be located in another software agent, “Logic tier”, which acts as a middleware when executing the application. It coordinates the application by making logical decisions and calculations, and processes data with communicating with the data tier in two directions [Ramirez, 2000]. In web-based GIS, map servers can be accepted as the logic tier. When the client makes a request for a map, the web server sends a request to the map server, and the map server generates a map in a bitmap image file format like JPG, PNG or GIF. After receiving the image file from the map server, web server delivers it to the client as a response [Figure 2.1]. This model is called “Multi-tier architecture”.

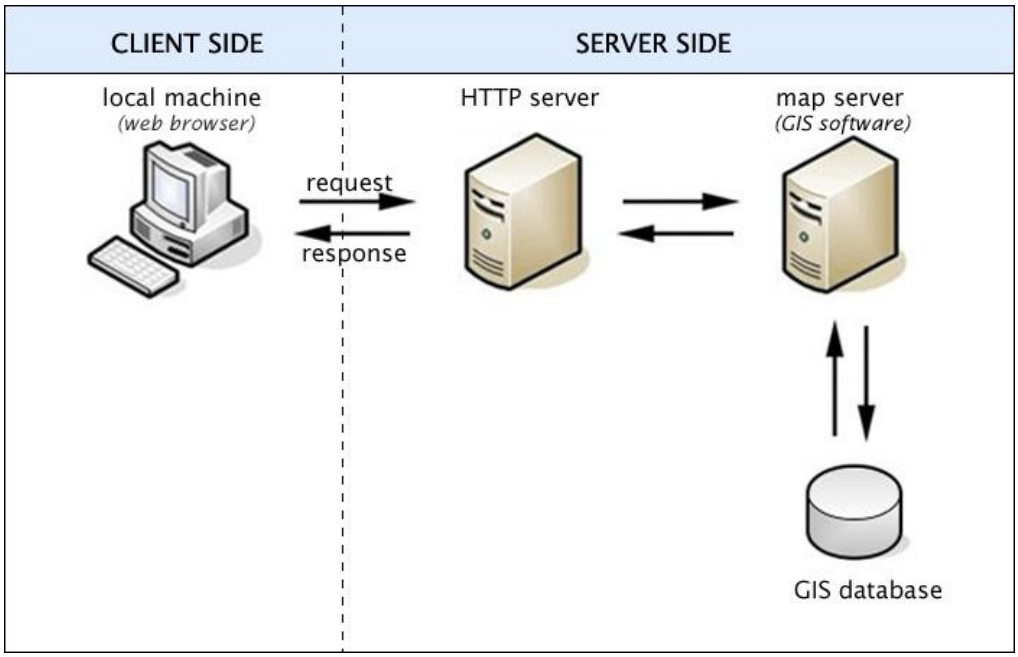


Figure 2.1 A typical approach for client-server architecture

2.2. Choosing the Best Architecture for Web Mapping

When choosing the best system architecture for web-based mapping systems, several factors should be considered, including available bandwidth, hardware and software cost, graphic quality, interactivity, client-server balance, weight of the data and file sizes.

Client-server balance is about the ratio of processes that each side undertakes and should be manipulated before designing the system. A thin client needs a heavy server while a thick client can handle many tasks without heavy network load during the interaction with the server.

Map servers generate maps according to the requests coming from the client, generally in bitmap image file format. The map data should be updated every time a request is done by the client, even if it is a simple zooming or panning command, and this may be done for multiple users simultaneously. That requires heavy network connections and sophisticated hardware configurations on the server-side. Besides, map server softwares are usually very expensive except the open-source ones such as MapServer. Thus, using a map server generally increases the cost of implementation. However, the client-side only needs a standard-minimal hardware

configuration with a fast network connection since most of the process occurs on the server-side.

Another approach is implementing most of the mapping functions on the client side. That will need a fat-client, which means a relatively stronger client computer and a standard HTTP server. All the map data will be downloaded in the client computer where the map interactions and data rendering functions take place. Hence, this approach is not suitable for large data sets.

Graphical quality also depends on the system architecture by means of output image format. The map image can be in either raster or vector file format. Since the size of the raster images are defined in pixels and each pixel stores the information of the graphic, they usually have large file sizes, even for a small image. Also, they lose image quality when they are rescaled to larger sizes and have degradation due to pixelation. On the other hand, vector image files have low file sizes and can be rescaled to very high values without losing their visual quality. It is possible to create graphical objects such as lines, polygons, circles or paths, and all the attributes and properties of these objects can be accessed within the document and linked to database tables, which enhances interactivity.

Choosing the most proper programming language is another important issue before constructing a web-based GIS system. An open-source server-side scripting language which is relatively easy to be developed and has a fast access to various database systems can be a good choice.

2.3. Web-based Mapping at Other Universities

For this study, web-based mapping applications of seventeen universities are studied, five of which are Turkish universities. There are also ten from USA, one from Canada, and one from Germany [Table 2.1]. These universities usually use GIS for planning, designing and construction purposes. Navigation through the campus becomes easier for universities especially with large areas, and getting detailed information about the buildings and services is possible with this technology. The larger the area and the population of a university campus, the more it needs an interactive campus map.

Table 2.1 List of the universities examined

Turkish Universities :

- Anadolu University
  http://harita.anadolu.edu.tr/
- Ege University [Figure 2.2]
  http://egeweb.ege.edu.tr/harita/
- Hacettepe University
  http://hun.edu.tr/harita/
- İstanbul Technical University
  http://www.karto.itu.edu.tr/kampus/index.html
- Pamukkale University
  http://www.pau.edu.tr/Pau30/harita/harita.html

Canadian University :

- University of Calgary
  http://ucmaps.ucalgary.ca/website/CampusWeb/viewer.htm

German University :

- Dresden Technical University [Figure 2.3]
  http://navigator.tu-dresden.de/navigator/leitsystem.xsql?neu=map
  http://www.carto.net/papers/svg/campus_dresden/

USA Universities :

- Emory University
  http://emap.fmd.emory.edu/website/campus/index.htm
- Marshall University **
  http://map.marshall.edu/
- Massachusetts Institute of Technology
  http://whereis.mit.edu/map-jpg
- Northern Illinois University
  http://www.webmap.niu.edu/framesetup.asp
- University of Arizona
  http://iiewww.ccit.arizona.edu/uamap/map.asp
- University of California, Riverside
  http://www.campusmap.ucr.edu/campusMap.php
- University of Georgia [Figure 2.4]
  http://maps.uga.edu/website/htmlviewer/hyperlink/viewer.htm
- University of Missouri-Columbia
  http://accessibilitymap.missouri.edu/
- University of Oregon
  http://map.uoregon.edu/
- University of Utah
  http://www.map.utah.edu/index.jsp

***Last accessed in September 2008*
Others are last accessed in March 2009

Most of the maps contain search options by building name, building ID or department name. Some of them show the names of the buildings with combo boxes, either grouping or directly listing them all. Generally, zoom (in/out/full extent), pan, identify, measure buttons are used. Some maps have different selection options and buffering capabilities, and most of them have legends [Table 2.2].

Table 2.2 Some capabilities of the studied campus maps

	Photo	Info	List of buildings	Search	Layer control	Aerial photo	Find path	Measure distance	Buffer	Zoom special	Measure area
Anadolu University	x		x								
Dresden Technical University	x	x	x	x			x				
Ege University	x	x	x	x		x	x				
Emory University	x	x	x	x		x				x	
Hacettepe University			x								
İstanbul Technical University			x								
Massachusetts Institute of Technology	x	x	x	x		x					
Northern Illinois University			x		x			x		x	x
Pamukkale University			x								
University of Arizona	x	x	x	x	x						
University of Calgary		x	x	x	x	x		x	x	x	
University of California, Riverside			x		x						
University of Georgia		x		x	x			x	x	x	x
University of Missouri-Columbia			x		x						
University of Oregon				x	x	x					
University of Utah	x	x	x	x	x	x					

Common layers shown on these maps are buildings, roads, computer labs, gates, parking spaces, ramps, memorials, recycling bins, elevators, handicapped parking and restrooms, food services, healthcare, landmarks, wireless Internet access, transportation (bus routes, bus stops), telephones, aerial imagery, and ATM's. Campus maps of University of Arizona and University of Missouri-Columbia have detailed information about accessibility facilities. The layers include elevators, handicapped parking, accessible entrances, accessible with accessible restrooms, accessible without accessible restrooms, ground floor accessible with accessible restrooms, ground floor accessible without accessible restrooms, not wheelchair accessible, accessible entrances, automatic doors, accessible parking, elevators, chair lifts, and ramps.

The maps are generally published by ESRI Internet Map Server products [Figure 2.4], supported by ASP, .NET, and Flash. In addition, Ege University uses Google Maps with efficient search options [Figure 2.2], and Dresden Technical University has an alternative campus map created by SVG [Figure 2.3].

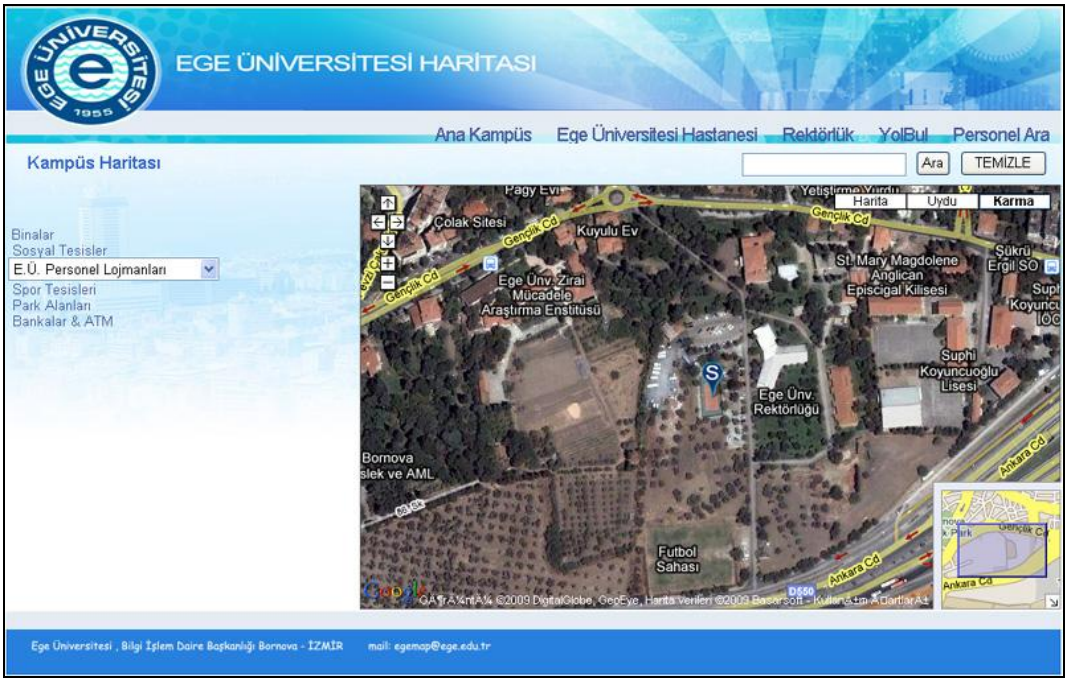


Figure 2.2 Campus map of Ege University

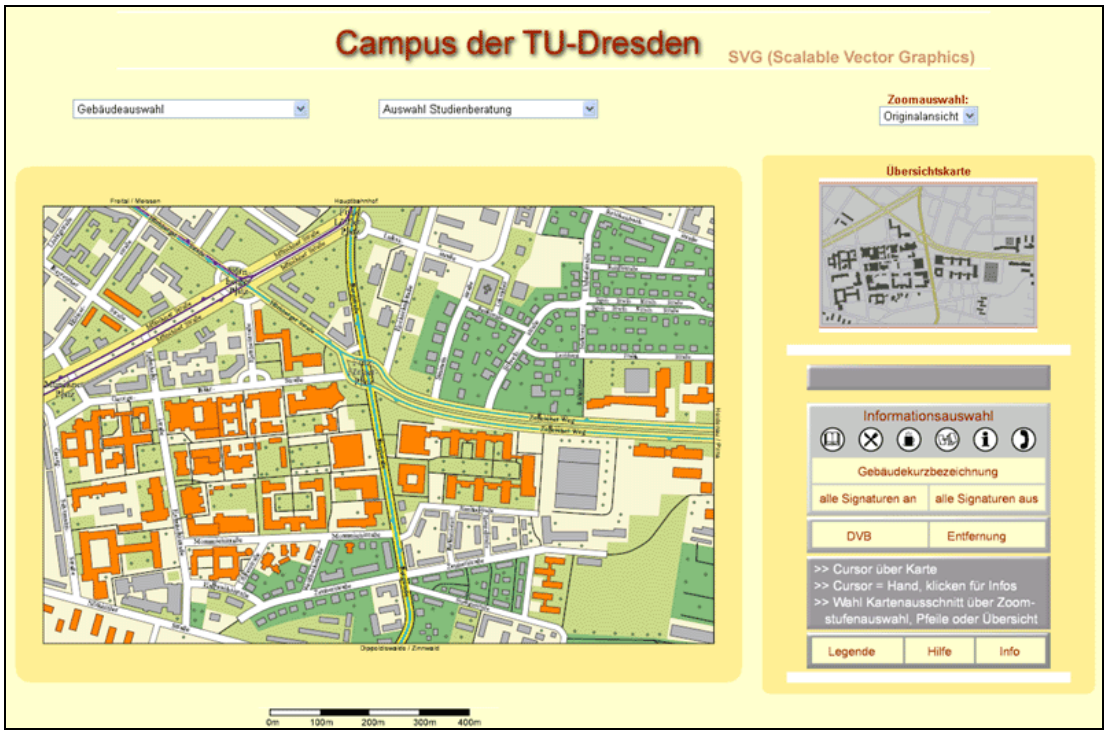


Figure 2.3 SVG Map of Dresden Technical University

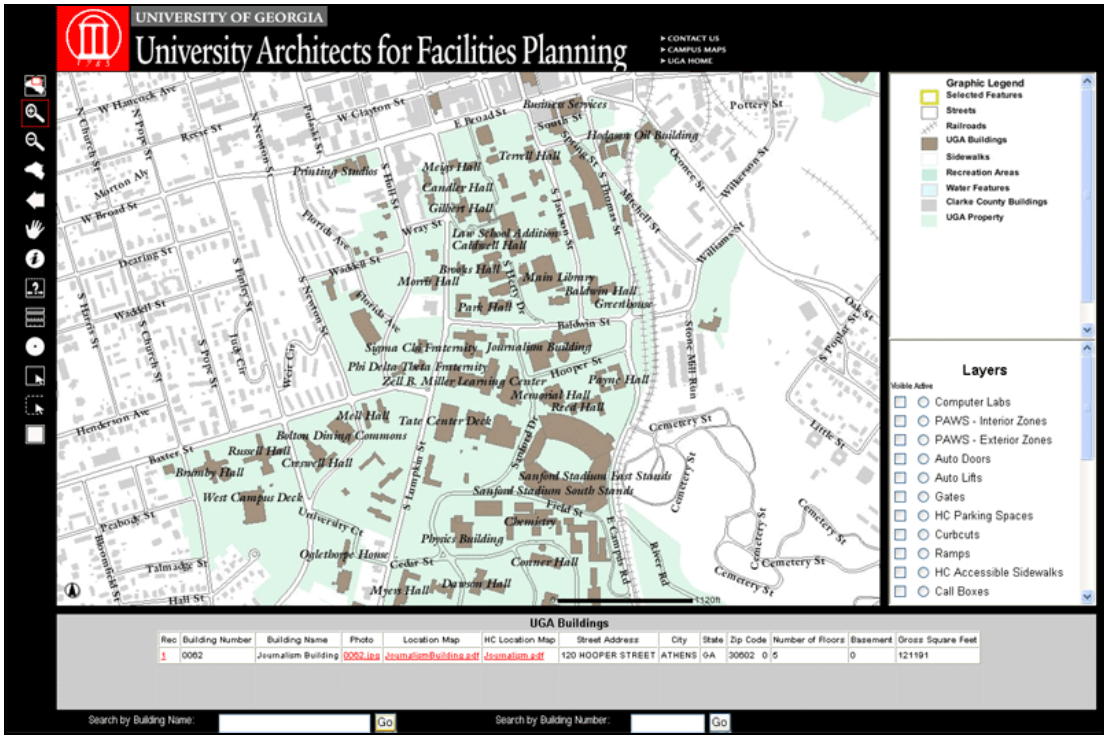


Figure 2.4 Campus map of University of Georgia

CHAPTER 3

DESIGNING THE SYSTEM

Before designing the system, it was decided to host the full system in METU Computer Center servers. METU Campus hardware and software infrastructure consists of IBM, HP and SUN servers with operating systems Unix AIX, Linux, HP-UX and Solaris [METU Computer Center, 2008]. Maintenance of these servers is supported by Technical Support Group of METU Computer Center. Constructing a map server in these systems can increase software cost and give extra responsibilities to this group for installing the software and maintenance of the server. In order to avoid that, an XML based, open-source system that is compatible with the hardware and software infrastructure of the campus can be implemented. Scalable Vector Graphics, which is XML based, can be a good choice for publishing maps on the Internet. It is possible to create interactive maps linked to large datasets stored in any database system. Map interactivity and communications between the map and the database can be provided by JavaScript and PHP scripting languages.

3.1. Markup Languages

A markup language is a set of annotations inserted in a text document to indicate how it will be structured and formatted. It is a collection of information about the apperance of the text file when it is printed and presented. What markups are allowed and how they will be interpreted should be specified by the markup language. HTML (HyperText Markup Language) is the most well-known markup language and used for encoding the web documents to be published through the HTTP. XML, which will be explained in the next section, is another example for markup languages and widely used nowadays.

Formatting is specified by some markup instructions inserted in certain places in the text, which are called “elements”. Elements are represented with “tags” by enclosing their names with angle-brackets “< >”. They identify and characterize the text portion between the starting and closing tags.

Tags can either be pre-defined or user-defined. HTML uses a set of familiar elements to format text portions, such as bold, italic, font or paragraph. For example, the text between “This is bold!” is marked up with tag, and it is displayed in bold letters when published.

Some elements have one or more “attributes” to describe their additional properties. These attributes are defined in the related tags by matching their names with their properties. For example, color, font type and size of a text portion in an HTML document can be formatted by changing the color, face and size attributes of the font element that tags that text [Figure 3.1].

HTML source :

```
<font size="+1" face="Comic Sans MS" color="gray">This text is
bigger, its font type is Comic Sans MS, and the color is
grey!</font>
```

Output :

This text is bigger, its font type is Comic Sans MS, and the color is grey!

Figure 3.1 Formatting a text portion in an HTML document

3.2. Overview of XML

E**X**tensible **M**arkup **L**anguage (XML) is a markup language like HTML, and it is a World Wide Web Consortium (W3C) specification for web documents containing structured information. XML is called “extensive”, because it is self-descriptive. There is no predefined tags like HTML has, users can create and define their own elements. Another issue that XML differs from HTML is that, it is designed to store and transport the data, not for displaying them.

Being a text-based format does not only require small file sizes, but also provides platform independency. Most systems store data in different formats and environments. Data exchange among these incompatible architectures and upgrading to new systems are very difficult. However, XML provides flexibility to connect various environments, databases and web services since it can be read by different applications. Therefore, XML can be defined as “a software and hardware independent tool for carrying information” [w3schools, 2009].

Studying the developement of XML based technologies shows us that XML seems to have a strong place in future application developements. Today, there are some other popular markup languages which are derived from XML and used for different applications, such as Scalable Vector Graphics (SVG), OGC recommended Geography Markup Language (GML), Keyhole Markup Language (KML) of Google, and Rich Site Summary (RSS) for web syndication. In this study, SVG is used to create vector based maps, and its structure and some properties are explained in the following sections.

3.3. Overview of Scalable Vector Graphics (SVG)

Scalable **V**ector **G**raphics (SVG) is an XML based standard for describing stylable two-dimensional vector graphics developed by the World Wide Web Consortium (W3C), which is a non-profit international consortium established for developing web standards and guidelines.

Many corporations participated in the developement of SVG, including Adobe Systems Inc, Corel Corporation, Microsoft Corporation, Autodesk Inc, Macromedia, Apple, Sun Microsystems, and IBM. SVG has been a W3C recommendation since 2001, and SVG 1.1 is the current specification that forms the core of the developement of the new one, SVG 1.2 [W3C, 2003].

Scalable means that the quality of the image is independent with the scale and remains the same in different display resolutions [Clarke, 2005]. Having this property, size and scale of SVG objects can be modified without losing the image quality, unlike the bitmap GIF and JPEG formats. These objects do not have degradation due to compression and not limited by fixed pizel sizes. They preserve clarity and sharpness when magnified by the local users [Behr, et al., 2006].

SVG supports three types of graphic objects: vector graphic shapes (paths consisting of straight lines and curves, polylines, rectangles, circles, polygons, ellipses, etc.), raster images (GIF, JPG, PNG), and text. Simple objects can be created with a few lines of text [Figure 3.2]. It is also possible to create more sophisticated graphical features that can be composited into rendered objects, styled with Cascading Style Sheets, grouped, transformed, animated, and displayed with clipping paths, alpha masks, transparency, and filter effects.

