ONLINE VISUALIZATION OF SPATIAL DATA

A PROTOTYPE OF AN OPEN SOURCE INTERNET MAP SERVER WITH BACKEND SPATIAL DATABASE FOR THE SWISS NATIONAL PARK

Thomas Hächler

Diploma Thesis

Department of Geography
University of Zurich
2003

Advisor:
Dr. Britta Allgöwer

Faculty Member:
Prof. Dr. Robert Weibel

ACKNOWLEDGMENTS

I would like to thank all persons who supported me during this thesis project, especially:

- Dr. Britta Allgöwer, for making this project possible, for supervision and revision of this thesis and for providing the data of the GIS Swiss National Park.
- Prof. Dr. Robert Weibel, for supporting this thesis and for its final revision.
- Dr. Martin Galanda and Dr. Alessandro Cecconi, for conceptual contributions and literature hints.
- dipl. zool. Daniel Wirz, for most useful advice with installation issues, helpful discussions about databases and system architecture, and for the revision of the technical chapters of this thesis.
- PD Dr. Herbert Bitto, for interesting discussions about topology and spatial databases, and for the revision of the respective chapter of this thesis.
- The Open Source Community, for developing the software components used in this project and for unselfishly sharing ideas and advice.

My special thanks are dedicated to Yvonne Leuenberger, for supporting and motivating me, and for her patience and love.

Finally, I am forever grateful to my family, especially my parents Margrit and Roland Hächler, who made my education possible and who supported me during all my life.

TABLE OF CONTENTS

PART I: INTRODUCTION AND APPROACH

1 INTRODUCTION 1

1.1 MOTIVATION 1
1.2 THESIS ORGANIZATION 2

2 GIS, THE INTERNET AND WEBGIS 5

2.1 BRIEF HISTORICAL OVERVIEW 5
2.2 INTEROPERABILITY AND STANDARDS 7
2.2.1 WMS Implementation Specification 7
2.2.2 GML Implementation Specification 8
2.2.3 Simple Features Specification For SQL 8
2.3 APPROACHES IN WEB MAPPING 9
2.5 CATEGORIZATIONS IN WEB MAPPING 10
2.5.1 Types of web mapping goals 10
2.5.2 Types of web mapping applications 12
2.5.3 Types of web maps 15
2.6 DEFINITIONS / TERMS AND TECHNOLOGIES 16

3 THESIS APPROACH 19

3.1 ANALYSIS OF DEFICIENCIES 19
3.1.1 A new Cubic Model 19
3.1.2 Requirements for Web Maps 20
3.1.3 Commercial Software vs. Open Systems 21
3.1.4 Research Challenges 22
3.2 CONCLUSIONS 23
3.3 RESEARCH OBJECTIVE 25

PART II: WEBGIS ARCHITECTURE

4 CLIENT-SIDE TECHNOLOGY AND GIS FUNCTIONALITY 27

4.1 CLIENT-SIDE GIS OPERATIONS 27
4.1.1 HTML 27
4.1.2 JavaScript 28
4.1.3 Plug-Ins 28
4.1.4 Java Applets 29
4.1.5 SVG 29
4.2 CLIENT-SIDE CONTEXT PRESERVATION 29
4.2.1 HTML Forms / HTTP-Request and -Response 30
4.2.2 Cookies 30

5 SERVER-SIDE TECHNOLOGY AND GIS FUNCTIONALITY 31

5.1 SERVER-SIDE GIS OPERATIONS 32
5.1.1 Common Gateway Interface 32
5.1.2 Server Plug-Ins 32
5.1.3 Other Possibilities 33
5.2 SERVER-SIDE CONTEXT PRESERVATION 33
5.3 LOAD BALANCING 34
5.4 DATA STORAGE 34

6 EXTENDED RELATIONAL DATABASE MANAGEMENT SYSTEMS 39

6.1 FROM RDBMS TO EXTENDED RDBMS 39
6.1.1 Requirements of Extended RDBMS 40
6.2 TOPOLOGY, REPRESENTATION AND LOGICAL MODELS 40
6.2.1 Topology 41
6.2.2 Geometric Representation of Spatial Objects 41
6.2.3 Spatial Abstract Data Types 44
6.3 DATA ACCESS METHODS 46
6.3.1 Traditional Access Methods 46
6.3.2 Spatial Access Methods 50

PART III: WEBGIS PROTOTYPE

7 METHODS 55

7.1 REQUIREMENTS 55
7.2 DATA 56
7.3 SOFTWARE COMPONENTS OF THE PROTOTYPE 57
7.3.1 Basic Libraries 58
7.3.2 Apache Web Server 59
7.3.3 PHP 59
7.3.4 PostgreSQL 59
7.3.5 PostGIS 61
7.3.6 UMN MapServer 63
7.4 ARCHITECTURAL OVERVIEW AND CONFIGURATION 65
7.4.1 Basic Setup 66
7.4.2 Setup of the Prototype 70
7.5 PROGRAMMING 70
7.5.1 Drawing Layers 70
7.5.2 Dynamic Legend 71
7.5.3 Information and Query page 73

8 THE WEBGIS SNP PROTOTYPE 75

8.1 BASIC FUNCTIONS, TOOLS AND ELEMENTS 76
8.1.1 Main Map 77
8.1.2 Reference Map 78
8.1.3 Graphic Scale 78
8.2 IMPLEMENTED FUNCTIONS AND TOOLS 78
8.2.1 Dynamic Legend 78
8.2.2 Information and Query Page 79
8.3 EXAMPLE USE CASE 81
8.4 USE AND ROLE OF CLASSIC MAP ELEMENTS IN WEB MAPS 86
8.4.1 The Legend and the Concept of Status and Visibility 86
8.4.2 Numeric and Graphic Scale 87
8.4.3 Other Elements 88
8.5 DISCUSSION AND EVALUATION 88

9 CONCLUSIONS 91

9.1 ACHIEVEMENTS 91
9.2 INSIGHTS 92
9.3 OUTLOOK 92

DATA 103

LIST OF FIGURES

FIGURE 1: HISTORICAL OVERVIEW 5
FIGURE 2: MONOLITHIC VS. 3-TIER ARCHITECTURE 6
FIGURE 3: MAP USE CUBE 11
FIGURE 4: TYPES OF WEB MAPPING APPLICATIONS 14
FIGURE 5: CUBE OF WEB MAP TYPES 19
FIGURE 6: LOCATION OF THE PROTOTYPE WITHIN THE CUBE OF WEB MAP TYPES 23
FIGURE 7: STANDARD 3-TIER ARCHITECTURE 31
FIGURE 8: DATA STORAGE ARCHITECTURE 35
FIGURE 9: ALTERNATIVE DATA STORAGE ARCHITECTURES 36
FIGURE 10: TOPOLOGICAL RELATIONS BETWEEN TWO REGIONS 41
FIGURE 11: SPATIAL DATA MODELS 42
FIGURE 12: GEOMETRIC REPRESENTATION 44
FIGURE 13: SPATIAL AND GEOGRAPHIC DATA MODEL 45
FIGURE 14: BINARY SEARCH METHOD 47
FIGURE 15: INDEXING WITH A DENSE INDEX 48
FIGURE 16: INDEXING WITH A SPARSE INDEX 48
FIGURE 17: BINARY TREE 49
FIGURE 18: FIXED GRID 51
FIGURE 19: SPATIAL INDEXING WITH A LINEAR QUADTREE 52
FIGURE 20: PROTOTYPE SOFTWARE COMPONENTS 57
FIGURE 21: PROTOTYPE SYSTEM ARCHITECTURE 65
FIGURE 22: SIMPLIFIED HTML TEMPLATE FILE 67
FIGURE 23: SIMPLIFIED TEMPLATE FILE WITH PHP CODE 68
FIGURE 24: SIMPLIFIED MAP FILE 69
FIGURE 25: DYNAMIC LEGEND 71
FIGURE 26: THE WEBGIS PROTOTYPE APPLICATION 75
FIGURE 27: MAIN MAP IN JAVA MODE 76
FIGURE 28: INFORMATION PAGE 80
FIGURE 29: DYNAMIC ATTRIBUTE QUERY (1): SELECTION OF THE THEME 81
FIGURE 30: DYNAMIC ATTRIBUTE QUERY (2): SELECTION OF THE ATTRIBUTE 81
FIGURE 31: DYNAMIC ATTRIBUTE QUERY (3): PRODUCED SQL STRING 81
FIGURE 32: DYNAMIC ATTRIBUTE QUERY (4): TABLE WITH QUERY RESULTS 82
FIGURE 33: DYNAMIC ATTRIBUTE QUERY (5): VISUALIZATION OF QUERY RESULTS 83
FIGURE 34: DYNAMIC ATTRIBUTE QUERY (6): REFINED SQL STRING 84
FIGURE 35: DYNAMIC ATTRIBUTE QUERY (7): VISUALIZATION OF SUBQUERY RESULTS 85
FIGURE 36: DYNAMIC ATTRIBUTE QUERY (8): AGGREGATION OF SUBQUERY RESULTS 85

LIST OF TABLES

TABLE 1: GEOMETRIC DATA TYPES IMPLEMENTED IN POSTGIS 62
TABLE 2: OPERATORS IMPLEMENTED IN POSTGIS 62
TABLE 3: JAVA MODES 76
TABLE 4: NAVIGATION FUNCTIONS 77
TABLE 5: TOOLS 77
TABLE 6: LEGEND FUNCTIONS 79

ABBREVIATIONS

ADT	Abstract Data Type
API	Application Programming Interface
ASP	Active Server Pages
CGI	Common Gateway Interface
DBMS	Database Management System
DML	Data Manipulation Language
DOM	Document Object Model
ER	Entity-Relationship Model
GEOS	Geometry Engine – Open Source
GIS	Geographic Information Systems
GiST	Generalized Search Tree
GML	Geography Markup Language
GUI	Graphical User Interface
GVIS	Geographic Visualization
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity
JSP	Java Server Pages
JTS	Java Topology Suite
MBB	Minimum Bounding Box
OGC	Open GIS Consortium
PHP	PHP: HyperText Preprocessor
RDBMS	Relational Database Management System
SAM	Spatial Access Method
SNP	Swiss National Park
SQL	Standard Query Language
SVG	Scalable Vector Graphics
UMN	University of Minnesota
W3C	World Wide Web Consortium
WMS	Web Map Service
WWW	World Wide Web
XML	Extended Markup Language

ABSTRACT

While the usage of internet maps has grown in the last years, GIS and Cartography have evolved technically. Today, many software solutions for the publication of web maps are available. Some of them are commercial and proprietary; others are non-commercial and have been developed for only one specific purpose and data type. Furthermore, the Open Source community has provided many tools, enabling the setup of multi-component solutions following open standards.

Part I of this thesis includes the introduction and shows the current state of the art in Web Mapping. An own approach is worked out by introducing a new cubic model for the categorization of the types of web maps, followed by an analysis of deficiencies, the conclusions for this thesis and finally the research objectives.

Part II examines the theoretical background of WebGIS architecture and shows how GIS functionality can be implemented on both, client-side and server-side. Also the topic of data storage and Extended Relational Databases is covered intensively.

Part III describes the WebGIS SNP Prototype that has been implemented for the visualization of GIS data of the Swiss National Park. The resulting Web Map is interactive, usable intuitively and provides an adaptive interface. The basic functionality has been extended with a dynamically programmed Information and Query Page, enabling the user to perform attribute queries and visualizing the results in the map. Furthermore, specific aspects concerning the use of classic map elements in Web Maps have been addressed. The concept of Status and Visibility is newly introduced as a result. In the consequence, a dynamically programmed legend with dual functionality has been implemented.

The WebGIS SNP Prototype has shown that it is possible to set up a working interoperable multi-component application consisting of Open Source software products for the visualization of spatial data.

PART I: INTRODUCTION AND APPROACH

1 Introduction

While internet usage has grown from 61 million to 625 million internet users in the period of 1996 to 2001, the growth rate of internet map usage has been even higher [Pet01]. At the same time, Geographic Information Systems (GIS) have evolved from universal tools for specialists to systems that can be used with only marginal expert or programming knowledge [DZ99]. In the last few years, most commercial GIS products have been enhanced with map servers and now form the new category of Online GIS solutions. Furthermore, also stand-alone Map Servers that serve as a powerful tool for web mapping purposes are available.

1.1 Motivation

While more and more organizations offer more and more web maps of all kinds, only few national parks have done so yet. In a qualitative analysis conducted in March 2002, the World Wide Web (WWW) has been searched for the websites of national parks. There are countless national parks, and many of them do not have a website, especially the ones located in Africa, Asia and South America. In North America, the term “National Park” is used in a broader meaning and includes historic sites or recreational areas where fishing or hunting is allowed. For this reason, only European parks have been considered. Of the total 59 parks found, 16 websites were not loadable, mostly because of outdated URL’s. From the remaining 43, 5 consisted of text only, 9 included texts and images, 7 consisted of text and one or several static or clickable maps, and 22 had text, images and maps together. This shows that if National Parks are present on the WWW, they often use static view-only maps for promotion purposes. The maximum of interactivity in those maps is provided by clickable maps, where the result is a static map in a different scale.

It is therefore a challenge to visualize the GIS data of the Swiss National Park (SNP) interactively on the World Wide Web. In a survey among the users of the GIS of the Swiss National Park, over 40% requested the possibility to interactively generate maps of the GIS data, and 29% were interested in performing easy analyses online [SNP97]. Therefore, the setup of a web mapping application that can be used intuitively and that includes certain simple GIS functions was chosen as a case study for this diploma thesis. Considering the special character of the GIS-SNP [AB92] with two physical locations (Zernez and Zurich), the GIS data, the outdated available software and the financial aspects, a solution from the public domain (Open Source) would be desirable.

The main goal of this thesis project is to set up a working prototype of a web mapping application in order to visualize the GIS data of the Swiss National Park. The more detailed research objectives will be specified at the end of chapter 3 (section 3.3), in response to the state of the art (chapter 2), the analysis of deficiencies (section 3.1) and the conclusions drawn (section 3.3).

1.2 Thesis Organization

Part I of this thesis consists of three chapters. An introduction and the motivation for the thesis project are located in the first chapter, including the organization of this thesis. The second chapter describes the sate of the art; it gives a short historical overview of GIS, the internet and WebGIS, treats the topic of open systems and standards, presents approaches in Web Mapping as well as different kinds of categorizations and ends with some definitions. In the third chapter, the approach of this thesis is worked out. An analysis of deficiencies is performed; a new cubic model is introduced, requirements for Web Maps are considered, the use of commercial and open systems is discussed and research challenges are identified. This leads to conclusions and the setup of the research objectives for this thesis.

Part II is dedicated to WebGIS architecture, supplying the theoretical aspects that are important for understanding how WebGIS applications work and how GIS functionality can be provided over the internet as a platform. The implementation of client-side technology and GIS functionality is addressed in chapter four. Server-side technology and GIS functionality is covered in the fifth chapter. Storage and retrieval of spatial data in Extended Relational Database Management Systems is the topic of chapter six.

Part III is the practical part of the thesis. In chapter seven, the requirements for the prototype are specified, the used data are described and the software components of the prototype are listed, as well as the architectural overview, configuration and special programming issues. The prototype application and the implemented functionality are presented in chapter eight, where also a discussion and evaluation takes place. Finally, the conclusions of this thesis are the topic of chapter nine.

2 GIS, the Internet and WebGIS

2.1 Brief historical overview

GIS evolved since the 1960s, and four main phases of development can be identified: innovation (research and development by some few pioneers, 1960-1970), the start of the diffusion (mainly in research and administration agencies, 1970-1980), diffusion (adaptation by commercial users, 1980-1990) and the boom (maturity of the software, time of the users, 1990-today) [DZ99].

Paper maps as the final product of cartography used to work (and still do so) as a medium for storage and presentation of spatial data. These two functions were split with the introduction of on-screen maps and their corresponding databases, which led to new presentation options like 3D and animated maps [KO96].

Both, the development of GIS and computer cartography, have been parallel processes [Jon97], [DZ99]. Therefore, with the rise of the internet, the next step was to make the new computer technologies available in desktop GIS and computer cartography. Figure 1 shows a historical overview of the development in cartography and GIS, based on the sources cited above and [VBW+00].

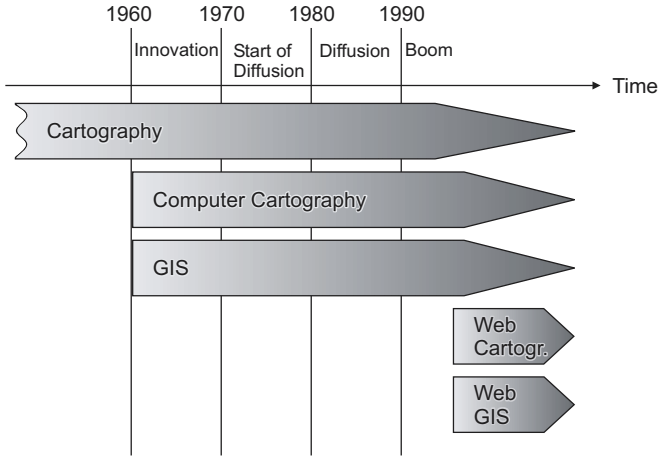


Figure 1: Historical Overview

Traditional Standalone GIS applications consist of one software package running on one machine, with geodata stored at the same machine (monolithic architecture, Figure 2a). Today, other characteristics are required, since many contributors and users may be involved. The main difference between a Standalone and an Online GIS is the separation of the user interface, processing and data storage. These elements are normally distributed over several machines. While the user interface (receiving information and displaying it for the user) is located at the web client (client-side), the processing (supplying the information) and the data storage take place at the web server (server-side). [GB01]

As illustrated in Figure 2b, a presentation tier, an application tier and a database tier can be identified [VBW+00], [Puc01], [Str01], [HSS01]. The standard 3-tier architecture is not only valid for Online GIS Systems, but is followed by most WebGIS applications.

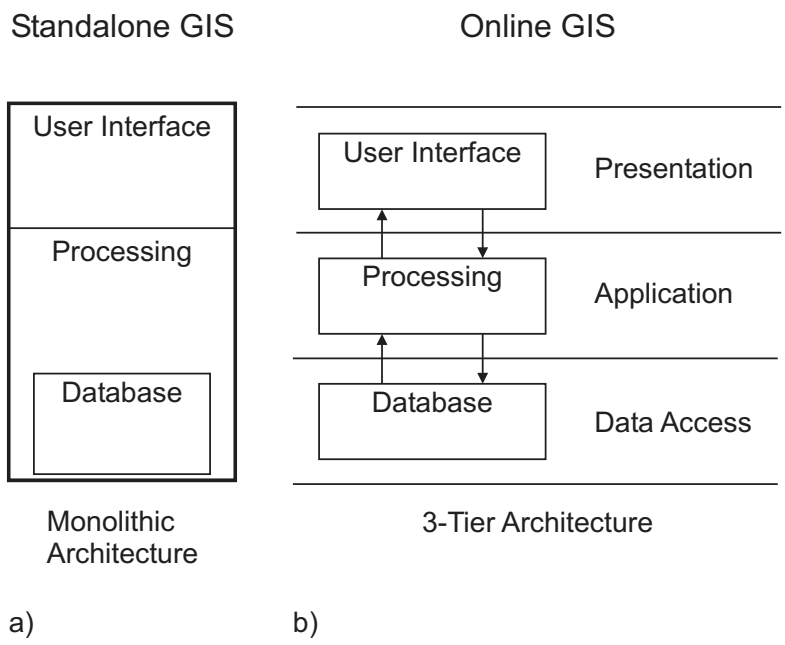


Figure 2: Monolithic vs. 3-Tier Architecture (partially after [Str01])

Looking at the history of the development of Online GIS applications, three phases can be distinguished [Str01]:

Generation 0: Standalone Implementations of server-side applications, mostly tailored for particular datasets and specifically developed for individual tasks.

Generation 1: Geoinformation-Web-Architectures, i.e. Map Servers on the server-side and Plug-Ins on the client-side, mostly based on closed architectures. Every Map Server requires a specific client.

Generation 2: Open systems that include interfaces based on standards guaranteeing integrated solutions.

The ongoing efforts to make spatial information accessible worldwide make clear that a dynamic process is changing the traditional understanding of GIS applications and services.

2.2 Interoperability and Standards

The Open GIS Consortium (OGC), currently consisting of more than 230 companies, government agencies and universities, has developed several publicly available specifications for the field of geoprocessing, such as the Geography Markup Language (GML) Implementation Specification, the Simple Features Specification for SQL or the Web Map Service (WMS) Implementation Specification [OGC02a]. An example of a WMS and GML based interoperable web mapping system has been described in [SVS+01].

2.2.1 WMS Implementation Specification

The specification is described by the OGC as following: “This OpenGIS® Standard specifies the behavior of a service that produces geo-referenced maps. This standard specifies operations to retrieve a description of the maps offered by a service instance, to retrieve a map, and query a server about features displayed on a

map" [OGC02b]. The specification shall support interoperable solutions and enable systems to access spatial information from different sources [OGC02a], [Str01].

Three operations are defined by the specification: *GetCapabilities* returns service-level metadata, *GetMap* obtains a map image and *GetFeatureInfo* (optional) returns information about particular features shown on a map. The first operation makes the creation of customized maps consisting of several map layers requested from distributed Map Servers possible. The aggregation of the data supplied by the different Map Servers into one service is handled by Cascading Map Servers.

The specification also has its limitations. The standard is only applicable to pictorial renderings of maps in a graphical format, and not to retrieval of feature or coverage data values. Besides that, the GetFeatureInfo operation, which enables a Web Map Server (WMS) to answer queries about feature information, is only optional.

2.2.2 GML Implementation Specification

"The Geography Markup Language (GML) is an XML encoding for the modeling, transport and storage of geographic information including both the spatial and non-spatial properties of geographic features" [OGC02c]. Implementers can either store geographic application schemas in GML, or use GML only for schema and data transport (by converting the data from another storage format on demand). The kinds of objects provided in version 3.0 for describing geography include features, coordinate reference systems, geometry, topology, time, units of measure and generalized values.

