

STUTT GART UNIVERSITY OF APPLIED SCIENCES

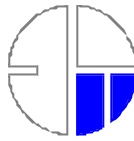
## **M.Sc Thesis**



# **Developing a GIS-based Geo-Portal with Scalable Vector Graphics (SVG) for Accessing Environmental Information of Baden- Württemberg**

**By**  
**Sudhir Kumar Reddy Maddirala**

**Stuttgart, March 2003**



FACHHOCHSCHULE **HOCHSCHULE FÜR**  
STUTT GART *TECHNIK*

---

STUTT GART UNIVERSITY OF APPLIED SCIENCES

## **Developing a GIS-based Geo-Portal with Scalable Vector Graphics for Accessing Environmental Information of Baden-Württemberg**

**A dissertation**

**Presented in partial fulfillment of the requirements for  
the degree of Master of Science in the  
Department of Geomatics, Computer Science and Mathematics  
of the University of Applied Sciences in Stuttgart**

**By**

**Sudhir Kumar Reddy Maddirala**

**700870**

**Fachhochschule Stuttgart – Hochschule für Technik**

**University of Applied Sciences**

**March 2003**

### **Supervisors:**

Professor Dr. Wolfgang Huep

Professor Dr. -Ing Franz-Josef Behr

Dr.M.Haase, Andrian Raiber (FAW)

### **Approved by:**

---

02.05.2003, Supervisor

**Master Course Photogrammetry  
and Geoinformatics**

### **Declaration**

The following Master thesis was prepared in my own words without any additional help.  
All used sources of literature are listed at the end of the thesis.

I hereby grant to Stuttgart University of Applied Sciences permission to reproduce and to  
distribute publicly paper and electronic copies of this document in whole and in part.

Stuttgart, 02.05.2003

---

(Sudhir Kumar Reddy Maddirala)

## Acknowledgment

First of all I would like to thank God for the grace strength provided to me for completing this course. The project would not have been completed without the help of several people. Also it is a pleasure to acknowledge the free rein given to me by Prof. Wolf Gang Huep in pursuing this Study. I thank him as well as Profs Dr. -Ing Franz-Josef Behr, Dr.M.Haase, Andrian Raiber (*FAW*, Research Institute for Applied Knowledge, ULM) for the discussions, suggestions and encouragement.

Many colleagues chipped in with words of wisdom at different stages of my studies and it was a great time with them around: special mention goes to Anand reddy, Harshavardhan reddy,Venkatesh,Bandaru Sreekanth, Kolitha Ratnapriya, Hein Hanifa, David Kuria, Md. Jezan, Joe Nyuyin, Md. Al-Barwani, Thummala Dhananjaya.

Finally, I dedicate this thesis to my parents, brothers and sister, Krishna reddy, Pushpalatha reddy and Satish kumar reddy and Suresh kumar reddy, who have jovially supported me and my endeavours and whose vision infact shaped me.

#### ABSTRACT

Geographical maps are widely used on the World Wide Web to visualize various kinds of spatial related data. Their purposes are as different as their appearances. With the progress of web techniques, the visualization of maps is becoming more advanced and sophisticated. To attract the users attention and to make their web experience more interesting, the use of interaction has increased with the expansion of www.

The goal of this thesis work is to access the environmental documents over the Internet-based information systems, which has the same functionality as the GIS software.

As the GIS community sees an expanding need for Internet extensions, SVG will play an important role in providing powerful yet economical solutions. In order to demonstrate the capabilities of this new XML grammar, GeoTechnologies, Inc has developed a number of sample Internet GIS applications.

This thesis project is research on the feasibility, mechanisms and potential of SVG and then implement an appropriate SVG based geo-portal. The portal shall provide access to documents via thematic tree and map functionality as well. Additionally, correlation analysis shall be performed and visualized by SVG techniques.