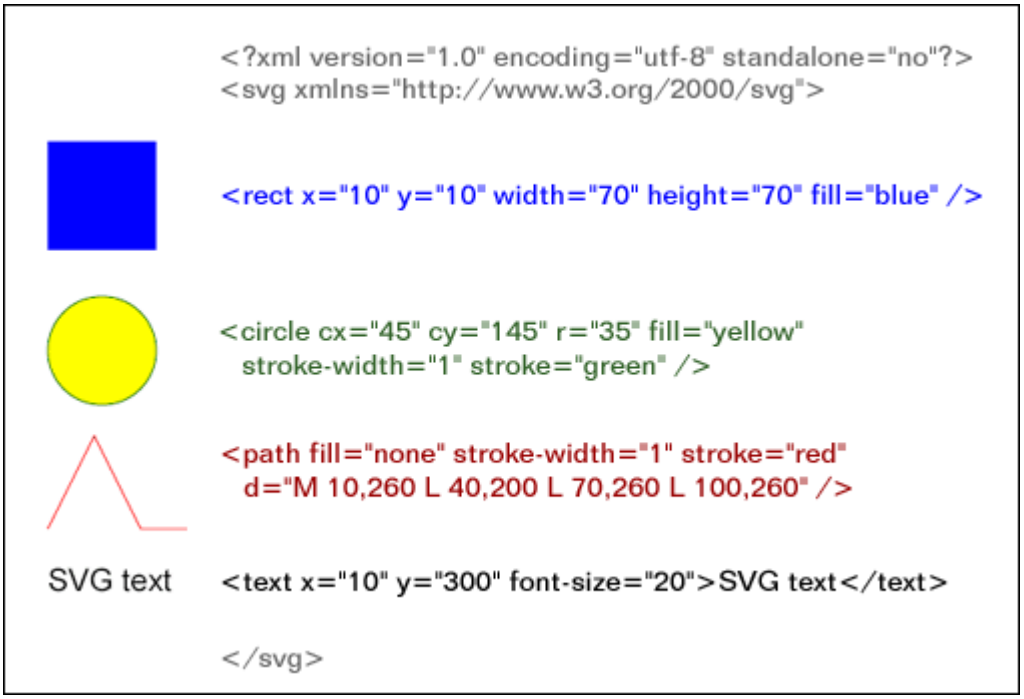


Figure 3.2 SVG source code of simple graphical objects

SVG documents can be viewed in any web browser provided that a plug-in, an SVG viewing program, is installed to the client side. Most commonly used one is Adobe SVG Viewer, and can be downloaded freely from Adobe's web site [Adobe, 2009]. Today, most of the popular web browsers except Internet Explorer have built-in SVG support. Also, many mobile phones have begun to support SVG format.

A wide variety of applications can be provided with SVG technology. SVG can be used for carrying different types of information such as statistical and technical diagrams, graphs, and maps.

3.4. Web-based mapping with SVG

Today, most mapping systems include displaying raster images which are created and delivered by servers according to the requests of the clients. These GIF or JPG bitmap formats are in lack of display quality when they are scaled to larger resolutions, and occupy larger file sizes that cause heavy network connections and long download durations. On the other hand, being a vector graphic standard, Scalable Vector Graphics is very suitable for Internet mapping applications since it provides excellent graphics which compound to form high quality maps scalable to any resolution with smooth apperance and reduced file sizes.

Interactivity is an important issue for Internet mapping applications. In traditional systems [Figure 2.1], maps are re-rendered after the requests of the users with map navigation tools, such as zooming, panning and layer control [Seff, 2002]. However, SVG maps can be controlled on the client side, once downloaded and viewed on the web browser, without sending requests to the server for a new map image again and again [Figure 3.3]. Map features are defined by each graphical object in the SVG document, and they can be linked to database tables which store the map's both spatial and non-spatial data. The interactions between these objects in the SVG document are achieved by some scripting languages. Most map controls such as zooming, panning, controlling the layers, getting info and thematic mapping can be implemented with JavaScript. These interactions can be improved by assigning some event handlers to the map features, in some events when the mouse is clicked over or its arrow is moved onto an element.

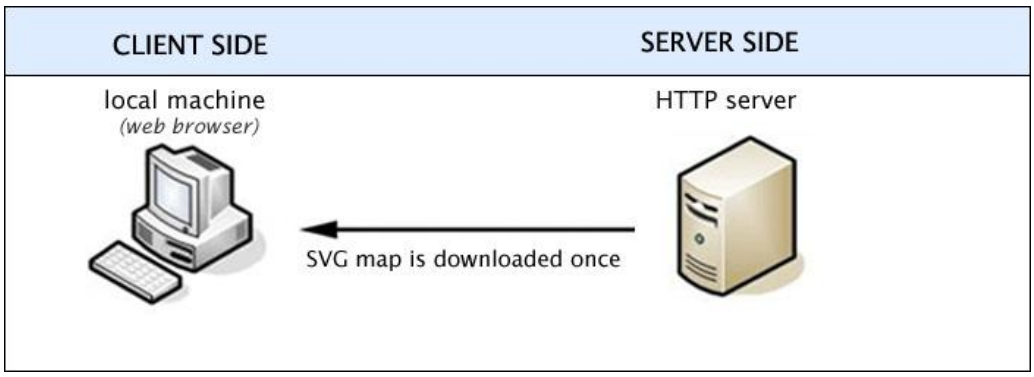


Figure 3.3 SVG map delivery

Sharing data and knowledge through the Internet is very useful for developement of many technologies, especially GIS. Since SVG is open-source and text based, map features and their attributes can be read and shared with the other GIS developers. It is possible to view and download the source code of the SVG documents by right-clicking the mouse over the graphics by choosing “View SVG Source Code” option. After downloading the source code, the map elements can be modified and published locally.

Using open source systems also reduces implementation costs. When SVG is used, a web and database server can be sufficient to distribute geographic data. There will be no need for setting up a GIS application server that requires very expensive softwares.

Being an XML based format provides several advantages to SVG. Combination with other technologies and communication with other platforms is possible [Behr, et al., 2006]. SVG documents can easily be styled with Cascading Style Sheets (CSS) and Extensible Stylesheet Language (XSL) to modify the appearance of the map. Animations and multimedia presentations can be prepared with Synchronized Multimedia Integration Language (SMIL). SVG text is selectable and searchable, and available to be indexed by search engines [Seff, 2002]. Also, SVG can be used as the output file format of GML, which is developed by Open GIS Consortium for storage and transport of geographical information. GML enhances interoperability and share of spatial data coming from different platforms [Chen, et al., 2004].

SVG does not offer full capabilities of GIS functions but it is an efficient and inexpensive way for representing spatial information on the Internet. In order to create SVG maps some extensions for two leading GIS softwares have been developed by external sources, SVGMapper for ArcView and Map2SVG for MapInfo. Also, shapefiles can be converted to SVG files with the exporting tool of ArcGIS software.

3.5. Study Area

Middle East Technical University (METU) is a Turkish state university, and its campus is located in the south-west of Ankara, on Eskişehir Highway, and 7 kms from the city center [Figure 3.4]. It has a total area of 4500 hectares [METU General

Secretariat, 2008], including the campus area of about 200 hectares, and extends to the south, near Gölbaşı Village. METU land also covers Yalıncak Village and Lake Eymir, which is near Gölbaşı, with an area of 120 hectares, and 20 kms from the city center. All the faculties and departments are in the main campus except Graduate School of Marine Sciences that is located in Erdemli-Mersin and Northern Cyprus Campus, and they are out of concern in this study.

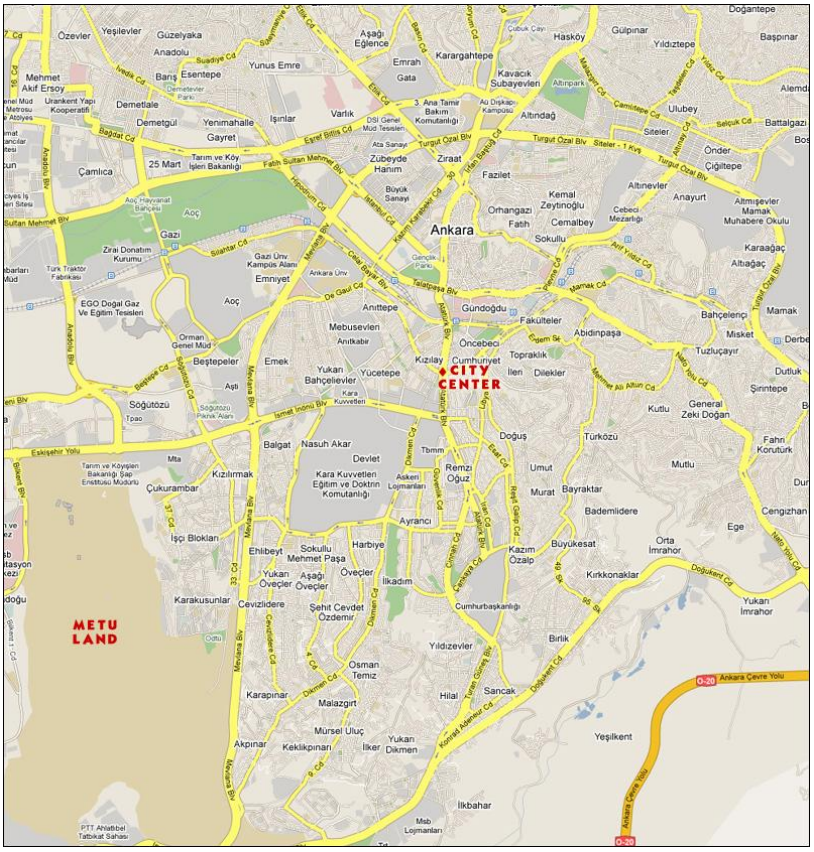


Figure 3.4 Location of METU in Ankara [Google]

METU was founded in 1956, with School of Architecture and City Planning, at a temporary location near the Parliement Building, having 40 students and four instructors. Construction of the campus in its present location began in 1962, and the classes started at the new campus in October 1963 [METU, 2009].

In 1958, it is considered to afforest 75% of METU land in order to prevent existing erosion and provide a green area in Ankara. Since 1960, nearly 10 million trees have been planted by the effort of METU staff, students, and other volunteers like

some military and diplomatic groups. METU forest is the largest human-made forest ecosystem of Turkey, maybe even in the world. This ecosystem shelters many types of trees, over 140 species of birds and wild animals. In 1995, it is announced as "1st Degree Natural Site Area" by Ministry of Culture and Tourism [METU General Secretariat, 2008], and re-forestation programme of METU was rewarded by "Aga Khan Award for Architecture" in 1993-1995 term [Aga Khan, 2009].

The construction of new buildings and afforestation have never stopped since 1960's, accordingly the student population and campus area have been increasing. The first academic buildings were built on the center of the campus, in an an elliptic area from north to south, as seen in Figure 3.5.

Figure 3.5 Satellite image of METU

Most of the sports facilities, shopping center, Cultural and Conventional Center, health center, banks, post office, dormitories and residential buildings are located on the east side of that ellipse. METU Technopolis, ODTÜ-KENT, a new Sports Center and some academical buildings are some examples of the new ones that have been built on the western part, going far away from the campus center. There are also METU Primary and High Schools, and some technical buildings such as workshops, greenhouses, and printing house on the north-west of the campus.

Today, METU campus has 304.228 square meters of floor space as of date [METU, 2009]. This study covers 295 of campus buildings, with different usage types [Table 3.1].

Table 3.1 Categories of campus buildings

Usage Type	Number of Buildings
Academic	63
Administrative	8
Bank and ATM	10
Cultural	5
Dormitory	22
Education	8
Food and Drink	10
Health	2
METU Technopolis	16
Research	6
Residential	62
Shopping	5
Social	12
Sports	27
Technical	39
TOTAL	295

METU has five faculties (Faculties of Architecture, Arts and Sciences, Economic and Administrative Sciences, Education, and Engineering) with 47 undergraduate programs, and five graduate schools (Graduate Schools of Applied Mathematics,

Informatics, Marine Sciences, Natural and Applied Sciences, and Social Sciences) with 97 masters and 55 doctorate programs [METU, 2009]. Every year approximately 8000 new students are registered to different departments of Middle East Techical University, and the campus has a student population of around 20000 each semester. In Figure 3.6, change in the population of registered students in fall and spring semesters of last 20 years are shown as a bar chart.

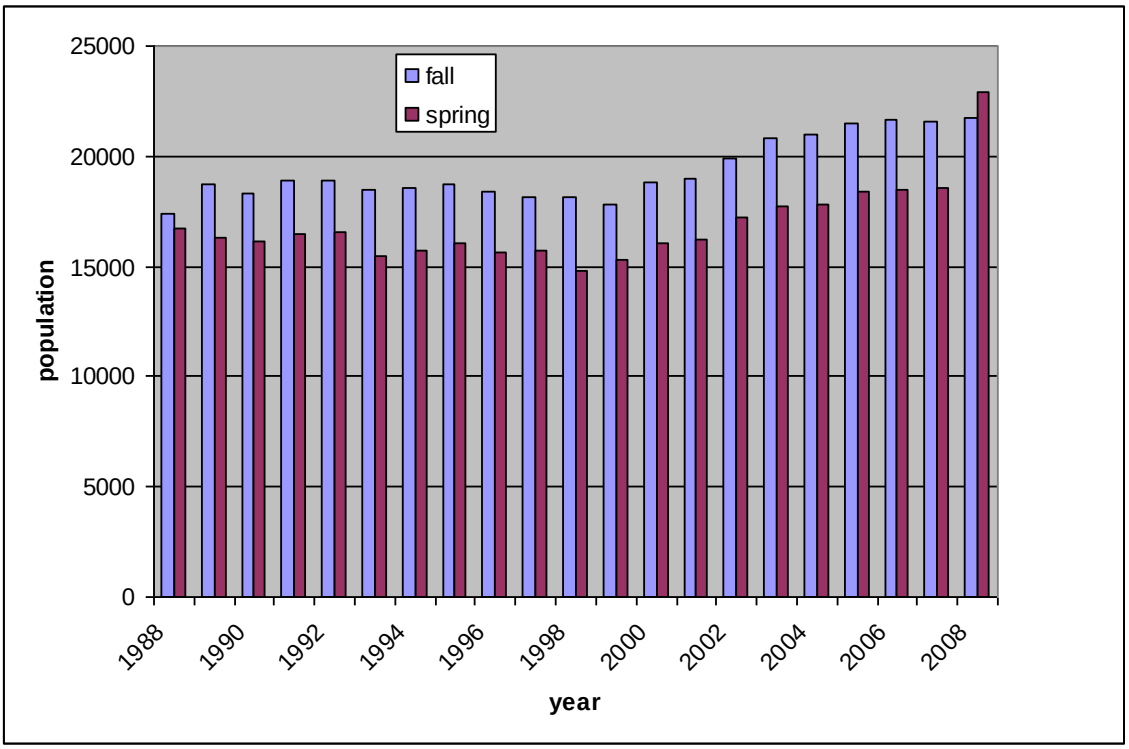


Figure 3.6 Student populations per semester from 1988 to 2008

In the campus, some departments have more than one building, and also, some buildings contain more than one department. For example, Department of Civil Engineering has seven buildings [Figure 3.7] while Faculty of Architecture Building has three departments, which are Departments of Architecture, City and Regional Planning, and Industrial Design. Also, students generally take classes from different departments, hence visit several buildings.

Figure 3.7 Buildings of Civil Engineering Department

By the end of 2000, after the completion of METU Twins and Halıcı Software, METU became the owner of the first science and research park of Turkey, METU Technopolis. In addition to these buildings, Silver Blocks, Silicone Blocks [Figure 3.8], Gallium Blocks and SATGeb buildings were constructed. According to the statistical results of 2007 viewed on the official web page of METUTech, it occuppies 110 hectares of METU land with 87.000 m^2 of indoor floor area and hosts 220 companies and 3730 employees. Activity areas of METUTech companies are mainly Information Technologies, telecommunication, electronics and defence industry. There are also other technology areas such as biotechnology, aerospace, environment, nanotechnology, and advanced materials [METUTech, 2009].

Figure 3.8 METU Technopolis [METUTech, 2009]

Further information about the campus will be given in the next sections, with the explanation of the data types and map layers.

3.6. Map Layers

The campus map was designed to present nine main layers, which are buildings, campus area with forest, satellite image, roads, transportation points, campus gates, wireless zones, disabled facilities, and parking lots.

The building layer carries the campus buildings and it is seperated into new sublayers according to their usage types. These sublayers are academic, administrative, cultural, social, education, health, METU Technopolis, research, residential, shopping and technical buildings, eating places, sports facilities, dormitories, banks and ATM's. Buildings that have similar categories are combined to minimize the number of layers for a simpler view. These sublayers are “Cultural and Social Buildings“, “METU Technopolis and Research Buildings“, and “Health, Shopping, Food and Drink”. Some buildings have more than one unit which have different categories. For these buildings, usage type of the dominant one is chosen. Tennis courts, football, basketball and volleyball fields are assumed as buildings and joined with the other sports facilies such as gymnasiums and swimming pools to create sporting areas sublayer.

For this research, campus area and road layers have only display purposes and have no attributes. Campus area is the base map, and it also shows forestry areas which have denser tree population and filled with a tree pattern. Road layer carries some graphical details such as traffic circles and refuges.

The main layers, their descriptions and corresponding object types of SVG are listed in Table 3.2.

3.7. Data Sources

There were three different data types for METU campus; a satellite image, shapefiles, and attribute data.

Table 3.2 Main map layers

Layer	Description	Object type
Buildings • Academic buildings • Administrative buildings • Health, shopping, food and drink • Cultural and social buildings • Residential buildings and dormitories • Technical buildings • Technopolis and Research • Education buildings • Banks, ATM's and Post Office	Building name; usage type; list of departments, units, companies, offices and shops; classrooms, amphitheaters, seminar rooms, laboratories; computer laboratories, pay phones, kiosks, cüzdanmatik machines; picture of the building	Polygon
Campus area	Terrain and forestry area with no attribute	Polygon
Satellite image	Satellite image of the campus in "jpg" file format	Raster
Roads	Campus roads with no attribute	Polygon
Parking lots	User types (stickers), disabled parking space	Polygon
Transportation points	EGO bus, METU rings, minibus and taxi stops, schedules and destinations	Point
Campus gates	Name and description of the campus gates	Point and text
Sports facilities	Swimming pools, tennis courts, gymnasiums, football, basketball and volleyball fields	Polygon
Wireless zones	Center of the wireless network areas	Circle
Disabled	Buildings that contain facilitaties for disabled people (Ramps, lavatories, parking spaces, elevators)	Polygon

All the layers were built on the same spatial reference of the satellite image which was obtained from General Command of Mapping in May 2007, in ERDAS IMAGINE format, with projected coordinate system of Transverse Mercator, and based on WGS 84. Spatial data of most buildings were obtained from Güllüoğlu (2005), which were in shapefile format having Transverse Mercator Projection ED 50 for Turkey. All the roads, lately constructed buildings and the other features were added to these data by digitizing them from the satellite image, in shapefile format again.

Some of the attribute data were obtained from different administrative and academic units through the help of General Secretariat of METU, who asked them with an official document, and some of them were gathered from their web sites. Some of the existing and missing information were verified and collected, and pictures were taken by visiting most of the buildings [Table 3.3].

Table 3.3 Data sources of the buildings

Building Type	Data Source
Library	Directorate of Library and Documentation
Guesthouses and housings	Office of Social Facilities
Dormitories	Office of Dormitories
Sports facilities	Office of Sports
Pay phones	Office of Telephones
Cultural and Conventional Center	Cultural and Conventional Center Administration
Academic and administrative buildings	METU web site Personal visit Department secretaries
Classrooms	METU Web Site Personal visit
Religious facilities	Office of Domestic Services
Parking lots	Office of Domestic Services
Transportation points	Office of Domestic Services Office of Transportation
Wireless zones, kiosks, cüzdanmatik	Computer Center
Kindergarten	Office of Kindergarten
Shopping Center	Personal visit
Disabled facilities	METU Disablity Support Coordination Unit

3.8. Dealing with SVG documents

After the satellite image was digitized by using ArcGIS 9.0, the shapefiles were converted to SVG documents with export function of ArcGIS. This function creates the graphical objects in path element of SVG, which are defined in <path> tags. Path is the general element of SVG which provides drawing of the outline of various graphical objects and common basic shapes like rectangles, circles and polygons, and it can be filled or stroked with several style definitions.

Attribute “d” of the path element defines the outlines of the shapes with a list of commands containing some letters and numbers. The letters give instructions for drawing shapes with some given points whose screen coordinates are represented by numbers. The most commonly used instructions are M (moveto), L (lineto), and Z (closepath). The code in Figure 3.9 draws a path with a line from point (75,0) to point (0,125), another line from the previous point to (150,125), and closes the shape which results as the triangle in the same figure.

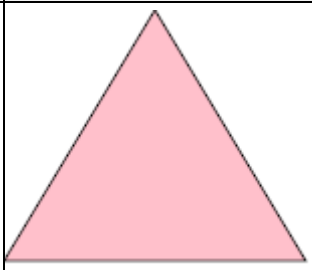
Source code	Result
<svg width=”100%” height=”100%” version=”1.1” xmlns=”http://www.w3.org/2000/svg”> <path **d=**”M75 0 L0 125 L150 125 Z” **fill**=”pink” **stroke**=”black” **stroke-width**="1" /> </svg>	

Figure 3.9 Drawing a SVG shape with path element

Styles of the objects are also defined with several attributes in path element of each object. Attributes “fill” and “stroke” decide background and outline colors of the graphic. Another commonly used attribute is “stroke-width”, and it assigns the width of the line outlining the figure [Figure 3.9]. There are other stroke attributes that come with the SVG document that has been exported from ArcGIS and give additional styling properties to the lines, such as stroke-miterlimit, stroke-linecap, and stroke-linejoin, which modify the junction and closing properties of the paths [Figure 3.10].

```
<path clip-path="url(#SVG_CP_1)" fill="none" stroke="#FF0000"
stroke-width="1.00808" stroke-miterlimit="10" stroke-
linecap="round"
stroke-linejoin="round" d="
M214.72048,342.30993L233.00989,340.58182L233.44192,349.79841
L233.87395,357.86292L215.72856,359.30302L215.72856,357.57491
L215.44054,354.83873L214.86449,346.19818L214.72048,342.30993"/>
```

Figure 3.10 Original object definition of a road

All of the path elements in the initial SVG document characterize their fill and stroke attributes in their tags. Common definitions repeat themselves within the document for the similar objects. For example, all the road objects have the same color and width properties. Since SVG is a plain text file format, excessive amount of text increases the file size. The simpler is the code, the smaller size the file has. Therefore, the source code must be simplified as much as possible by deleting unused definitions and eliminating duplicated attributes by generalizing them with a single attribute, class attribute, which is related to the style definition in the same document [Figure 3.11].

```
<path class="road" d="
M214.72048,342.30993L233.00989,340.58182L233.44192,349.79841
L233.87395,357.86292L215.72856,359.30302L215.72856,357.57491
L215.44054,354.83873L214.86449,346.19818L214.72048,342.30993"/>
```

Figure 3.11 Simplified object definition of a road

Since all the similar objects have common style definitions, a simple change in the related class will effect all the display properties of the objects at the same time. In Figure 3.12, there are two different classes, roads and academic buildings. The path element in Figure 3.11 has a class of “road”, which is linked to “.road” class in the style definition. Any change in “.road” class will effect all of the elements that have a class value of “road”.

```
<style type="text/css">
   <![CDATA[
      .road {
            stroke:#e8e800; fill:#cccccc
            stroke-width:0.47994; stroke-miterlimit:10;
            stroke-linecap:round; stroke-linejoin:round;
            }

      .academic {
            stroke:#6E6E6E; fill:#ffcc99
            stroke-width:0.3; stroke-miterlimit:10;
            stroke-linecap:round; stroke-linejoin:round;
            }
   ]]>
</style>
```

Figure 3.12 Style definition

Points are not supported as an object type by SVG, but they can be represented as small discs, squares or bitmap images. If specific template objects or images will be used repeatedly, symbol elements can be rendered, and duplicated definitions are avoided.