2.2.3 Simple Features Specification For SQL

The definition of a standard SQL schema that supports storage, retrieval, query and update of simple features is the purpose of this specification. Simple features have spatial (geometry valued) and non-spatial attributes. They are stored as rows in tables of a Relational Database Management System, called feature tables. Two target SQL environments are distinguished: SQL92 and SQL92 with Geometry

Types. Both Implementations extend the SQL92 information schema to support standard metadata queries returning the list of features in a database, the list of geometry columns for a feature table and the spatial reference system for a geometry column. [OGC99]

2.3 Approaches in Web Mapping

Two major approaches have – until now – mostly been considered for the creation of web based maps. One of them is the use of commercial software. Besides all advantages of a full featured software product, also several disadvantages must be considered. In most cases they concentrate on general applications and purposes, do not allow an optimization of the maps or explorative analyses and are quite expensive [CSW99], [Für01]. In addition, there is dependence from a commercial vendor, since proprietary solutions are bound to specific products or environments such as database servers or Internet Map Servers [CSW99].

The other approach focuses on the development of new, non-commercial web based visualization software [CSW99], [CSW00]. One example is the "CartoApplet" [Cec99], which has been designed at the Department of Geography at the University of Zurich for the cartographic visualization of statistical data on the internet. It has been extended through "EVisA" (Enhanced Visualization Application) [She00]. The goals of these projects were the development of a cartographic visualization tool for web clients with concentration on a specific data type (statistical data on areas), focusing on cartographic functionality, and the minimization of a dependence from commercial and non-standardized software. Such tailored visualization tools are suitable for specific applications that are mostly developed for a special audience. They are justified under the conditions that the functionality is designed for a particular purpose, the application concentrates on optimized cartographic functions, explorative functions are offered, the source code is open and the software avoids the use of proprietary components [CSW99], [CSW00]. The consequence of these conditions is that each application

with all its functionality has to be developed from scratch, especially because no components from commercial systems can be used.

2.5 Categorizations in Web Mapping

Many authors use different kinds of terms related to web mapping, which may lead to confusion. Therefore, this section gives a systematic overview and examines three different kinds of categorizations. But first of all it must be considered that web mapping always has something to do with *visualization*, which is a widely usable term. *Scientific visualization* has got the meaning of computer technology for making scientific data and concepts visible. As discussed in [Mac95], "the critical aspects of visualization as a concept are not even fixed for individual researchers". Thus, it is suggested regarding *geographic visualization* (GVIS) as a fuzzy category. According to [Mac94], the terms *cartographic visualization* and *GVIS* are both used for spatial visualization in the context of scientific research in earth sciences, in which maps are used as a primary tool. GVIS is hereby the better term to use, since it implies a broader spectrum of possibilities, for example the use of remotely sensed images together with maps.

As there are many approaches of how to handle the concept of GVIS, in the following subsections of this chapter three of them are used to build a chain consisting of map use (goals), mapping tools (applications) and map categories. However, there are no solid borders within the classifications, and especially placing applications into categories is difficult.

2.5.1 Types of web mapping goals

In a first step a model is needed which allows to separate different kinds of map use for the identification of the visualization approach that has to be chosen for the individual goal.

[Mac94], created a three-dimensional model of human – map interaction space. The so-called map-use cube (Figure 3) consists of three continua: "from map use that is private (tailored to an individual) to public (designed for a wide audience); map use

that is directed toward revealing unknowns (exploration) versus presenting knowns (presentation); and map use that has high interaction versus low interaction" [Mac95].

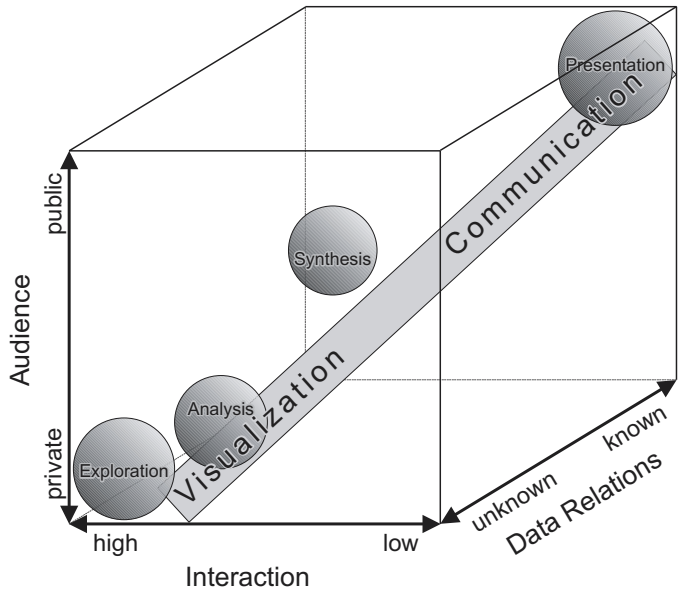


Figure 3: Map Use Cube (after [Mac94], [MK01])

As the model has evolved over the years, the three axes of the cube have been (re)named. *Audience* used to have no name in the original model and distinguishes between individual map use for own needs and previously prepared maps for a public audience. *Data relations* used to be unnamed as well and means whether a user is generally searching some information or accessing particular spatial information. *Interaction* used to be called "human – map interaction" and refers to the possibility for the user to change the map presentation.

In this space of map use, GVIS can be identified in the highly interactive, private and exploratory corner, while on the opposite *cartographic communication* is placed in the corner with the contrary characteristics. Starting in the former corner, exploration is referred to as a process, in which the use starts without much knowledge about the underlying data, but interactivity provides the tools for the

search of structures and trends. On the line to the opposite corner, also analysis and synthesis can be defined between the extremes. Presentation in a traditional sense for public use and based on known data is placed in the latter corner.

This model fits with the model for map-based scientific visualization defined in [DiB91], which makes a difference between *private visual thinking* and *public visual communication*: The process of scientific research starts with a few specialists exploring their data for finding answers to their questions. With the work progressing to a wider circle of peers, they are communicating ideas to others. This leads to a dissemination of the research from the private realm of scholarly inquiry to the public realm of scholarly and popular communication. The described exploratory, confirmatory, synthetic and presentational visual methods can be found again as spheres in the map-use cube (exploration, analysis, synthesis and presentation). Both models have in common that the number of representation options is reduced as the public end of the visualization continuum is approached [Mac94]. It is therefore important to see that maps are – in this context – more than making data visible. They are instruments in the scientific process of cognition that help to create new ideas [DiB91], [MK01].

2.5.2 Types of web mapping applications

In a second step, it is important to distinguish the types of applications that can be used for the chosen goal.

A common organization of application categories in Web GIS is a functional categorization [Gar99], [Für01]:

Geodata Server: Geodata from an archive can be accessed and searched. Download services for offline processing on the client side are provided (Figure 4, 1). An example can be found at the website of the United States Geological Survey (USGS), http://www.usgs.gov.

Map Server: Static or interactive maps are served after a request from the client. Map Server software is running on the server-side, often as a CGI program.

Static Map Server: Previously prepared raster maps (often exported from a Desktop GIS) are offered to the user (Figure 4, 2a).

Visualization Map Server: A new map is created with every request from the client, but the concentration of the functionality is on the cartographic visualization of geodata. No specific GIS functionality is offered (Figure 4, 2b).

Interactive Map Server: Similarly to Visualization Map Servers, maps are served over the internet. In this case, the user can change several parameters of the map. Attribute and/or spatial queries can be offered (Figure 4, 2c).

Map based Online Information Systems: Thematic or spatial queries are offered via text or interactive input through the map. The concentration lies on attribute queries from a database (Figure 4, 3). Examples are online route planners and traffic information systems, e.g. http://www.mapquest.com.

Online GIS: In the last years, all major vendors of commercial GIS software have released an Online GIS version of their products. Thus, most Desktop GIS functions are available over the internet [FRK+01] (Figure 4, 4). Many examples can be found at http://www.geoplace.com/gr/webmapping.

GIS Functions Server: Remote access to the functions of a GIS Server is offered. The request and the data have to be sent from the client and are processed on the server. The results (not necessarily a map) are sent back to the client (Figure 4, 5).

A web mapping cube similar to the one discussed previously has been proposed in [Her01]. The first axis is the *database connectivity*, which is represented by the two possible values *d* and *D*, depending on the presence of a database connection. The

second axis shows the degree of *interaction* with the values of *i* and *I,* depending on the amount of interaction possibilities. The third axis describes *visualization*, which is not used in the sense of GVIS but means how adequate (*v* or *V*) the visual communication of a theme is. Using this cube, the above categories of web mapping applications can be placed within the three-dimensional space (Figure 4).

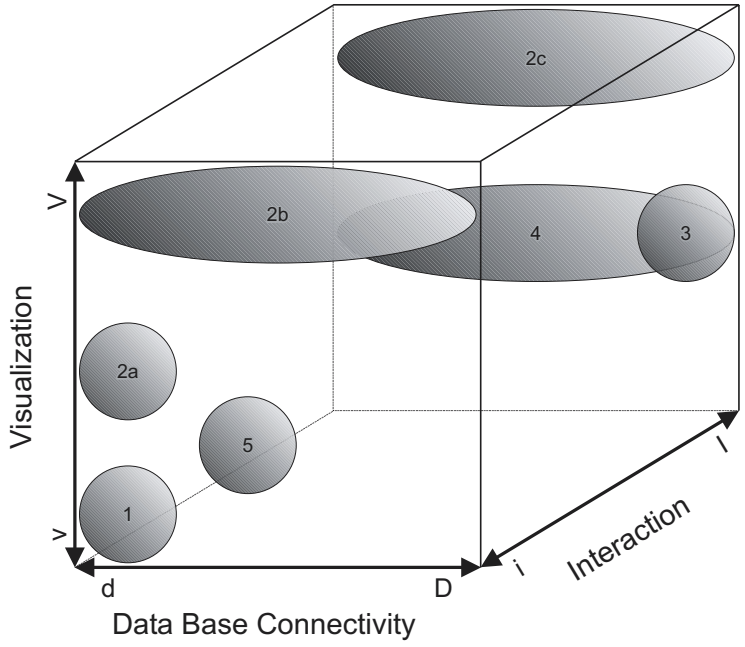


Figure 4: Types of Web Mapping Applications 1) Geodata Server, 2a) Static Map Server, 2b) Visualization Map Server, 2c) Interactive Map Server, 3) Map based Online Information Systems, 4) Online GIS, 5) GIS Functions Server (enhanced after [Her01])

Geodata Servers (1) provide no interaction, database connectivity or visualization. Static Map Servers (2a) are a little bit stronger on the visualization side, since they mostly provide raster maps that have been exported from a desktop GIS and deal with a certain topic. Visualization Map Servers (2b) are, as implied by their name, providing good visualization, but do not support much interaction to the user. They can be connected to a database, but not necessarily have to. Interactive Map Servers (2c) have similar characteristics, but are interactive. Map based Online Information Systems (3) supply much interaction and are connected to a database, but the

visualization is standardized. Online GIS (4) are interactive, concentrate more on data analysis than visualization and can be connected to a database. GIS Functions Servers (5) have similar characteristics like Geodata servers, except that there is more interaction due to the fact that parameters are submitted with a request and the according results are sent back.

2.5.3 Types of web maps

With the chosen goal and application, the visualization approach has also to consider which type of web map is the appropriate form of the cartographic product.

The types of web maps are discussed in [Für01], where the original classification of [Kra01a] was expanded. The latter is, in a first level, divided into *static* and *dynamic* maps. In a second level, each of these divisions is subdivided into *view only* and *interactive* maps. The expanded classification adds a new top layer, consisting of maps produced *before* or *after request*.

Maps that have been produced before user request have been finished by the cartographer first and then made available on the internet. On the contrary, maps that are produced after a user request will be created following the rules previously set by the cartographer. Static maps are unchangeable (frozen). Dynamic maps show a movement (animation). View only maps can not be changed in their presentation by the user, while interactive maps provide some kind of interaction possibilities to the user [Für01]. Interactivity is discussed heterogeneous in web mapping literature. For example, in [Dic01], static and view only maps are put into the same category. It is postulated that zoom and pan are user actions but no interactive elements, since they are the same as viewing a paper map from further or closer distance; the pixels are just enlarged and no additional information is provided, while interaction must make new information or map elements visible. According to [Ric98], the zoom function in static maps can at least make fonts more readable. The classification of [Kra01a] and [Für01] on the other hand implies

implicitly that static maps can be view only or interactive, since already the request of a map can be interpreted as an interactive element.

2.6 Definitions / Terms and Technologies

There exists an amount of terms related to web maps. But what is “web mapping”, and what is the difference between “Web GIS”, “Online GIS” and “Internet GIS”? Many authors use the latter terms synonymous, which is sometimes leading to confusions. In this section, some of the most important terms are listed with the attempt to explain for a better understanding and to define them at the least for the context of this thesis.

Web Map: Basically, every map available on the WWW can be called a web map. This includes the simplest version of a scanned paper map that is integrated in a website [Kra01a]. Web Maps can also include some classical GIS functionality (like querying attribute data by clicking the map), which expands the classical understanding of cartography as a process unrelated to data handling.

Web Mapping: The creation, distribution and use of web maps [Asc00] is called Web Mapping and implies the process of creation rather than the product [Dic01] or the application that has been used. Visualization aspects are the main subject of mapping [Dic01].

Web GIS: A Web GIS application includes some more GIS functions than a Web Map. An important difference is the access to attribute data. The user should be able to perform functions like attribute queries, search functions, area and distance measuring or the construction of buffer zones [Dic01].

Online GIS: An Online GIS application provides the functionality of a standalone Desktop GIS program, but online via the internet. The difference is that user interface, processing and data storage separated and mostly located on different servers.

Internet GIS: This can be regarded the top level category of all applications or systems providing GIS functionality over the internet, although most applications only use the WWW.

Map Server: The term can be understood from different points of view: If, in a network with several computers, one machine has the only purpose of hosting the Map Server program, this computer can be called a (hardware) Map Server. But mostly, the term is used for Map Server programs (software) that do the map creation and perform some server-side GIS functions. The basic functions of a Map Server are visualization, navigation and query [Für01]. Map Server Software can also be distinguished concerning economic and legal aspects: Commercial Map Servers are proprietary, while Open Source Map Servers are free of cost and their source code can be downloaded and modified. Depending on functional aspects, the three subcategories of Static, Interactive or Visualization Map Servers can be applied.

GIS Functions: Zoom and pan within a map are not considered true GIS functions [Dic01]. Some GIS functionality like data acquisition and data processing are not a particular issue in Web GIS, and visualization functionality (see cartographic functions) is treated here separately. Twenty universal GIS operations have been derived by [Alb96], organized in six categories: *Search* (interpolation, thematic search, spatial search, classification/reclassification), *Location Analysis* (buffer, corridor, overlay, Thiessen/Voronoi), *Terrain Analysis* (slope/aspect, catchment/basins, drainage/network, viewshed analysis), *Distribution/Neighborhood* (cost/diffusion/spread, proximity, nearest neighbor), *Spatial Analysis* (multivariate analysis, pattern/dispersion, centrality/connectedness, shape) and *Measurements*. [Sch02] also proposes a list of GIS analysis functions, consisting of: *Measure*, *Query* (identification, search by attribute, search by geometry), *Reclassification* (aggregation: dissolve, merge), *Overlay* (quantitative, qualitative), *Buffering*, *Neighborhood* (filtering,

cost/diffusion/spread), *Spatial Interpolation* (deterministic, stochastic), *Terrain Analysis* (slope, concavity, viewshed, watersheds, catchment basins), *Networks* (shortest path, location-allocation, trace lines), *Statistics* (univariate, bivariate, multivariate). However, not all GIS functions are suited for all kinds of data and projects.

Cartographic Functions: The most important cartographic functions are changes of the symbology and layer control. Also exploratory data analysis can be considered a cartographic function [Sch02].

3 Thesis Approach

3.1 Analysis of Deficiencies

The second chapter of this thesis has described the state of the art in Web Mapping. This chapter includes an analysis of the important points, shows where the gaps are and draws the conclusions.

3.1.1 A new Cubic Model

The categorization of web maps presented previously lacks a graphical visualization for better understanding. Thus, a new cubic model for the organization of the types of web maps is proposed here: The Cube of web map types (Figure 5).

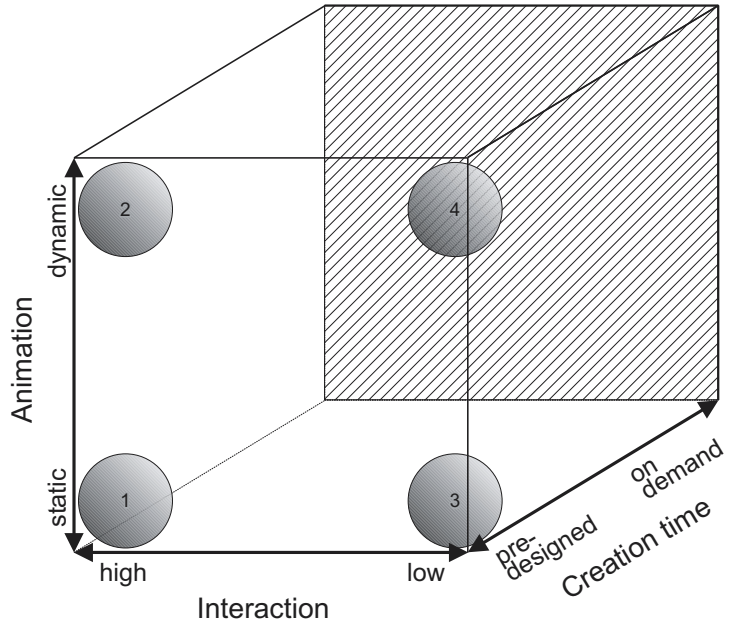


Figure 5: Cube of Web Map Types 1) Static Vector maps, 2) Dynamic vector maps, 3) Static raster maps, 4) Dynamic raster maps

The dimension of *animation* considers if a map has *static* or rather *dynamic* character. *Interaction* (manipulation possibilities for the user) can be *high* or *low*.

The *creation time* of a map is also included into the model, making a difference whether a map has been *pre-designed* before being accessible over the web or if the map creation happens *on demand* (upon a user request [Cec03]). Static vector maps (1) implicitly provide interactivity (e.g., adaptive zooming). Dynamic vector maps (2) are similar, but include some kind of animation (e.g., moving map elements). Static raster maps (3) are not interactive, since they are mostly a raster image of another cartographic product (e.g., a screenshot or scanned). Dynamic raster maps (4) include animation (e.g., movies).

However, all the categorization models presented have in common that there are "no clear boundaries" [Mac94] inside of the cubes, and that a classification can only be "a snapshot in time, (…) and certainly not carved in stone" [Kra01a]. This is because quantitative aspects are missing and every distinction has to be qualitative and finally subjective.

The only clear distinction that can be made in the above model is the one of the creation time. A map has to be created either before a user request, or after, but there is no possibility in between. On demand web mapping is a challenge, because maps are created automatically upon user request, meeting the user's preferences as well as the technical display specifications [Cec03]. This is why the most interesting space in the cube is on its back side (Figure 5, marked with a hatching). Dynamic maps on demand are generally a very new thing; most animated maps are produced before user request [Für01]. Static maps with low interaction on demand are seldom, e.g. maps that have been produced by a GIS Functions Server. Mostly, static maps on demand include interactive elements. This is the area where most Web Maps produced by Web GIS applications (e.g., Map Servers) are located.

3.1.2 Requirements for Web Maps

Since, with the introduction of computer cartography and databases, the classical dual map functionality of presenting and storing spatial data has been split [KO96], [MK01], the new requirements for maps are *presentation*, *exploration* and *communication*. This leads to the requirements for new mapping tools: Technical

aspects of map production are simplified through new user interfaces or even automated, which requires a new degree of *intelligence*. For getting an intuitive graphical user interface towards the data, also *interactivity* must be provided. Because maps are presented and explored over the WWW, and also communication about those maps happens over the internet, *internet-ability* is required, too. [SR01]

Internet-ability must be provided by using up to date techniques and programming languages. This also enables interactivity. Intelligence in user interfaces does not only mean that an application must be usable intuitively, but also that in the sense of dynamic programming. This means that the application reacts on changes in the underlying data or on inputs the user has given, and modifies the user interface accordingly. This behavior is seldom in Web Maps, since most developers tailor the application to the underlying data.

3.1.3 Commercial Software vs. Open Systems

As already mentioned, most vendors of commercial GIS software products have released their own Online GIS solution. It is not easy to get an overview over all the many products and their functions. One website listing some of the products currently available on the market is http://www.geoplace.com/gr/webmapping. While, at the time of beginning this thesis, all products offered basic functionality like pan, zoom, and attribute queries, more complex GIS analysis functions had to be programmed [LR00]. Additional GIS functionality that can be found in commercial Online GIS software is in most cases not necessary for Web Mapping purposes [Für01], or partially not suitable, e.g. for Atlas Information Systems with environmental vector data [Sch02], but also for related cartographic applications. The differences between the products are based on how much the cartographic developer can influence the design of the application [Für01]. Also the costs of several thousand US$ for each product and the dependence from a commercial vendor must be considered.

There is a trend in recent development from closed and proprietary systems towards open systems with standardized interfaces. Yesterday's monolithic GIS applications

with file based data storage already have become today's 3-tier client-server models with database connection. Future development will lead to distributed applications with open interfaces. Not only the data will be distributed, but also the software will consist of interoperable components. [Str01]

Commercial products do not only have the disadvantage of their high costs, but they also are inflexible. Their source code and thus their functionality can not be changed because of their terms of license. An adaptation of the software towards the user's needs is impossible. Technical support is available, but in the case of a malfunction in the software the source of error can not be tracked down by the user and the problem will – if ever – only eventually be solved in the next release. Open Source software does in the opposite give the opportunity to find and solve problems to expert users, who contribute to the community by submitting their patches for inclusion into the next release and by giving advice to other users through the mailing lists. Furthermore, when the work for this thesis project started, there was no up to date version of commercial Online GIS or Web Mapping software available at the Department of Geography of the University of Zurich.

3.1.4 Research Challenges

With the use of the WWW as a medium for the visualization of geographic data, new cartographic questions arise. While research was first primarily done about the role of the internet as a medium for the distribution of maps and the aspects of the use of Web Maps, it finally dealt with questions about the design and realization of cartographic web applications [Gar01]. Modern cartography deals with a complex process of geospatial information organization, access, display and use – thus problems in human-computer interaction and in enabling dynamic map and map object behaviors shift into the focus of cartographic design and research [MK01]. The ICA Commission on Visualization and Virtual Environments therefore identifies four research challenges: Representation, Visualization-Computation Integration, Interfaces and Cognitive/Usability Issues [MK01].