I concentrate on further development of the Web Information System and also research on the feasibility, mechanisms and potential of SVG and then implementation of an SVG based geo-portal. The Information System should have following functionalities:

- Fast, efficient, interactive and dynamic system.
- Having a friendly user interface with interactive maps and reference regions, navigation aid for the user with the functionalities calling of documents over space reference, different possibilities of accessing the document, visualization of the sample places in the map and alternatives, visualization of document types, graphical information presentation i.e., a system capable of assembling, storing, manipulating, and displaying geographically referenced information, i.e. data identified according to their locations.
- Providing access to documents via thematic tree and map functionality so that user can access the documents over an interactive map and /or by tree functionalities.

- Easy accessing data what user want and their graphical representation that means to provide a scientific investigation.
- Correlation analysis shall be performed and visualized by SVG techniques.

---

**Table of Contents**

	Page No
<b>Table of contents</b> .....	I
<b>Abstract</b> .....	IV
<b>Chapter 1</b> .....	1
1.Introduction.....	1
1.1 History of the Project.....	2
1.2 Goals of the work.....	3
1.3 Requirements of the System.....	4
<b>Chapter 2</b> .....	5
2.Technical bases.....	5
2.1 XML.....	5
2.2 Document Object Model (DOM).....	7
2.3 JavaScript.....	9
2.4 Scalable Vector Graphics (SVG).....	9
2.4.1 What is SVG?.....	11
2.4.2 Client-Side Application.....	12
2.4.3 Event Handlers.....	13
2.4.4 SVG Document Structure.....	14
2.4.4.1 Standalone SVG.....	14
2.4.4.2 Embedded SVG.....	15
2.4.5 SVG Features.....	16
2.4.6 The advantages of an XML-based vector graphic Language..	17
2.4.7 The Benefits of a Text-based vector formats for the WEB...	18
2.4.8 Dynamic Theme Generation.....	18
2.4.9 Coordinate system and Theme generation.....	19

---

2.4.10	Clipping and Masking.....	20
2.4.11	Basic Geometric Elements.....	22
2.4.12	Text.....	22
2.4.13	Colors, Fill samples, Color courses, Transparencies, Line Types	23
2.4.14	Interaction, Scripting.....	23
2.4.15	Metadata and Extensibility.....	24
2.5	Server side technique.....	25
2.5.1	Web Server.....	25
2.6	Serverside scripting, Java servlets and CGI.....	27
2.7	Common Gateway Interface (CGI).....	27
2.8	Standard and Open source Technology.....	29
2.9	SVG and GML.....	30
<b>CHAPTER 3</b>	.....	32
3.	Installation of Web-Information System.....	32
3.1	Basic Requirements.....	32
3.1.1	Apache Web Server.....	32
3.1.2	Adobe SVG Viewer.....	33
3.1.3	Active Perl.....	33
3.1.4	Perl Modules DBI and DBD-ODBC.....	34
3.1.5	Creating an ODBC connection.....	35
3.2	GUI-Graphical User Interface.....	37
3.3	System Architecture.....	38
<b>CHAPTER 4</b>	.....	40
4.	Implementation and design of Web-Information System.....	40
4.1	Generating SVG map.....	40
4.1.1	Steps to Install SVG mapper.....	40



4.1.2	Features of SVG mapper.....	41
4.2	Showing Coordinates on Mouse move.....	43
4.3	Legend.....	46
4.4	Overview Map.....	48
4.5	Panning.....	50
4.6	Accessing Documents.....	51
4.6.1	Tree View.....	51
4.6.1.1	Main Features of Tree View.....	52
4.6.1.2	How to setup a Tree Structure.....	52
4.6.1.3	Workflow.....	56
4.6.1.4	Accessing documents of Regions.....	58
4.6.1.5	Accessing documents of Sub-Regions.....	60
4.6.1.6	Accessing documents of Counties (Kreis)...	61
4.6.1.7	Accessing documents of Cities.....	62
4.6.2	Accessing Documents by Clicking over the Interactive Map	63
4.6.2.1.	Overview Process.....	64
4.6.2.2.	Accessing documents of Regions.....	64
4.6.2.3.	Accessing documents of Counties.....	68
4.6.2.4.	Accessing documents of Cities.....	70
4.7	Changing Colors of the Layers over the map.....	71
4.8	Scale Bar.....	72
4.9	Generalization.....	74
4.10	Conclusions.....	80
4.11	References.....	82

---

## ABSTRACT

Geographical maps are widely used on the World Wide Web to visualize various kinds of spatial related data. Their purposes are as different as their appearances. With the progress of web techniques, the visualization of maps is becoming more advanced and sophisticated. To attract the users attention and to make their web experience more interesting, the use of interaction has increased with the expansion of www.