In order to show the areas that have dense tree population, a pre-defined pattern is used, which is the SVG element for filling or an object with other graphical objects or symbols. At first, a tree symbol was created by drawing some paths, then a pattern was defined with using this symbol. Lastly, forestry area was filled with that pattern [Figure 3.13].

Source code

```
//Symbol
<symbol id="tree" overflow="visible"
style="fill:green;stroke:green;stroke-width:2">
      <path d="M0 -15 l0 30"/>
      <path d="M0 -15 q0 3.5 5 3.5"/>
      <path d="M0 -15 q0 3.5 -5 3.5"/>
      <path d="M0 -9 q0 6 8 6"/>
      <path d="M0 -9 q0 6 -8 6"/>
      <path d="M0 0 q0 7 10 7"/>
      <path d="M0 0 q0 7 -10 7"/>
</symbol>

//Pattern
<pattern id="pattern1" patternUnits="userSpaceOnUse" width="10"
height="10">
      <use xlink:href="#tree"
      transform="translate(3,3) rotate(10) scale(0.3)" />
</pattern>

//Rectangle filled with the pattern
<rect x="10" y="20" fill="url(#pattern1)" width="100"
height="100" stroke-width="1" />
```

Tree symbol	Tree pattern
	

Figure 3.13 SVG symbol and pattern

Graphics that are related with each other can be organized with a grouping element in order to express the layers. All groups consist of several paths or other elements collected to form the same layer, between <g> and </g> tags. They must be given an identifier to be controlled, and a name for the layer. In Figure 3.14, “building” and “road” layers are created by combining similar objects.

```
<g id=”building”>
      <path id=”1” d="M 0 0 L10 20Z” />
               ....
      <path id=”2” d="M 1 3 L15 30Z” />
</g>

<g id=”road”>
      <path id=”1” d="M 5 8 L20 40Z” />
               .....
      <path id=”2” d="M 10 20 L30 60Z” />
</g>
```

Figure 3.14 Layer definitions with grouping

3.9. Organization of the Data

SVG data can be organized with two different approaches. They can either be carried as a whole content in a static SVG document, or graphical objects can be processed individually after storing their attributes with their unique identifiers in data tables, isolated from the document.

The choice depends on the variety of attributes, and the number of features. If the graphical objects do not have any information other than their shape attributes, a database related system may not be necessary. On the other hand, for more complex structures, separating the attributes from the SVG document provides efficient data update, simplifies the source code, and hence reduces the file size. Figure 3.15 shows how the attributes of the path elements are stored in the columns of a table.

Source code

```
<g>
  <path name="MM Building" id="1" fill="red" d="M 0 1 L 2 3 L 4 5 Z"/>
  <path name="Computer Center" id="2" fill="blue" d="M 6 7 L 8 9 Z"/>
  <path name="Library" id="3" fill="green" d="M 2 4 L 6 7 10 8 Z"/>
</g>
```

Attribute table

id	name	Color	Path
1	MM Building	Red	M 0 1 L 2 3 L 4 5 Z
2	Computer Center	Blue	M 6 7 L 8 9 8 7 5 Z
3	Library	Green	M 2 4 L 6 7 10 8 Z

Figure 3.15 Table-based mapping with SVG

In this study, both shape and attribute data are stored in the database, because some layers have large sets of attributes which can be updated frequently. When the map is downloaded initially, only the graphical and identifier data are delivered from the database. These attributes draw, style and identify the map features, and create the basic map with a small file size that can be downloaded quickly. Further information is retrieved only when it is required.

Geodata are stored as geometry attributes in shape tables, which are specialized according to their feature types [Figure 3.16]. These are :

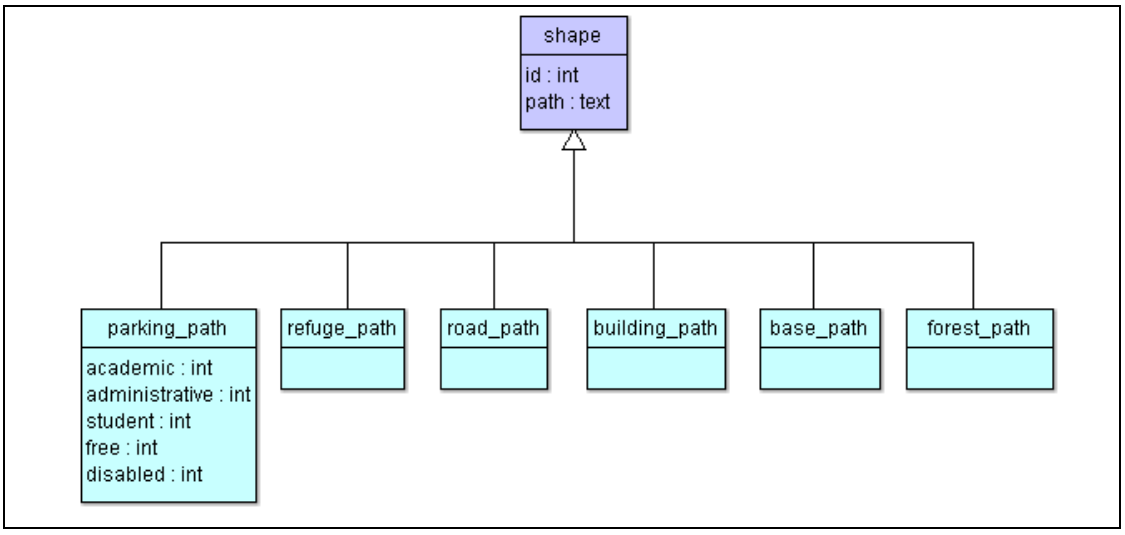


Figure 3.16 Shape tables

- BUILDING_PATH (*id*, path) : identifier and path attributes of campus buildings [Table 3.4]

Table 3.4 Sample view from building_path

id	Path
1	M248.27506,538.30648L251.87533,538.88252L250.00319...
2	M255.47561,534.99427L255.3316,534.99427L252.88341,...
3	M287.59008,513.82492L287.59008,513.39289L288.45414...

- ROAD_PATH (id, path) : identifier and path attributes of campus roads
- REFUGE_PATH (id, path) : identifier and path attributes of road details such as refuges, traffic circles, etc.
- FORESTRY_PATH (id, path) : identifier and path attributes of forestry area
- BASE_PATH (id, path) : identifier and path attributes of base area
- PARKING_PATH (id, path, academic, administrative, student, free, disabled) identifier and path attributes of parking lots, logical values of user types [Table 3.5]

Table 3.5 Sample view from parking_path

id	Path	acad	adm	stud	free	disab
1	M215.29653,321.28458L213.7124,321.28458L213.7124,...	0	0	0	1	0
2	M188.9425,318.83642L194.27091,307.31569L197.72718...	0	0	0	1	0
3	M188.65448,271.88942L189.66256,271.16937L188.5104...	1	1	0	0	0

These tables have similar table structures with two columns; one of them is the identifier, and the other one stores the content that will be inserted in the “d” attribute of the path tag. Tables parking_path, which has additional columns, and also building_path can be connected with the attribute tables, while road_path, refuge_path, forestry_path and base_path have only display purposes and no relationships with the others.

Information about the map features are presented in either Turkish or English, depending on the choice of the user. Therefore, non-spatial data are stored in two languages, in separate columns of the same table of corresponding entity types.

Building layer is created with the combination of several attribute tables and building_path shape table. They are joined with a common foreign key, which references the primary key of building table :

- BUILDING (bid, name, name_tr, *type_id*, description, description_tr, url) : identifier, name, identity number of the usage type, brief description about the building, and url adress [Table 3.6]

Table 3.6 Sample view from building

Bid	Name	name_tr	type_id	description	desc_tr	url
152	Tennis Courts Buffet	Tenis kortları Büfe	7	Tel:2101972	Tel:2101972	
341	Mechanical Workshop	Atölye	15	Paint	Boya	
38	Food Engineering	Gıda Mühendisliği	1			http://fde.metu.edu

- CLASSROOM (*bid*, name) : identifier of the building, list of the classroom and amphitheater codes
- TYPES (type_id, type, type_tr) : identity number and name of the usage types in both English and Turkish
- BUILDING_UNITS (entity_no, *bid*, name, name_tr, *type_id*) : index number, identifier of the building, name of any entity in a building (academic and administrative units, offices, rooms, facilities, eating places, shops, offices, companies, etc.), identity number of the usage type [Table 3.7]

Table 3.7 Sample view from building_units

entity_no	bid	Nme	name_tr	type_id
17	90	MM Canteen	MM Kantini	7
18	90	Department of Engineering Sciences	Mühendislik Bilimleri Bölümü	1
19	90	Graduate School of Social Sciences	Sosyal Bilimler Enstitüsü	1

- BUILDING_FACILITIES *(bid, type_id*) : identifier of the building, identity number of the telecommunication or disabled facility type
- PARKING_BUILDING (*pid, bid*) : identifier of the building, identifier of the parking lot which references parking_path table

For a better view of search results, classroom table was not normalized, and the classroom codes were listed together in the same cell for each building.

The usage types are stored in the types table in both Turkish and English. The "type_id" columns of building and building_units tables reference this table, and make it possible to create new sublayers of building by categorizing them according to their usage types. Table of building_units matches the identifiers of the buildings with the names and usage types of anything that are located in them.

Some disabled and telecommunication facilities of the buildings are organized together in table of building_facilities, which assigns type numbers of the facilities to their corresponding buildings. These types are entrance ramp, lift and handicapped restroom for the disabled, and kiosk, cüzdanmatik, pay phone and computer laboratory for the telecommunication facilities.

The table of parking_path stores both spatial and non-spatial data of the parking lots [Table 3.5]. User types of the parking lots are determined by the color of the sticker that is taken from Office of Domestic Services of METU, and they can either be academic (red), administrative (green), student (yellow), or others. The table also gives information about the existence of any handicapped parking space. A parking lot can be shared by more than one building, and a building may have more than one parking lot. Therefore, table of parking_building was created to match the parking lots with the nearest buildings to them.

These tables will be joined with each other when a phrase is searched to find the location of a building or a classroom, or detailed information about a building is requested. By referencing the table of types, buildings and their entities can be categorized, and facilities will be listed. Their table structures and relationships can be seen in Figure 3.17.

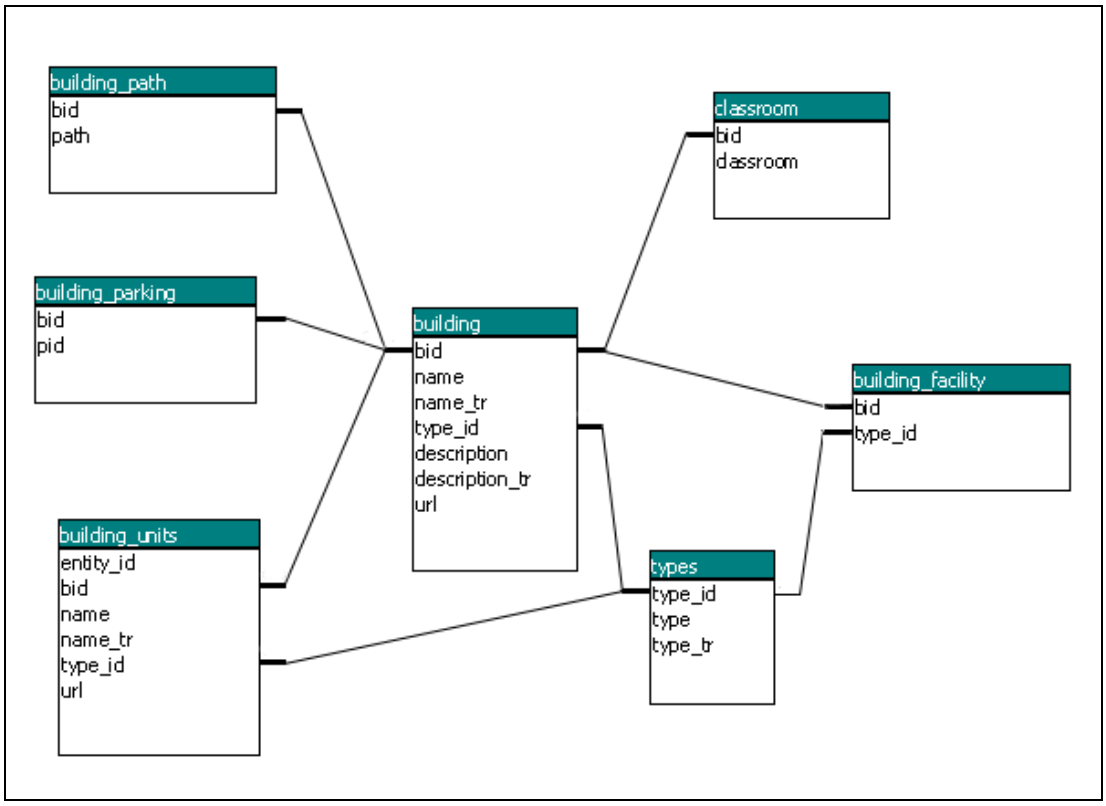


Figure 3.17 Relations of the tables that create the building layer

Transportation, wireless network points, and campus gates are represented by symbols which are rendered from bitmap images, and viewport coordinates of these symbols are stored in separate tables:

- TRANSPORTATION (tid, x, y) : identifier of the transportation point, x and y coordinates of the symbol [Table 3.8]
- TRANSPORTATION_INFO (entity_no, *tid*, description, description_tr, *type_id*) : index number and identifier of the transportation point, description about the transportation vehicle such as schedules and destinations, type number of the transportation type (EGO bus, campus ring, taxi, minibus) [Table 3.9]
- WIRELESS (wid, x, y) : identifier of the wireless network point, x and y coordinates of the symbol
- CAMPUS_GATES (name, x, y, description) : unique name of the gate as the identifier, x and y coordinates of the symbol, description about the location of the gate

Table 3.8 Sample view from transportation

tid	x	y
1	316	320
2	321	363
3	322	410
4	321	406
6	323	405
8	322	459
9	301	499
10	289	469

Table 3.9 Sample view from transportation_info

entity_no	tid	type_id	description	Description_tr
17	1	19	Tunus 10:40 11:40	Tunus 10:40 11:40
18	1	19	Kızılay 9:30 10:30 11:30	Kızılay 9:30 10:30 11:30
19	2	21	Ayrancı-weekdays until 18:00	Ayrancı-haftaiçi 18:00'a kadar

CHAPTER 4

SYSTEM DEVELOPEMENT

The system architecture was built on a multi-tier system as illustrated in Figure 4.1. The client is the presentation tier and considered as a thick client since most of the map browsing processes take place on that side, and consists of web browser, SVG plug-in, SVG documents and JavaScript functions.

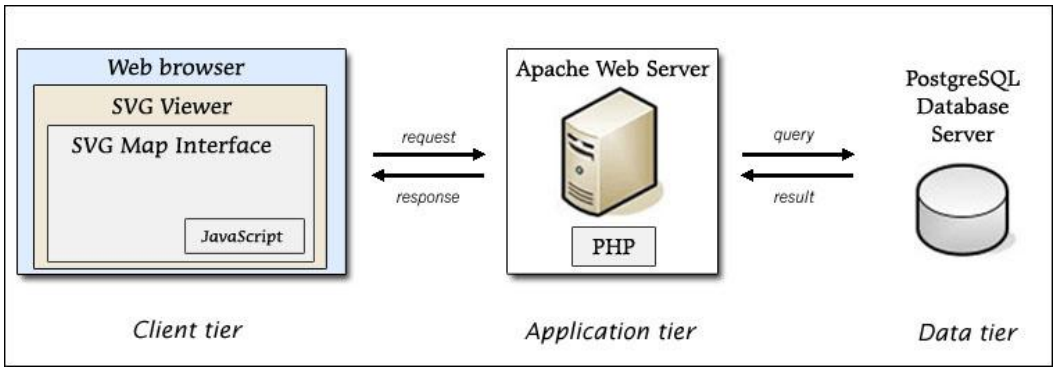


Figure 4.1 System Architecture

Data transmission between the client and the database is provided by the application tier, which is the Apache web server with PHP: Hypertext Preprocessor support. Unlike the traditional web based GIS systems, a map server does not exist. Map data are stored in data-tier, which is the PostgreSQL database, and delivered by PHP scripts to produce the map.

After the initial map is rendered by PHP scripts and displayed on the web browser, map navigation tools and layer control can be used without any server access. Graphical data are completely installed to the client, and all map browsing functions performed locally, preserving the graphical quality. However, searching operations and getting detailed information about the features require contact with the web

server. Search operations result with a double-sized new drawn map on which the searched feature is centered and highlighted. On the other hand, getting information functions keep the main map in the parent document, opens a new window that makes requests from the server, and displays the attribute data.

The system elements are explained in detail in the following sections.

4.1. Development of the Client-side

When dealing with web-based applications, the maps are displayed on the web browser of the local computer, which can be accepted as the client side. According to the report of Market Share (2009), Microsoft Internet Explorer (66,10%), Mozilla Firefox (22,48%) and Safari (8,21%) are the most widely used browsers all over the world. In order to view SVG documents with Internet Explorer, an SVG compatible plug-in has to be installed at once. On the other hand, latest versions of Firefox and Safari have SVG support and they do not need any plug-in to be installed to display SVG content. Adobe SVG Viewer (ASV) is the most full-featured SVG plug-in [SVG Wiki, 2009], and was installed to the local machines that were used for testing of the implementation with Internet Explorer.

All of the operations are viewed from the main page which has a PHP file format. This web document consists of search section and the layer control. It also displays the map content as an SVG document which is located in another PHP file and embedded in the main page.

Figure 4.2 represents the main page, ‘index.php’, divided into three sections. Querying section and layer control are located on the right side of the web page. On the left part, the map and its navigation tools are displayed as if they are in the same document. However, map data are created and controlled in the child document, named as ‘embed.php’. All map browsing processes except the layer control take place in ‘embed.php’ without any interactions with the server and the parent page. On the other hand, parent page controls the child with searching options and layer control. There are also some information files which are opened as pop-up windows, activated by the info button of the map tools in the SVG document.

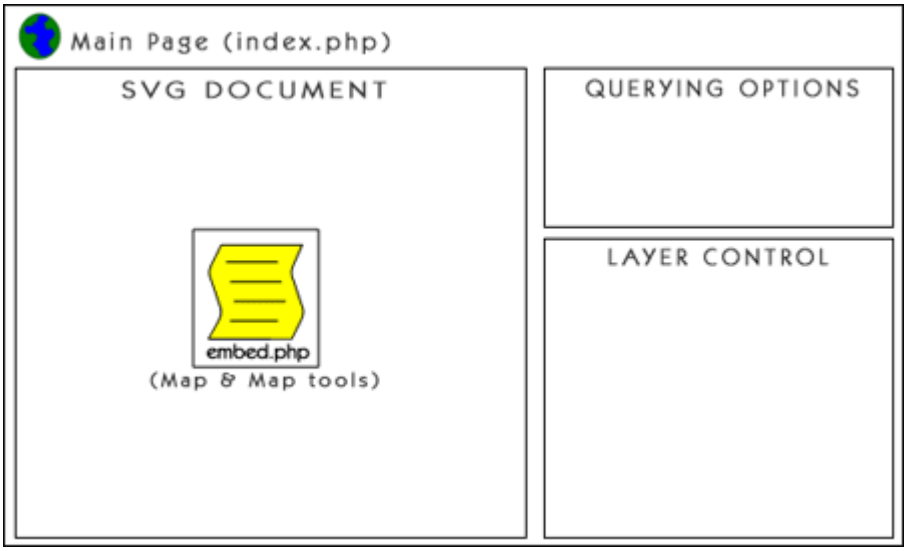


Figure 4.2 Representation of the main page

Although the child document has ".php" file extension, the output is an SVG document whose structure is summarized in Figure 4.3. There are two different SVG fragments which are nested. The map is located in the child SVG content, named as "map". After the style, symbols, patterns and JavaScript functions have been defined, map is drawn with groups containing path, image and text elements of SVG.

Interactivity capabilities were customized by JavaScript functions, which were originally created for this application. These functions comprise map browsing tools, info button, layer control, and mouse events, and controlled by the buttons in the toolbox, placed in the parent SVG content.

Order of the objects in SVG documents is very important, because the objects defined in the bottom of the document overlay the upper ones if they are located in the same areas. Therefore, the toolbox is created at the bottom of the document.

```
<SVG id="main">
        <SVG id="map">
                <symbol>
                    //Symbol definitions
                   //Pattern definitions
                </symbol>
                <style type="text/css">
                   //Style definitions
                </style>
                <script type='JavaScript'>
                   //JavaScript functions
                   //Map navigation tools
                   //Mouse events
                </script>
                <g id="mainmap">
                   <g id="base">...</g>
                   <g id="forrest">...</g>
                   <g id="academic">...</g>
                   <g id="administrative">...</g>
                        ...
                   <g id="roads">...</g>
                </g>
        </SVG>
<g id="toolbox">
        Zoom in button
        Zoom out button
        ...
</g>
</SVG>
```

Figure 4.3 Document structure of the SVG file

4.1.1. SVG Canvas

SVG content is rendered in an infinite space, which is called “SVG canvas”. By specifying width and height of the canvas in the outermost SVG definition tag, SVG canvas can be delimited with a finite region, “SVG viewport”, in unit of pixels. Viewport is the initial user coordinate system, and its origin matches the top-left corner of the SVG canvas [W3C, 2003] [Figure 4.4]. Within the viewport, many new fragments can be defined with different coordinate systems, and it can be controlled by “transform” attribute. It is possible to use a transformation matrix in the transform attribute to keep the original map coordinate system. Transform attribute has also translate, rotate, skew and scale properties. Translate and scale are the most frequently used ones for controlling the map display. Scale states a scale factor which is multiplied by the original size of the document in both x and y dimensions,

and translate creates a new coordinate system by assigning the new coordinates of the origin.