As far as it is known to the author, no research has been done yet about the classic map elements in interactive, static, on demand Web Maps. This opens two questions that can be identified as research challenges in modern Web Cartography: How are classic map elements used in interactive, static, on demand Web Maps? And what is their role in those Web Maps? This thesis project attempts to contribute to developing infrastructure enabling to answer these questions.

3.2 Conclusions

The conclusions from the state of the art and the above analysis of deficiencies are as follows:

- Dynamic on demand Web Mapping is a very new subject of research, but the available GIS data of the Swiss National Park do not enable or require such technology. The prototype application will thus take place in the interactive and static corner of the Cube of Web Mapping Types (Figure 6).

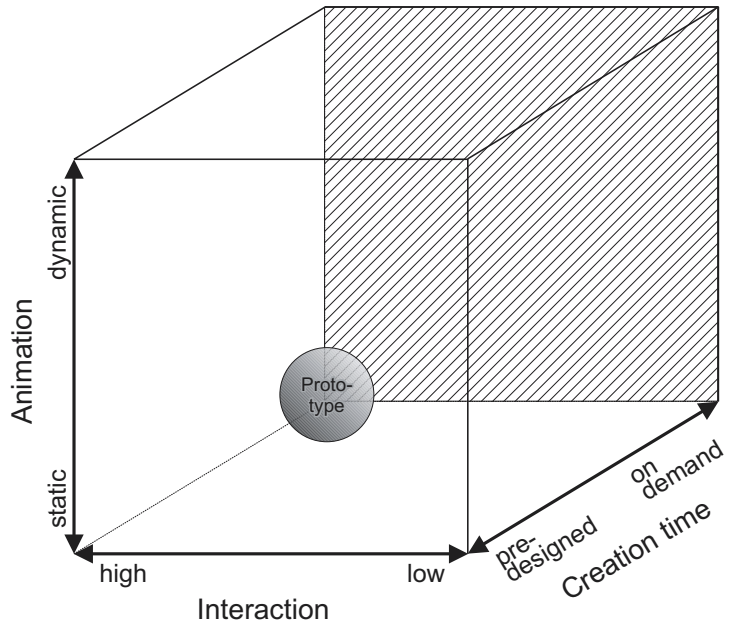


Figure 6: Location of the Prototype within the Cube of Web Map Types

- Dynamics can not only be provided in the cartographical sense of animation in maps. Dynamic programming works with the use of variables and general cases rather than explicitly naming everything. This enables the creation of intelligent user interfaces that automatically adapt on changes.
- The questions identified as research challenges can only be answered "by doing", i.e. while implementing the prototype. Only the usage of Open Source software gives the developer enough control over the application.
- Concerning the two standard approaches in Web Mapping (use of commercial software and development of own specialised applications), a different approach has to be chosen here: It makes most sense to use a product that has already been developed, but can be adjusted and extended according to the individual needs. For most commercial systems, this is impossible because of their terms of license. The logical consequence is to use software that is distributed under an Open Source license.
- The only software known to the author that meets the demands formulated in these conclusions and in the following research objective is the "MapServer" of the University of Minnesota. It works well together with the PostgreSQL database, which can be spatially enhanced with PostGIS. These are all products distributed under an Open Source license.

3.3 Research Objective

The GIS data of the Swiss National Park shall be visualized by an interoperable multi-component prototype application with open interfaces:

- The resulting Web Map shall have interactive character.
- Basic functionality (visualization, navigation and query) must be included.
- The user interface must be usable intuitively.

The chosen Map Server shall be checked for its functionality, which has to be expanded with additional functionality in order to fulfill the following requirements:

- Attribute queries must be possible.
- The results of attribute queries should be visualized in the map.

As a conclusion, the following conditions have to be met:

- The underlying data (geometry and attributes) have to be stored in a database.
- The user interface must be adaptive (the elements of the user interface adapt to the actual situation).

While the visualization of the data of the Swiss National Park is regarded as a case study, a use case shall be realized. It will be specified in section 7.1, where also the requirements for the prototype will be listed in more detail. Furthermore, the questions about the use and role of classic map elements in interactive, static, on demand Web Maps have to be addressed “by doing” while designing the graphical user interface.

PART II: WEBGIS ARCHITECTURE

4 Client-Side Technology and GIS Functionality

In Standalone systems, one monolithic GIS application performs all processes needed, and the entire user action is stored in the computer's memory. Because of the separation of the three tiers, two new problems appear: a) What kind of functionality should be processed where, and b) How can the user action be transferred from the user interface to the server.

4.1 Client-Side GIS Operations

Following the approach of providing GIS functionality over the internet as a platform, the browser's limited capabilities need to be extended. In the following, some possibilities of implementing GIS functionality on the client-side are presented [LR00], [GB01], [Köb01], [Wir01], [W3C03].

4.1.1 HTML

The Hypertext Markup Language (HTML) is used for describing the content of a webpage. Files written in HTML are requested from the server and transferred via the Hypertext Transfer Protocol (HTTP) to the client, where they are displayed by the web browser. HTML documents can include images and hyperlinks, which make jumping to other documents possible. One simple possibility to implement geographic queries is the use of image maps. Images included in HTML documents can not only be just displayed or work as hyperlinks to other documents. They can also have sensitive areas defined by coordinates of the areas' bounding polygons. Like this, one image can provide several hyperlinks. Image maps are also called clickable maps.

HTML is, although specified by the World Wide Web Consortium (W3C), not browser independent, because the different suppliers of browser software often

make small changes of additions to the standard. This can lead to problems in correct data handling with some browsers, which is a serious problem to web publishers. Dynamic HTML is the newest development but only partly specified and supported only by the newest browsers. It allows changing elements of a web document during the presentation, and is not really an extension of HTML but a combination of new HTML tags and options, style sheets and programming.

4.1.2 JavaScript

JavaScript is an interpreted scripting language and is designed for performing various functions in web documents, such as providing interactivity, creating HTML content dynamically and controlling the browser's behavior. The code is embedded into HTML documents and interpreted by the browser at runtime (client-side JavaScript). JavaScript is often used to validate data entered into HTML forms before submission or to make static HTML documents interactive by using event handlers that perform an action at a certain event. Client-side JavaScript also plays an important role in Dynamic HTML programming.

4.1.3 Plug-Ins

Plug-Ins are helper applications that extend the functionality of web browsers. For example, if vector formats like the Scalable Vector Graphics (SVG) are used, the appropriate plug-in will take over the interpretation of the data. Plug-Ins have to be downloaded first and are then installed on the user's computer. They are specific concerning platforms, operating systems and browsers, but have the advantage of being widely available, mostly free of cost and accessing local data. The disadvantage is the need to download and install the plug-ins the first time a new format appears, since otherwise the latter can not be interpreted. Plug-ins also need to be updated if newer versions are available. Besides that, they can be a potential security problem, because of their location on the local hard disk and access to the system resources of the client.

4.1.4 Java Applets

Java is an object-oriented programming language. Applets written in Java are embedded in a HTML document and interpreted by virtual machines implemented into web browsers. The Java Platform provides a predefined set of classes, which are available for all Java programs and makes this technology independent from the underlying operating system. Java applets can only access the server from which they were downloaded automatically and are therefore secure. Via the Java Database Connectivity (JDBC), Java can be used to access databases. Though Java Applets are considered portable across platforms, the virtual machines in web browsers are sometimes malfunctioning.

4.1.5 SVG

Scalable Vector Graphics (SVG) is a new language for describing two-dimensional graphics in the Extensible Markup Language (XML). Both XML and SVG have been specified by the W3C and are therefore non-proprietary formats. More exactly, XML is providing its Document Object Model (DOM), and SVG is what could be called an XML dialect. At the moment, the use of SVG still requires a Plug-In for the web browser. The advantage of SVG is that the processing (map calculation) takes place at the client. But this requires all geometric data being sent in readable format (XML) over the internet to the client first, which makes the data potentially readable for everyone. Nevertheless, SVG is a promising new standard. Flash as an alternative vector format is, on the opposite, proprietary.

4.2 Client-Side Context Preservation

A problem to web developers in general, but to developers of WebGIS applications in special, is keeping track of the user's action. This is caused by the interaction via the HTTP, which is connectionless (client and server are connected only for the duration of the transaction) and stateless (the server forgets about the client after the transaction) [VBW+00], and the separation of the three tiers mentioned previously.

In multi-user systems, several requests from a user may arrive on the server one after another, but also other users may send requests at the same time. Thus, an incoming request must be related to a former for keeping up the context of the session with a particular user.

Unlike in standalone systems, where all user action is stored in the computers memory while the GIS application is running, in WebGIS applications this information needs to be transferred from the user interface to the processing application.

4.2.1 HTML Forms / HTTP-Request and -Response

Context can be preserved in the easiest way with HTML forms. With every submission of a form, the respective variables (key-value pairs) are transferred as a HTTP-Request to the server, where they are processed and a HTTP-Response is routed back to the browser by the web server. Also images can be defined as form fields and cause the variables to be submitted after a user's click on the image. The image coordinates of the mouse pointer are submitted with the rest of the form variables, which is how many interactive maps work. The image coordinates need to be converted into geographic coordinates by the processing application. The connectionless and stateless nature of the HTTP makes the server forget about a request that has been processed, thus the variables must be re-submitted with every new request.

4.2.2 Cookies

JavaScript can also be used to create Cookies, HTML headers that pass data between server and client. Since HTML is memory-less, the use of Cookies is another strategy that is often used for preserving the history of the user's actions, for example in online shops. The variables and their values are stored on the user's hard disk. The fact that users can disable the use of Cookies in the browser's preferences makes Cookies unreliable.

5 Server-Side Technology and GIS Functionality

On the server-side, there exist different possibilities of implementing GIS functionality. Figure 7 shows the classic 3-tier internet architecture. A host computer connected to the internet has a web server program running and waiting for requests of websites from a client. After processing, the server sends the resulting data back to the client via the HTTP. Hosts are often also called servers because they mostly have the main purpose of running the server program. Since delivering web pages is the web server's main role, server-side GIS functionality needs to be processed by utility programs linked to the web server. The applications linked to the web server can have access to attribute or spatial data stored either in files or in a database (DB) via a Database Management System (DBMS).

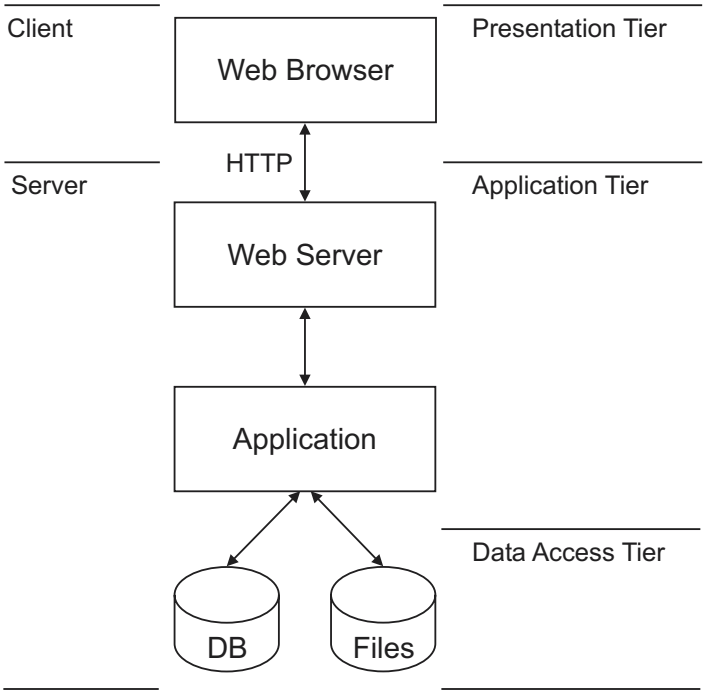


Figure 7: Standard 3-tier Architecture

5.1 Server-Side GIS Operations

Also on the server-side, there is more than one possibility to add GIS functionality. However, most WebGIS applications use the Common Gateway Interface (CGI). The different methods are examined in the following in more detail [LR00], [GB01], [Köb01], [Wir01].

5.1.1 Common Gateway Interface

Referring to Figure 7, the application processing the spatial data is in most cases a CGI program. It can access HTTP-Request parameters (key-value pairs for each variable). The CGI program decodes the incoming data and does the requested processing. The output is sent back to the client as a HTTP-Response by the web server. CGI programs can be used to provide database access or can work as map servers producing interactive maps. The advantages of the use of CGI programs are independence from a programming language (although PERL is used by most programmers) and the separation from the web server, which makes erroneous CGI programs unable to influence the web server. The disadvantage is that processes are forked, database connections have to be closed and re-established for each request, and much time is spent to set up the environment for CGI scripts before requests are processed. This leads to a decreasing performance with an increasing number of parallel users.

5.1.2 Server Plug-Ins

Server Plug-Ins provide an alternative to CGI programs. They are binary extensions for web servers, statically linked or dynamically loaded into them on demand, and enable the parsing of scripting languages on the server-side. Scripts that are intermixed with the HTML document are parsed on the fly when the document is requested from the web server. Optionally, whole files with functions coded in a scripting language and stored on the server can be included. Precompiled documents can even provide a better performance. This is a powerful method for creating dynamic web content. Server Plug-Ins can also work as an interface

towards a DBMS. Popular examples of Server Plug-Ins are PHP (acronym for "PHP: HyperText Preprocessor"), PERL, Active Server Pages (ASP). They have the advantage of a performance gain compared to CGI programs, since no new process has to be started with every request. Also, they are universally compatible with any client, because only HTML is sent back from the web server. However, they are server-platform-specific and thus vendor-specific.

5.1.3 Other Possibilities

Other methods of implementing GIS functions on the server-side are the web server Application Programming Interface (API), which allows the extension of the web server functionality, but also Java Server Pages (JSP) and Java Servlets. Java Servlets also answer HTTP-Requests, but with improved response times, scalability and session management. They do not terminate and thus enable persistent database connections. Java Server Pages go one step further and also separate presentation-logic and application-logic; JSP are used for presentation and maintained by the webmaster, while Java Servlets are maintained by the developer.

5.2 Server-Side Context Preservation

There is also a server-side alternative to the use of cookies, which are unreliable because they can be disabled by the user. When using PHP, session control can be maintained by a cryptographic random number that is generated by PHP and used as a session ID (SID). It can be stored as a cookie on the client-side, or be added to the URL. The SID acts as a key that allows registering session variables which are stored in files on the server. The advantage is that the application also works for users who have disabled cookies in their browser settings. The files can be stored in a temporary directory on the server and can be deleted periodically by an automatic mechanism (a cron job on UNIX platforms, for instance). Also Java Servlets keep the context for requests and sessions.

5.3 Load Balancing

Depending on the place where most of the (geospatial) processing is done, a difference can be made between server-centric and client-centric paradigms [VBW+00]. The connectionless and stateless nature of the HTTP initially caused the thin-client/thick-server approach. The advantage is that there is no need for additional plug-ins and the platform independence is preserved [Köb01], at least at the client-side. The application is under almost complete control of the developer, but has the main disadvantage of a performance problem arising with an increasing number of requests. Today, with increasing available bandwidth and the rise of markup languages such as XML, GML or SVG, the development seems to tend towards thick-client/thin-server systems, where Plug-ins and Applets can be used to solve the performance problem on server-centric systems.

5.4 Data Storage

Data used in a GIS can be stored in different ways. Early GIS software packages used to be built on top of proprietary file systems (Figure 8a). Both geospatial and attribute data are stored in files controlled by the application, and the functions are defined on this data. This way of data handling is violating the principle of *data independence*, which postulates that the user should only interact with a representation of the data independently from the physical storage. Thus several problems arise, such as data security (user access control) and concurrency control (synchronization of concurring access to data from several users). [HSS01], [RSV02]

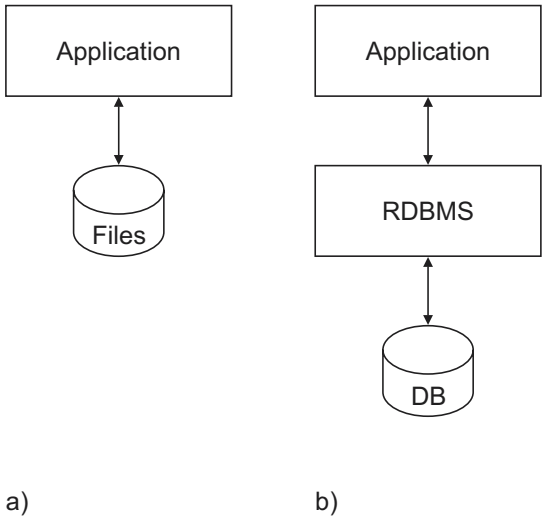


Figure 8: Data Storage Architecture a) Proprietary file systems, b) Relational Database Management Systems (RDBMS) (after [RSV02])

Another option for data storage and management could be the use of a Relational Database Management System (RDBMS), as shown in Figure 8b. The advantages would be a representation of the features by relations and Structured Query Language (SQL) based querying. But the relational model is not powerful enough for handling the structural complexities of spatial data. The independence principle is violated and the performance is poor. [SS98], [RSV02]

Because the data storage architectures mentioned above are unsuitable for GIS purposes, other approaches have to be considered: The Loosely Coupled Approach and the use of Extended or Object-Oriented RDBMS [SS98], [RSV02].

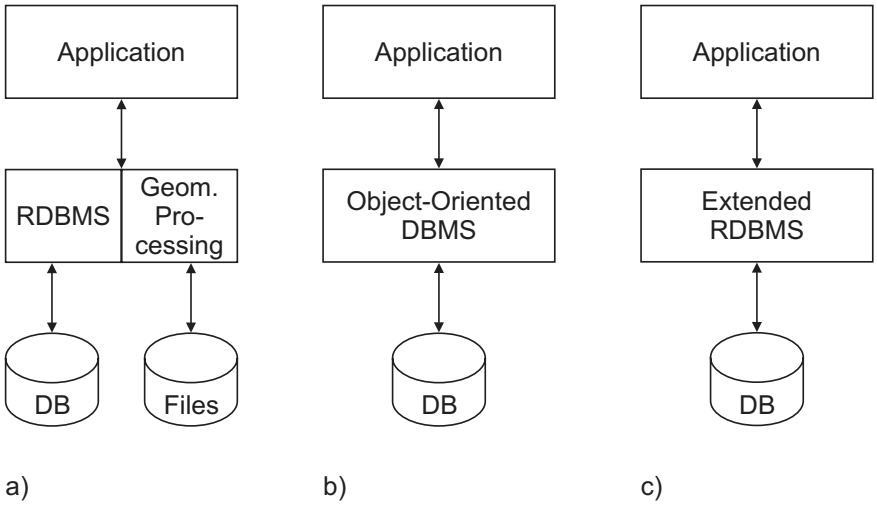


Figure 9: Alternative Data Storage Architectures a) Loosely coupled, b) Object-Oriented Database Management System, c) Extended Relational Database Management System, (after [RSV02])

In the Loosely Coupled Approach, the management of attribute and spatial data is separated (Figure 9a). While attribute data can be handled by a RDBMS, spatial data are handled by a separate subsystem that cares about geometric processing. This strategy is followed by most of the commercial GIS software packages, but also suffers from several disadvantages, namely the coexistence of heterogeneous data models, a partial loss of basic DBMS functionality (recovery, querying, and optimization) and the need to do the processing in the application layer.

For some years, Object-Oriented DBMS (OODBMS) seemed promising (Figure 9b). In this approach, object classes are defined which are collections of objects of the same abstract data type (ADT). Following the concept of encapsulation, these objects can only be accessed through the operators attached to their class. Classes can inherit their operators (methods) and attributes to subclasses. The object-oriented model also implies object identity, complex object support and user defined data types. In the mean time, recent development seems to have directed

the attention towards object-oriented programming languages and Extended RDBMS. [Gün98], [HSS01]

Extended RDBMS (Figure 9c) unify the modeling concepts of Object-Oriented DBMS and the advanced technology of RDBMS. This is why they are also called Object-Relational DBMS (ORDBMS). New types and operations are added to RDBMS. The extension of SQL enables the manipulation of spatial data and provides new spatial types (point, line, and region). Besides this, other DBMS functions are optimized for efficient spatial data handling. [SS98], [HSS01], [RSV02]

6 Extended Relational Database Management Systems

This chapter introduces issues relevant to spatial databases. It is of special interest for this thesis project, because not only attributes, but also the geometries of some themes shall be stored in a database. Thus, this chapter is important for understanding how spatial data can be stored in a database and – even more important – how it can be retrieved efficiently.

The first section explains why RDBMS must be extended for the use in GIS applications and what the requirements are. In the second section, the representation of spatial data and the logical models in a spatial database are examined. It is mainly based on [RSV02]. The third section gives an overview of access methods. For a better understanding of the spatial access methods, which are needed in order to find the results of spatial queries efficiently, traditional access methods are introduced in a first subsection.

6.1 From RDBMS to Extended RDBMS

Relational Database Management Systems are based on a data model that provides the infrastructure needed for the generation of a computer based model of the real world. In order to be able to model the relevant part of the real world in a database, a semantic model is needed. Among the semantic models, the Entity-Relationship-Model (ER) is the most common in modern RDBMS. It consists of Entities (different concepts of the world that shall be modeled) and Relationships between those entities. Attributes are used to characterize the entities and – in some dialects of the ER model – relationships. Keys are the minimal amount of attributes needed for an unambiguous characterization of an entity amongst all entities of the same type. On the conceptual level, the Relational Model is the most commonly used. Relations are sets of Tuples, which are a collection of (atomic) attribute values.

They can be represented as Tables, with rows corresponding to data objects of the real world and with columns stating their attributes. [KE01], [RP02]

In the vocabulary of geospatial modeling, entities are *geographic objects* (also called features) which consist of a description and a spatial component. The description is a set of descriptive (alphanumeric) attributes. The spatial component itself can contain geometry and topology and does not correspond to any standard data type available in RDBMS. Therefore, their representation requires powerful modeling. This is leading to spatial data models, amongst which three basic data types are used: *points* (zero-dimensional objects), *lines* (one-dimensional objects) and *regions* (two-dimensional objects). [RSV02]

6.1.1 Requirements of Extended RDBMS

Generally speaking, data input, storage, retrieval, analysis, display and selection need to be possibly performed on both spatial and attribute data. Therefore, in a spatial DBMS the representation and manipulation of geometric information with traditional data should be integrated (at the logical level), and data storage and processing need to be efficient (at the physical level). The logical data representation should thus be extended to geometric data.

6.2 Topology, Representation and Logical Models

Collections of *spatial objects* (isolated spatial components of geographic objects) can be represented in a database in different ways. In this context, the relationships among these objects become interesting. The first subsection therefore introduces the topological relationships among spatial objects. In the second subsection, the three most commonly used representation models for collections of spatial models are presented, which mainly differ in the way how topological relationships are expressed. The third subsection looks at how the representation of geographic objects can be supported, using Spatial Abstract Data Types to extend the relational data model.

6.2.1 Topology

Topological relationships between objects are orientation-independent (invariant under topological transformations like translation, rotation or scaling in the Euclidean plane) [RSV02]. They differ in terminology, but generally include equivalence, partial equivalence, containment, adjacency and separateness [Jon97]. In [Ege96], eight relations between two regions in $\mathbb{R}^2$ have been identified (Figure 10):

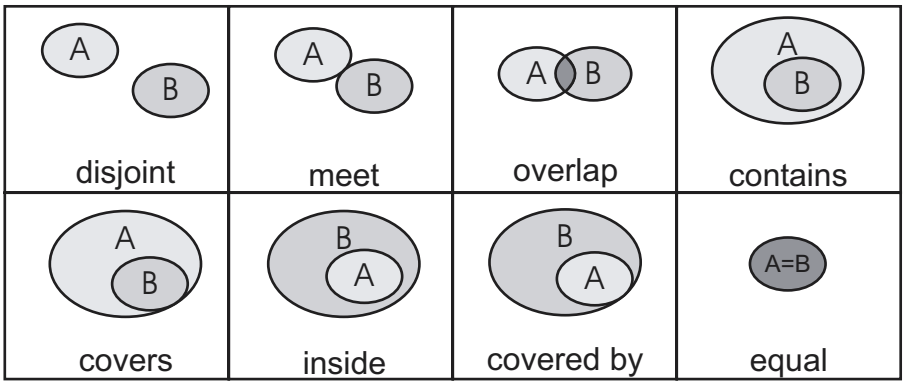


Figure 10: Topological Relations between two Regions (after [Ege96])

6.2.2 Geometric Representation of Spatial Objects

As already mentioned, there are three spatial data models (Figure 11) for representing the geometry of spatial objects, but also of the topological relationships among the objects included in such a collection.

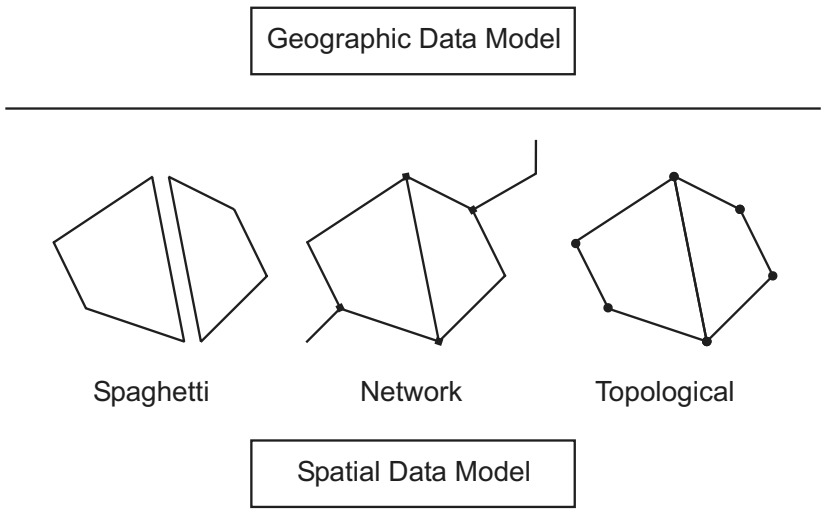


Figure 11: Spatial Data Models

Spaghetti Model

This Model describes the geometry of every object independently, without storing any topological information. Its simplicity, the easy input of new objects and the possibility to represent heterogeneous data types are important advantages. But there are also main disadvantages like the lack of topological information which therefore has to be computed on demand. And due to the independent representation of each object, redundancy is implied. This means that for example common boundaries of polygons are stored twice, which can decrease the performance of queries in large databases or lead to inconsistency. Figure 12a shows the example of two polygons (*P*) represented in the Spaghetti Model. They can be described by an ordered list of coordinate pairs, such as:

- P1: < [1, 4], [2, 2], [4, 1], [3, 6] >
- P2: < [4, 1], [6, 3], [5, 5], [3, 6] >

The Spaghetti Model is implemented in the Simple Features Specification of the OGC.

Network Model

Network representation is more complicated and additionally requires the concepts of nodes (points connecting arcs) and arcs (polylines starting and ending at a node). So the relevant data types are regular points, nodes, arcs, polygons and regions. In *planar networks*, all intersections of edges are nodes; while in *nonplanar networks*, intersections are not necessarily created with every crossing. Nodes are stored with their coordinates and a list of arcs they connect, arcs with their starting and ending node and a list of regular points they include. This model provides a description of networked topology, enabling optimal-path searches and connectivity tests. Topological information about polygons (closed polylines) and regions (sets of polygons) is missing. Figure 12b illustrates the representation of nodes (n) and arcs (a) in the Network Model:

- n_1: [[2,2], < a_1, a_2, a_3 >]
- a_3: [n_1, n_3 < [1,4] >]

Topological Model

This approach is similar to the network model and featuring the same types, but is always describing a planar network. There exist adjacent polygons, since arcs also include the two polygons sharing it as a common boundary. Because polygons are represented by a list of arcs, and arcs are including information about the polygons they belong to, a certain redundancy is implied. However, the geometry of the objects is stored only once. This concept of sharing objects solves the problem of possible inconsistency, making maintenance and updates easier. The computation of topological queries is more efficient. However, some operations like line scanning are much slower than in the spaghetti model because of the complexity of the structure. And pre-computation of a part of the planar graph is necessary for the insertion of new objects. Figure 12c presents the representation of polygons and arcs in the Topological Model:

- P_1: [< a, b, c, g >]
- g: [n_1, n_2, P_1, P_2, < >]

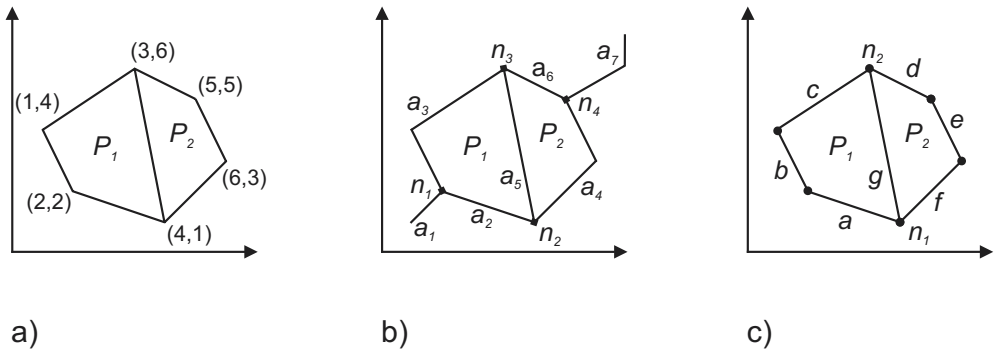


Figure 12: Geometric Representation a) Spaghetti Model, b) Network Model, c) Topological Model

The main difference between the Spaghetti Model and the other two is that the former is missing the storage of topological information basing on the identity of parts of objects (e.g., arcs or nodes), while the latter two do not. However, because of the storage of the coordinates of each point, topology is included also in the Spaghetti Model, but just implicitly.

6.2.3 Spatial Abstract Data Types

As already mentioned, new spatial data types are required for a representation of geometric objects. But not only their structure, also the operations performed on those types need to be defined. This is called the concept of *Abstract Data Types* (ADT). ADT are encapsulated, i.e. they are only accessible through the operations defined on them. More exactly, on objects of a certain type, a set of operations is defined. These operations are serving as an interface towards the user, to whom the structure of the data type is hidden. Therefore, a query language can be extended with geometric functionality, independently of a specific representation or implementation.

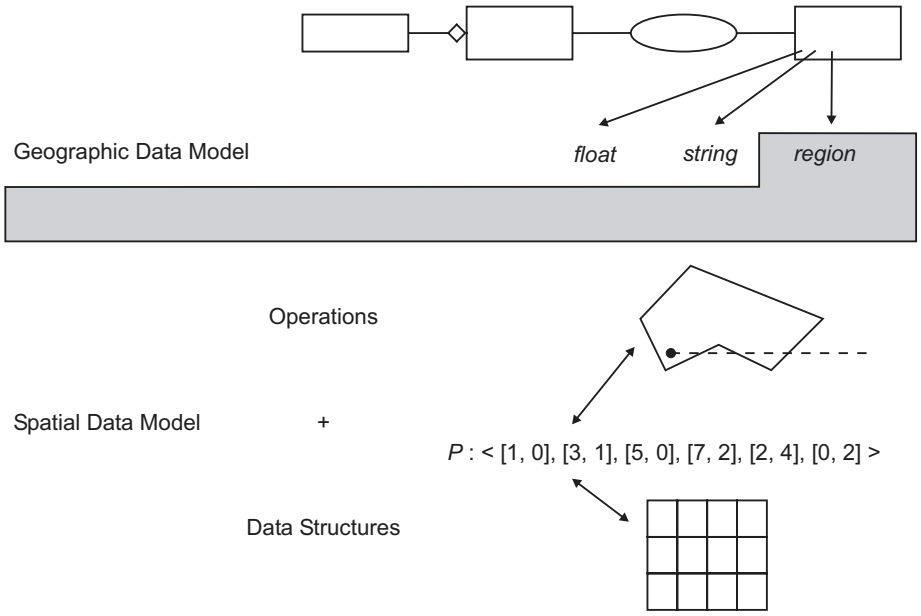


Figure 13: Spatial and Geographic Data Model (after [RSV02])

The concept of ADT is related to the one separating spatial objects from geometric objects, as already mentioned earlier. As can be seen in Figure 13, a lower level is built by the *spatial data model*, providing data structures and operations on them. At an upper level, the *geographic data model* takes place. Here, the new types such as *region*, *line* or *point* appear next to all other traditional types like *float* or *string*. A geographic object can now be described with values of the system-own types, extended with spatial data types. Any spatial data model described in the previous section may be used to actually store the spatial objects, as long as they provide the same operations.

The implementation of the spatial operations depends on the representation of the spatial objects. It must also be kept in mind that the result of every operation must be either an atomic type of the DBMS (e.g. *real* or *integer*), or one of the abstract types. This means that, for instance, a result of a query can not contain points, lines and polygons, if they can not be represented within one type. The reason for this is the functional view of queries as a composition of operations. Another thing to

think of is the semantics of operations. Depending on the dimension, an operation can have different meanings. For example, the intersection of two lines can result in a segment (or a set of segments) the lines have in common, or it can be the set of points of intersection.

6.3 Data Access Methods

Extended RDBMS do not only need new data types and operations. In order to process spatial data efficiently, also new Spatial Access Methods (SAM) are required. While RDBMS use classical indexes for the retrieval of alphanumeric data, Extended RDBMS provide different access methods for spatial data, in which objects are selected location-based or phenomenon-based (or both in combination) [Jon97]. In the following, the first subsection introduces traditional data access methods. Its purpose is to provide the theoretical basics for a better understanding of the second subsection, where some of the numerous kinds of SAM are described.

6.3.1 Traditional Access Methods

The simplest traditional method to access an entry in a file would be the *sequential* access, in which every record in a file is read until the one of interest is found. This method may have a poor performance, since if, in the worst case, the record of interest is the last entry, all previous entries have to be accessed first. More direct access can be reached if either the address (an associated serial number) of the record is known or if a constituent *key* field has been defined, whose content enables the DBMS to find the according record directly. The *primary key* is the field which is used the most for identifying a record, but also *secondary keys* can be used. Of course, the values of these key fields must be unique. The access methods most commonly used in RDBMS are listed in this subsection, based on [Jon97]. The examples deal with a list of names that has been chosen for illustration.

Binary search

The records within a file are sorted according to the contents of the primary key field. Given a value to search for (*search key*), this method starts with the record

placed in the middle of the file and compares the two keys (Figure 14, step 1). If the values are not equal, the method jumps to the middle record of the upper or lower half part of records (step 2), depending on whether the search key is bigger or smaller than the primary key value just checked. Like this, the amount of records to search is always halved until the record of interest is finally found (step 3).

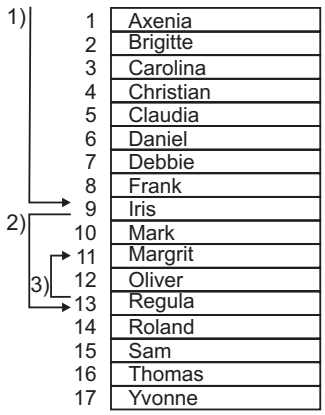


Figure 14: Binary Search Method

Hash addressing

Hashing algorithms are used to transform key field values into record numbers. This is done for all records and results in the records being allocated to a unique record number. If searching for a specific record, the search key value is transformed into the record number where the data are stored, and the record can be accessed directly by its address.

Indexing

Another method is the introduction of an index, ordered by the values of the key field. In *dense indexes* (Figure 15), there is an entry consisting of the key field and the record number of the corresponding record in the main file, which is also sorted by the values of the primary key. The record number is also called a *pointer*, because it points to the record in the main file. Indexes are smaller than the main files and can therefore be read much faster. But in large files this can still take too

long, which is why *sparse indexes* only register a part of the key fields (Figure 16). This requires a sequential reading of the index until a key field value is found, which is greater than or equal to the one of interest. Like this, direct access is provided to the address of a record within the main file, from which a short sequential search has to be done until the target is found.

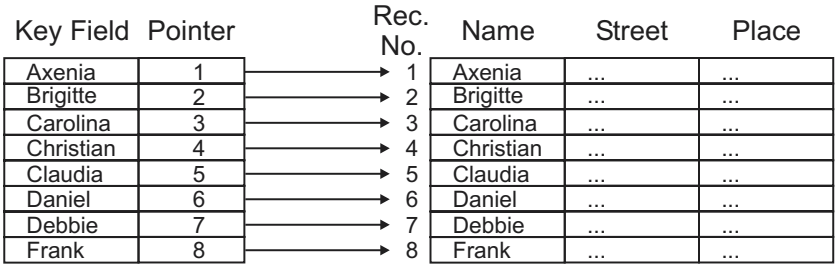


Figure 15: Indexing with a Dense Index (after [Jon97])

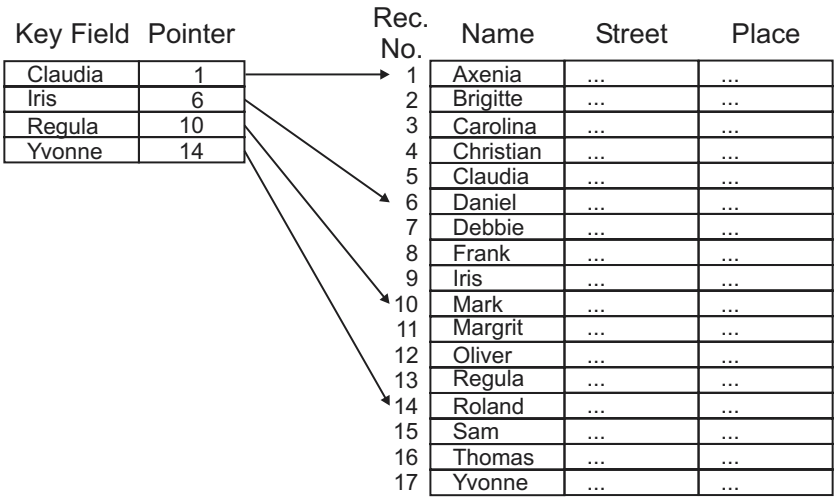


Figure 16: Indexing with a Sparse Index (after [Jon97])

If a sparse index becomes too long, a new index can be laid on top of the first index, leading to a hierarchical structure. It is also possible to have more than one index on the same file, with a *primary index* being based on the primary key, and a *secondary index* on a secondary key field. If some records share a common

property in the secondary key field, it is furthermore possible to index just one record per property, if there is a pointer in the main file addressing the next record with the same property. The records chained together are called *linked lists* and are also used to access records in a specific order. This approach enables easy updates of records, which are inserted at the correct logical place, without having to be stored physically near their logical neighbors. Thus, this method is also used to chain records ordered by the primary key.

Trees

Also with linked lists, it can possibly take a long time to read through all records sharing a common property, if there is a large amount of data to handle. For this reason, tree structures have been introduced to provide an alternative way, depending on the object of the search. In a *binary tree*, each record contains – besides data in a key field – two pointers to other records whose items are respectively higher and lower to the sequence (Figure 17).

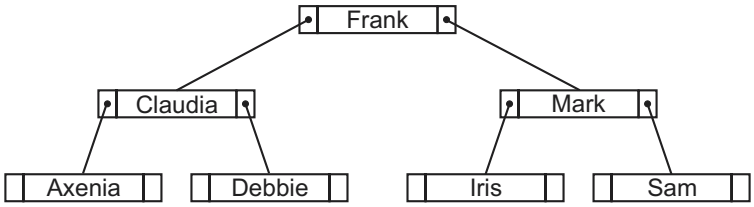


Figure 17: Binary tree (after [Jon97])

Starting at the root record, the search is descending along the tree structure, turning left or right at each record (also called a *node*), until the record of interest is reached. The fact that the number of records to read is proportional to the logarithm to the base 2 of the total number or records makes this method very useful. In order to create complete database index mechanisms with nodes storing key fields and pointers to records of corresponding database files, the concept has been extended. The resulting *B-tree* is the classic structure for indexing in RDBMS. Its nodes can contain a variable number of pointers, which makes the tree *balanced*. This means

that the height of the tree (number of levels) stays the same for all leaf nodes when inserting or deleting data, hence the tree structure is adaptable. The height of a tree affects the number of pointers that need to be followed and is thus the main factor influencing performance.

There exist several variations of the B-tree. The *B*-tree*, for instance, is a broader tree that reaches a smaller height due to increased branching [HR01]. Another example is the B^+*-tree*, in which the nodes contain only separators, while the leaves store all keys and are sequentially chained [ES00].

6.3.2 Spatial Access Methods

B-trees and hashing are designed for alphanumeric data without spatial properties. Thus new access methods have to be designed that use spatial properties for accessing spatial data efficiently [AG97]. While B-tree properties rely on a total order of key values, geometric objects are better ordered preserving object proximity, i.e. objects close in the 2D plane should be close in the index [RSV02].

There exist countless methods and algorithms for spatial data access, and authors classify them according to different criteria. Some of the most important SAM have their origin in one of the traditional indexing methods [AG97] discussed in the previous section. However, two basic approaches can be identified in spatial indexing [Jon97], [RSV02]. The first of them is based on regular partitioning of the data space, more or less independently of the distribution of the objects in the 2D plane. This may lead to a subdivision of an object's geometry into several cells. Examples of access methods following this approach are *Grids*, *Quadtrees* and *Grid Files*. On the opposite, the second approach adapts to the distribution of the objects in space and uses different strategies. The concept of *divide and conquer* is used to subdivide the space into half-spaces by selecting point or linear objects, as for example in the *k-d-tree*, a multidimensional binary-search tree [Jon97], [AG97]. Another strategy is the use of *minimum bounding boxes* (MBB) stored in nodes of a search tree, such as the *R-tree* and its variants.

Grids

A two-dimensional regular grid structure (*fixed grid,* Figure 18) is dividing the space into cells (*buckets*), representing a block of secondary storage where all objects laying in the same cell are stored [AG97]. This method is only efficient for uniformly distributed point data, since many points in a few cells might overflow the capacity of a storing page [RSV02]. *Grid files* have been developed allowing a dynamic structure by splitting the cells according to the density of objects [HR01].