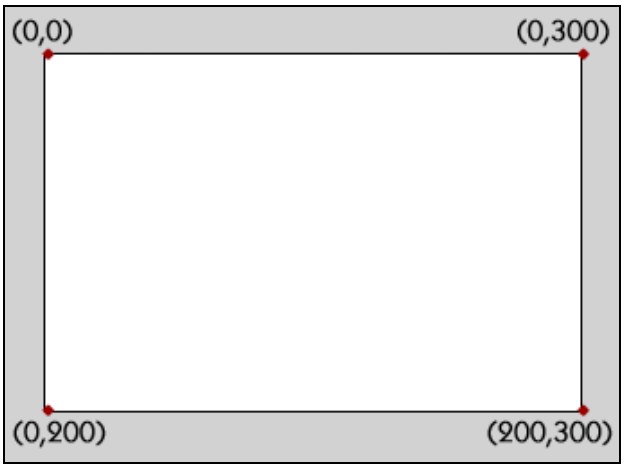


Figure 4.4 Initial coordinate system of a SVG viewport

In this study, the original SVG document which was exported from ArcGIS has a width of 594.73701 pixels and a height of 842.25827 pixels initially, but the viewport is adjusted to 650 and 600 pixels respectively. The canvas is divided into two main fragments, main map and the toolbox.

4.1.2. Main map

The main map shows eighteen layers which are formed by the group element and controlled by the map tools with JavaScript functions. Each group is a collection of similar graphical objects created by the path, symbol, image and text elements of SVG. The groups are also combined and re-grouped as a “main group” in order to be browsed by the map tools as a single group, preventing that an object will be processed individually when the map is zoomed or moved.

Showing the whole document as the initial map would not be a good display because of the small and composite appearance of the map features in the original SVG document [Figure 4.6.a]. Therefore, a scale factor was defined as 1.5 in both dimensions. Also, the map was translated by (-100,-150), in x and y directions respectively, to centerize the main map. As a result, a new coordinate system was defined with an origin of (100,150) [Figures 4.5 and 4.6.b].

```
<g id="maingroup" onmousedown="if (click==false) clicked(evt); else click=false;"
onmousemove="move_object(evt)" onmouseup="click=false"
transform="translate(-100,-150) scale(1.5,1.5)">
```

Figure 4.5 Manipulation of the transform attribute

a. Original SVG document	b. Rescaled and translated SVG
	
transform="translate(0,0) scale(1,1)" origin(0,0)	transform="translate(-100,-150) scale(1.5,1.5)" origin(-100,-150)

Figure 4.6 Transformation of the map elements

4.1.3. Map Navigation Tools

Map browsing capabilities contain zoom in, zoom out, zoom to full view and move buttons. These buttons are displayed over the main map as another layer in the same SVG document. Zoom buttons run by controlling the transform attributes, scale and translate, while the move buttons only control the translate property of the main map group. There is also an info button in the toolbox that gives detailed information of an object when it is clicked on [Figure 4.7].

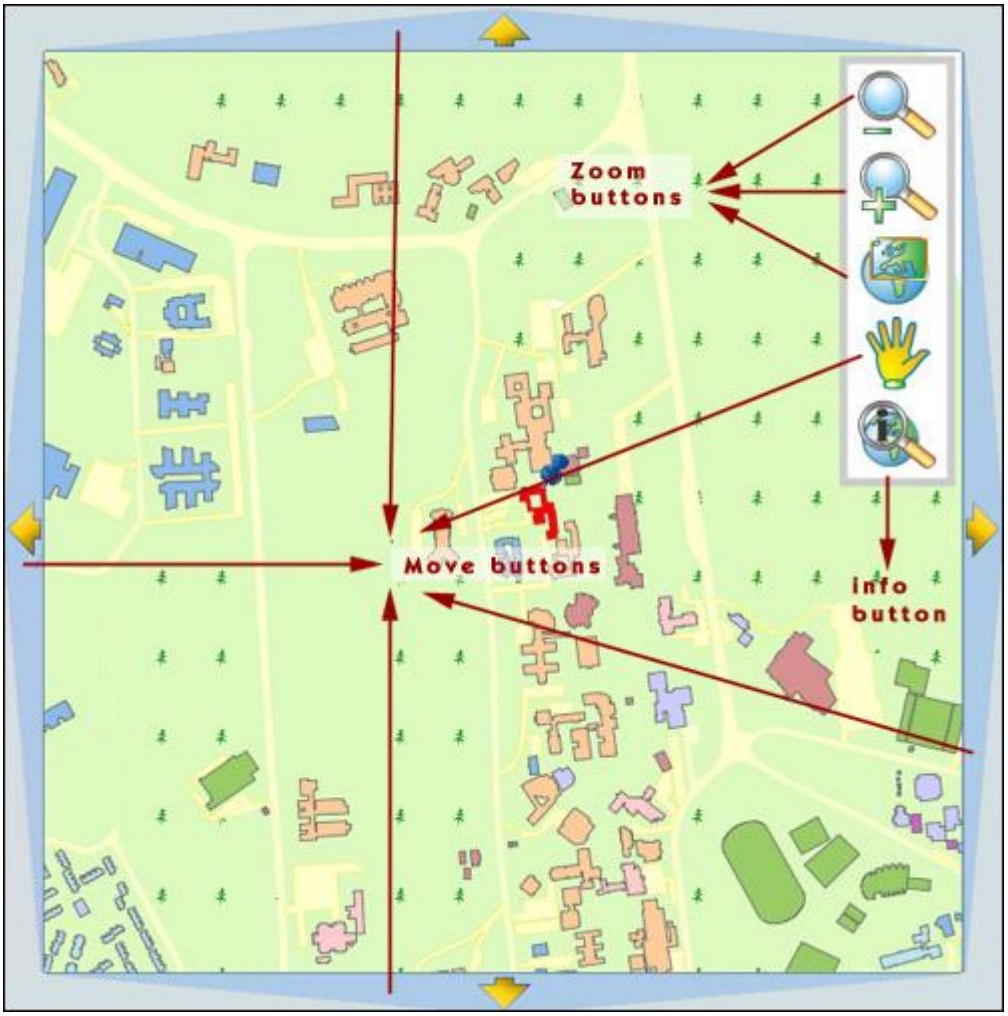


Figure 4.7 Main map with the toolbox

Zoom in changes the previous scale by multiplying with 1.25, which means that the map is enlarged by 25%. Reversely, zoom out rescales the map with a factor of 0.8, and reduces the size by 25%. After changing the dimensions, the origin is still matched to the original coordinate of the canvas, and the center of the map slides to right and down directions. In order to avoid that dislocation, the map content is also translated according to the formula in the related JavaScript function [Figure 4.8]. Zoom to full view restores the map by re-assigning the transform and scale values to their original ones.

```
function zoom(evt) {
---
//trans[0]=original x coordinate of the origin
//trans[1]=original y coordinate of the origin
//trans0=new x coordinate of the origin
//trans1= new y coordinate of the origin
//scale_val=scale factor
//650=>width of the viewport
//600=>height of the viewport

//Zoom in :
trans0=parseFloat(trans[0])-scale0*parseFloat((scale_val-1)*605/2);
trans1=parseFloat(trans[1])-scale1*parseFloat((scale_val-1)*610/2);
//Zoom out
trans0=1/scale_val*parseFloat(trans[0])+parseFloat(scale_val*(scale_val-1)*605/2);
trans1=1/scale_val*parseFloat(trans[1])+parseFloat(scale_val*(scale_val-1)*610/2);---
}
```

Figure 4.8 Zoom in and zoom out functions

There are two different pan options. The users can either scroll the map with down-clicking the mouse or click the arrows on the edges of the map to move in four different directions. The main idea is still the same, both move options change the origin of the present coordinate system. Dragging changes the origin according to the user's mouse movements while moving with clicking the arrows changes it with adding a pre-defined value, which is 75 pixels.

The identifier number of the object elements which are encapsulated with object defintions are extracted with JavaScript functions. When the info button is active and the mouse is clicked over an object, a pop-up window is opened and these identifier numbers are passed to these windows by PHP "GET" method. Only building, transportation points and parking lot layers have information windows, and their window content differs for each object type. In the new window, id number is linked to the database and all of the attributes of the object are gathered. The most detailed one is the building window, which shows the results by joining several tables, and also a picture of the building if it has one, after checking it from the related table, "building".

In addition to object ID's, names of the buildings are also retriewed from the database and defined by "name" attribute. Another JavaScript function is written for showing the building names at the top of the map, when the pointer of the mouse is moved over the objects. If only the name of the building is required, it will be shown instantly, and there will be no need to open an info window and wait for the results gathered from the database.

All those buttons are bitmap images and defined in <image> tags with their file paths, sizes, names and coordinates of their locations on the canvas. Each button is encapsulated with <a> linking tag and they are all linked to their main group, "toolbox" [Figure 4.9]. All of them have only display purposes, except "id" attribute of the <a> tag. It is the most critical one among them, because it is the one which identifies the active button to the JavaScript map navigation function. This function defines its events for map navigation according to the "id" that is passed from the "toolbox" group.

```
<g id="toolbox">

....

<a id="zoomout"  transform="translate(514,35)">
       <image name="Zoom Out" xlink:href="zoomout.gif" x="0" y="0"
      width="52" height="45"  />
</a>

....
</g>
```

id :	İdentifier
transform, x, y :	relative coordinates
width="52" height="45" :	size of the image
name="Zoom Out" :	label of the button describing its function
xlink:href="zoomout.gif" :	file path of the image

Figure 4.9 Toolbox

4.1.4. Layer Control

There are eighteen layers in the application, and they are controlled from the main page with checkboxes and a JavaScript function [Figure 4.10]. All layers have a visibility attribute in their container group tags, which can be “visible” or “hidden”. The function changes the visibility attributes according to the conditions of their checkboxes. When the box is checked or unchecked, the visibility attribute changes as visible or invisible, and the layer is displayed or removed respectively.

Figure 4.10 Layer Control

Most of the layers are checked in the default document, and some of the layers are locked, so they cannot be made invisible by the users. However, the locked layers must be listed among the others since legend is combined with layer control to save space on the web page.

4.2. Serving SVG files with PHP

While scripting with PHP, SVG elements are dynamically created from the data stored in the tables. As mentioned before, “d” attributes of path elements are the instructions of drawing the graphical shapes. These attributes are stored in path tables in the database to be gathered and drawn by the outputs of some PHP scripts. These scripts extract path and attribute data from the database, draw graphical objects, and create layers by containing them in group element. Each group is created with a specified code segment, and has a different style, which is linked to CSS definitions [Figure 3.4] within the same document. Building layer is nested by creating sublayers according to their categories. In addition to path elements, identifier numbers and names of the buildings are also retrieved from the attribute tables for these objects.

A short PHP code can draw an entire layer by retrieving the paths from the database. The output of the script in Figure 4.11 creates a layer for all the academical buildings in a code of more than 400 lines and 50000 characters.

```
<g id="academic" onmouseover="onmouseover(evt)"
onmouseout="onmouseout(evt)">
<?
$sql="SELECT * FROM building_path,building_info
      WHERE category='Academic'
      AND building_info.bid=building_path.bid";
$result=@pg_query($baglanti,$sql);
while($row=pg_fetch_array($result)) {
      echo "<path class=\"academic\" d=\"";
      echo $row["path"];
      echo "\" name=\"";
      echo $row["name"];
      echo "\" id=\"";
      echo $row["bid"];
      echo  "\" />";
}
?>
</g>
```

academic : Name of the layer
Academic : Usage of the building defined in the database
academic : Style class of the academic buildings

Figure 4.11 A whole layer which shows all the academic buildings in the campus

Figure 4.12 shows PHP search string of Department of Civil Engineering K3 Building, its result in SVG object definition, and vectorial image output that is displayed on the SVG viewer. Attribute “d” draws the outline of the polygon, “fill” defines the filling color of the shape, “id” is the identity number of the object, and “name” gives the name of the building which is displayed at the top of the map when the mouse pointer comes over the object. Searched object is zoomed and centered on the SVG viewport after being created by manipulating the transform attribute of the main content. Additional attributes can be listed on the pop-up window opened by mouse clicking on the object. In this window, PHP scripts request the attribute data of the object by sending its identity number to the database.

PHP Script :

```
$sql="SELECT * FROM building_path,building_info
      WHERE building_path.bid=12
      AND building_info.bid=building_path.bid";
$result=@pg_query($baglanti, $sql);
while($row=pg_fetch_array($result)) {
echo "<path fill=\"red\" d=\"";
echo $row["path"];
echo "\" name=\"";
echo $row["name"];
echo "\" id=\"";
echo $row["bid"];
echo  "\" />";
}
```

SVG object definition :

```
<path fill="red"
d="M315.96025,480.27077L317.68839,479.98275L316.5363,471.05418
L321.14465,470.47814L323.01679,484.87906L318.40844,485.4551
L317.97641,482.14289L316.24828,482.4309L316.24828,482.86293
L313.36805,483.15095L312.93602,480.27077
L315.96025,479.83874L315.96025,480.27077z"
name="Department of Civil Engineering K3 Block"
id="12" />
```

Result :

Figure 4.12 Search results of a single object

4.3. Map Querying

Classrooms, names of the buildings, facilities, and all other units or rooms including laboratories, cafeterias or administrative offices can be searched by the text box on the main page. The tables “building”, “building_units”, “building_facilities”, “types” and “classroom” are joined with the search query. Also, there is a list box which lists the name of the buildings according to their usage types.

AJAX functions and PHP work together to gather search results in a synchronized way. When entering a letter or a word in the text box, a result list box becomes visible and lists the results containing those letters. Every change in the text box modifies the results simultaneously without any connection with the server and the database, and as a result, there will be no time delays [Figure 4.13].

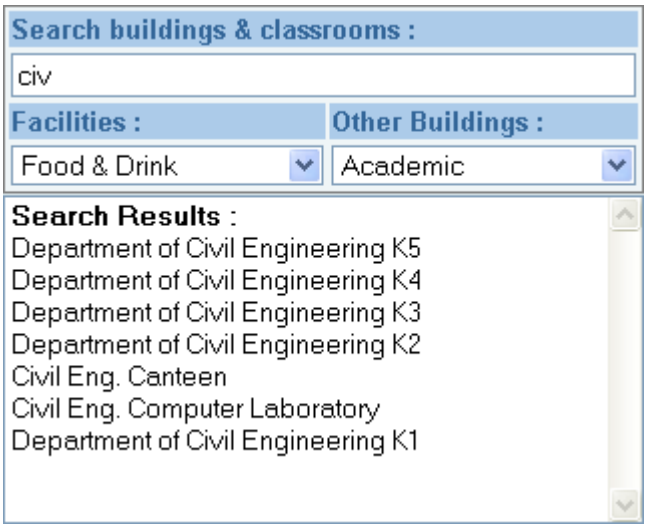


Figure 4.13 Searching with a phrase

Buildings and facilities can also be listed according to their usage types by the same way. Altering the selected option changes the results instantly [Figure 4.14].

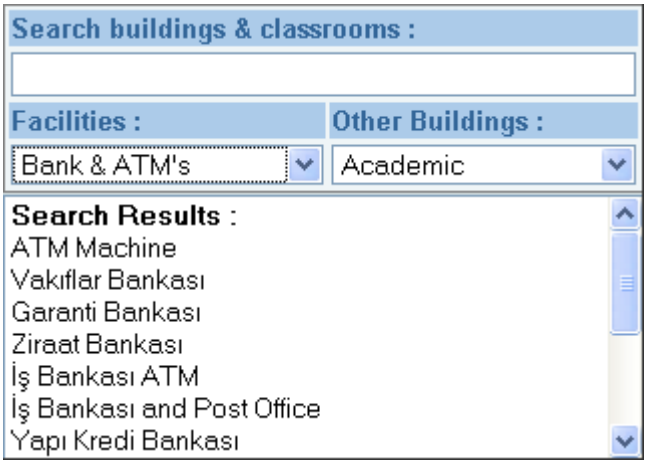


Figure 4.14 Listing by usage types

Object identity numbers are also carried in the value attributes of the option tags. When a result on the list box is clicked on, id number is passed from the embed tag to “embed.php”. In the embed tag, coordinates of the starting point of the polygons are queried from the path table. Finally, identity number, and (x,y) coordinates are sent to the SVG document by the GET method of PHP. That is the only moment that the map is reloaded. Now the map is rescaled by multiplying with 3, all the layers are downloaded again, and the searched building is centered and painted with red color in order to be recognized by the user.

4.4. Summary of the System Design

Figure 4.15 summarizes the implementation of the study in a flowchart. At first, data were collected from different sources in different formats. Spatial data were edited and digitized by GIS tools, converted from shapefile to SVG format, and transfered to the database. Attribute data were also organized and stored in data tables separately. Both spatial and attribute data were delivered from the database by PHP scripts, which also draw the vector graphics to create the map, and provide search capabilities. JavaScript functions were used to form the map navigation tools, and user interface was prepared with graphic and web page editing softwares. After all of these modules were put together, the web document was finally published on the Internet.

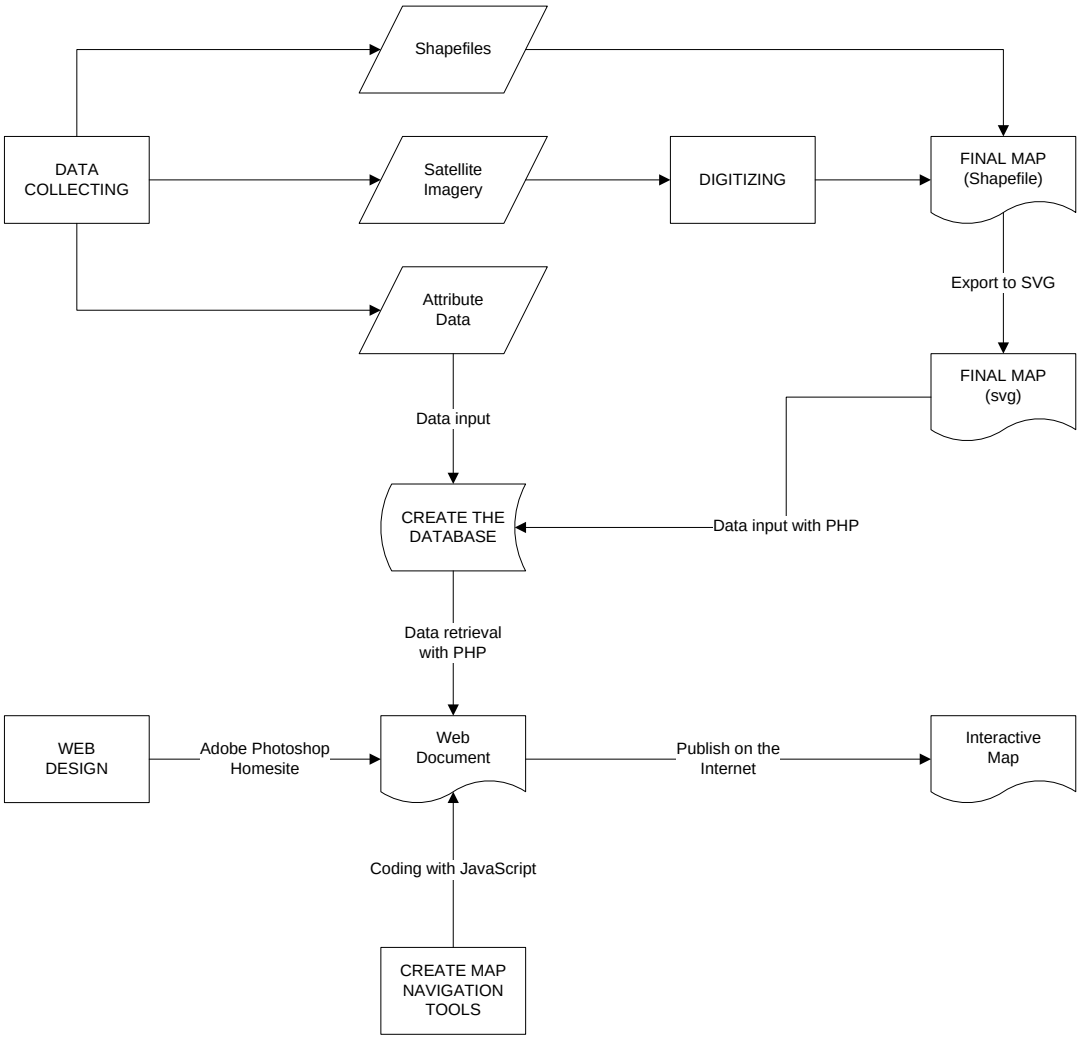


Figure 4.15 Flowchart of the implementation

CHAPTER 5

RESULTS

5.1. User Interface

The map is avaliable at www.gis.metu.edu.tr in both English and Turkish languages, and hosted by Linux servers with Apache, PHP 5.0 and PostgreSQL database server located in METU Computer Center [Figure 5.1]. A web browser with an SVG plug-in is sufficient for viewing the map from the Internet. Recently, most of the web browsers have an SVG support in their latest versions, and a plug-in is not needed to be installed.

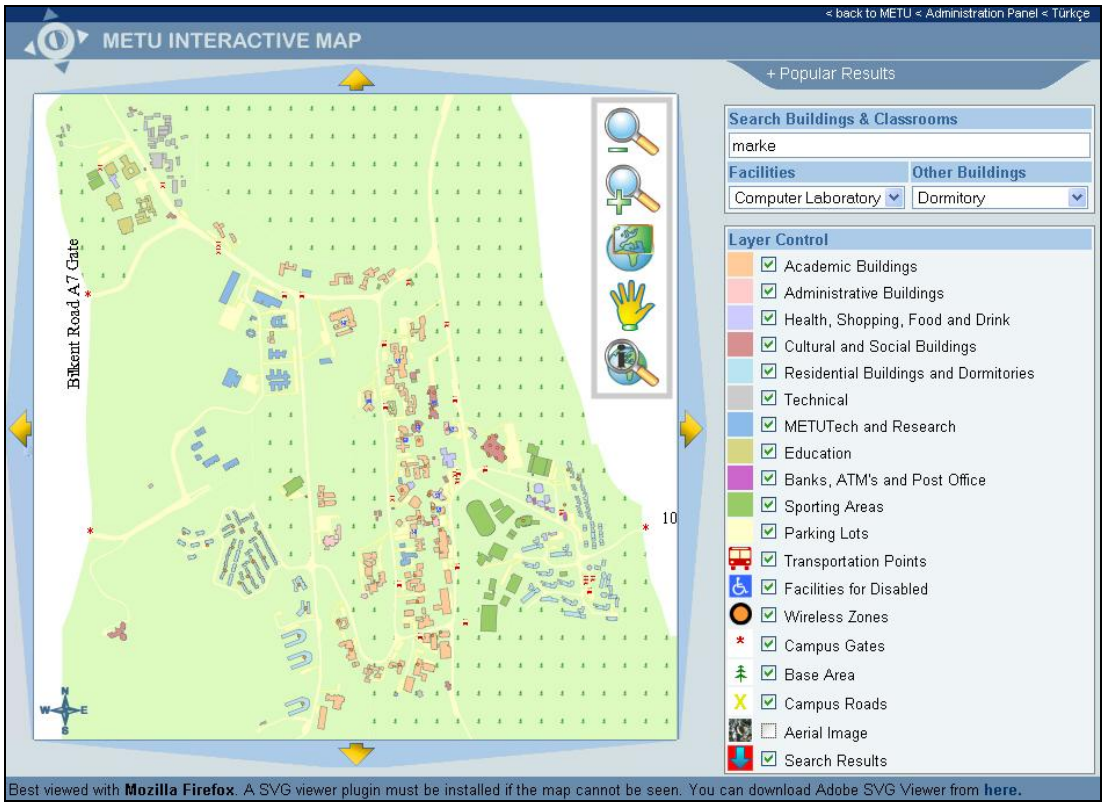


Figure 5.1 User interface of the application

As explained in the previous chapters, the application has four sections; main map, map navigation tools, layer control and search section.

The main map is a vector image in SVG format, and controlled by the map tools which come within the same SVG document [Figure 5.2]. Once the main page is accessed, all spatial data, identity and name attributes of the buildings are downloaded to the user's web browser with a size of about 800 KB in a few seconds, depending on the network bandwidth. While navigating the map with the map tools, no time delays and round trips to the web server occur, since all these functions operate locally.

Figure 5.2 The map magnified with zoom level of 8

On the top side of the map, there is a hidden text area, which becomes visible when the pointer of the mouse is moved over an object. This area guides the users by showing the functions of the map tools, names of the buildings, or usage types of the other map features, and disappears back if the pointer is moved out [Figure 5.2].

Zoom level is set up to two levels for zooming out, but limitless for zooming in. Therefore, the map can be enlarged many times to show any detail that user wants to see. No matter how large the map is, image quality does not change, since the graphics are in vector format [Figure 5.2]. Zoom to full view button resets the map to its original extent.

Info button opens a pop-up window with information about the object, on which the mouse is clicked. There are three different pop-up windows for three different feature types; transportation points, buildings, and parking spaces.

Transportation points window shows the types of the transportation vehicles, and gives a detailed description about the destinations and schedules of them [Figure 5.3].

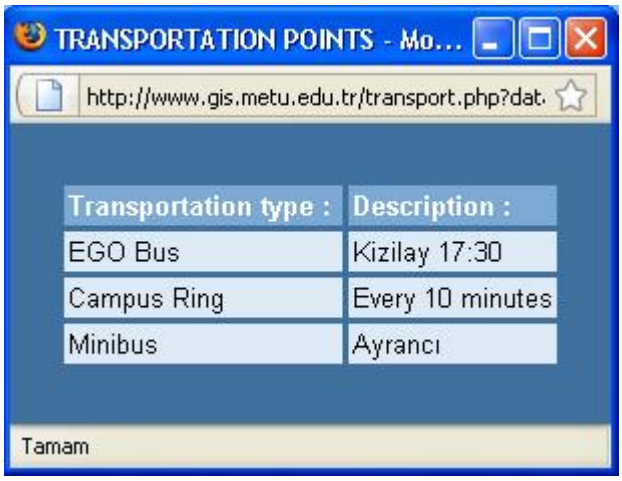


Figure 5.3 Information window for transportation points

Building window lists the building name, usage of the building, name of the units, facilities for disabled people (lift, ramp, handicapped restroom and parking space), and all other entities that the building contains, such as laboratories, rooms, classrooms, eating places, and so on [Figure 5.4]. It also shows a picture of the building.

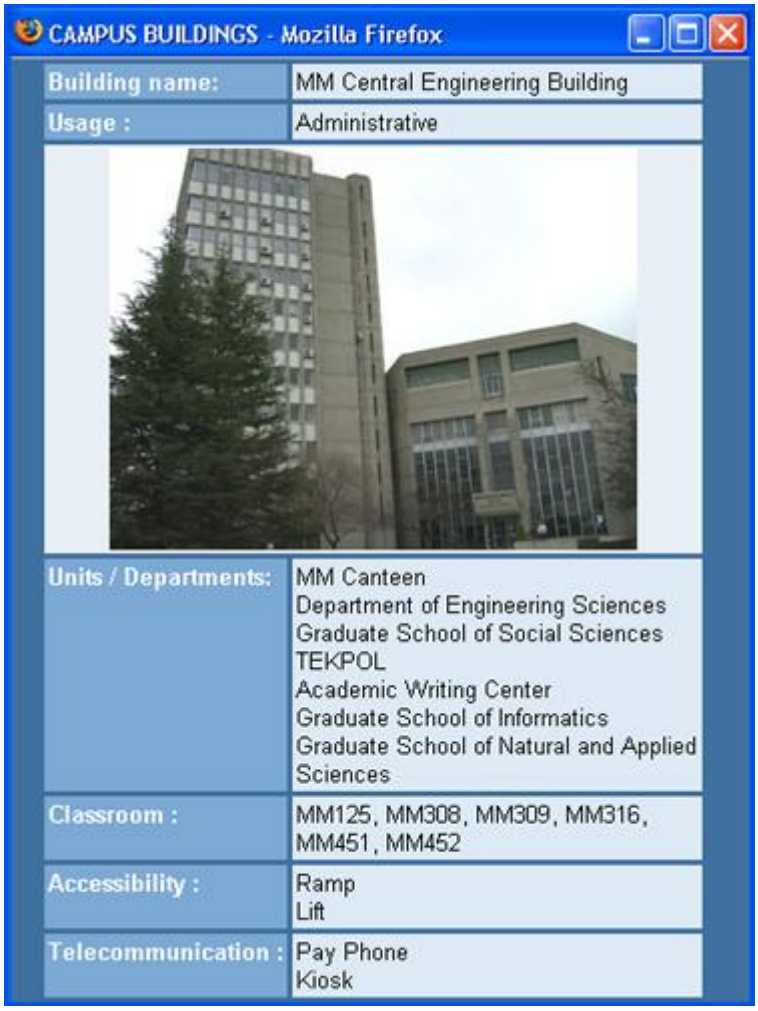


Figure 5.4 Info window for buildings

Lastly, the parking space window gives information about the sticker colors and user types that can park there, the name of the nearest building, and whether it contains a disabled parking space or not [Figure 5.5].

Figure 5.5 Info window for parking lots

Move and info buttons change each other's activity status when they are clicked on. As default, only info button is active. When the move button is clicked on, info button becomes inactive, and vice versa. Otherwise, pop-up windows will be opened when dragging the map. Active button is distinguished with its colorless new appearance [Figure 5.6].

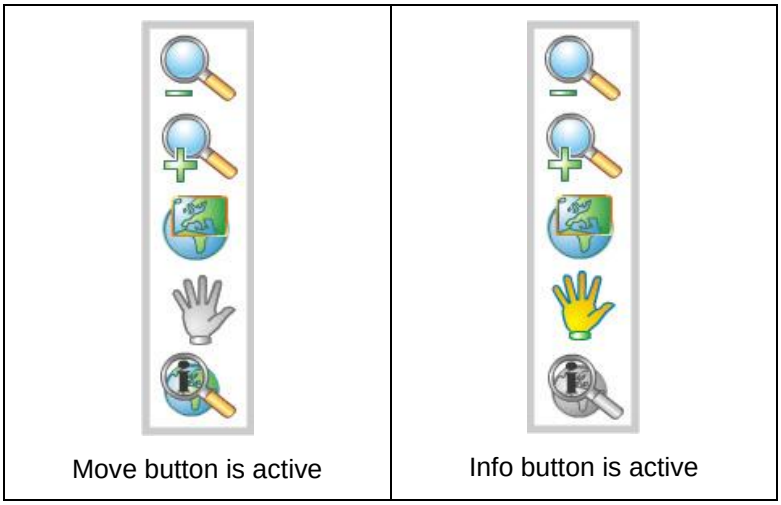


Figure 5.6 Activation of the buttons

As mentioned in Chapter 4, the map can also be moved by clicking on the arrows on the four edges of the map [Figure 4.7]. Unlike the move tool, info button is not deactivated when these arrows are used to move the map.

Results of both search options come in the same list box [Figures 4.13 and 4.14]. As an example, a user wants to find "Civil Engineering Building K1" [Figure 5.7]. It is not needed to write the full name of the building. The letters written in the search box changes the results in the listbox simultaneously, and phrase "civ" is enough to achieve the name of the building. Then the user has to click on that name to view the location of that building on the map.

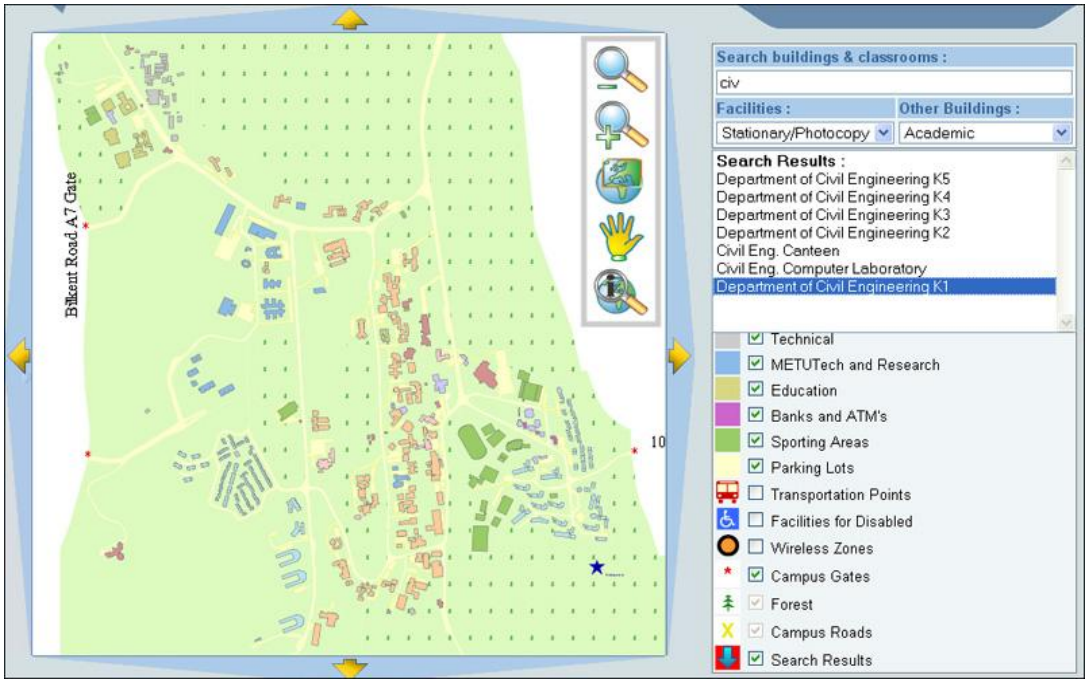


Figure 5.7 Results of the phrase "civ"

After clicking on that result, the map is magnified, graphic of the building is pointed by an arrow, highligted with a different color, and centered on the map to display the result more remarkably [Figure 5.8].

Figure 5.8 Location of the building "Civil Engineering K1"

There are eighteen layers, eleven of which are the building sublayers categorized as their usage types [Figure 4.10]. Once the map is loaded, some of the layers are invisible in order to avoid a complicated view. These layers are transportation points, facilities for disabled, and wireless zones. Layer control is a local process as well. If further layers are desired to be seen, they can be switched on from the layer control section, or switched-off by unchecking again, without dealing with the web server. Forest and campus roads layers are locked, so they cannot be turned off. Search results are also displayed on a separate layer, and can be removed.

5.2. Editing Interfaces

The map can be edited by some authorized users from various departments and units of METU, logging in the system by METU user accounts, and each assigned for different features. Editing interfaces are opened in new pop-up windows for buildings, transportation points and parking spaces, having the same content of the information windows, and they have additional elements such as text boxes and buttons to change the content.

Administrator of the parking lots can only update the types of the users, which can be administrative and academic staff, student, or handicapped. The other users can only use free parking lots [Figure 5.9].

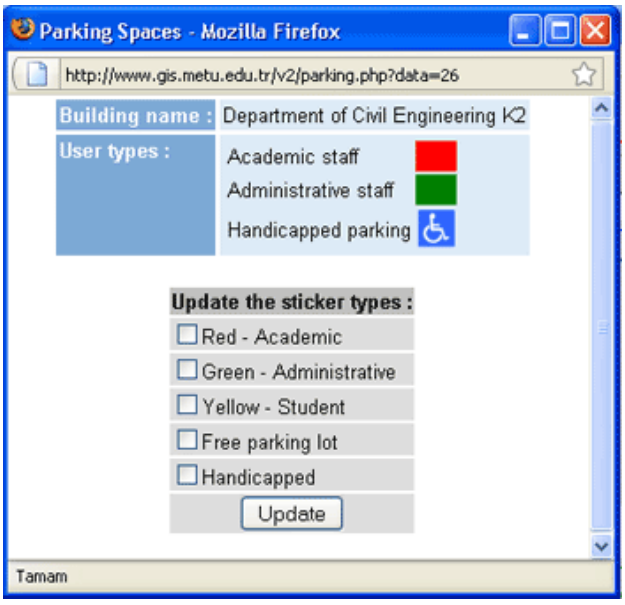


Figure 5.9 Parking space editing interface

Editing interface of the buildings contains many entry fields. Figure 5.10 shows data update of Faculty of Economic and Administrative Sciences Building B. Its name, description, usage, classrooms and disabled facilities can easily be updated from this interface. Also, entities and departments can be deleted or new ones can be added with their usage types.

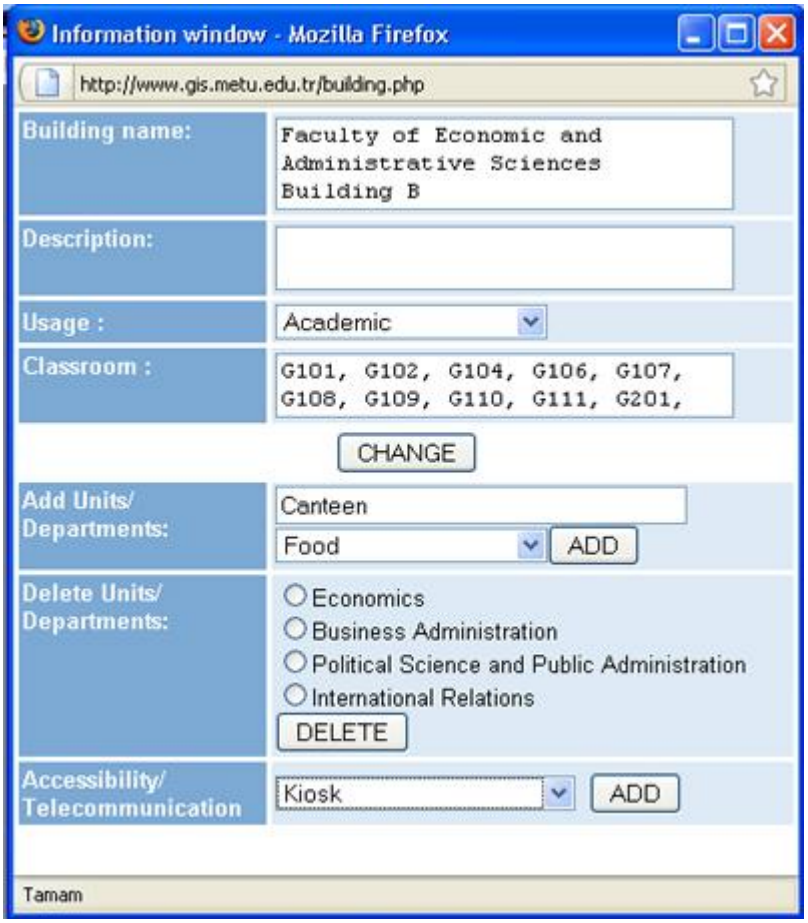


Figure 5.10 Buildings editing interface

Types of the vehicles used for transportation in METU campus, their schedules and destinations can be edited and deleted by transportation points editing interface [Figure 5.11].

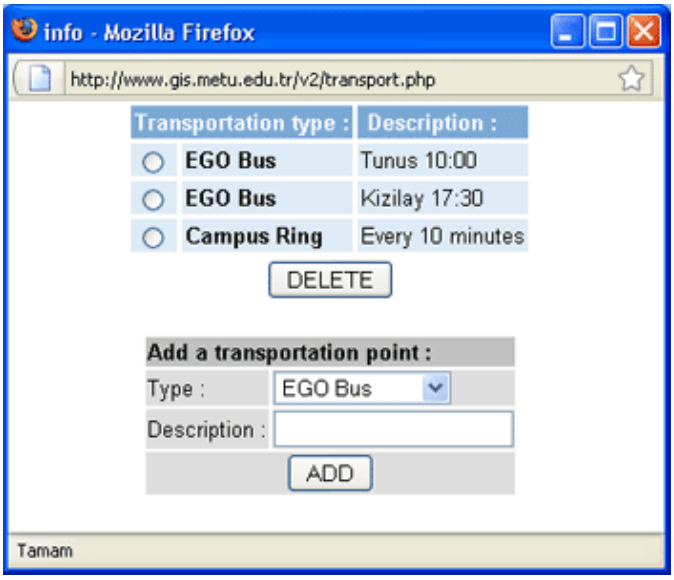


Figure 5.11 Transportation points editing interface

5.3. Usability Study

The purpose of usability testing is to determine the satisfaction and efficiency levels of the application to improve its usability.

Satisfaction is related with the impressions of the users about the interface, and can be quantified by applying questionnaires. Efficiency is a performance measure, and computed with the time to complete a task [Duchowski, 2007].

It is aimed to test performance of the map and search tools, diagnose the usability problems on the interface, and figure out :

- Which tools are used more likely?
- Is it difficult to use them?
- Are they located in right places?
- Do the users need guidance and usage tips to use them?
- Is there any progress for using the map tools during the test?

5.3.1 User Questionnaires

After the application was implemented, a questionnaire form [Appendix B] about the user interface was published on the Internet, and its URL address was sent to the members of some GIS related mailing lists. The members were requested to examine and test the web page, fill the form, and also add their comments and suggestions.

In the form, Likert scale was used for rating the answers. Percentage of agreement was calculated by summing strongly agree and agree percentages. The questions, format of the scale, results in frequencies, and percentages of agreement can be viewed from Figure 5.12.

According to the results, map graphics, user interface and map content did not satisfy the participants entirely. On the other hand, map tools were easily understood and used.

The users were also requested to write down some comments. Some of the incoming feedbacks are :

- The colors should be sharper instead of pastel ones.
- There should be a “check/uncheck all” option on the layer control.
- The buildings should be hidden as default, and added by the users.
- Zoom functions should also be controlled by the mouse scroll.
- Raster image can be used as the base map.

After the evaluations, some changes were applied to the map according to the suggestions of the users.

Questions	Strongly disagree	Disagree (%)	Neither ag. Nor disag.	Agree	Strongly Agree	% agree
METU needs an application like this.	2	1	2	10	2	**90,20**
Map content is sufficient	4	6	21	11	4	**40,38**
I didn't find it difficult to learn how the map works	1	3	3	15	1	**86,79**
I didn't find it difficult to **understand the function** of this button :						
	1	1	4	49	1	**96,36**
	1	1	4	49	1	**96,36**
	2	4	9	9	2	**72,73**
	2	1	3	8	2	**89,09**
	2	2	1	6	2	**90,91**
	1	1	2	50	1	**96,30**
I didn't find it difficult to **use** this button :						
	2	3	3	8	2	**84,91**
	1	4	3	8	1	**84,91**
	1	1	11	5	1	**75,47**
	2	2	7	6	2	**78,85**
	1	3	2	6	1	**88,24**
	1	1	4	8	1	**88,24**
I liked the map graphics	4	5	15	15	4	**55,56**
I liked the user interface	4	5	13	18	4	**59,26**
I didn't have difficulty to use the search options	2	5	4	15	2	**79,63**

Figure 5.12 User Questionnaire with the percentage agreement results

5.3.2. Eye Tracking Methodology

The usability of the web interface was tested by eye tracking methodology, which is the study of measuring and evaluation of the eye movements to examine the visual

attention of the people. According to Montello (2002), tracking the eye movements of the users is one of the most emprical approaches to map psychology. Eye tracker is the main instrumental device of these studies and capable of recording eye movements and positions.

The tests were performed in Human Computer Interaction laboratory of METU Computer Center [Figure 5.13]. The laboratory is divided into two parts with a one-way mirror. One of them is the test room having microphones, amplifiers, cameras, and a computer with an eye tracker device and screen recording capabilities. The other one is the operator room, where the participants and screen of the test computer are observed. The test room is soundproof and the participants cannot see inside the operator room from the mirror.

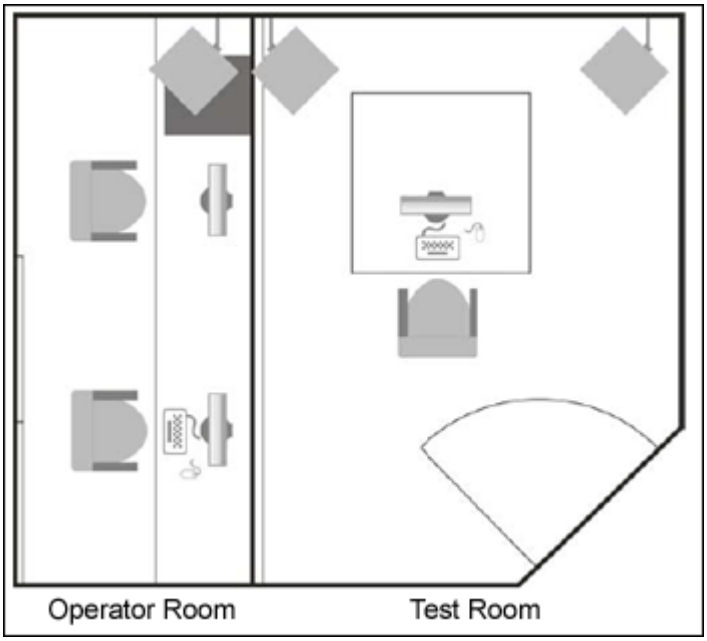


Figure 5.13 HCI Laboratory of METU Computer Center [ODTÜ, 2009]

Fixation, which is moment that the eyes are focused and stabilized on a particular area or point, is the main metric of eye tracking studies. Visual attention can be quantified by the numbers of fixations within a region arbitrarily defined on the interface [Duchowski, 2007], which are called Aeras of Interest (AOI).