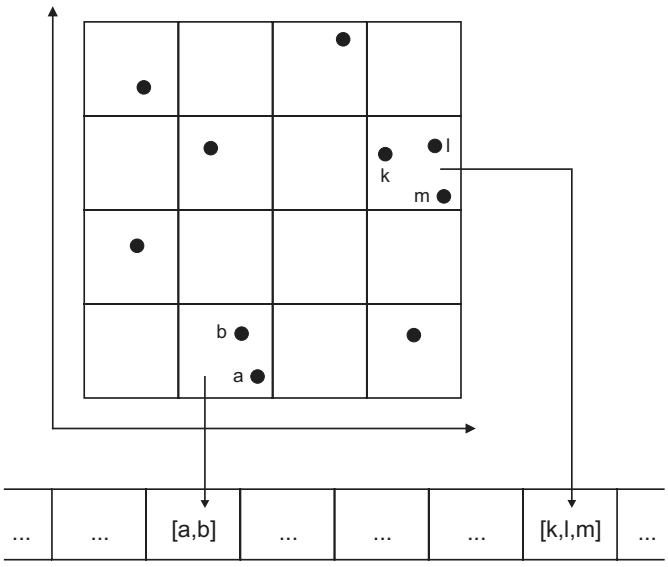


Figure 18: Fixed Grid (after [RSV02])

Quadtrees

In *quadtrees*, the space is divided into quadrants, which can be subdivided recursively until a condition of data storage is met. This results in storage overhead due to the big number of pointers, which is why *linear quadtrees* use listed records of the node data to become better suited to database storage schemes. Figure 19 shows an example with a maximum capacity of four points per cell. Many different *space filling curves* have been proposed to define an order to the cell labeling, which partially preserves proximity (neighbor cells are labeled with similar numbers and stored close together). The cells of a linear quadtree can be indexed

with a B-tree using their order as a key, providing a good performance especially in point queries. [Jon97], [RSV02]

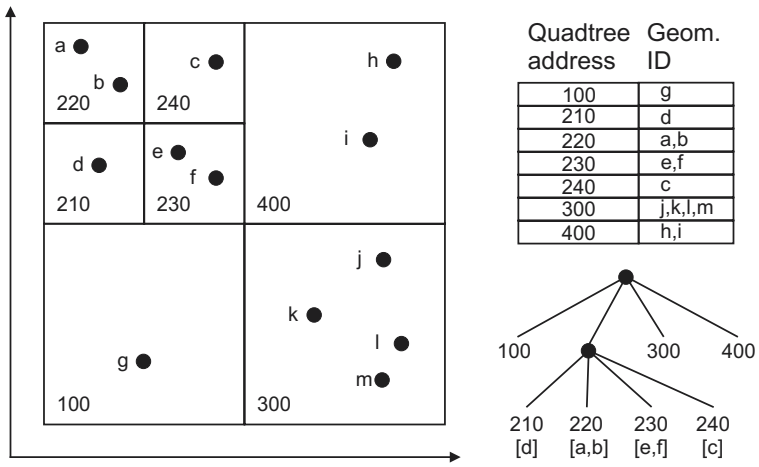


Figure 19: Spatial indexing with a Linear Quadtree (after [RSV02], [Jon97])

R-trees

This method was introduced in order to provide a dynamic index structure for objects in a multidimensional space. The tree is height-balanced and similar to a B-tree, with nodes containing the tuple-identifier of an object and its minimum bounding box. Non-leaf nodes contain a pointer to a lower level node and the MBB of all boxes stored in it. A search of the tree begins at its root and checks for overlapping of the search area with the MBB in each node, until in an end-leaf a record qualifies. Because of the hierarchical structure of MBB containing lower level MBB (and being contained by higher level MBB), several branches may have to be searched. In difference to previously discussed methods such as the quadtree, there exists an adaptation to skew data distribution. The tree remains balanced, since new entries are inserted into the leaves and overflowing nodes are split, which propagates up the tree. [Gut84], [RSV02]

One issue occurring with R-trees is that the boxes containing the objects may not be optimal, causing too much *coverage*, which is defined as the total area of all MBB of all leaf tree nodes. In addition, the effect of *overlapping* refers to the total space being contained within several leaf MBB. Given the assumption that most spatial databases are rather static, this problem can be addressed by *packing* R-trees, i.e. pre-organizing the objects and MBB before creating the R-tree index. The objects are grouped with the nearest-neighbor method and the leaf MBB are treated as objects themselves; that way, the algorithm works from bottom to top, creating the root at the end. Later inserts or deletions of objects can still be done. [RL85]

Another approach to avoid overlapping is followed when using R^+-trees. With the *clipping* technique, newly inserted rectangle objects are split and stored in several leaves. This leads to object duplication, increasing the tree size. But for point object data, a straight path down the tree is followed, and fewer nodes have to be visited, which increases performance. [RSV02]

Generalized Search Trees (GiST)

In order to extend search trees for maximum flexibility towards all kinds of new applications, development was concentrated on two main research approaches: 1) New data structures (specialized search trees) have been developed for specific problems (queries), for example the R-tree. 2.) Existing search trees have been extended, so that they support new data types. But regardless of the data type, B^+-trees only support queries containing linear range predicates (=, <, >) and, similarly, R-trees only support queries containing n-dimensional predicates (equality, containment, overlap). Hence a third possibility has been presented, which unifies some of the most common search trees (such as the B^+-tree, R-tree, hb-tree, TV-tree, Ch-tree, partial sum tree and others). The *GiST* is an object oriented, flexible structure, allowing new data types to be indexed and new queries natural to the types to be performed. It is implemented in one single piece of program code. The keys may be arbitrary, but are in practice coming from a user-implemented object class that provides a set of methods required by the GiST.

Those methods are sufficient to extend the GiST structure towards a specific SAM, allowing it to be used e.g. as an R-tree. This ability makes the GiST not only extremely extensible, but also a good tool to perform analyses, comparisons and optimizations of search trees. [HNP95], [HR01]

PART III: WEBGIS PROTOTYPE

7 Methods

This chapter describes the *WebGIS SNP* prototype that has been implemented in the practical part of this thesis. The first section specifies the requirements for the prototype application. A short description of the data is given in the second section. In the third section, the software components of the system are described in detail. The fourth section presents an overview over the system architecture of the prototype, referring to the client- and server-side technologies and data storage possibilities examined in the previous chapters. It also gives a short introduction into the basic setup of a MapServer application. The fifth section describes how important functions of the prototype have been implemented.

7.1 Requirements

Besides the research objectives formulated in section 3.3, the WebGIS prototype should have the following functionality:

- Basic functionality:
  - Zoom in and out, zoom to selected extents
  - Pan
  - Query (requesting information by clicking on an object in the map)
  - Changeable map size
- Advanced Functionality:
  - Attribute queries (querying attributes stored in the database through a separate user interface)
  - Visualization of the attribute query results in the map
  - Graphic aggregation of the query results in the map

- Interaction (the user can change parameters and the application reacts on changes)
- Adaptive user interface (the elements of the user interface adapt to the actual situation and behave according to the data that are currently available and the rules that must be respected)

These requirements lead to the consequence that both, the general user interface and the specialized interface for attribute queries must be programmed as dynamically as possible. Furthermore, the following use case shall be realized: A user should be able to perform attribute queries on several layers, enabling him to explore the underlying data stored in a database. This shall result in a list of query results on one hand, and in a visualization of the query results in the map on the other hand.

7.2 Data

The detailed description of the GIS data that have been used is listed in the Appendix. Generally said, the data are either raster or vector data; the former are geo-referenced raster images, and the latter originate from ESRI Arc/INFO vector data sets or Shapefiles. Some of them have been imported into the PostGIS database with the shp2pgsql tool.

The maximum extent covered by WebGIS SNP is limited by the following coordinates:

X_{min}: 785000 X_{max}: 835000 Y_{min}: 150000 Y_{max}: 199999

These are values in the Swiss Coordinate System with map units in meters and the following origin in the city of Berne:

X: 600000 = 7° 26' 22.5" N Y: 200000 = 45° 57' 88.66" E

The fact that all data are in German has heavily influenced the decision to create the GUI in the same language. This single language strategy was chosen because

German query results would have been intermixed into an English web page otherwise.

7.3 Software Components of the Prototype

The software components presented in this section form a multi-component prototype with open interfaces that could also be used for a distributed application with the components running on different servers, as discussed in chapter 3. Concerning the hardware, the server that has been used is a Sun Ultra-60 with 2 sparcv9 296 MHz processors and 1024 MB RAM. The operating system is Solaris 7. On this machine, all software components were configured, compiled and installed as illustrated in Figure 20. It is notable that virtually all software components in use on the server are distributed under Open Source licenses. They are discussed in the following subsections.

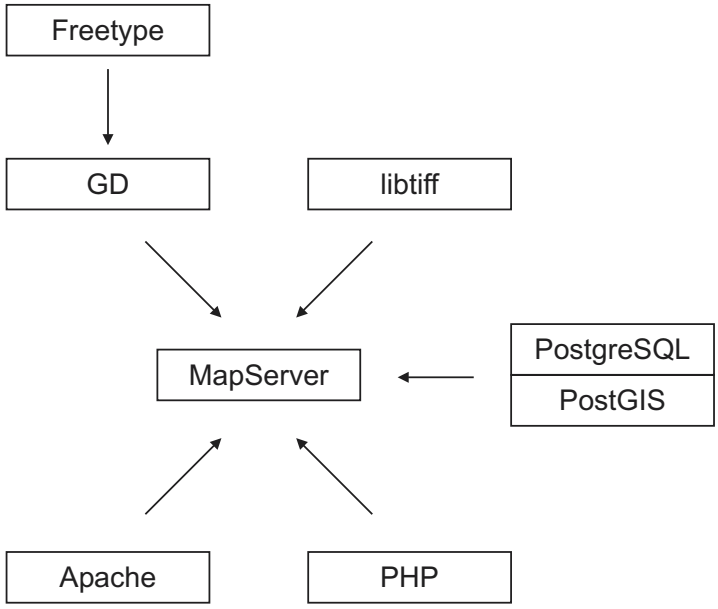


Figure 20: Prototype Software Components

7.3.1 Basic Libraries

In order to finally install and run the *UMN MapServer*, a few basic libraries have to be installed first. To be more precise, only the *GD* library is mandatory, but it requires the *zlib* and *libpng* libraries (which are the only parts that had already been installed on the server by the system administrators). GD also needs to be compiled against *Freetype* in order to enable TrueType font support with MapServer. Only *libtiff* is optional. All these components are Open Source.

Freetype

Freetype is a font engine, giving access to most important font properties and TrueType tables. It defines several kinds of structures that are used to manage the various abstractions that are required to access and display fonts. For more information, refer to http://www.freetype.org.

zlib

zlib is a general purpose data compression library. Its data format is portable across platforms. More information can be found at: http://www.gzip.org/zlib/.

libpng

libpng is the official PNG reference library. The PNG (Portable Network Graphics) file format enables a lossless, portable and well-compressed storage of raster files, providing a patent-free replacement of GIF. It uses zlib to compress and decompress PNG files. More information can be found at: http://www.libpng.org/pub/png/libpng.html.

GD

GD is an ANSI C library for the dynamic creation of images. It creates images of the type PNG and JPEG, but not GIF (anymore), because of patents issues with the compression algorithm for this format. More information can be found at: http://www.boutell.com/gd/.

libtiff

libtiff is a library for reading and writing TIFF (Tag Image File Format) files. More information can be found at: http://www.libtiff.org.

7.3.2 Apache Web Server

The Apache HTTP server is a well known Web Server that is maintained as an Open Source project by the Apache Software Foundation. It has been the most popular web server on the internet since 1996 and runs on UNIX/LINUX and Windows operating systems (http://httpd.apache.org).

7.3.3 PHP

PHP is an Open Source scripting language and has become very popular for dynamic web development (http://www.php.net). Since PHP is used on the server-side (see also chapter 5), it must be installed on the server. This is mostly done by compiling it into the Apache web server, either statically linked or as a dynamic module, with the advantage of a certain performance gain. Because of known current thread-safety issues with the module, the PHP/MapScript module of the UMN MapServer (see section 7.3.6) can at the moment of writing only be run if PHP has been installed as a CGI program. This requires additional configuration of the Apache web server.

7.3.4 PostgreSQL

PostgreSQL is an Extended RDBMS that has its roots in the POSTGRES project started 1986 at the Computer Science Department of the University of California at Berkeley. Since 1994, the original query language PostQUEL was replaced with SQL, and the software was released as an Open Source Project with the name Postgres95. In 1996, the new name PostgreSQL was chosen, reflecting the relationship between the original and the newer version with SQL capability. Features of PostgreSQL are constraints, triggers, rules and transactional integrity. It is extensible by the incorporation of the additional concepts of inheritance, data

types and functions. The spatial data model in use is the Spaghetti Model. *Point*, *line*, *line segment*, *box*, *path*, *polygon* and *circle* are included in the set of available geometric data types. A *region* type is missing. To avoid ambiguity when creating new geometric objects, the *'(Representation)'::type* notation is used (e.g., *'((0,0),(1,1))'::box* to create a rectangle, *'((0,0),(1,1))'::lseg* to create a line segment). A large set of native support functions and operators is implemented, but not listed here in detail. Four indexing methods are implemented into PostgreSQL: B-tree, Hash, R-tree and GiST. [RSV02], [PS03b]

In the following subsections, some of the capabilities of PostgreSQL are explained in more detail, showing the possibilities for a user to extend the system [PS03a].

Functions

Functions in PostgreSQL can most easily be written in SQL (*SQL Functions*). The *Procedural Language Functions* PL/pgSQL, PL/Tcl, PL/Perl and PL/Python are available by loadable modules, but also user-defined procedural languages can be developed. *Internal Functions* are written in the C programming language. They are predefined and have been statically linked into the PostgreSQL Server. *C Language Functions* are written in C and compiled into dynamically loadable objects. Such *shared libraries* may be loaded by the server on demand, which is the difference to internal functions that are statically linked.

Types

Types in PostgreSQL can be *Base Types* or *Composite Types*. While the latter are created with the creation of a table, the former are defined in a programming language and can be subdivided into *built-in types* that are compiled into the system and *user-defined types*. To define a new type, an input function and an output function must be written first.

Operators

The query planner can optimize queries that use an Operator. Operators can be created on top of an underlying function.

Interfacing Extensions to Indexes

After the creation of user-defined functions, types and operators, the system must be told how to use them with indexes. This has to be done by modifying some of PostgreSQL's system catalogs. *Access Method Strategies* are used to express the relationships between operators and the way they can be used to scan indexes. *Operator Classes* define the input data types for each of the operator classes. *Access Method Support Routines* are additional administrative routines internally used by the access methods.

Index Cost Estimation Functions

In order to optimize queries, the query planer needs a cost estimation function for every index access method. These functions are written in C and provide the cost of all disk and CPU costs associated with scanning the index.

7.3.5 PostGIS

PostGIS (http://postgis.refractions.net) is a spatial extension to PostgreSQL, enabling PostgreSQL to be used as a backend spatial database for GIS. It allows geographic objects to be stored in the database and includes support for GiST-based R-tree spatial indexes and functions for basic analysis of GIS objects. PostGIS follows the OpenGIS Simple Features Specification for SQL and will be submitted for compliance testing at version 1.0 (current version at the time of writing is 0.7.4). PostGIS is Open Source software and takes advantage of the extensibility of PostgreSQL (discussed in the previous section).

Geometric Data Types

Table 1 lists the available types provided by PostGIS, as they are specified by the OGC in [OGC99].

Geometry Type	**Representation**
Point	'POINT (10 10)'
LineString	'LINESTRING (10 10, 20 20, 30 40)'
Polygon	'POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))'
Multipoint	'MULTIPOINT (10 10, 20 20)'
MultiLineString	'MULTILINESTRING ((10 10, 20 20), (15 15, 30 ,15))'
MultiPolygon	'MULTIPOLYGON (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60)))'
GeomCollection	'GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))'

Table 1: Geometric Data Types implemented in PostGIS

Operators

PostGIS supports OGC's features and representations, but currently not the various comparison and convolution operators. Implemented operators are shown in Table 2, where A and B represent geographic objects.

Operator	**Description**
A &< B	A's mbb overlaps or is to the left of B's mbb.
A &> B	A's mbb overlaps or is to the right of B's mbb.
A << B	A's mbb is strictly to the left of B's mbb.
A >> B	A's mbb is strictly to the right of B's mbb.
A ~= B	A is geometrically equal to B (A and B are the same feature).
A ~ B	A's mbb is completely contained by B's mbb.
A && B	A's mbb overlaps B's mbb.

Table 2: Operators implemented in PostGIS

Functions

Two kinds of functions are implemented, but not listed here in detail: OpenGIS functions as specified by the OGC [OGC99] and other functions, which are mainly support functions and not needed by general users.

As PostGIS is still under development (tough rapidly evolving), some limitations still apply: 1) Topological relationships can at the moment only be tested between the mbb of two objects. An exception is the ~= operator which compares the actual geometries. 2) Named spatial relationship predicates for testing spatial relations between geometric objects are still missing, such as: Relate(), Touches(), Contains(), Crosses() and Disjoint(). PostGIS at the moment also lacks most spatial operators that support spatial analysis, like: Buffer(), Intersection(), Union(), Difference() and SymDifference().

This will change when the currently ongoing integration of GEOS (Geometry Engine – Open Source) has been completed. GEOS (http://geos.refractions.net) is porting the Java Topology Suite (JTS) to a C^{++} library, which will be used to provide more advanced topology operations to PostGIS. JTS is an Open Source Java API of 2D spatial predicates and functions and conforms to the Simple Features Specification for SQL (http://www.vividsolutions.com/jts/jtshome.htm).

7.3.6 UMN MapServer

The “MapServer” (http://mapserver.gis.umn.edu) of the University of Minnesota (UMN) has been developed in cooperation with NASA and the Minnesota Department of Natural Resources. It was chosen because of its many advantages: This Open source product has already been successfully used in several other projects [VBW+00], [Für01], [Puc01], [SVS+01]. MapServer is developed for UNIX/LINUX operating systems, but can also be run under Windows. It is not a full-featured GIS system, but provides core functionality to support a variety of web mapping applications. Its most important features include:

- Vector formats supported: ESRI shapefiles, simple embedded features, ESRI ArcSDE (alpha release)
- Raster formats supported (8-bit only): TIFF/GeoTIFF, GIF, PNG, ERDAS, JPEG and EPPL7
- Quadtree spatial indexing for shapefiles
- Fully customizable, template driven output
- Feature selection by item/value, point, area or another feature
- TrueType font support
- Support for tiled raster and vector data (display only)
- Automatic legend and scale bar building
- Scale dependent feature drawing and application execution
- Thematic map building using logical or regular expression based classes
- Feature labeling including label collision mediation
- On-the-fly configuration via URLs
- On-the-fly projection

On-the-fly projection is supported through PROJ4, a USGS cartographic projection library (http://www.remotesensing.org/proj/). Additional vector input is supported through OGR, a simple features library for the display of various vector data formats (http://gdal.velocet.ca/projects/opengis/). These features are optional and must be compiled into the system, as well as support for ArcSDE, Oracle Spatial or PostgreSQL/PostGIS.

MapServer is also compliant with OGC's Web Map Service Implementation Specification 1.1.0. It can be used as a WMS client in order to include map layers from remote WMS servers into MapServer applications, but can also work as a WMS compliant server, from which clients can build customized maps.

While the classic MapServer application is a CGI program, the MapServer C API can also be accessed via MapScript in several popular programming languages such

as Perl, Python, Tk/Tcl and Java. For the WebGIS SNP prototype, the PHP/MapScript module was used. This module is an Open Source development that has now become an integral part of the MapServer distribution. It makes MapServer's MapScript functions and classes available via PHP (http://www2.dmsolutions.ca/webtools/php_mapscript/index.html). Furthermore, the ROSA Java Applet (http://www2.dmsolutions.ca/webtools/rosa/index.html) is in use, providing additional client-side functionality like dragging images for box zoom on a map.

7.4 Architectural Overview and Configuration

After the description of data, hardware and software, Figure 21 illustrates an overview of the system architecture including all major components.

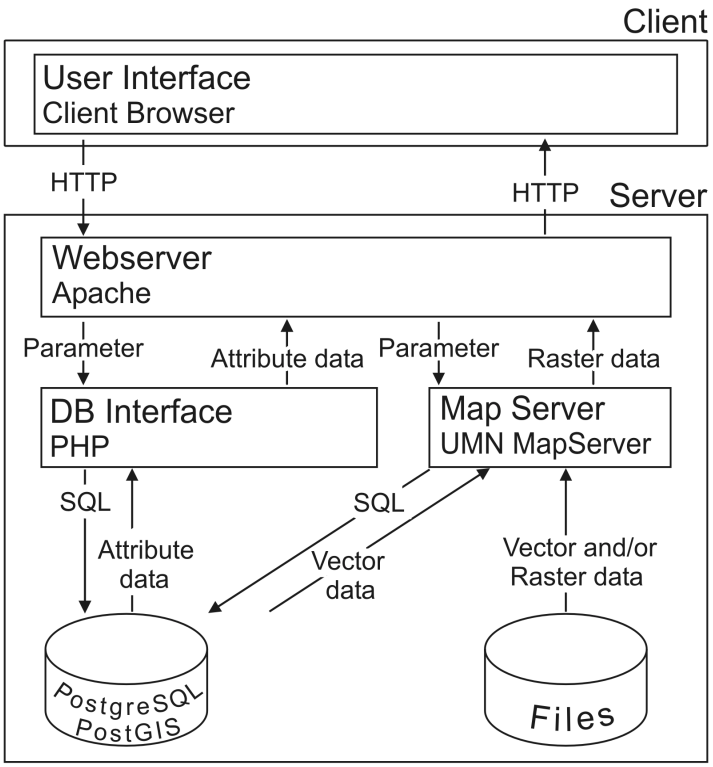


Figure 21: Prototype System Architecture

A request from the client is sent through the HTTP to the Apache web server. The PHP scripts are parsed and interpreted by the PHP CGI program or Apache module, and the results are included into the web document. Parameters for attribute queries are passed to PHP, which is used as an interface to the PostgreSQL database. As the result of such a query, attribute data are passed back the same way and are embedded into the HTML code of a web document. If a map request reaches the web server (simultaneously or independently), it is passed to MapServer's PHP/MapScript module, which processes the spatial data (read either from a file or from the PostgreSQL/PostGIS database) and creates a raster PNG file. The raster file is also embedded into the HTML code by the Apache web server and is finally sent back to the client, where it is displayed by the local web browser.

There are two things to note for explanation. First, PHP can be compiled and installed as both, an Apache module and a CGI program, at the same time. PHP/MapScript related requests are then (necessarily) processed by the CGI program, while all common PHP scripts can be handled by the faster module. However, this only works if they are separated in two different files with different file name endings, so that Apache can be configured to treat them separately. Second, the PHP/MapScript module completely replaces the MapServer CGI program. It is dynamically loaded into a PHP script and provides the same (and more) functions as if parameters had been sent to the CGI program.

7.4.1 Basic Setup

The basic setup of a MapServer application relies on two major components:

Template File

The *Template File* controls the display of MapServer output in a web page. More precisely, the design of the graphical user interface (GUI) and the way how users can interact with the application are defined here. In the most trivial case, when the MapServer CGI program is used, the template file is a normal HTML page that can

be designed like any other web page. An example of a simplified HTML Template File is shown in Figure 22.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head></head>
<body>
  <form method="GET" action="/cgi-bin/mapserv">
  <input type="hidden" name="map" value="[map]">
  <input type="hidden" name="imgext" value="[mapext]">
    <table>
      <tr>
        <td>
          <input type="image" name="img" src="/temp/[img]"></td>
        <td><img src="/temp/[legend]"><p><img src="/temp/[ref]"></td>
      </tr>
    </table>
  </form>
</body>
</html>
```

Figure 22: Simplified HTML Template File

It contains a form that collects a few parameters like the extent of the map and the layers to display via hidden input fields. The HTML form itself includes some input fields of type *image* that have a parameter name in brackets as their value. This is the case for the main map, the reference map, the scale bar and the legend. After a request has been processed and the resulting map has been produced as a raster image stored on the server, the parameter in the brackets is replaced with its value, which is the path to the respective file. The client browser then displays the image normally.

In the more advanced case, if the PHP/MapScript module is used, the Template File also contains PHP code, recognizable by the surrounding “<?php” and “?>” tags. An example of a simplified Template File with PHP is shown in Figure 23.

```
<HTML><HEAD>
<?php include("webgis.php"); ?>
<TITLE>WebGIS SNP</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<FORM METHOD=GET NAME="myform">
    <TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0">
      <TR>
        <TD BGCOLOR="#C1D8E3"> </TD>
        <TD BGCOLOR="#C1D8E3">
          <TABLE WIDTH="100%" BORDER="0" CELLSPACING="1" CELLPADDING="2">
            <TR BGCOLOR="#FFFFFF">
              <TD ALIGN="CENTER"><IMG SRC="images/checkmark.gif"></TD>
              <TD ALIGN="CENTER"><IMG SRC="images/icon_eye.gif"></TD>
              <TD><FONT FACE="Arial, Helvetica, sans-serif" SIZE="2"><B>
                Basic Themes</B></FONT></TD>
            </TR>
            <?php DrawBasicLayersLegend(); ?>
            <TR BGCOLOR="#FFFFFF">
              <TD ALIGN="CENTER"><IMG SRC="images/icon_checkmark.gif"></TD>
              <TD ALIGN="CENTER"><IMG SRC="images/icon_eye.gif"></TD>
              <TD><FONT FACE="Arial, Helvetica, sans-serif" SIZE="2"><B>
                Additional Themes</B></FONT></TD>
            </TR>
            <?php DrawAdditionalLayersLegend(); ?>
            <TR BGCOLOR="#FFFFFF">
              <TD ALIGN="CENTER"><INPUT TYPE=IMAGE SRC="images/icon_redraw.gif"></TD>
              <TD COLSPAN="2"><FONT FACE="Arial, Helvetica, sans-serif" SIZE="2">
                Redraw Map</FONT></TD>
            </TR>
          </TABLE>
        </TD>
        <TD BGCOLOR="#C1D8E3"> </TD>
      </TR>
    </TABLE>
(...)
```

Figure 23: Simplified Template File with PHP Code

The difference in this Template File is that a specific PHP function is called for every map component. Such a function is responsible for e.g. the creation of the map and can be defined within the same file, but is more advantageously placed in a physically different file that is included. This way, the design of the user interface and the internal functionality is held apart, which makes the maintenance of the code much easier.

Map File

The *Map File* can be regarded the configuration file and controls all other aspects a MapServer application has to deal with: the layers to display, the display

parameters (how shall the layers be displayed) and the query parameters (which layers can be queried). The example of a simplified Map File in Figure 24 shows how a couple of general settings like extent, status, and size are defined in the upper part. In the lower part, a layer with one class is described. Map Files are standard ASCII text files.

```
# Start of map file #

NAME mainmap
STATUS ON
SIZE 400 400
EXTENT 785000 150000 835000 199999
UNITS METERS
IMAGETYPE PNG

(...)

# Definition of FOREST Layer

LAYER
  NAME Forests
  TYPE POLYGON
  STATUS ON
  MINSCALE 20000
  MAXSCALE 200000
  METADATA
    "DESCRIPTION"   "Forests"
  END # METADATA
  CLASSITEM "veg_type"
  CLASS
    NAME 'FOREST'
    EXPRESSION "12"
    COLOR 0 255 0
 END #Class
(...)