There are additional metrics, such as observation and fixation length, time to first fixation, mouse click count. Observation length is the duration that a user looks at an AOI, starting from a fixation within the AOI, and ended with another fixation outside the AOI, while fixation length is the mean duration of all fixations within an AOI [Tobii, 2008]. High frequencies and long durations of fixations expose the difficulty of the searching tasks [Rayner, 1998].

The strategy for usability testing of the study is to define AOI's on the user interface, assign some tasks corresponding to objects within these AOI's, recruit a group of participants with a minimum number of 5 [Nielsen, 2000] [Figure 5.14] to accomplish the tasks, and analyze the eye movements of the evaluators that fall in those areas after the testing process.

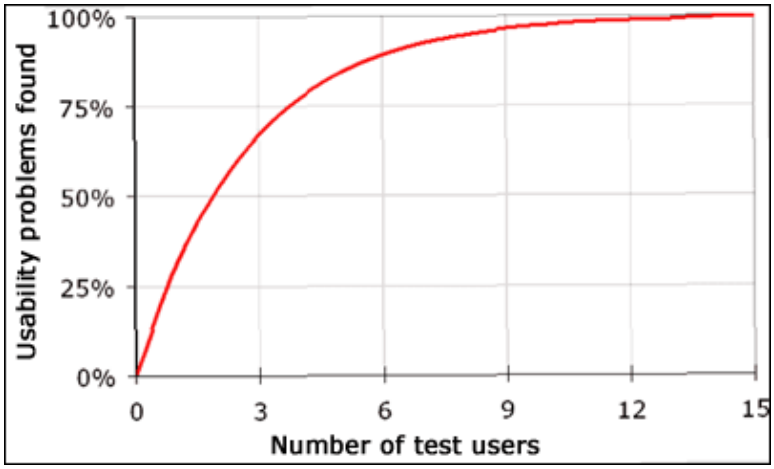


Figure 5.14 Adequate number of evaluators for usability testing [Nielsen, 2000]

12 participants were recruited for the test, which were composed of students, graduates, administrative and academic staff of METU, and had never seen the tested web page before. Four of them were students of Geodetic and Geographical Information Technologies (GGIT), and there were four female users. They were asked to complete eight tasks, and also rate each one by ease of use by using 5 point Likert scale :

1. There is a dark blue star on the right bottom side of the map. Try to read the text near the star.
2. Find the classroom named as TDB-2.

3. View the nearest wireless netwok points to the shopping center.
4. Where can you park your car near the Department of Metallurgical and Materials Engineering building with a student (yellow) sticker?
5. What else is there in the building of the dry cleaner?
6. Which vehicles can you embark in front of the Department of Biological Sciences?
7. View the names of the classrooms of Department of Food Engineering.
8. List the eating places in the campus.

The question of the Likert scale was “What is the difficulty level of this task?”, and the format was 1: very difficult, 2: difficult, 3: normal, 4: easy, and 5: very easy. After the test, they also filled the same questionnaire form mentioned in the previous section.

While trying to accomplish the tasks, eye movements of the participants were recorded by a 50 Hz non-intrusive eye tracker. Task completion rates and usage of the map tools were also captured by the computer.

5.3.3. Analysis of the Test Results

Eye tracking metrics were calculated by the eye tracker software, which is also the test design, evaluation and visualization tool of the eye tracking device. Each session can be replayed, saved as video or image files, and results of the tests can be statistically analyzed and exported to Excel, SPSS and Matlab.

There are several strategies to accomplish each task. In order to evaluate which tools are easier and used more often, seven AOI’s are defined on the user interface [Figure 5.15] :

- map : Main map
- toolbox : Toolbox (zoom in, zoom out, full view, get information, move)
- nav : Navigation arrows (go to east, west, north and south)
- s_building : Search by phrase
- s_type : List by type
- layer_control : Layer control
- all : Entire user interface

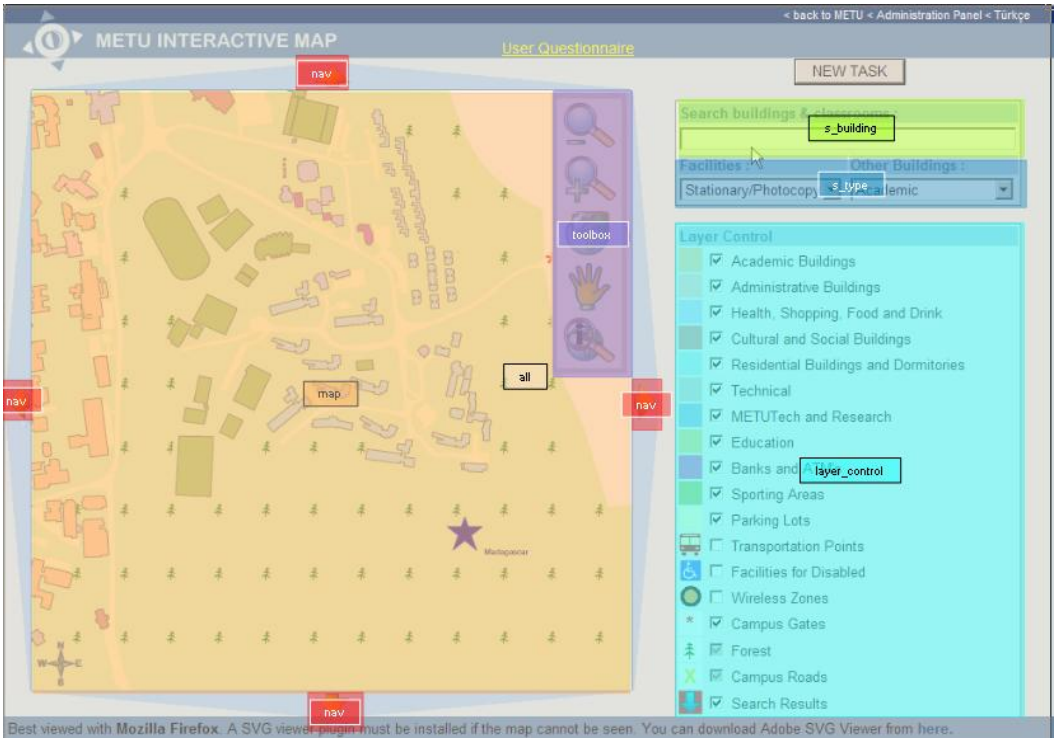


Figure 5.15 Areas of Interest

Observation length of the users in seconds and task completion success rate for each task are listed in Table 5.1. Eye movements of some participants could not be recorded during the task due to a calibration error of the device. Data of these tasks were not used for the statistical analysis even if they were completed successfully, but included in the user rankings and percentage of success values. The uncompleted task sessions and outliers were also excluded from the analysis data.

In order to determine the outliers, the suspected values were calculated with the mean and standard deviation of the individual task data. If the value differed from the mean by bigger than 2.5 times of the standard deviation, it was discarded.

There is a peak on the completion time of Task 4, which was ranked as the most difficult one [Figure 5.16]. The most possible solution of the task is to find the building with one of the search options first, activate the information button on the toolbox, then click on the parking lots near them. The building was found easily, but the users did not expect that the parking lots could have an information window. Furthermore, the arrow that shows the results is so big that it overlays a small region of the target. However, in the next tasks, they recognized the information window of

the buildings and transportation points easily. After Task 4, completion rates of the remaining tasks decrease rapidly, and user ranks of ease increase, which shows that the users learned how to use the map after a challenging task.

Table 5.1 Task rates (s), percentage of success, and mean user ranking

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
User 1	124,16	46,88	83,92	Fail (1)	59,47	41,40	28,92	25,06
User 2	49,60	15,31	47,46	172,50	16,40	51,99	29,83	140,02
User 3	58,03	92,89	67,17	Fail (1)	45,13	46,13	17,08	36,74
User 4	95,71	386,98	81,20	490,01	148,84	51,00	57,06	20,26
User 5	71,80	19,41	50,89	158,92	50,62	39,21	20,91	7,49
User 6	67,00	64,63	41,67	Fail (1)	44,24	90,44	33,06	Err (2)
User 7	73,25	38,27	37,76	185,54	43,85	53,99	26,77	17,57
User 8	51,71	49,09	67,02	161,75	163,31	81,45	24,68	15,59
User 9	49,50	33,56	139,74	193,13	34,77	57,82	54,91	54,61
User 10	42,42	37,04	22,69	95,94	35,33	Fail (1)	28,00	34,26
User 11	68,08	35,52	53,32	142,59	29,43	33,96	38,35	44,30
User 12	69,81	51,41	177,38	Fail (1)	212,48	55,19	32,11	14,85
Mean	63,35	44,00	62,98	158,62	61,03	54,78	32,64	27,07
S.D.	15,09	21,43	31,45	32,41	48,43	17,16	12,23	14,93
Success(%)	100	100	100	66,67	100	91,67	100	100
Rank of ease	4,25	4,18	4,50	2,33	3,83	4,50	4,50	4,67

(1) : Uncompleted tasks
(2) : Unrecorded task due to a technical error
6 values (6,25 %) were outlied

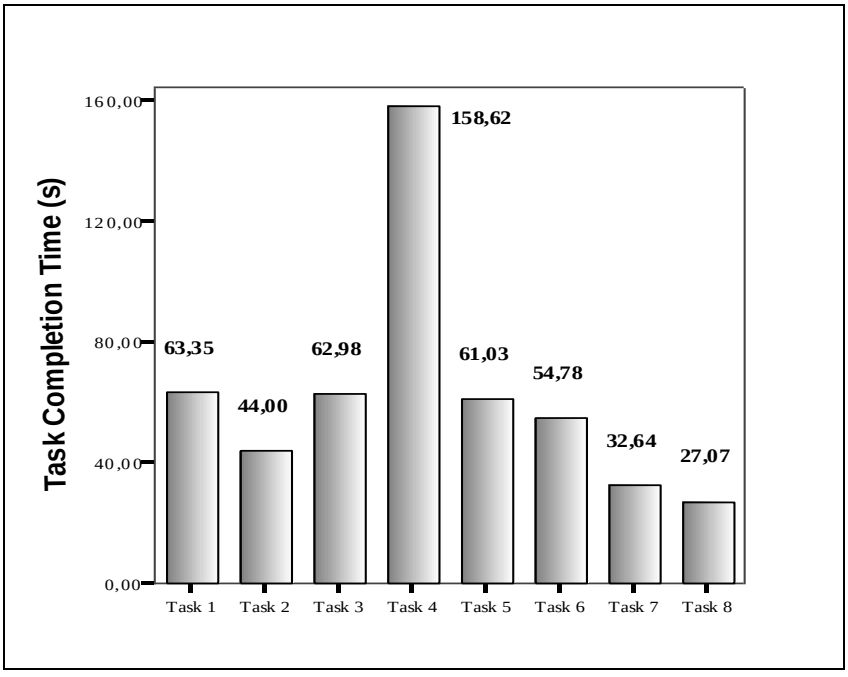


Figure 5.16 Mean task completion times in seconds

Task completion rates of GGIT students, who mostly have more experience with maps, were compared with the other users. The results of female and male participants were also compared. However, no significant difference was recognized between all the user sets after applying 2-tailed t-test for equality of means at 95% of confidence interval [Table 5.2].

Table 5.2 Task completion durations of independent variables

Variable	Mean	Standard Deviation	N	p
Male	59,37	45,33	57	0,802
Female	56,90	33,40	27	
GGIT student	64,88	45,14	29	0,317
Other Users	55,25	39,77	55	

Figure 5.17 shows avarage fixation durations of each AOI according to the tasks. Long fixation times expose the difficulty of the focused item. Decreasing number of the durations shows that the users had less difficulty to use the tools during the last tasks.

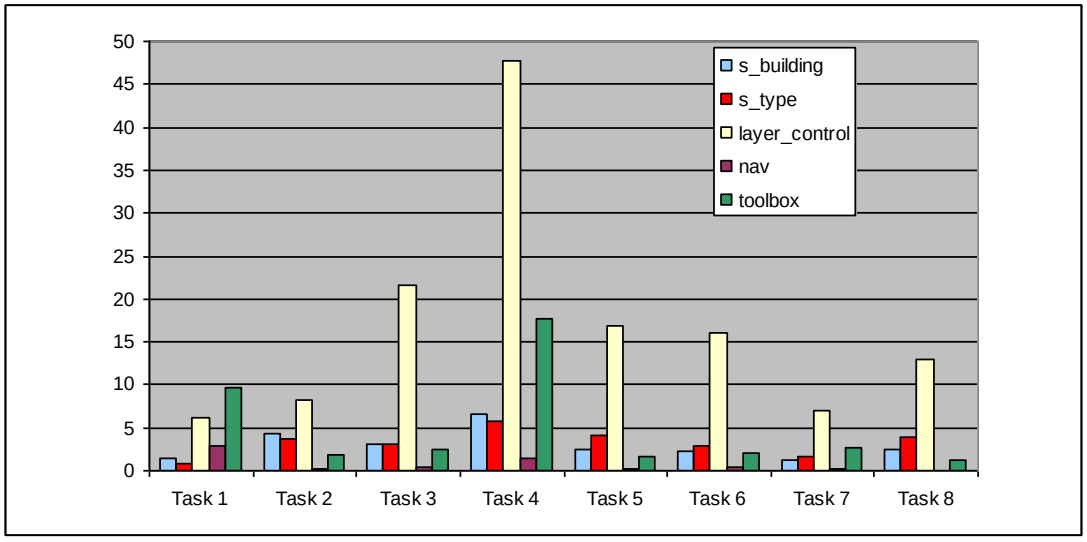


Figure 5.17 Mean fixation durations of AOI's in seconds

Except the first one, all tasks can be accomplished by common strategies. AOI's of s_building, s_type, layer_control, toolbox and nav are the main tools for finding the

target locations. In Figure 5.18, the mean mouse click counts of these AOI's are displayed in a pie chart.

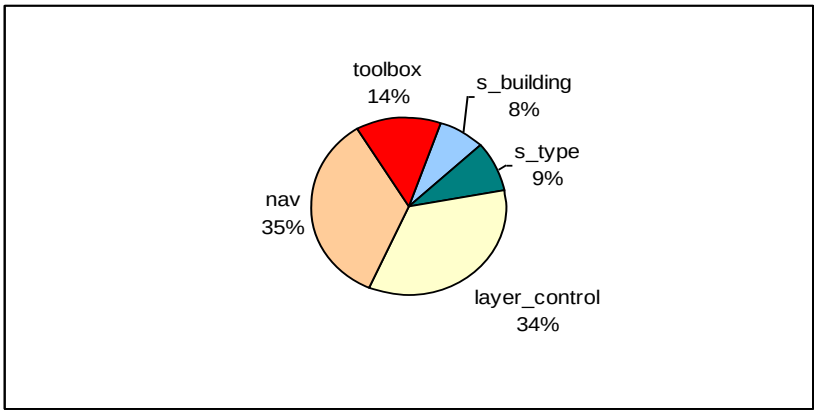


Figure 5.18 Percentage mouse click counts of the map tools

The AOI's s_building and s_type have similar characteristics, but it is more reasonable to use s_type as a category filter, and s_building for known phrases, for instance, building names or classroom codes. On the other hand, layer control is not an efficient tool for filtering the text results, its function is to filter map graphics only. However, looking at the mouse click counts of the map tools, layer control was the most commonly used one among the map tools, although the search boxes would inevitably used in almost all the tasks. In fact, the map offers a large set of data available for searching by using these elements. The reason of this situation can be understood from Tables 5.3 and 5.4, which display mean time values to first fixation and mean number of fixations, respectively. Low first fixation times and high fixation numbers indicate high attraction, and search boxes have higher time to first fixation values and lower fixation numbers than the others. Since they were recognized late and slightly, they were used seldom comparing with the others.

Table 5.3 Time to first fixation values (s) of each AOI

	T 1	T 2	T 3	T 4	T 5	T 6	T 7	T 8	**Mean**
s_building	28,97	20,18	39,78	36,19	12,77	12,18	13,85	11,78	**21,96**
s_type	25,24	14,28	19,72	26,53	14,44	16,14	8,81	8,70	**16,73**
layer_control	1,65	7,85	2,88	7,41	10,06	5,96	6,27	7,20	**6,16**
nav	28,85	38,61	49,61	102,53	25,71	26,62	30,87	.	43,26
toolbox	8,48	8,99	13,43	25,40	15,16	37,51	14,56	10,54	16,76

Table 5.4 Fixation counts of each AOI

	T 1	T 2	T 3	T 4	T 5	T 6	T 7	T 8	**Mean**
s_building	4,92	8,58	5,83	16,75	5,92	4,64	3,75	6,09	**7,06**
s_type	2,83	9,42	7,33	15,88	10,42	7,09	3,83	10,91	**8,46**
layer_control	19,25	19,17	41,83	105,88	32,25	39,82	15,92	30,00	**38,01**
nav	4,08	0,83	0,83	3,13	0,75	0,64	0,50	0,00	1,35
toolbox	21,08	5,67	6,83	45,75	6,08	6,36	6,17	3,91	12,73

After the evaluation of the tests, it was estimated that some minor changes on the user interface would increase the effectiveness of the application. A short tutorial with some usage tips was prepared to inform the users about the content of the map and capabilities of the tools. Search areas were made more remarkable with an attractive color.

For the best results, usability tests must be repeated, if in future the interface is changed in an extreme level.

5.4. Conclusion

Scalable Vector Graphics (SVG) offers an alternative way for displaying spatial data on the world wide web. It is an open standard, does not need a map server, and compatible to run in many servers and database systems.

When the map is accessed from the Internet, all graphical data are downloaded to the client side in advance. In addition to the spatial data, limited size of non-spatial data are delivered from the database in order to reduce the start up time. These are the names of the buildings and identifiers of the entities. Unless a query is made with the search section, there will be no interaction with the web server. Therefore, all map browsing operations occur locally without creating a network load. This is the most significant advantage of using an SVG based system instead of a map server, which usually responds to all kinds of demands coming from the client side. The map images are re-rendered by the map server even if the map is zoomed, moved, or the layers are switched.

Except the desktop side, which requires some GIS softwares to create the geographical data, cost of establishing such a system is very low. First of all, no additional software, database systems, and application servers are needed. Since

SVG is an XML derivation, its applications can be performed and stored in many systems, integrated with other web standards, and coded by various programming and scripting languages such as PHP, Java, and ASP. For that reason, it is relatively simple to set up and maintain to use SVG for GIS projects. In this study, the implementation was developed by using entirely open source products and standards, without any modification to the server side.

Source code of the client-side can easily be obtained from the “View Source” tool of the web browser that displays the map. This code includes all map features in SVG format, with their identifier and name attributes, style sheets, and JavaScript functions that control the map browsing operations. The web page can also be downloaded to a local computer directly. Therefore, SVG data of METU map, and all map navigation scripts developed by JavaScript are shared and can be used by everyone.

Raster images have large file sizes and they lack interactivity. Also, enlarging them results in degradation, or a significant increase in file size if the quality is wanted to be preserved. However, SVG promises dynamic and more sophisticated graphics which can be rescaled to unlimited values in low file sizes, without losing their display qualities, because of having vector image format. Additionally, they can be labeled, animated, and linked to other objects. Figure 5.19 compares double sized view of raster and vector format of the same image. Vector image keeps its sharpness while the raster one have pixel deterioration.

Shape properties of the SVG objects can easily be customized by changing their class definitions in the style sheet. Every class is assigned to a layer, and any change of a class definition directly affects the apperance of a layer simultaneously. For instance, color of all the academic buildings will change at the same time if a new background color is assigned in the class definition of the academic building layer. Therefore, display properties of the map can also be modified dynamically.

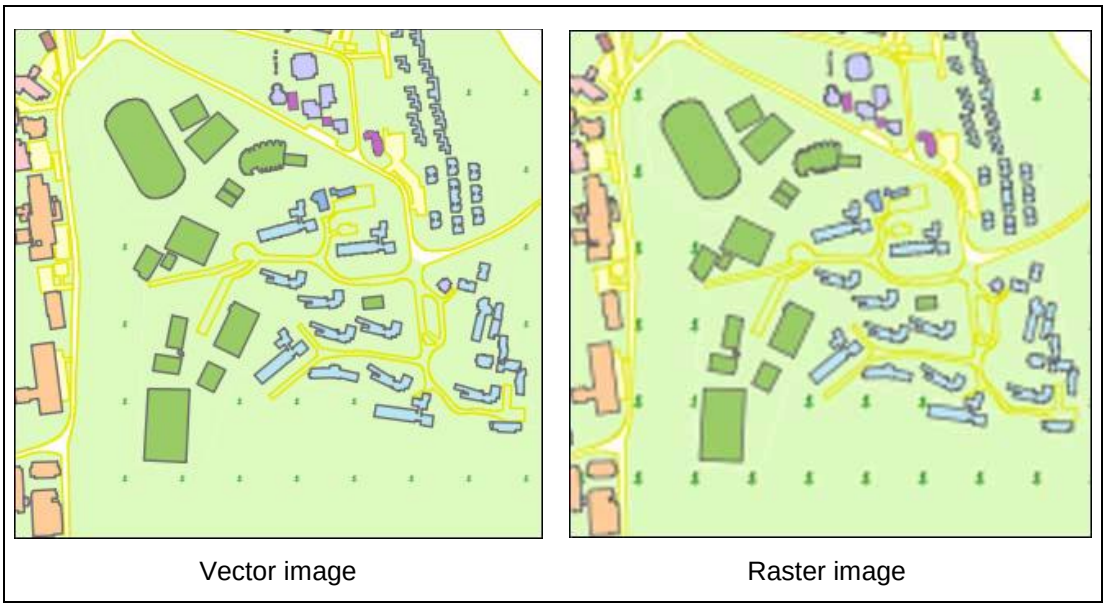


Figure 5.19 Comparison of display qualities of vector and raster images

JavaScript functions provide high level of interactivity to SVG documents by enhancing the dynamic capabilities of SVG objects. However, using such client side scripting languages usually causes cross-browser problems. This means that, a JavaScript function that works on a specific browser may not run properly on other browsers. Different versions of the same browsers can also react differently. Therefore, the implementation was tested on most commonly used browsers and their versions, and some modifications had to be made to terminate the errors after the test. Currently, the application is compatible with Internet Explorer 6 and 7, Mozilla Firefox 3+, and Safari 3+ on both Windows and Linux platforms.