END # LAYER
END # Map File
```

Figure 24: Simplified Map File

If the PHP/MapScript module is used, a big variety of classes, properties and methods are available, that allow changing or configuring most aspects of an application *on the fly*. For example, the status or minimum extent of a layer can be changed out of a PHP script, but also a whole new layer can be added.

7.4.2 Setup of the Prototype

For the prototype, there exists one main Template File (*webgis.phtml*). It is basically a HTML file with JavaScript functions defined in its head. PHP functions are stored in a plain PHP file (*webgis.php*), which is included into the Template File every time the application is loaded and itself includes other PHP files *on demand*. For instance, this is the case if the display of the Legend becomes more complicated because the themes stored in the PostGIS database require specific treatment (see later explanations). The Map File (*webgis.map*) includes layers that are not specified there in detail, but by the functions in the special PHP files just mentioned.

7.5 Programming

All the functionality of the WebGIS SNP Prototype that can not be regarded as MapServer's basic functionality was programmed in PHP. The following subsections give descriptions on how the problems have been solved, while the resulting functionality is presented in a later chapter.

7.5.1 Drawing Layers

In order to draw a map correctly, MapServer needs to know which layers are active. At startup, this is read from the Map File, but the parameters for the layers can be changed on the fly, e.g. if the user deactivates a layer. Thus, at every reload of the map, all layers have to be checked for their parameters, which is done by looping through them. Common layers (from ESRI Shapefiles) do not cause specific problems; their parameters are read from the HTTP form variables that have been submitted when the map is reloaded. More challenging is the handling of the PostGIS layers, where there are two subcategories to handle: The layers that are loaded from the spatial database when activated, or the layers generated dynamically as the visualization or aggregation of an attribute query result.

If one of the three PostGIS layers (vegetation, geology, and tectonics) is active, the extra functionality needed for setting the parameters is delegated to a separate file (*postgislayer.php)*. The same happens if dynamically generated layers are active (*dynqueryres_layer.php*, *dynattribute_layer.php*). Basically, there are two problems that need to be solved by such a file: First, a data statement (SQL string) has to be created and set, for fetching the right geometries from the spatial database for their visualization in the Main Map later on. Second, several database queries are done for getting the information needed for later display in the Legend. This requires a loop through all classes of each layer, where the class names and colors are received by querying the database and are set at the same time. For dynamic layers, the procedure is more complicated; much more variables have to be passed and set together, and the previous settings of the user need to be remembered and reinserted into the form (context preservation).

7.5.2 Dynamic Legend

Two functions are controlling the behavior of the legend (Figure 25).

Figure 25: Dynamic Legend

One of the functions is shorter and is called only for the background Themes which – as a constraint – shall never have more than one single class and are not allowed to be stored in the PostGIS database. The other function is more complicated, because it handles layers with several classes and from different sources (ESRI Shapefiles and two kinds of layers stored in the PostGIS database).

Concerning the PostGIS layers, there are two subcategories to handle: The standard layers that are loaded from the spatial database when activated, or the layers generated dynamically as the visualization of an attribute query result or its aggregation. This is explained in more detail in the following.

In a first step, a loop is done through all layers, to check if they are visible at the current map scale and determine their Visibility and Font Style properties. In a second step, the function only deals with the layers consisting of one single class and handles them differently, depending on the properties defined in the first step. The third step does the same for all layers that have two or more classes. In a fourth and fifth step, also the two kinds of PostGIS layers are treated.

A separate row in the Legend is created dynamically for each layer. In the first column, the checkbox needs to be defined, so that the user can activate each layer. Layers that are already activated must stay in this Status until the user or the application deactivates them. Activating some layers may result in them covering others completely. In order to prevent this, the latter layers are deactivated automatically: If the event of a click on the checkbox happens, a dynamically created JavaScript call is done for one of the several functions that deactivate the respectively other layers concerned. An icon is drawn in the second column, if the layer has only one class and is visible. In the third column, the layer name is specified and surrounded by the font tags determined earlier. Then, in the fourth column, a hyperlink is created individually, which enables to open the Information and Query Page when the user clicks on the question mark icon next to the respective layer name. It is populated with the following parameters: Layer name, layer type (point, line, polygon, raster, annotation or query), connection type

(specifies the data source), table (name of the attribute table for later database queries) and SID (PHP Session ID, needs to be passed for keeping track of the user's settings in the application).

The reason for the transmission of these variables is the dynamic nature of the Information and Query Page, where they are either displayed or needed for querying the database. Or, in the case of the Session ID, some settings like the map size and the active layers must be remembered for a later visualization of the query results (context preservation).

For layers with several classes, the function basically does the same as described above for one class, but leaves the column with the icon empty. Then, each class is looped, and a legend icon is created for it. All the classes are listed in additional rows below the layer's name, as long as the layer is active and visible.

Layers resulting from a Dynamic Attribute Query (Figure 25 shows a screenshot of the Legend when the result of a query on the tectonics theme is visualized) are treated separately, because many additional parameters need to be remembered for further redrawing of the layer. The reason for this is that Dynamic Layers are defined by an SQL string acting as input data source, which requires them to be filled in hidden form variables dynamically for recreating the SQL string after submitting the form (e.g., clicking on the map).

Since the database is queried several times, the dynamic creation of the legend and the visualization of attribute query results are slow in performance.

7.5.3 Information and Query page

The Information and Query Page opens in a separate browser window and consists of frames that display hyperlinks and information. As it has been mentioned in the previous section, it gets parameters originating from the Dynamic Legend. Depending on the chosen layer, it can act differently, offering just basic information about how to use the application and links to the metadata page of the Swiss National Park, or enabling a Dynamic Attribute Query.

The Dynamic Attribute Query Page consists of a form that offers the user to choose one of the supplied possibilities at a time (for illustration, see the Example Use Case described later). That way, a new variable is generated and the HTML form is submitted immediately by a JavaScript function. The submission causes a query in the database that looks for the new available possibilities for selection by the user dynamically. The page then reloads and displays the enlarged form with new possibilities to select. The user is guided step by step through the query process. At the end, all variables are chained together and form a SQL string that is used to perform the query the user wants to do. For the visualization or graphic aggregation of the query results all the parameters plus the session ID are passed back to the main application in the original browser window, which closes the circle.

8 The WebGIS SNP Prototype

The application as it presents itself in the browser at first access of the web page is shown in Figure 26.

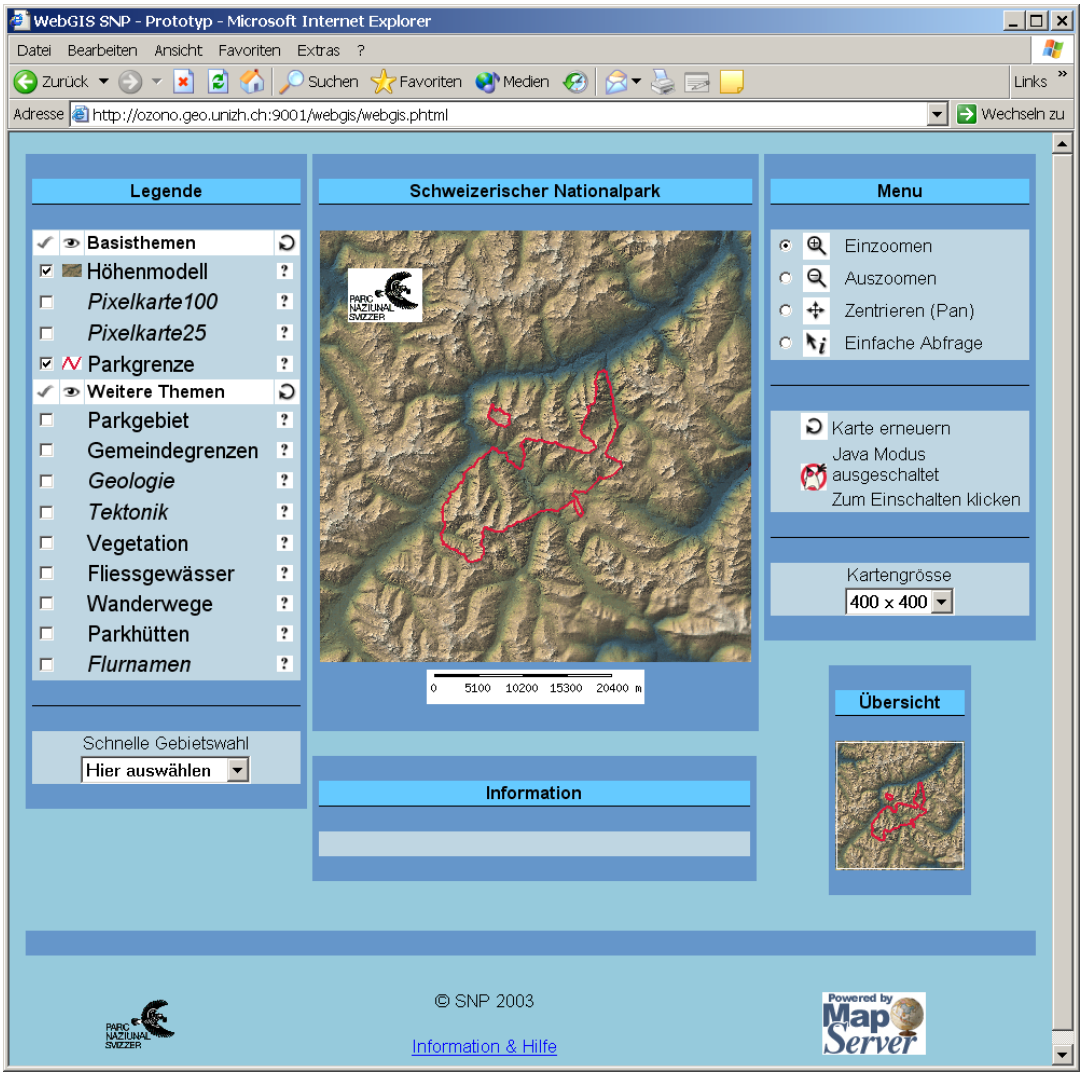


Figure 26: The WebGIS Prototype Application (Data © GIS-SNP)

8.1 Basic Functions, Tools and Elements

This section describes what can be called the basic functionality of the UMN MapServer, except the ROSA Java Applet which is provided by DMSolutions. However, as it has been described earlier, each of these and all other elements of the map has specifically to be enabled by designing the user interface in the Template File and configuring the behavior of the map and its layers in the Map File.

Two modes are implemented, which influence the look of the user interface and are called “Java Mode off” and “Java Mode on”. An icon informs the user about the mode that is active, and a click on it switches to the respectively other mode (Table 3). Figure 27 illustrates the different look of the Main Map with Java Mode on.

Java Mode off		If the icon in the Menu is red, Java Mode is off.
Java Mode on		If the icon in the Menu is green, Java Mode is on.

Table 3: Java Modes

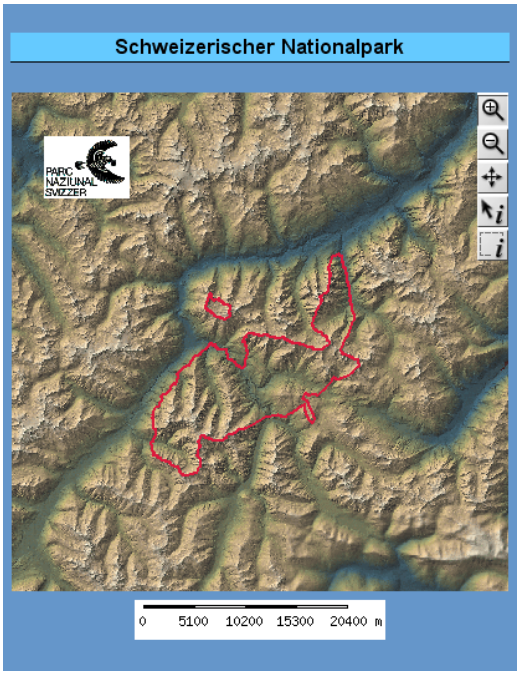


Figure 27: Main Map in Java Mode (Data © GIS-SNP)

Java Mode off is the default, because otherwise, users having a browser without Java virtual machine would just see an empty map the first time they load the web page. The icons for the Navigation Functions and Tools are then displayed in the Menu, where radio buttons can be clicked for selecting them. With Java Mode on, the ROSA Java Applet is used to display the map. It enables placing the icons as buttons inside of the Main Map and allows extended image functionality such as drawing a rectangle over the map. Furthermore, one more function is available: Area Query. Table 4 lists the *Navigation Functions*. Table 5 lists other *Tools*.

Zoom in		A mouse click on the map will zoom in with factor 2 and recenter the map at this point. If Java Mode is on, a rectangle can be drawn with pressed mouse button. This results in a zoom to the marked extent.
Zoom out		A mouse click on the map will zoom out with factor 2 and recenter the map at this point, unless a part of the map extent is outside of the extent defined for this application.
Pan		A mouse click on the map will recenter the map around this point, unless a part of the map extent is outside of the extent defined for this application.

Table 4: Navigation Functions

Point Query		A mouse click on an object in the map queries the respective dataset. The Information Table below the Main Map is then filled with information about that object, which is also highlighted in the map with red color. If more than one queryable layer is displayed, information is given for all objects located at this point.
Area Query		If Java Mode is on, it is possible to query several objects in a rectangle area. All available information about them is listed in the Information Table.

Table 5: Tools

8.1.1 Main Map

The activated and visible Themes are displayed in the Main Map. It is created by MapServer as a raster image in PNG format. Since it is an interactive map, every click on it causes parameters being sent to the server and a reloading of the map. Its

size can be changed in a pull down menu (“Kartengrösse”) and varies from 400*400 to 800*800 pixels. Since the size of the Main Map influences the position of the Menu and Reference Map, the application starts up with the smallest size, allowing the user to customize the application depending on his screen size at any time.

8.1.2 Reference Map

The Reference Map is a minimized version of the Main Map and gives an overview about the extent currently displayed there. A white rectangle marks this extent and enables an easier orientation of the user. If, after zooming in to a large scale, the rectangle would only be a point, its look changes into a cross. With Java Mode off, it is also possible to use the Zoom and Pan Functions in the Reference Map for quick navigation.

8.1.3 Graphic Scale

The graphic scale (scale bar) is located under the Main Map and is also an image in PNG format. It is rendered by the MapServer application at the same time as the Main Map, and the distances in the two images can therefore be compared.

8.2 Implemented Functions and Tools

The functionality described in this section has been implemented in order to extend the basis MapServer functionality.

8.2.1 Dynamic Legend

While legends provided by MapServer are just raster images with the activated layers, and standard legends mostly just consist of checkboxes for switching layers on or off, the legend presented here is a new development and completely dynamic. Number and properties of the layers are not hardcoded, but automatically detected on startup and every time the map is redrawn: All layers are checked individually for their Status, Visibility, Font Style, Query ability and input data type

(raster/vector). For layers with several classes, the respective class names and symbol color is detected as well.

The multifunctional legend thus enables the user not only to change the status of the map layers (by activating or deactivating its checkbox), but also to check the meaning of a symbol or whether a layer is visible at the current scale or not. Furthermore, the legend offers the possibility to get more information about the layer or to do an advanced query by opening the Information and Query Page. Table 6 lists all functionality appearing in the legend.

Status		A hook (or cross, depending on the browser) in the checkbox means that the Status of a layer is "on".
Visibility		The symbol for layers consisting of only one class is displayed in one row with the layer's checkbox and name. If a layer has several classes, they are displayed and labeled separately in additional rows.
Name		The name of a Theme is displayed either in normal or italic font style, depending on its Visibility.
Redraw Map		After the user has selected or deselected a Theme, a click on this icon causes the map to be redrawn.
Information/ Attribute Query		Clicking on this icon opens the Information Page in a separate browser window. Dependent on the layer, an attribute query on the layer's attributes and visualization of the query results can be done.
Quick View		Several areas of interest are listed in this pull down menu. Selecting one of them reloads the map, zooming in or out to the respective extent.

Table 6: Legend Functions

8.2.2 Information and Query Page

The Information and Query Page opens in a new browser window and can be reached via a hyperlink at the bottom of the main application page or by clicking on one of the legend's question mark icons next to a layer name. Depending on the layer, it looks a little different. There are two possibilities: For standard layers, information and helping explanations about the functionality are offered, as well as additional information such as hyperlinks to the metadata page of the Swiss

National Park or to the web pages of the prototype software components. For PostGIS layers, the advanced query possibilities are explained, which provide two special functions:

"Attribut-Tabelle": Lists the Attribute Table for the layer that has been specified by clicking on the respective question mark icon in the Legend. This table is the result of a pre-defined database query and gives the advanced user an overview over the attributes available for a theme (Figure 28). It can also be created in a new browser window.

"Datenbank-Abfrage": The Dynamic Attribute Query is the most complicated function in the WebGIS SNP prototype and allows the advanced user to query the Vegetation, Geology and Tectonics Themes. No user needs specific knowledge about databases or SQL; the selection of the variables happens intuitively. The results of such a query can also be visualized or graphically aggregated. A detailed example is illustrated in the Example Use Case section.

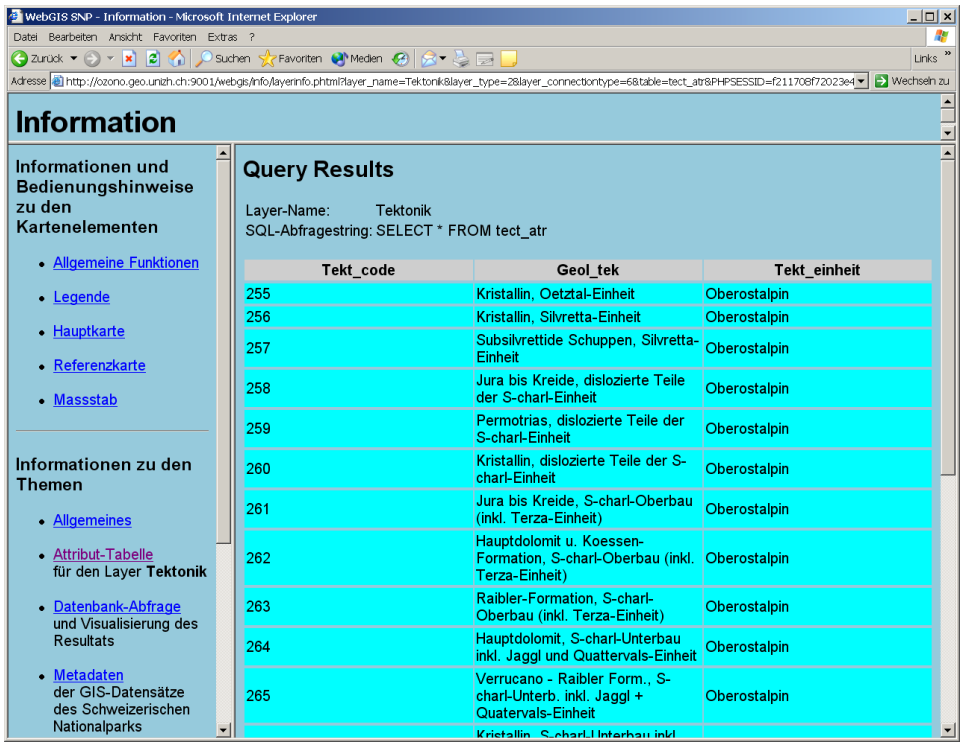


Figure 28: Information Page (Data © GIS-SNP)

8.3 Example Use Case

This section is dedicated to a comprehensive example of the implemented functionality. The use case that has been defined previously is followed step by step: Figure 29, Figure 30 and Figure 31 show the sequence of selections that allow the user to select a theme and an attribute. A simple SQL string is produced and the user can decide which action to perform next: Submitting the Query or refine it with a Subquery.

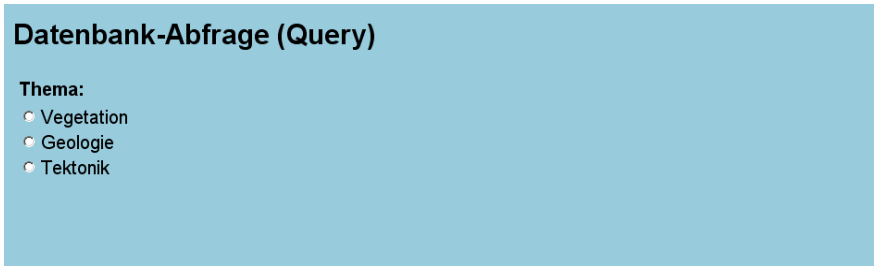


Figure 29: Dynamic Attribute Query (1): Selection of the Theme

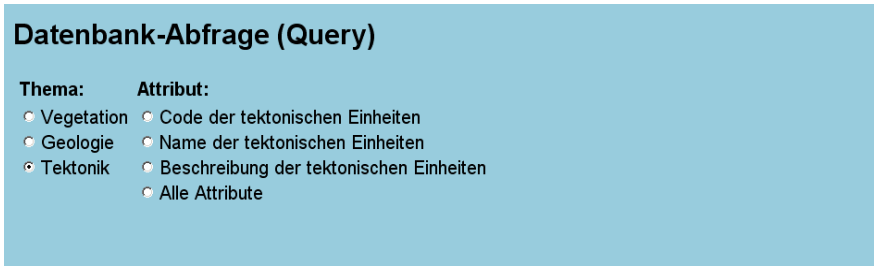


Figure 30: Dynamic Attribute Query (2): Selection of the Attribute

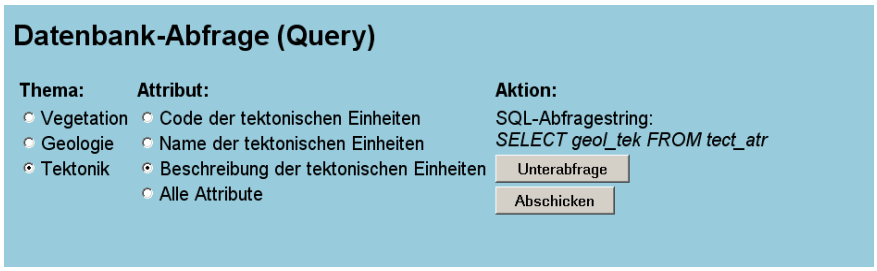
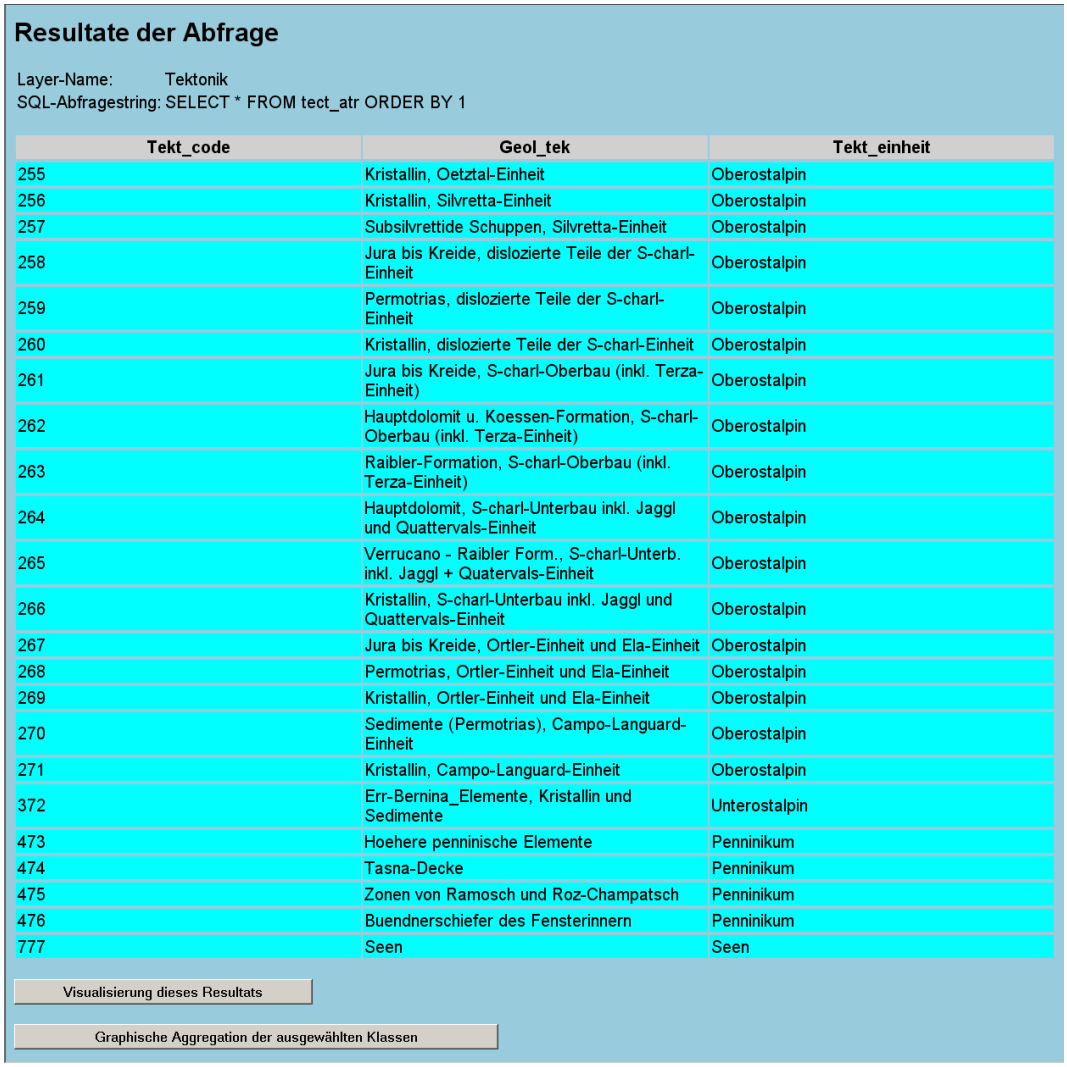


Figure 31: Dynamic Attribute Query (3): Produced SQL String

The results of a Query are listed in a dynamically created table in a separate browser window (Figure 32).



Resultate der Abfrage

Layer-Name: Tektonik
SQL-Abfragestring: SELECT * FROM tect_atr ORDER BY 1

Tekt_code	Geol_tek	Tekt_einheit
255	Kristallin, Oetztal-Einheit	Oberostalpin
256	Kristallin, Silvretta-Einheit	Oberostalpin
257	Subsilvrettide Schuppen, Silvretta-Einheit	Oberostalpin
258	Jura bis Kreide, dislozierte Teile der S-charl-Einheit	Oberostalpin
259	Permotrias, dislozierte Teile der S-charl-Einheit	Oberostalpin
260	Kristallin, dislozierte Teile der S-charl-Einheit	Oberostalpin
261	Jura bis Kreide, S-charl-Oberbau (inkl. Terza-Einheit)	Oberostalpin
262	Hauptdolomit u. Koessen-Formation, S-charl-Oberbau (inkl. Terza-Einheit)	Oberostalpin
263	Raibler-Formation, S-charl-Oberbau (inkl. Terza-Einheit)	Oberostalpin
264	Hauptdolomit, S-charl-Unterbau inkl. Jaggl und Quattervals-Einheit	Oberostalpin
265	Verrucano - Raibler Form., S-charl-Unterb. inkl. Jaggl + Quatervals-Einheit	Oberostalpin
266	Kristallin, S-charl-Unterbau inkl. Jaggl und Quattervals-Einheit	Oberostalpin
267	Jura bis Kreide, Ortler-Einheit und Ela-Einheit	Oberostalpin
268	Permotrias, Ortler-Einheit und Ela-Einheit	Oberostalpin
269	Kristallin, Ortler-Einheit und Ela-Einheit	Oberostalpin
270	Sedimente (Permotrias), Campo-Languard-Einheit	Oberostalpin
271	Kristallin, Campo-Languard-Einheit	Oberostalpin
372	Err-Bernina_Elemente, Kristallin und Sedimente	Unterostalpin
473	Hoehere penninische Elemente	Penninikum
474	Tasna-Decke	Penninikum
475	Zonen von Ramosch und Roz-Champatsch	Penninikum
476	Buendnerschiefer des Fensterinnern	Penninikum
777	Seen	Seen

Visualisierung dieses Resultats

Graphische Aggregation der ausgewählten Klassen

Figure 32: Dynamic Attribute Query (4): Table with Query Results (Data © GIS-SNP)

They can be either visualized with the classes listed in the table, or a graphic Aggregation of those classes could be done. While the latter will be illustrated later, the former case is presented in (Figure 33). The browser window containing the main application jumps on the top of all other windows and the page reloads. The classes are visualized in the Main Map and a new entry in the Legend has been created.

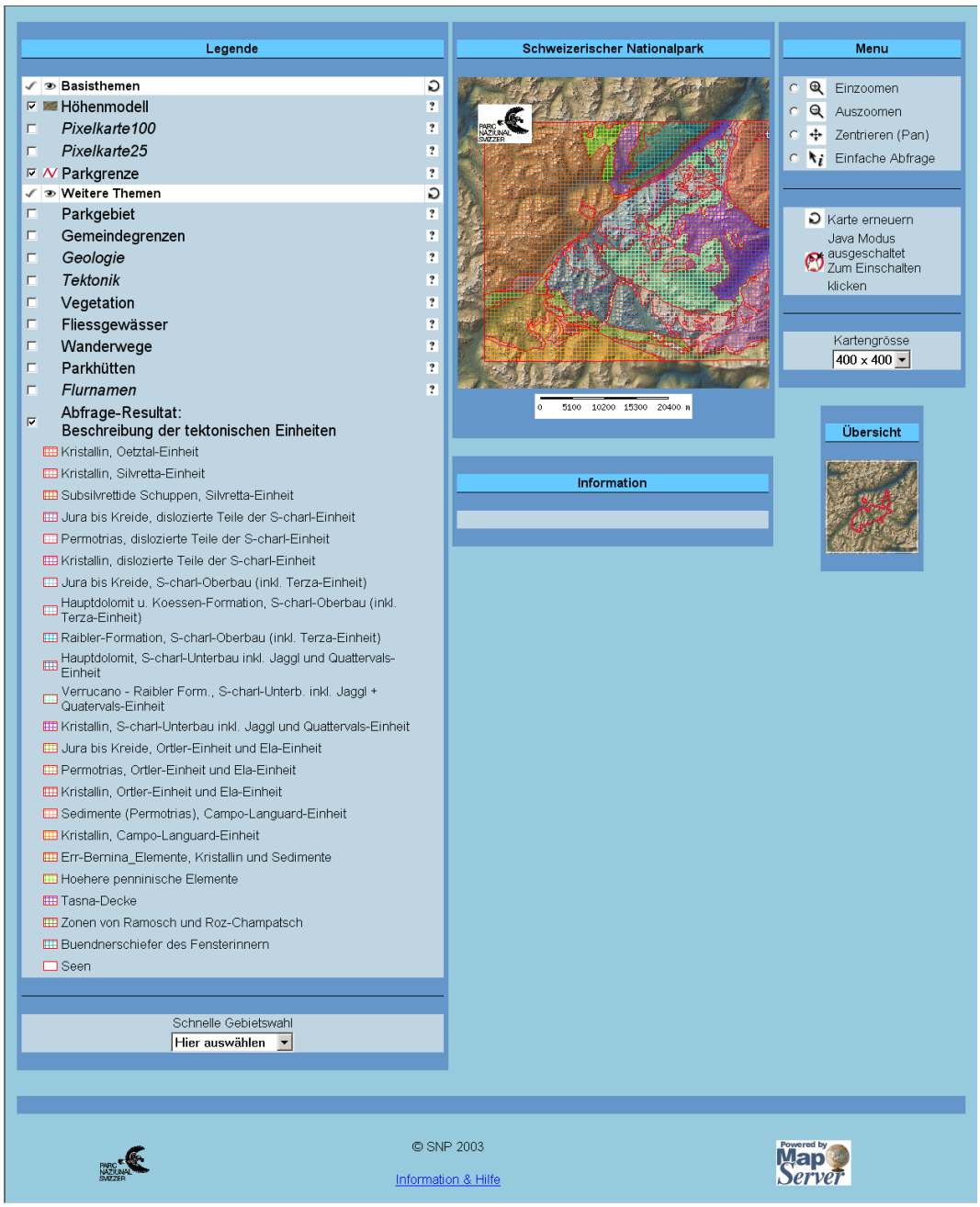


Figure 33: Dynamic Attribute Query (5): Visualization of Query Results (Data © GIS-SNP)

If the Subquery button in Figure 31 is pressed, the HTML form is enlarged and the user gets the possibility to choose a second attribute. After having done so, an Operator and a Value must be selected. If the Values were of numeric character,

more Operators would be provided (such as <, <=, > and >=). As can be seen in Figure 34, the former SQL string has been extended with a subselect statement. At this point, a second Subquery could be done for further refinement, or the actual SQL string can be submitted.

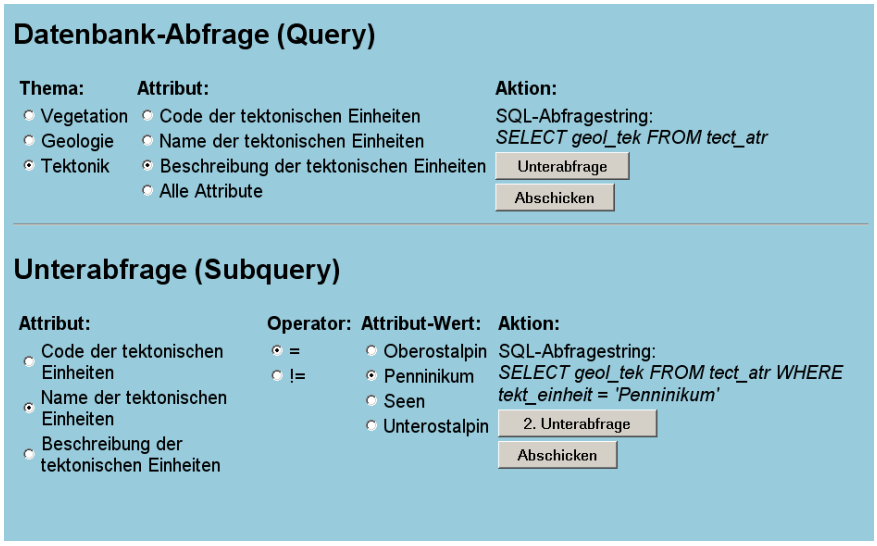


Figure 34: Dynamic Attribute Query (6): Refined SQL String

In this case, the number of results is reduced significantly to four classes. Again, the results can be visualized with all classes displayed separately (Figure 35), or graphically aggregated (Figure 36). In both cases, the map is zoomed to the respective extent, with a buffer of 10%. The Aggregation of the Query results is only of graphical nature, since in this application neither attribute nor geometric data are changed like it is possible in a GIS application.

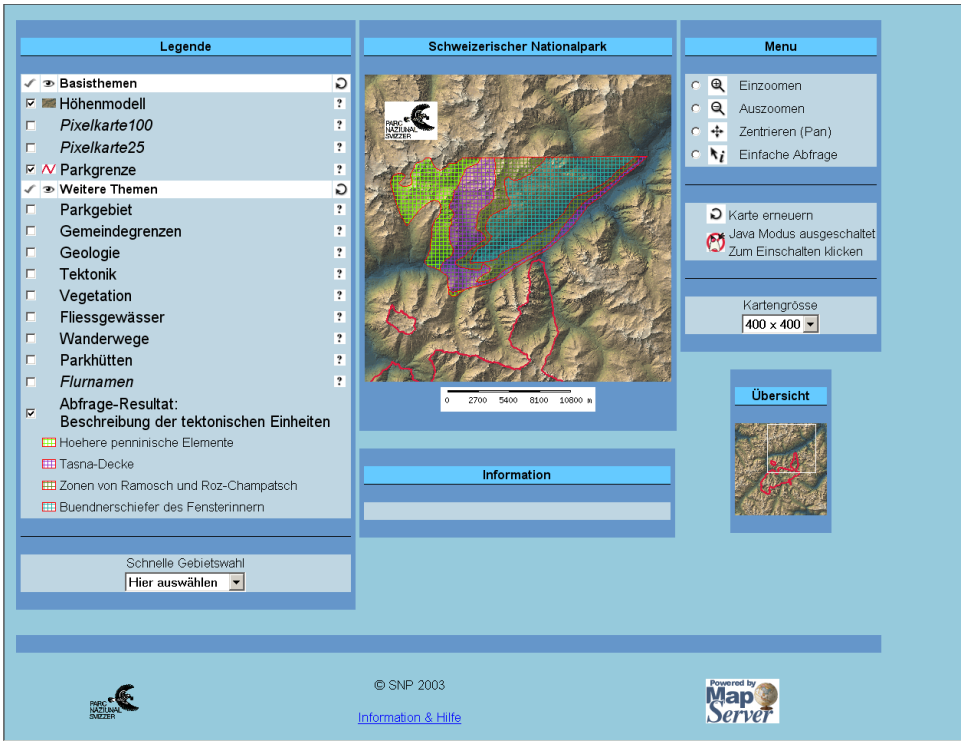


Figure 35: Dynamic Attribute Query (7): Visualization of Subquery Results (Data © GIS-SNP)

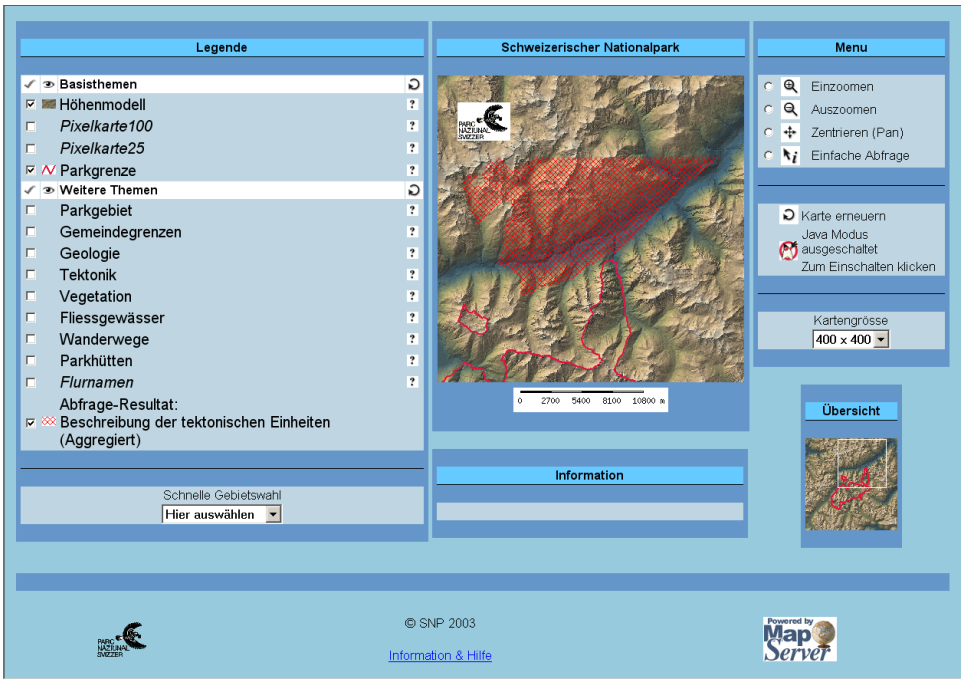


Figure 36: Dynamic Attribute Query (8): Aggregation of Subquery Results (Data © GIS-SNP)

8.4 Use and Role of Classic Map Elements in Web Maps

The Questions identified as research challenge in modern Web Cartography are: How are classic map elements used in interactive, static, on demand Web Maps? And what is their role in those Web Maps?

8.4.1 The Legend and the Concept of Status and Visibility

In paper maps, the legend serves as a key, explaining the meaning of the symbols that occur in the map. This is the classic legend function, which is also needed in interactive Web Maps. But, in difference to traditional paper maps, interactive Web Maps use the technique of thematic layers that can be activated of deactivated by the user. Creating a new map element for layer control is in most cases unsuitable, because it occupies space on the website (which is always rare) and forces the user's attention to be laid on one more tool. In practice, most interactive Web Maps use the legend for layer control, as it is familiar from the user interface of Desktop GIS programs like ESRI's ArcView. This leads to a dual functionality of the legend – classic function and layer controlling function – that has up to now not been reflected in the design of the legend. In order to do so, the concept of *Status* and *Visibility* has been developed.

Status and Visibility

It is proposed to distinguish between the Status and the Visibility of a map layer. If a layer's Status is "on", this means that it is generally activated and can be visualized, but it does not necessarily mean that it is visible, too. The reason is that some layers have a minimum and/or maximum scale, because their occurrence in the map only makes sense at certain scales. If the actual map scale is outside of the respective layer's minimum or maximum scale, these layers will not be displayed. Therefore, the same behavior has to be chosen for the legend symbols, since it is not appropriate to display them if they do not appear in the map. The legend thus can not remain static but must dynamically be recreated with every map redraw.

Font Style

The font style in use for the display of the layers' names has a supportive function to the above concept. Layers that are not visible in the current map do not have an icon and are labeled in italic instead of normal font. This lets the user intuitively estimate if the activation of a layer at the present scale would make sense, i.e. if it would be visible if its Status would be switched to "on". It also tells the user why an already activated layer is not visible at the moment (e.g., after zooming in) and thus supports the appearance/disappearance of the symbols visually.

Raster Layer Icons

In order to be consequent, also raster layers (input data in raster format, e.g. TIFF) do have an icon to show their visibility in the actual map. The icon is the same for all raster layers and has been created from a minimized area of a raster map. It is clear that the colors in this icon do not have any cartographic meaning; the icon does not represent a cartographic symbol. But it shows the user that a raster layer is visible and makes the legend consequently following the same rules for all layers.

8.4.2 Numeric and Graphic Scale

The numeric scale represents the relation of a distance in a map and the respective distance in nature [HGM02]. This works well for traditional paper maps, but becomes problematic with maps that are displayed on a computer screen. The distance on the map then is dependent on the hardware of the user, since all screens have different resolutions. It is at the moment not possible to detect the users screen resolution via scripting languages, so an estimated constant value of 72dpi is used by MapServer. The result of a scale calculation then looks very precise, with several floating numbers after the comma. In fact, this scale is only correct for screens with exactly the estimated resolution. Ironically, the numeric scale is therefore in most cases less precise than the graphic scale (scale bar). It is therefore not recommended to use the numeric scale in web maps. The same aspects are to keep in mind with distance measuring, since from the distance on the screen a

calculation of the distance in nature is made. Although they would be easy to implement, no numeric scale and no tool for distance measurement are offered in this application.

8.4.3 Other Elements

Other additional elements that are important for traditional paper maps are mostly missing in Web Maps. This mainly concerns elements that are located in the margin, like north arrows, the title, the border itself, the author/editor and the production date. The title is mostly missing inside of the website, but it can be located in the top of the border of the browser's window. The author/editor and the production date have lost their importance. The owner of the website is responsible for its content and mostly holds the copyright. The author has become the developer of the application. The user creates his own map from the data that are available, at the time he chooses.

8.5 Discussion and Evaluation

The WebGIS SNP prototype shows that it is possible to set up a working interoperable multi-component application consisting of Open Source software products. However, besides all advantages of Open Source software, also a few drawbacks have been experienced during this project. Since all components are individual software that is developed by different groups of the Open Source community, a new release of one component featuring additional desired functionality may result in a cascade of updating other components. The fact that they must be configured, compiled and installed individually makes the maintenance of such systems a time consuming process.

The requirements specified for the prototype could all be fulfilled. However, interactivity is a controversially disputed term among web cartographers. The basic functionality of zooming, panning and querying was easy to implement. The objective of an intuitive interface was reached by trying to keep it as "logic" as possible. Its use should be especially easy for users who are familiar with other

Web Maps. In one thing it differs strongly from other Web Maps and desktop GIS or drawing programs: Respecting the concept of Status and Visibility that has been proposed in this thesis, the legend has a dual functionality. For this reason, the interface might in this point appear non-intuitive on a first look. But this is overweight by the benefits of the concept, which are a better overview of the layers that are displayed at the moment and of the ones that could be visualized at the current scale. The interface for attribute queries can be considered to be most intuitive, since the user can graphically do his choice and so step by step collects the variables for a final SQL query string. He thus doesn't need any experience with SQL or databases in order to get the results he is interested in. Also the visualization of the query results can be reached with one single click.

However, both, the separate user interface for attribute queries and the dynamic legend have the drawback that several database queries have to be performed in order to either get the final query results or to get the layers' names, classes and icons. Not to forget that additionally, also the geometries have to be extracted from the database and drawn in the map. While the PostgreSQL/PostGIS database seems to perform well, PHP as a CGI program is the bottleneck. Persistent connections to the database would be desirable. Besides, the machine where all software components have been installed is neither configured as a GIS applications server nor as a database server; it is intensively used for general purposes by the staff of the department. Thus, only selected layers of the Web Map have been stored in the database completely (i.e., geometries and attributes) and made available for attribute queries. Nevertheless, it could be shown exemplarily that it is possible to store entire layers in a spatial database, which is an important step towards becoming independent from proprietary file formats.

As a last point, the adaptive user interface has to be mentioned. It can generally be said that the more dynamically an application is programmed, the better. This enables the application to be maintained by a webmaster, without needing the knowledge of a developer for e.g. every new layer that should be added. On the

other hand, the behavior of the interface can only be controlled by general rules, and the actual manifestation of it may look sub-optimal (e.g., see Figure 33, where the legend becomes very long because of the big number of classes resulting from an attribute query). The problem is the same as for the map itself: the developer sets up rules and provides the data – the map is then produced on demand upon the user's request, without any possibility to ensure its cartographic correctness.

9 Conclusions

9.1 Achievements

The main goal of this thesis was the implementation of a working prototype that is located in the interactive, static, on demand corner of the Cube of Web Map Types, a model that has been introduced for the approach of this thesis. The visualization of GIS data of the Swiss National Park with this application could be reached with the WebGIS SNP prototype, basing on MapServer and PostgreSQL/PostGIS. It consists of several software components and follows open standards. The interactive Web Map of the Swiss National Park includes basic functionality like zoom, pan and simple query, which is satisfactorily supported by MapServer. The basic functionality has been extended with the possibility of a database attribute query for layers stored in the PostgreSQL/PostGIS database. This has been solved with a separate graphical user interface (Information and Query Page) which intuitively leads the user to a list of query results without needing any knowledge in SQL. A later visualization or graphic aggregation of the query result in the Web Map is also possible. The graphical user interface of the Web Map also respects the dual functionality of legends in interactive Web Maps, combining the classic legend function with the controlling function for the layers. Both, the database attribute query and the legend, are programmed dynamically, which enables the application to react on changes in the data and thus makes it intelligent.

Furthermore, the use and role of classic map elements has been researched: Numeric scale calculations and distance measuring are not appropriate in web maps. Legends do not only have to play the traditional role known from paper maps. They are also used to control the status of the layers, which gives them a dual functionality. Moreover, the concept of Status and Visibility for map layers has been introduced in order to respect the fact that not all layers should be visualized at all scales, but still they should be generally available for the application. In addition, the consequence of maps on demand, especially with dynamically

produced layers, is that also the legends can not remain static. They must be dynamically recreated with every map redraw in order to provide only the symbols that occur in the map and the functionality that is useful and making sense at any time.

9.2 Insights

As already mentioned, this project has shown that it is possible to set up a working interoperable multi-component application consisting of Open Source software products for the visualization of spatial data. One of the insights is, that also the installation and maintenance of the system consume a big part of the time that has to be put into such a project. System architecture issues play an important role for the functioning and the performance. Furthermore, the more ideas are added to an application, the more they need to be weighed against each other: The realization of a new cartographic concept can make a user interface appear less intuitive at a first look, although the original intention was to make it more intuitive.

9.3 Outlook

The WebGIS SNP prototype and this thesis contribute to the efforts of the Swiss National Park to find the appropriate solution for a comprehensive presentation of its GIS data (including metadata) on the WWW – an issue which is currently still under discussion.

Future improvements of the prototype should primarily address the performance, which is sometimes poor when redrawing the legend and map for the visualization of layers that are created dynamically. This could be reached by not using a CGI program or by using a different programming language (e.g., Java).

While some recent developments in web mapping tend towards client-centric applications using SVG, Map Servers are still a good solution if the control over the application shall mainly stay in the hand of the developer. However, vector output (PDF and SWF) is one of the features expected in the next version 3.7 of the UMN

MapServer, accompanied by 24-bit image support. Concerning the dynamic character of applications, future solutions might possibly go one step further and not only build the legends, layers and classes dynamically, but also detect the available data from different sources. A Map File could thus be almost empty, accompanied by a program or script filling in the configuration parameters of the data dynamically. Besides, with GEOS fully integrated into PostGIS, a backend database with the complete Simple Features functionality will soon be available. The near future will therefore bring a set of powerful tools, enabling the setup of a WMS capable Interactive Map Server with a backend spatial database – all Open Source!

BIBLIOGRAPHY

[AB92] ALLGÖWER, B., BITTER, P.: Konzeptstudie zum Aufbau eines Geographischen Informationssystems für den Schweizerischen Nationalpark (GIS-SNP). Jahresbericht GIS-SNP 1992. Arbeitsberichte zur Nationalparkforschung. Wissenschaftliche Nationalparkkommission/Nationalparkdirektion, Zürich, Davos, 1992

[AG97] ADAM, N. R., GANGOPADHYAY, A.: *Database Issues in Geographic Information Systems*. Kluwer, Boston, 1997.

[Alb96] ALBRECHT, J. H.: Universal GIS Operations for Environmental Modeling. In: *Proceedings, Third International Conference/Workshop on Integrating GIS and Environmental Modeling*, Santa Fe, 1996. URL http://www.ncgia.ucsb.edu/conf/, 3.07.2003.

[Asc00] ASCHE, H.: Zum Nutzungspotential konventioneller und netzbasierter Atlanten: Welchen Mehrwert bieten die Neuen Medien? In: *Proceedings, Symposium Web.mapping.2000*, FH Karlsruhe - Hochschule für Technik, Karlsruhe. pp. VIII.2-VIII.12, 2000.

[Cec03] CECCONI, A.: Integration of Cartographic Generalization and Multi-Scale Databases for Enhanced Web Mapping. Dissertation, Geographisches Institut, Universität Zürich, 2003.

[Cec99] CECCONI, A.: Kartographische Darstellung von statistischen Daten im Internet. Diplomarbeit, Geographisches Institut, Universität Zürich, 1999.

[Cra99] CRAMPTON, J. W.: Online Mapping: Theoretical Context and Practical Applications. In: Cartwright, W., Peterson, M.P., Gartner, G. (eds.) *Multimedia Cartography*. Springer, Berlin, Heidelberg, New York, London, pp. 305-314, 1999.

[CSW00] CECCONI, A., SHENTON, C., WEIBEL, R.: Verwendung von Java für die kartographische Visualisierung von statistischen Daten auf dem Internet. *Kartographische Nachrichten* 50(4), pp. 151-162, 2000.

[CSW99] CECCONI, A., SHENTON, C., WEIBEL, R.: Tools for Cartographic Visualization of Statistical Data on the Internet. In: *Proceedings, 19th International Cartographic Conference*, Ottawa, Sect. 5, pp.59-69, 1999.

[DiB91] DIBIASE, D.: Visualization in the Earth Sciences. *Geotimes* 36(7), pp. 13-15, 1991.

[Dic01] DICKMANN, F.: *Web-Mapping und Web-GIS*. Westermann, Braunschweig, 2001.

[DZ99] DICKMANN, K., ZEHNER, F.: *Computerkartographie und GIS*. Westermann, Braunschweig, 1999.

[Ege96] EGENHOFER, M. J.: Spatial-Query-by-Sketch. *Proceedings, IEEE Symposium on Visual Languages*, Boulder, pp. 60-67, 1996.

[ES00] ESTER, M., SANDER, J.: *Knowldege discovery in databases: Techniken und Anwendungen*. Springer, Berlin, Heidelberg, New York, London, 2000.

[FRK+01] FÜRPASS, C., RIEDL, A., KRIZ, K., JORDAN, P., PARTL, F.: Suitability of a Mapserver from a Cartographic Perspective. In: *Proceedings, 20th International Cartographic Conference*, Beijing, pp. 2371-2379, 2001.

[Für01] FÜRPASS, C.: Mapserver als Hilfsmittel zur Datenvisualisierung im Internet - Erläutert anhand des Internetprojektes AtOS, der Internetversion des Atlas Ost- und Südosteuropa. Diplomarbeit, Universität Wien, 2001.

[Gar01] GARTNER, G.: Raumbezogene Visualisierung und Präsentationen im Internet: Stand und Tendenzen. In: *Proceedings, Symposium Web.mapping.2001*, FH Karlsruhe - Hochschule für Technik, Karlsruhe, pp. II.1-II.16, 2001.

[Gar99] GARTNER, G.: Multimedia GIS and the Web. In: Cartwright, W., Peterson, M.P., Gartner, G. (eds.) *Multimedia Cartography*. Springer, Berlin, Heidelberg, New York, London, pp. 305-314, 1999.

[GB01] GREEN, D., BOSSOMAIER, T.: *Online GIS and Spatial Metadata.* Taylor & Francis, London, 2001.

[Gün98] GÜNTHER, O.: *Environmental Information Systems*. Springer, Berlin, 1998.

[Gut84] GUTTMAN, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. *Proceedings of the Association for Computing Machinery, Special Interest Group on Management of Data (ACM SIGMOD) Conference*, Boston, pp. 47-57, 1984.

[Her01] HERRMANN, C. M.: Webmapping – Thesen, Beispiele und Tendenzen. *Kartographische Nachrichten* 51(6), pp. 279-285, 2001.

[HGM02] HAKE, G., GRÜNREICH, D., MENG, L.: *Kartographie. Visualisierung raum-zeitlicher Informationen.* deGruyter, Berlin, New York, 2002.

[HNP95] HELLERSTEIN, J. M., NAUGHTON, J. F., PFEFFER, A.: Generalized Search Trees for Database Systems. *Proceedings of the 21st VLDB Conference*, Zurich, pp. 562-573, 1995.

[HR01] HÄRDER, T., RAHM, E.: *Datenbanksysteme: Konzepte und Techniken der Implementierung*. Springer, Berlin, Heidelberg, New York, London, 2001.

[HSS01] HEUER, A., SAAKE, G., SATTLER, K.-U.: *Datenbanken – kompakt.* mitp, Bonn, 2001.

[Jon97] JONES, C. B.: *Geographical Information Systems and Computer Cartography*. Longman, Harlow, 1997.

[KE01] KEMPER, A., EICKLER, A.: *Datenbanksysteme. Eine Einführung*. Oldenbourg, München, Wien, 2001.

[KO96] KRAAK, M.-J., ORMELING, F. J.: *Cartography: Visualization of Spatial Data*. Longman, Harlow, 1996.

[Köb01] KÖBBEN, B.: Publishing maps on the Web. In: Kraak, M.-J., Brown, A. (eds.) *Web Cartography: developments and prospects*. Taylor & Francis, London, pp. 73-86, 2001.

[Kra01a] KRAAK, M.-J.: Settings and needs for web cartography. In: Kraak, M.-J., Brown, A. (eds.) *Web Cartography: developments and prospects*. Taylor & Francis, London, pp. 1-6, 2001.

[Kra01b] KRAAK, M.-J.: Trends in cartography. In: Kraak, M.-J., Brown, A. (eds.) *Web Cartography: developments and prospects*. Taylor & Francis, London, pp. 9-19, 2001.

[LR00] LEUKERT, K., REINHARDT, W.: GIS-Internet Architectures. *International Archives of Photogrammetry and Remote Sensing (IAPRS)*, Amsterdam, Vol. XXXIII, Part B4, S. 572-578, 2000.

[Mac94] MACEACHREN, A. M.: Visualization in Modern Cartography. In: MacEachren, A.M., Taylor, D.R.F., (eds.) *Visualization in Modern Cartography*. Elsevier, Oxford, pp. 1-12, 1994.

[Mac95] MACEACHREN, A. M.: *How Maps Work*. The Guilford Press, New York, 1995.

[MK01] MACEACHREN, A. M., KRAAK, M.-J.: Research Challenges in Geovisualization. *Cartography and Geographic Information Science* 28(1), pp. 3-12, 2001.

[OGC02a] OGC (OPEN GIS CONSORTIUM, INC.): URL http://www.opengis.org, 24.12.2002.

[OGC02b] OGC (OPEN GIS CONSORTIUM, INC.): Web Map Service Implementation Specification. Version 1.1.1. URL http://www.opengis.org, 24.12.2002.

[OGC02c] OGC (OPEN GIS CONSORTIUM, INC.): OpenGIS® Geography Markup Language (GML) Implementation Specification. Version 3.00. URL http://www.opengis.org, 18.02.2003.

[OGC99] OGC (OPEN GIS CONSORTIUM, INC.): OpenGIS® Simple Features Specification For SQL. Version 1.1. URL http://www.opengis.org, 18.02.2003.

[Pet01] PETERSON, M.: Webmapping: State of the Art. In: *Proceedings, Symposium Web.mapping.2001*, FH Karlsruhe - Hochschule für Technik, Karlsruhe, pp. I.2-I.8, 2001.

[PS03a] The PostgreSQL Global Development Group: PostgreSQL 7.2 Programmer's Guide. URL: http://www.postgresql.org, 21.3.2003.

[PS03b] The PostgreSQL Global Development Group: PostgreSQL 7.2 User's Guide. URL http://www.postgresql.org, 21.3.2003.

[Puc01] PUCHER, A.: Datenbankgestützte Visualisierung im Internet - Anwendungen im grossen Massstab mittels Mapserver-Systeme. Diplomarbeit, Universität Wien, 2001.

[Ric98] RICHARD, D.: Web maps – Karten im Internet. *Vermessung Photogrammetrie Kulturtechnik* 8, 1998. URL HTTP://WWW.VPK.CH/VPKOL.HTML, 3.2.2003.

[RL85] ROUSSOPOULOS, N., LEIFKER, D.: Direct Spatial Search on Pictorial Databases Using Packed R-trees. *Proceedings of the Association for Computing Machinery, Special Interest Group on Management of Data (ACM SIGMOD) Conference*, Austin, pp. 17-31, 1985.

[RP02] RECHENBERG, P., POMBERGER, G. (eds): *Informatik-Handbuch*. Hanser, München, Wien, 2002.

[RSV02] RIGAUX, P., SCHOLL, M., VOISARD, A.: *Spatial Databases with Application to GIS.* Morgan Kaufmann, San Francisco, 2002.

[Sch02] SCHNEIDER, B.: GIS-Funktionen in Atlas-Informationssystemen. Dissertation, ETH Zürich, 2002.

[She00] SHENTON, C.: Kartographische Visualisierung multivariater, statistischer Daten in einer verteilten, objekt-orientierten Umgebung. Diplomarbeit, Geographisches Institut, Universität Zürich, 2000.

[SNP97] SWISS NATIONAL PARK: GIS-SNP-Benutzeridentifikation II. Internal paper. 1997

[SR01] SCHMIDT, D., RINNER, C.: Intelligent, interaktiv, internetfähig – die neue Karten-Generation. In: Herrmann, C., Asche, H. (eds.) *Web.Mapping 1: Raumbezogene Information und Kommunikation im Internet.* Wichmann, Heidelberg, pp.90-99, 2001.

[SS98] STEFANAKIS, E., SELLIS, T.: Enhancing Operations with Spatial Access Methods in a Database Management System for GIS. *Cartography and Geographic Information Science* 25(1), pp. 16-32, 1998.

[Str01] STROBL, J.: Online GIS – das WWW als GIS-Plattform. In: Herrmann, C., Asche, H. (eds.) *Web.Mapping 1: Raumbezogene Information und Kommunikation im Internet.* Wichmann, Heidelberg, pp.18-29, 2001.

[SVS+01] SHEKHAR, S., VATSAVAI, R.R., SAHAY, N., BURK, T.E., LIME, S.: WMS and GML based Interoperable Mapping System. *Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems (ACMGIS)*, Atlanta, 2001.

[VBW+00] VATSAVAI, R.R., BURK, T.E., WILSON, T.B., SHEKHAR, S.: A Web-based browsing and spatial analysis system for regional natural resource analysis and mapping. *Proceedings of the 8th ACM International Symposium on Advances in Geographic Information Systems (ACMGIS 2000),* Washington D.C., 2000.

[W3C03] W3C (WORLD WIDE WEB CONSORTIUM): URL http://www.w3.org, 21.02.2003.

[Wir01] WIRZ, D.: Dynamic Web Publishing with Java and Oracle. Lecture script, Department of Geography, University of Zurich, 2001.

APPENDIX

Data

The data that have been used for the WebGIS prototype application are listed here in order of appearance in the Legend.

"Höhenmodell": A colored TIFF image, visualizing the digital elevation model 25 of the SNP. Geo-referenced by a TIFF World file (TFW). The original file (24 bit, 12 MB) was resized (8 bit, no compression, 8 MB). © GIS Swiss National Park.

"Pixelkarte100": Scanned Topographic Map of Switzerland 1:100'000, GeoTIFF with World file, 508 dpi resolution. © Swiss Federal Institute of Topography. More information can be found at: http://www.swisstopo/en/digital/pixel.htm.

"Pixelkarte25": Scanned Topographic Map of Switzerland 1:25'000, GeoTIFF with World file, 508 dpi resolution. © Swiss Federal Institute of Topography. More information can be found at: http://www.swisstopo/en/digital/pixel.htm.

"Parkgrenze": Border of the Swiss National Park, available as Arc/INFO vector data set (line). The Coverage has been converted to a Shapefile with ArcView. © GIS Swiss National Park. More information can be found at: http://www.nationalpark.ch/

"Parkgebiet": Area of the Swiss National Park, available as Arc/INFO vector data set (polygon). The Coverage has been converted to a Shapefile with ArcView. © GIS Swiss National Park.

"Gemeindegrenzen": Borders of the political boundaries, available as Arc/INFO vector data set (polygon). The Coverage has been converted to a Shapefile with ArcView. © GIS Swiss National Park.

"Geologie": Geological Units according to the Geological Map "Dössegger" 1987, 1:50'000, available as Arc/INFO vector data set (polygon). The Coverage has been converted to a Shapefile with ArcView and then imported into the PostGIS database with the shp2pgsql tool. It contains 57 different geological units, their short description and code of tectonical units. © GIS Swiss National Park. More information can be found at: http://www.geo.unizh.ch/nationalpark/gis/daten/grundd/.

"Tektonik": Tectonical Units according to the Tectonical Map "Dössegger" 1987, 1:400'000, available as Arc/INFO vector data set (polygon). The Coverage has been converted to a Shapefile with ArcView and then imported into the PostGIS database with the shp2pgsql tool. It contains 22 different tectonical units, their short description and code of tectonical units. © GIS Swiss National Park. More information can be found at: http://www.geo.unizh.ch/nationalpark/gis/daten/grundd/.

"Vegetation": Vegetation Units according to the Vegetation Map "Zoller" 1992, 1:50'000, available as Arc/INFO vector data set (polygon). The Coverage has been converted to a Shapefile with ArcView and then imported into the PostGIS database with the shp2pgsql tool. It contains 39 different vegetation units, their short description and code, and altitude levels. © GIS Swiss National Park. More information can be found at: http://www.geo.unizh.ch/nationalpark/gis/daten/grundd/.

"Fliessgewässer": Waters of the Swiss National Park (rivers), available as Arc/INFO vector data set (line). The Coverage has been converted to a Shapefile with ArcView. © GIS Swiss National Park. More information can be found at: http://www.geo.unizh.ch/nationalpark/gis/daten/grundd/.

"Wanderwege": Footpaths within the Swiss National Park, available as Arc/INFO vector data set (line). The Coverage has been converted to a Shapefile with ArcView. © GIS Swiss National Park.

"Parkhütten": Huts/cabins within the Swiss National Park, available as Arc/INFO vector data set (line). The Coverage has been converted to a Shapefile with ArcView. © GIS Swiss National Park.