Another problem about the web browsers is forcing the users to install a SVG plug-in to view the map in advance. Fortunately, latest versions of many browsers have their built-in SVG support today, and display the SVG documents without any requirements.

SVG data can easily be updated since they are stored in the database. Each object can be modified individually, without accessing the source code of the SVG document. Preparing some specified user interfaces will make data editing easier. Therefore, SVG is suitable for the systems that require frequent data updates.

Before applying SVG to GIS, total number of map features must be considered. Large number of entities increases the file size and time delays, since whole document is downloaded on the client side. Organization of the graphical attributes will be rather difficult. On the other hand, variety and size of non-spatial data do not affect the system reversely. In this study, METU map has 488 entities in overall [Table 5.3], making a file size of about 800 KB. These numbers appear reasonable to implement this method.

SVG cannot afford all functionalities of GIS, but it is an open-source, flexible, dynamic and inexpensive way for publishing geographical information on the world wide web. Since the objective of this study is to show the locations of the structures and facilities of METU campus with an interactive map published on the Internet, this architecture seems to meet the requirements.

Table 5.5 Numbers of the campus entities

Entity	Number
base area	13
building	295
forest	9
parking lot	76
road	15
road details	3
transportation points	26
wireless network points	52
Total	489

5.5. Discussion

In table 5.6, campus areas, building counts and student population of METU and the studied universities are compared. Some of them cover large parks and recreation areas within their campus areas, especially the ones from the USA. On the other hand, METU campus has a compact structure, and seems to have a relatively small campus area although it has more buildings and an extensive land overall.

The universities that have large areas and a lot of buildings usually use map application servers to publish their campus map on the Internet. Some of these maps have extra functionalities which METU campus does not have, such as different selection options for zooming, measuring distances and areas, and finding the shortest path. However, these systems are relatively slow, since almost every operation of a map tool requires the map image be re-rendered by the server, which creates time delays and makes them relatively slow.

Table 5.6 Some numbers from the studied universities

University	Campus Area (hectares)	Number of buildings	Student Population
Middle East Technical University	**200**	**298**	**20000**
Ege University [Ölgen, et. al, 2004]	345	600	30000
Emory University [Emory University, 2009]	281	248	12700
University of Missouri [Mizzou, 2009]	555	161	30000
University of Oregon [UO, 2009]	119	60	20376
University of Utah [U of U, 2008]	607	225	30450

Having 345 hectares of campus area with 520 buildings and the student population of about 30000 [Ölgen, et. al, 2004], Ege University campus has similar properties with METU. It also has the most functional campus map among the Turkish universities that were studied for this research with its search options, fancy information windows and path finding capabilities. However, only 90 of the buildings are displayed on the map, and information about the facilities is limited.

Tree view menus are mostly preferred by the universities with many layer elements or sets of buildings, such as Emory, Ege and Missouri Universities, in order to list them. These menus may result a confusing view and expand the page when several branches are opened at the same time. On the other hand, METU map has two combo boxes which group the building and facility types, and list the members of each group at once on the same area, where the results of searching by phrase are also displayed. Sharing a limited area for all result sets saves place on the user interface and avoids page scrolling.

If a web site of a university is prepared in more than one language, its campus map should also has a multilanguage support, especially if the university is an international one. All of the universities that were studied for this research present the maps only in their mother languages.

Some universities published their campus maps in Shockwave Flash (SWF) format, which supports vector graphic format as well as SVG. Being a multimedia platform and a strong graphic editor, Flash offers intelligent graphics with better appearance, and rich animations. It is also possible to create animations in SVG documents by using SMIL language, but its browser incompability problems have not been solved yet. Flash movies can be more interactive by connecting them to database systems and map application servers with proper extensions and scripting languages, but not as flexible as SVG applications.

Despite its qualifications, excluding the University of Oregon, the inspected universities have not used the full capabilities of Flash in their campus maps. Although the maps have strong visual appearances, data content and map functionalities are relatively poor. Probably, they do not need a campus map since they have smaller area and fewer buildings, such as Pamukkale University. In addition, Hacettepe University and İstanbul Technical University have multiple campus areas, and their buildings and student populations are spread on different campus locations. Therefore, if there is no difficulty in finding a target in the campus, a sophisticated map with a search interface may not be needed.

In conclusion, area of the campus, number of the buildings, and variety of the facilities are important criteria before choosing the scope and methodology for web mapping applications. Since METU campus is one of the biggest ones among all the universities in the world, map content and capabilities must be also very rich.

5.6. Recommendation

It will be more purposeful if the system is accessible in outdoor areas, without depending on the personal computers and mobile telephones. Displaying the interactive campus map from several kiosks located on the most crowded regions in the campus will help the users find their current and target locations easily. Therefore, efficiency and usage of the system will be increased.

After the implementation has been used for a while, further needs can grow up. At that time, it will be possible to migrate to bigger systems, such as Web Feature Services (WFS) with Geography Markup Language (GML) encoding. A proprietary traditional system consisting of a map application server can also be constructed by using the raw data of METU, which are ready in shapefile format. In addition, 3D modelling of the campus can be integrated to the system, which will improve the visual interpretation of the application.

By the help of the data input and editing interface, current data can be collected and used for the future projects. However, usage of the system will be ineffective if data update is not a continous process. The worst possibility is to serve inaccurate information to the users. In order to avoid that, proper staff from the academic and administrative units must be assigned, encouraged and controlled to use this system.

Since the academic life started in the current campus of METU in 1963 [METU, 2009], construction and renovation of the buildings have never stopped. Also, there has been a continuous change in land use, and infrastructure of the campus, such as network, electricity and telephone cabling, heating, plumbing, lighting, ventilating and sewerage systems, water and gas lines. Several technical units are responsible for planning, controlling, construction and maintainance of them.

Geographical Information Systems give a great opportunity to integration of facility management operations of the university. Current geographical data of all utilities can be inserted to the system by some GIS softwares, and updated easily. Any single entry or a modification can be tracked from the Internet with specified interfaces instantly, and spatial data will be distributed from a centralized source.

Having such a system will provide better observations about the requirements and deficiencies of the campus, more accurate decisions for future planning, hence better service for the university. The management and the administrative units will be able to access the spatial data everywhere, make improved spatial and statistical analysis, and receive reports and printouts easily.

If there is a malfunction on the underground network distribution, for instance, a leakage in one of the water pipes, or a power cut because of a corrupted electricty

cable, location of the problem can not be inspected easily from the ground. Integrating the infrastructure network into a centralized information system will provide the means to reach the sources of the problems, minimize the risk of digging the wrong areas or damaging other distribution lines accidentally, hence saves time and unnecessary expenses.

Since the locations of the assets can be viewed on the map, inventory will be tracked more accurately. Needs estimation can easily be performed, and redundant equipment will not be purchased. It will be possible to keep every modification or displacement process of the equipments in the database, and access the inventory history.
When an additional component is to be installed or a renovation be performed, size and amount of the equipment needed can be determined with the area calculation and distance measuring tools of GIS softwares. This helps in making better cost analysis and equipment estimation during the procurement process.

In order to carry out a facility management system properly, the technical and administration units have to work in a coordination, follow up the activities of each other before making decisions and plans, and GIS techniques would be very helpful for them in every step.

REFERENCES

Abel, D. J., Taylor, K., Ackland, R., Hungerford, S., 1998, "An Exploration of GIS Architectures for Internet Environments", Comput., Environ., and Urban Systems, Vol. 22, No. 1, pp. 7-23

Adobe, “Adobe SVG Viewer Download Area”, *http://www.adobe.com/svg/viewer/install/*, last accessed on March 3, 2009

Aga Khan Developement Network, “Aga Khan Award for Architecture - Awards 1993-1995”, available at *http://www.akdn.org/akaa_award6_awards.asp*, last accessed on March 12, 2009

Aydın, Y. E., 2006, "Web Based Multi Participant Spatial Data Entry in Crime Mapping”, M.Sc. Thesis, Geodetic and Geographical Information Technologies, Middle East Technical University, Ankara

Aydınoğlu, A. Ç., Yomralıoğlu, T., 2002, "Web Based Campus Information System", International Symposium on GIS, September 23-26, Istanbul

Behr, F.J., Hui, L., Hang, C., Weldeslasie, F., Zhuoer, C., 2006, "PHPMyWMS - an Open Source based, SVG oriented Framework for extended Web Map Services", Geospatial Information Technology, Proceedings of SPIE, Vol. 6421

Chen, Y., Gong, J., Jia, W., Zhang, Q., 2004, "XML-Based Spatial Data Interoperability on the Internet", ISPRS Congress, Istanbul 2004, Proceedings of Commission IV

Clarke, P., 2005, "Dynamic web-mapping using Scalable Vector Graphics (SVG)", 2005 ESRI International User Conference Proceedings

Duchowski, A. T., 2007, “Eye Tracking Methodology”, 2nd Edition, Springer, London

Emory University, *http://www.emory.edu*, last accessed on May 10, 2009

Foote, K. E., Kirvan, A. P., 1998, “WebGIS”, NCGIA Core Curriculum in Geographic Information Science, available at *http://www.ncgia.ucsb.edu/giscc/units/u133/u133.html*, last accessed on April 18, 2009

Google, “Ankara - Google Maps”, *http://maps.google.com/*, last accessed on April 5, 2009

Green, D., Bossomaier, T., 2002, "Online GIS and Spatial Metadata", Taylor & Francis, London, New York

Güllüoğlu N. ,C., 2005, “Evaulating Public Transportation Alternatives in the METU Campus With the Aid of GIS”, M.Sc. Thesis, Geodetic and Geographical Information Technologies, Middle East Technical University, Ankara

Hacettepe University, “General Information”, *http://hun.edu.tr/gorsel/yer.php?awk=2&dum=1*, last accessed on May 10, 2009

Kotzinos, D., Prastacos, P., 2001, “GAEA, a Java-based Map Applet”, Systems Analysis Modelling and Simulation, 41 (4), pp. 593-606

Market Share, “Browser Market Share”, *http://marketshare.hitslink.com/browser-market-share.aspx?qprid=0*, last accessed on May 10, 2009

METU, 2009, Middle East Technical University General Catalog 2007-2009, Ankara

METU Computer Center, “Campus Hardware Infrastructure”, *http://www.cc.metu.edu.tr/index_eng.php?go=tsg&sub=server*, last accessed on November 12, 2008

METU General Secretariat, “Office of Forestation and Landscape Planning”, *http://www.acdm.metu.edu.tr/*, last accessed on November 1, 2008

METUTech, *http://www.metutech.metu.edu.tr/*, last accessed in March 2, 2009

Montello, D. R., 2002, "Cognitive Map-Design Research in the Twentieth Century: Theoretical and Empirical Approaches", Cartography and Geographic Information Science, 29, pp. 283-304

Mozilla, “ActiveX”, available at *http://support.mozilla.com/en-US/kb/Activex*, last accessed on May 09, 2009

Mizzou, “About University of Missouri”, *http://www.missouri.edu/about/*, last accessed on May 10, 2009

Nielsen, J., 2000, “Why You Only Need to Test with 5 Users”, available at *http://www.useit.com/alertbox/20000319.html*, last accessed on May 14, 2009

ODTÜ, Bilgi İşlem Dairesi Başkanlığı, “İBE Broşürü”, available at *http://www.bidb.odtu.edu.tr/index.php?go=usg&sub=ibe*, last accessed on May 9, 2009

OGC Inc., 2006, “OpenGIS Web Map Service (WMS) Implementation Specification, Version: 1.3.0”, availlable at *http://www.opengeospatial.org/standards/wms*, last accessed on December 14, 2009

Ölgen, M. K., İnceoğlu, M. M., Cinsdikici, M., İkiz, M., 2004, “Ege Üniversitesi Kampüs Coğrafi Bilgi Sistemi”, 3rd GIS Days in Turkey, İstanbul

Putz, S., 1994, “Interactive Information Services Using World-Wide Web Hypertext”, Computer Networks and ISDN Systems, In Proceedings of the First International Conference on World Wide Web , Geneva

Ramirez, A. O., 2000, “Three-Tier Architecture”, available at *http://www.linuxjournal.com/article/3508*, last accessed on March 6, 2009

Rayner, K., 1998, "Eye Movements in Reading and Information Processing : 20 Years of Research", Psychological Bulletin, Vol. 124, No. 3, 372-422

Seff, G., 2002, “Scalable Vector Graphics and Geographic Information Systems”, Directions Magazine, available at *http://www.directionsmag.com/article.php?article_id=198&trv=1*, last accessed on March 11, 2009

Shea, G. Y. K., 2001, “A Web-Based Approach to the Integration of Diverse Data Sources for GIS”, M.Sc. Thesis, Surveying and Spatial Information Systems, University of New South Wales, Sydney

Su, Y., Slottow, J., Mozes, A., 2000, “Distributing proprietary geographic data on the world wide web - UCLA GIS Database and Map Server”, Computers & Geosciences 26, p:741-749

SVG Wiki, “Viewer Implementations”, *http://wiki.svg.org/Viewer_Implementations*, last accessed on February 1, 2009

Tobii, 2008, "Tobii Studio 1.2 User Manual", available at http://www.tobii.com, last accessed on May 8, 2009

U of U, 2008, “University of Utah Campus Master Plan”, available at *http://www.facilities.utah.edu/static-content/facilitiesmanagement/files/pdf/2008_UofU_CMP_1_Introduction.pdf*, last accessed on May 10, 2009

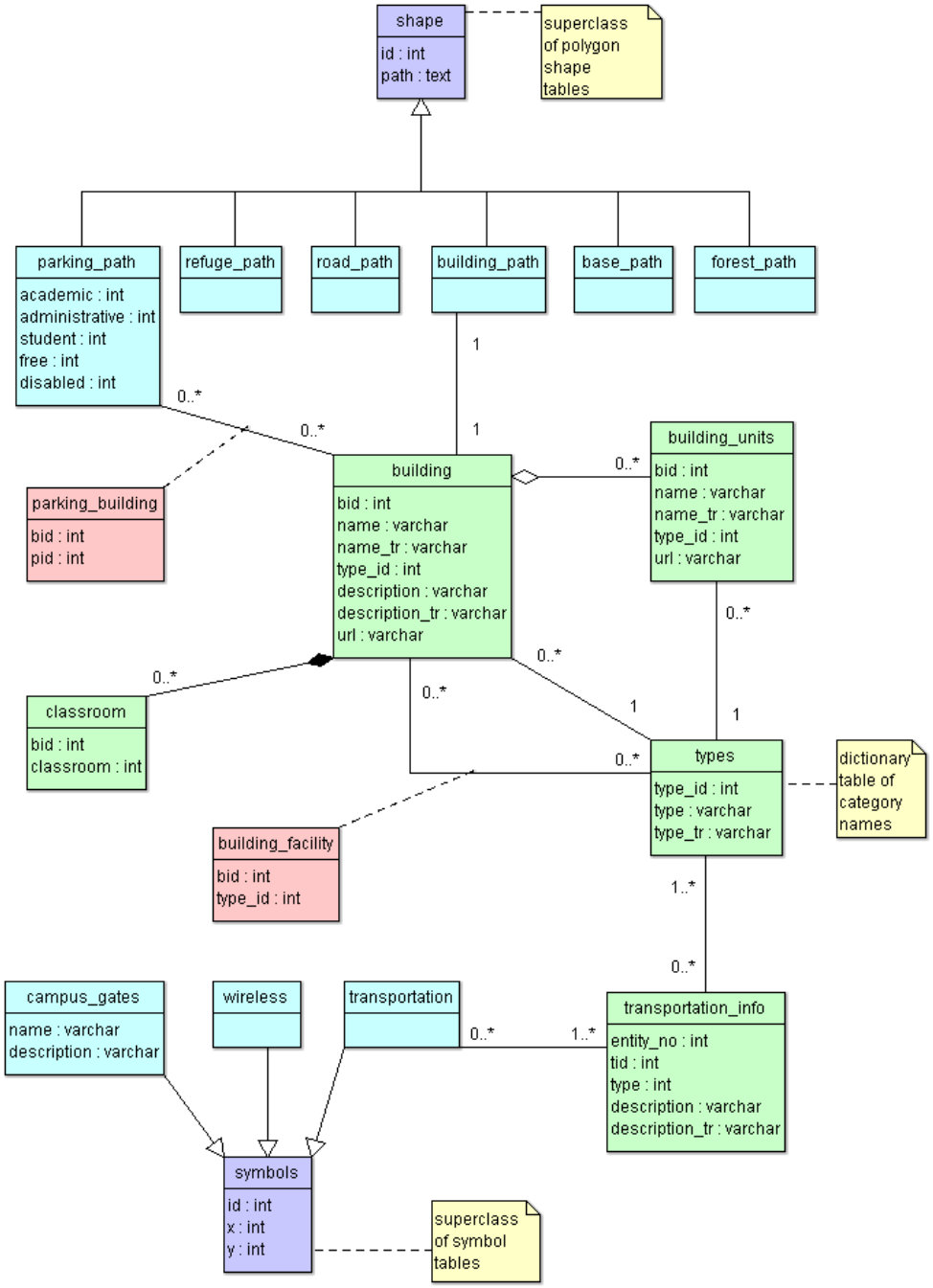
UO, “About the University of Oregon”, *http://www.uoregon.edu/about/*, last accessed on May 10, 2009

W3C, 2003, “Scalable Vector Graphics (SVG) 1.1 Specification”, available at *http://www.w3.org/TR/SVG11/*, last accessed on May 5, 2009

W3schools, “Introduction to XML”, available at *http://www.w3schools.com/xml/xml_whatis.asp*, last accessed on March 10, 2009

APPENDIX A

E-R DIAGRAM OF THE DATABASE IN UML REPRESENTATION



APPENDIX B

USER QUESTIONNAIRE

KULLANICI ANKETİ

	1	2	3	4	5
ODTÜ'nün böyle bir çalışmaya ihtiyacı vardır.	○	○	○	○	○
Haritanın içeriği yeterlidir.	○	○	○	○	○
Haritayı kullanmayı öğrenmekte zorluk çekmedim.	○	○	○	○	○
Aşağıdaki butonların görevlerini anlamakta zorluk çekmedim :					
	○	○	○	○	○
	○	○	○	○	○
	○	○	○	○	○
	○	○	○	○	○
	○	○	○	○	○
	○	○	○	○	○
Aşağıdaki butonları kullanmakta zorluk çekmedim :					
	○	○	○	○	○
	○	○	○	○	○
	○	○	○	○	○
	○	○	○	○	○
	○	○	○	○	○
	○	○	○	○	○
Haritanın grafiksel görüntüsünü beğendim.	○	○	○	○	○
Haritanın arayüz tasarımını beğendim.	○	○	○	○	○
Arama seçeneklerini kullanmakta zorlanmadım.	○	○	○	○	○

1 : Kesinlikle katılmıyorum
2 : Katılmıyorum
3 : Emin değilim
4 : Katılıyorum
5 : Kesinlikle katılıyorum

Eklemek istedikleriniz :

Ad soyad *(zorunlu değil)* :

Anketi Tamamla

APPENDIX C

JAVASCRIPT FUNCTIONS

```
function hide(element_name) {

svgobj =document.embeds['svg_map'].getSVGDocument().getElementById(element_name);
svgobj.setAttributeNS(null,'visibility', 'hidden');

}



function hilite_elem(checkbox, element_name){

svgobj =document.embeds['svg_map'].getSVGDocument().getElementById(element_name);

if (!checkbox.checked){
        svgobj.setAttributeNS(null,'style', 'display:none;fill-rule:evenodd');
        }
else {
        svgobj.setAttributeNS(null,'visibility', 'visible');
        svgobj.setAttributeNS(null,'style', 'display:inline;fill-rule:evenodd');                }


}

function switchpage(select) {

var index;

for(index=0; index<select.options.length; index++)
        if(select.options[index].selected) {
                if(select.options[index].value!="")
                        window.location.href=select.options[index].value;
                        break;
                        }
}



function layerAc() {

obj=document.getElementById("layers");
text1=document.getElementById("on");
text2=document.getElementById("off");
durum=obj.style.display;

if(durum=="none") {
        obj.style.display="block";
        text1.style.display="none";
        text2.style.display="block";
        }
```

```
if(durum=="block") {
	obj.style.display="none";
	text2.style.display="none";
	text1.style.display="block";
	}
}



var click = false, object ="";
	var x_obj = 0, y_obj = 0, x_trans = 0, y_trans = 0;
	var pan=false;
	var info=true;

	function zoom(evt) {

		//We find the active button for map navigation, and define its events :
		var type =evt.target.parentNode.getAttribute("id");
		var box=document.getElementById("map").getAttribute("viewBox").split(" ");
		var toolbox=document.getElementById("toolbox");
		var main=document.getElementById("map");
		//define the initial pan and scale values
		var scale_val=1;
		var pan_val=75;
		var pan_scale=50;

		//clicking the pan value disables the pan button
		/*
		if(pan==true)
			info=false;
		*/

		//this is the object defined by group <g> named as "all"
		var ponpon=document.getElementById("all");

		//We split scale and translate values from transform definition
		trans = ponpon.getAttributeNS(null , "transform");
		ind=trans.indexOf("scale");

		//find the scale value
		scale=trans.substring(ind,trans.length-1);
		scale=scale.substring(6,scale.length);
		//scale is an array showing scale of x and scale of y  of group <g> named as "all"
		var scale = scale.split(",");
		scale[1]=scale[1].replace(")","");

		//find the translate value
		trans = trans.substring(0,ind);
		trans = trans.substring(10,trans.length - 2);
		//translate is an array showing initial x and y coordinates of group <g> named as "all"
		var trans = trans.split(",");
		trans[0]=trans[0].replace("e(","");

		var select1=document.getElementById("pan_selected");
		var select2=document.getElementById("info_selected");

		switch (type) {

			//trans[0]=original x coordinate of the origin
			//trans[1]=original y coordinate of the origin
			//trans0=new x coordinate of the origin
			//trans1= new y coordinate of the origin
			//scale_val=scale factor
			//650=>width of the viewport
			//600=>height of the viewport
```

```
                        case "zoomin" :
                        //alert(scale[0]);
                        scale_val=1.25;
                        select1.setAttributeNS(null,"visibility","hidden");
                        select2.setAttributeNS(null,"visibility","hidden");
                        scale0=scale[0]*scale_val;
                        scale1=scale[1]*scale_val;
                        trans0=parseFloat(trans[0])-parseFloat((scale_val-1)*650/2);
                        trans1=parseFloat(trans[1])-parseFloat((scale_val-1)*600/2);
                        ponpon.setAttributeNS(null,"transform","translate("+trans0 + "," + trans1 +")
scale("+scale[0]*scale_val+","+scale[1]*scale_val+")");
                        break;

                        case "zoomout" :
                        scale_val=0.8;
                        select1.setAttributeNS(null,"visibility","hidden");
                        select2.setAttributeNS(null,"visibility","hidden");
                        trans0=parseFloat(trans[0])+parseFloat((1/scale_val-1)*650/2);
                        trans1=parseFloat(trans[1])+parseFloat((1/scale_val-1)*600/2);
                        ponpon.setAttributeNS(null,"transform","translate("+trans0 + "," + trans1 +")
scale("+scale[0]*scale_val+","+scale[1]*scale_val+")");
                        break;

                        case "fullextent" :
                        select1.setAttributeNS(null,"visibility","hidden");
                        select2.setAttributeNS(null,"visibility","hidden");
                        ponpon.setAttributeNS(null,"transform","translate(-100,-150)
scale(1.5,1.5)");
                        break;

                        case "pan":
                        //SVGDocument = LoadEvent.target.ownerDocument
                        //top.document.embeds[0].body.style.cursor='move';
                        select2.setAttributeNS(null,"visibility","hidden");
                        select1.setAttributeNS(null,"visibility","visible");

                        pan=true;
                        info=false;
                        break;

                        case "info":
                        info=true;
                        select1.setAttributeNS(null,"visibility","hidden");
                        select2.setAttributeNS(null,"visibility","visible");
                        break;

                        case "northarrow":
                        //alert(main.filename);
                        //window.focus();
                        //self.print();
                        //toolbox.setAttributeNS(null,"visibility","hidden");
                        //main.print();
                        //toolbox.setAttributeNS(null,"visibility","visible");
                        //setTimeout("window.print();",5000);
                        break;

                        case "south":
                        trans0=parseFloat(trans[0]);
                        trans1=parseFloat(trans[1])-pan_val;
                        ponpon.setAttributeNS(null,"transform","translate("+trans0 + "," + trans1 +")
scale("+scale[0]*scale_val+","+scale[1]*scale_val+")");
                        break;

                        case "east":
                        trans0=parseFloat(trans[0])+pan_val;
```

```
                        trans1=parseFloat(trans[1]);
                        ponpon.setAttributeNS(null,"transform","translate("+trans0 + "," + trans1 +")
scale("+scale[0]*scale_val+","+scale[1]*scale_val+")");
                        break;

                        case "west":

                        trans0=parseFloat(trans[0])-pan_val;
                        trans1=parseFloat(trans[1]);
                        ponpon.setAttributeNS(null,"transform","translate("+trans0 + "," + trans1 +")
scale("+scale[0]*scale_val+","+scale[1]*scale_val+")");
                        break;

                        case "north":
                        trans0=parseFloat(trans[0]);
                        trans1=parseFloat(trans[1])+pan_val;
                        ponpon.setAttributeNS(null,"transform","translate("+trans0 + "," + trans1 +")
scale("+scale[0]*scale_val+","+scale[1]*scale_val+")");
                        break;

                        return pan;
                        return info;
                }


        }



function onmouseover(evt) {


                //Mozilla sorunu icin sadece binalari aliyoruz,yoksa hepsi yanip sonuyor

                if(evt.target.parentNode.getAttribute("id")=="academic") {
                        var elem = evt.target;
                        elem.setAttributeNS(null,"fill-opacity",0.5);
                }

                node=evt.target.ownerDocument.getElementById("buildingName");
                recto=evt.target.ownerDocument.getElementById("infocan");

                new_x=evt.clientX;
                new_y=evt.clientY;

                b_name=evt.target.getAttributeNS(null,"name");
                node.firstChild.nodeValue=b_name;
                if(b_name!="")
                        recto.setAttributeNS(null,'visibility', 'visible');

        }

        function onmouseout(evt) {
                var elem = evt.target;
                elem.setAttributeNS(null,"fill-opacity",1);
                node=evt.target.ownerDocument.getElementById("buildingName");
                recto=elem.ownerDocument.getElementById("infocan");
                node.firstChild.data=null;
                recto.setAttributeNS(null,'visibility', 'hidden');

        }
```

```
function clicked(evt)
{



    if(pan==true) {
            click = true;

     object = evt.target.parentNode.parentNode;
     //alert(object.getAttributeNS(null , "id"));
     trans = evt.target.parentNode.parentNode.getAttributeNS(null , "transform");
              ind=trans.indexOf("scale");

              scale=trans.substring(ind,trans.length-1);
              scale=scale.substring(6,scale.length);
              scale = scale.split(",");
              scale[1]=scale[1].replace(")","");

              trans = trans.substring(0,ind);
              trans = trans.substring(10,trans.length - 2);
              //alert(trans);
     trans = trans.split(",");
     trans[0]=trans[0].replace("e(","");
     x_trans = trans[0] - 0;
     y_trans = trans[1] - 0;
     x_obj = evt.screenX - 0;
     y_obj = evt.screenY - 0;


    }
            var gok=evt.target.parentNode.getAttribute("id");
    if(info==true)


            var data=evt.target.getAttributeNS(null,"id");


            if(data) {
                                    if(gok=='transportation') {
                                            var url="http://www.gis.metu.edu.tr/transport.php?data="+data;
                                            x=300;
                                            y=150;
                                            }
                                    else if(gok=='parking') {
                                            var url="http://www.gis.metu.edu.tr/parking.php?data="+data;
                                            x=350;
                                            y=150;
                                            }
                                    else {
                                    var url="http://www.gis.metu.edu.tr/building.php?data="+data;
                                            x=420;
                                            y=500;
                                            }

            mywindow=open(url,"mywindow","status=0,toolbar=0,location=0,menubar=0,scrollbars=yes,wi
dth="+x+", height="+y);

            }

}


            function move_object(evt)
{
if (click)
            {
```

```
        xm = evt.screenX - 0
        ym = evt.screenY - 0

            object.setAttributeNS(null,"transform","translate("+( x_trans + xm - x_obj) + "," + ( y_trans +
ym - y_obj) +") scale("+scale[0]+","+scale[1]+")")
            }
}
```

APPENDIX D

PHP CODES

File : embed.php

```
<?
    header("Content-type: image/svg+xml");
   session_start();
            echo '<?xml version="1.0" ?>';
    echo '<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">';

include("connect.php");
include("php_functions.php");



?>


<svg width="605" height="610" viewBox="0 0 605 610" enable-background="new 0 0 605 610"
version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">


<svg width="605" height="605" viewBox="0 100 605 605" id="map" version="1.1"
xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">


<symbol id="broadLeave" overflow="visible"
          style="fill:none;stroke:green;stroke-width:2;">
  <path d="M0 15 l0 -10"/>
  <path d="M0 5 a6 6 0 1 1 -3.5 -10
                    a6 6 0 1 1 7 0 a6 6 0 1 1 -3.5 10z"/>
</symbol>

<symbol id="coniferous" overflow="visible" style="fill:green;stroke:green;stroke-width:2">

                                <path d="M0 -15 l0 30"/>
                                <path d="M0 -15 q0 3.5 5 3.5"/>
                                <path d="M0 -15 q0 3.5 -5 3.5"/>

                                <path d="M0 -9 q0 6 8 6"/>
                                <path d="M0 -9 q0 6 -8 6"/>
                                <path d="M0 0 q0 7 10 7"/>
                                <path d="M0 0 q0 7 -10 7"/>

                    </symbol>

<symbol id="bus">
<image  xlink:href="mapicons/busstop.gif"  width="3" height="4"  />
</symbol>
```

```
<defs>
<linearGradient id="ok" x1="0%" y1="0%" x2="0%" y2="100%">
<stop offset="0%" style="stop-color:rgb(0,255,255);
stop-opacity:1"/>
<stop offset="100%" style="stop-color:rgb(51,102,153);
stop-opacity:1"/>
</linearGradient>
</defs>

<symbol id="search">
<g transform="scale(0.4,0.3)">
<path
      id="path2540"
      d="M 6.1875 44.5 L 6.1875 56.6875 L 1.09375 56.6875 L 9.59375 67.75 L 18.09375 56.6875 L
12.96875 56.6875 L 12.96875 44.5z"
      style="fill:url(#ok);fill-opacity:1;stroke:#0c58a0;stroke-width:1;stroke-opacity:1" />

</g>

</symbol>

<symbol id="disabled">
<image  xlink:href="mapicons/disabled.gif"  width="3" height="3"  />
</symbol>

<pattern id="pat01" patternUnits="userSpaceOnUse" width="10" height="20">
  <use xlink:href="#broadLeave"  transform="translate(5,5) scale(0.2)  rotate(20)"/>
</pattern>

<pattern id="pat02" patternUnits="userSpaceOnUse" width="12" height="16">
                              <use xlink:href="#coniferous" transform="translate(3,3) scale(0.1)
rotate(10)"/>
                  </pattern>



<style type="text/css">
  <![CDATA[

          .road {stroke:#e8e800;stroke-width:0.47994;stroke-miterlimit:10;stroke-linecap:round;
                    stroke-linejoin:round;fill:none}
          .roads {stroke:#e8e800;stroke-width:0;fill:#ffffcc}
          .park {stroke:#e8e800;stroke-width:0.1;stroke-miterlimit:10;stroke-linecap:round;
                    stroke-linejoin:round;fill:#ffffcc}
          .forrest {stroke:#e8e800;stroke-width:0;fill:url(#pat02)}
          .bank {stroke:#6E6E6E;stroke-width:0.3;stroke-miterlimit:10;stroke-linecap:round;
                    stroke-linejoin:round;fill:#cc66cc}
          .building {stroke:#6E6E6E;stroke-width:0.3;stroke-miterlimit:10;stroke-linecap:round;
                    stroke-linejoin:round;fill:#8BBBEA}
          .academic {stroke:#6E6E6E;stroke-width:0.3;stroke-miterlimit:10;stroke-linecap:round;
                    stroke-linejoin:round;fill:#ffcc99}
          .residential {stroke:#6E6E6E;stroke-width:0.3;stroke-miterlimit:10;stroke-linecap:round;
                    stroke-linejoin:round;fill:#B8E5F2}
          .sports {stroke:#6E6E6E;stroke-width:0.3;stroke-miterlimit:10;stroke-linecap:round;
                    stroke-linejoin:round;fill:#99CC66}
          .administrative
                              {stroke:#6E6E6E;stroke-width:0.3;stroke-miterlimit:10;stroke-linecap:round;
                    stroke-linejoin:round;fill:#ffcccc}
          .food
                              {stroke:#6E6E6E;stroke-width:0.3;stroke-miterlimit:10;stroke-linecap:round;
                    stroke-linejoin:round;fill:#ccccff}
          .education
                              {stroke:#6E6E6E;stroke-width:0.3;stroke-miterlimit:10;stroke-linecap:round;
                    stroke-linejoin:round;fill:#D7D783}
          .cultural
```

```
                    {stroke:#6E6E6E;stroke-width:0.3;stroke-miterlimit:10;stroke-linecap:round;
                stroke-linejoin:round;fill:#D79191}
        .technical {stroke:#6E6E6E;stroke-width:0.3;stroke-miterlimit:10;stroke-linecap:round;
                stroke-linejoin:round;fill:#cccccc}

        .area { fill:green}
   .wireless {fill:#FF9933; fill-rule:evenodd; stroke:black; stroke-width:0.2; stroke-miterlimit:10;
                stroke-linecap:round; stroke-linejoin:round
   }
  ]]>
 </style>
<script type="text/javascript" xlink:href="maptools.js" />


        <g id="all" onmousedown="if (click==false) clicked(evt); else click=false;"
onmousemove="move_object(evt)" onmouseup="click=false"
        transform="
        <?

        if($_GET["x"] && $_GET["y"]) {
        $x=$_GET["x"];
        $y=$_GET["y"];
        $id=$_GET["id"];
        echo "translate($x,$y) scale(3,3)";
        //echo "translate(0,0) scale(1.5,1.5)";
        }
        else
        echo "translate(-100,-150) scale(1.5,1.5)";
        ?>
        ">


<g id="rect">
<rect x="10" y="100" width="550" height="500" style="fill:white" />
</g>

<g id="raster" visibility="hidden">
<rect x="10" y="100" width="550" height="500" style="fill:white" />
<image name="Zoom Out" xlink:href="quickbird.jpg" x="0" y="0" width="595" height="842"  />
</g>

<g id="base" visibility="visible">
<?


$sql="select * from base_area";
$result=@pg_query($baglanti, $sql);
while($row=pg_fetch_array($result)) {
echo "<path fill=\"#DDFABF\" d=\"";
echo $row["path"];
echo "\" />";
}

$sql="select * from forest_path";
$result=@pg_query($baglanti, $sql);
while($row=pg_fetch_array($result)) {
echo "<path fill=\"#DDFABF\" d=\"";
echo $row["path"];
echo  "\" />";
echo "<path class=\"forrest\" d=\"";
echo $row["path"];
echo  "\" />";
}
?>
</g>
```

```
<g id="roads">
<?


$sql="select * from roads";
$result=@pg_query($baglanti, $sql);
while($row=pg_fetch_array($result)) {
echo "<path class=\"roads\" d=\"";
echo $row["path"];
echo "\" name=\"Road";
echo "\" id=\"";
echo $row["rid"];
echo  "\" />";
}


$sql="select * from refuge_path";
$result=@pg_query($baglanti, $sql);
while($row=pg_fetch_array($result)) {
echo "<path fill=\"#DDFABF\" d=\"";
echo $row["path"];
echo "\" />";
}
?>

</g>

<g id="parking">
<?
$sql="select * from parking_path";
$result=@pg_query($baglanti, $sql);
while($row=pg_fetch_array($result)) {
echo "<path class=\"park\" d=\"";
echo $row["path"];
echo "\" name=\"Parking space";
echo "\" id=\"";
echo $row["pid"];
echo  "\" />";
}

?>



</g>
<?
function building($type,$class,$lang) {

                $baglanti=connect();

                $sql="SELECT * FROM building_path,building_info
                        WHERE type_id in ($type)
                        AND building_info.bid=building_path.bid";
                $result=@pg_query($baglanti, $sql);

                while($row=pg_fetch_array($result)) {

                        if($lang=='tr')
                                $name=$row["name_tr"];
                        else
                                $name=$row["name"];

                                echo "<path class=\"$class\" d=\"";
                                echo $row["path"];
                                echo "\" name=\"";
                                echo $name;
```

```
                                        echo "\" id=\"";
                                        echo $row["bid"];
                                        echo  "\" />";
                        }
}

?>

<g id="academic" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)"
visibility="visible">
<?

//Akademic buildings
building('1','academic','eng');

?>
</g>


<g id="administrative" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)">
<?
//Administrative buildings
building('2','administrative',$_GET["lang"]);
?>
</g>


<g id="food" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)">
<?
//Food, shopping, health
building('7,8,12','food',$_GET["lang"]);

?>
</g>

<g id="bank" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)">
<?
//Bank and ATM
building('3','bank',$_GET["lang"]);
?>
</g>



<g id="sports" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)" visibility="visible">
<?
//Sport areas
building('14','sports',$_GET["lang"]);
?>
</g>

<g id="technical" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)"
visibility="hidden">
<?
//Technical buildings
building('15','technical',$_GET["lang"]);
?>
</g>

<g id="residential" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)"
visibility="visible">
<?
//Housings and Dormitories
building('5,11','residential',$_GET["lang"]);
```

```
?>
</g>

<g id="cultural" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)">
<?
//Cultural and  Social
building('4,13','cultural',$_GET["lang"]);
?>
</g>

<g id="research" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)"
visibility="visible">
<?
//METUTech and Research
building('9,10','building',$_GET["lang"]);

?>
</g>


<g id="education" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)"
visibility="hidden">
<?


//Education
building('6','education',$_GET["lang"]);

?>
</g>




<g id="disabled" visibility="hidden">

<?

$sql="(select path from building_facilities bf, building_path b
        where bf.bid=b.bid
        and type_id in (26,27,28))
        union
        (select path from parking_path where disabled=1)
        ";
$result=@pg_query($baglanti, $sql);
while($row=pg_fetch_array($result)) {


$coords=center($row["path"]);
$x=$coords[0]-1;
$y=$coords[1]-1;

echo "<use x=\"$x\" y=\"$y\" xlink:href=\"#disabled\" />";

}

?>


</g>
```

```
<g id="transportation" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)"
visibility="hidden">
<?


$sql="select * from transportation";
$result=@pg_query($baglanti, $sql);
while($row=pg_fetch_array($result)) {
        echo "<use x=\"";
        echo $row["x"]+3;
        echo "\" y=\"";
        echo $row["y"]-3;
        echo  "\" xlink:href=\"#bus\"";
        echo " id=\"";
        echo $row["tid"];
        echo "\" name=\"Transportation point";
        echo "\" />";
}

?>

</g>

<g id="searchresult" onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)">
<?
$sql="SELECT * FROM building_path,building_info
        WHERE building_path.bid=".$_GET["id"]."
        AND building_info.bid=building_path.bid";
//$sql="select bid,path from group where bid=".$_GET["id"]."";
$result=@pg_query($baglanti, $sql);

while($row=pg_fetch_array($result)) {
echo "<path fill=\"#FF3333\" d=\"";
echo $row["path"];
echo "\" name=\"";
echo $row["name"];
echo "\" id=\"";
echo $row["bid"];
echo  "\" />";
$path=$row["path"];
}

$coords=center($path);
$x=$coords[2]-4;
$y=$coords[3]-21;


echo "<use x=\"$x\" y=\"$y\" xlink:href=\"#search\" />";

?>

</g>


</g>
 <g id="metin">
<rect x="25" y="130" height="25" width="500" fill="white" visibility="hidden" id="infocan"
fill-opacity="0.5" />
<text x="30" y="147" font-family="arial" font-size="14" font-weight="bold" id="buildingName"
fill="navy"> </text>

 </g>


</svg>
```

```
<rect x="12" y="5" width="580" height="596" style="fill:none;stroke:#698BAC;stroke-width:25; fill-
opacity:0" />
<g onclick="zoom(evt)" id="toolbox" onmouseover="onmouseover(evt)"
onmouseout="onmouseout(evt)" visibility="visible">

                    <a transform="translate(508,25)">
                        <rect x="0" y="10" width="65" height="255"
style="fill:#ffffff;stroke:#cccccc;stroke-width:5;" />
                    </a>

                    <a id="zoomout"  transform="translate(514,40) scale(1)">
                        <image name="Zoom Out" xlink:href="mapicons/zoomout.gif" x="0" y="0"
width="52" height="45"  />
                    </a>

                    <a id="zoomin" transform="translate(514,90) scale(1)">
                        <image name="Zoom In" xlink:href="mapicons/zoomin.gif" x="0" y="0"
width="52" height="45"  />
                    </a>

                    <a id="fullextent" transform="translate(514,140) scale(1)">
                        <image name="Full View" xlink:href="mapicons/fullextent.gif" x="0" y="0"
width="52" height="45"  />
                    </a>

                    <a id="pan" transform="translate(514,190) scale(1)">
                        <image name="Move" xlink:href="mapicons/pan.gif" x="0" y="0" width="52"
height="45"  />
                    </a>
                    <a id="info" transform="translate(514,240) scale(1)">
                        <image name="Info" xlink:href="mapicons/info.gif" x="0" y="0" width="52"
height="45"  />
                    </a>

                    <a id="east" transform="translate(0,30)">
                      <image name="Go to West" xlink:href="mapicons/west.jpg" x="0" y="0" width="25"
height="566"  />
                    </a>
                    <a id="north" transform="translate(0,0)">
                        <image name="Go to North" xlink:href="mapicons/north.jpg" x="0" y="0"
width="606" height="30"  />
                    </a>
                    <a id="west" transform="translate(580,30)">
                    <image name="Go to East" xlink:href="mapicons/east.jpg" x="0" y="0" width="25"
height="566"  />
                    </a>

                    <a id="south" transform="translate(0,586)">
                        <image name="Go to South" xlink:href="mapicons/south.jpg" x="0" y="0"
width="606" height="30"  />

                    </a>
                    <a id="north_arrow" transform="translate(30,540)">
                        <image id="North Arrow" xlink:href="mapicons/north.gif" x="0" y="0"
width="42" height="42" opacity="0.8" />
                    </a>
                    <a id="pan_selected" transform="translate(517,190)" visibility="hidden">
                        <image id="Move" xlink:href="mapicons/pan_selected.gif" x="0" y="0"
width="52" height="46"  />
                    </a>
                    <a id="info_selected" transform="translate(514,240)" visibility="hidden">
                        <image id="Info" xlink:href="mapicons/info_selected.gif" x="0" y="0"
width="52" height="45"  />
                    </a>
          </g>
</svg>
```

File : php_functions.php

```
<?

//x,y coordinatlarini ayiklayan fonksiyon
    function path($path) {
  $coord=explode("L",$path);
            $coord=explode("M",$coord[0]);
            $coord=explode(",",$coord[1]);
            $x=explode(".",$coord[0]);
            $x=$x[0];
            $y=explode(".",$coord[1]);
            $y=$y[0];
            $transx=315-3*$x;
            $transy=400-3*$y;
            $result=array("$transx", "$transy","$x","$y");
            return $result;
  }



//center of the polygon'u bulur

            function center($var) {

            //Delete M
            $var = substr($var, 1);
            //Replace M's with L, if any :
            $var=str_replace("M","L",$var);
            //Delete z
            $var = str_replace("z","",$var);


            //get rid of L
            $var=explode("L",$var);

            for($i=0;$i<count($var);$i++) {
            $erey=explode(",",$var[$i]);
            $x[$i]=$erey[0];
            $y[$i]=$erey[1];
            }

            $x_org=$x;
            $y_org=$y;

            sort($x);
            sort($y);
            $minx=$x[0];
            $miny=$y[0];
            rsort($x);
            rsort($y);
            $maxx=$x[0];
            $maxy=$y[0];

            for($k=0;$k<count($y_org);$k++) {
                        if ($y_org[$k]==$miny) {
                                    $key=$k;
                                    break;
                                    }
            }

            $xy=$x_org[$key];

            $avex=($minx+$maxx)/2;
```

```
		$avey=($miny+$maxy)/2;

		$result=array("$avex", "$avey","$xy","$miny");
		return $result;

		}

 function seticonv($var) {
		return iconv("UTF-8", "ISO-8859-9", $var);

 }


 function parking($var,$id,$type) {
		include("connect.php");
		if($var=='on')
			$sql="update parking_path set $type=1 where pid=$id";
		else
			$sql="update parking_path set $type=0 where pid=$id";
		pg_query($baglanti,$sql);

}
?>
```