The goal of this thesis work is to access the environmental documents over the Internet-based information systems, which has the same functionality as the GIS software.

As the GIS community sees an expanding need for Internet extensions, SVG will play an important role in providing powerful yet economical solutions. In order to demonstrate the capabilities of this new XML grammar, GeoTechnologies, Inc has developed a number of sample Internet GIS applications.

This thesis project is research on the feasibility, mechanisms and potential of SVG and then implement an appropriate SVG based geo-portal. The portal shall provide access to documents via thematic tree and map functionality as well. Additionally, correlation analysis shall be performed and visualized by SVG techniques.

I concentrate on further development of the Web Information System and also research on the feasibility, mechanisms and potential of SVG and then implementation of an SVG based geo-portal.

The Information System should have following functionalities:

- Fast, efficient, interactive and dynamic system.
- Having a friendly user interface with interactive maps and reference regions, navigation aid for the user with the functionalities calling of documents over space reference, different possibilities of accessing the document, visualization of the sample places in the map and alternatives, visualization of document types, graphical information presentation i.e., a system capable of assembling, storing, manipulating, and displaying geographically referenced information, i.e. data identified according to their locations.
- Providing access to documents via thematic tree and map functionality so that user can access the documents over an interactive map and /or by tree functionalities.
- Easy accessing data what user want and their graphical representation that means to provide a scientific investigation.

# Development of environmental status Reports on WEB by using SVG (Scalable Vector Graphics)

## CHAPTER 1

### 1.Introduction:

In the context of this work Web Information Systems were provided, which allows the user to access documents over an interactive map. For the implementation of this Web Information System, XML based language SVG (Scalable Vector Graphics) is used. Scalable Vector Graphics (SVG) is the emerging standard for the display of two-dimensional vector graphics on the Web. This opens very interesting possibilities for Cartographic applications in the Internet. SVG is a grammar in XML (eXtensible Markup Language), another W3C specification. Through the use of a scripting language (such as JavaScript), and access to the SVG Document Object Model (DOM), images can be made both interactive and dynamic within the Web browser. Since this System has no analysis functions, it is not called a GIS (Geographical Information Systems).

The most important functions of this system are:

- Call of documents over the space reference.
  - Different possibilities for the visualization of sample places in the map.
  - Interactive zooming and panning Functionality over the map.
1. Zooming can be performed in two ways. With a right click of the mouse button, a menu opens. Zooming in or zooming out can be selected; the maps will zoom-in/out in a preset scale. Three steps of zooming in or out are starting from the original view.
  2. The second method defines a zooming value in the List Box; select the Zoom Values in List Box. The map can be panned to a desired position by holding the 'Alt' key on the keyboard and using the right mouse button to move the map and also by dragging the Light Green rectangle Over the Overview Map.
- Interactively changing the colors of the layers over the Map.
-

- Accessing Documents via Tree structure.
- Selective Display of Geographic Features, one can select and deselect layer(s) on the map by checking and un-checking the relevant check boxes on the right of the map.
- Visualizing the Scale Bar.
- Visualizing the coordinates of the map.

The system is coupled with an ACCESS database, which contains information about the Environmental documents, which can be accessed through Web Information System.

### **1.1 History of the Project**

Decision-makers from politics, government, science and business depend on reliable, up-to-date environmental information in order to promote effective means of environmental protection and sustainable development. The Environmental Information Systems department at FAW Institute, ULM supports users from these communities by applying techniques based on advanced information technology on the business, state, and global levels. Since its foundation, FAW has been continuously involved in the design and development of the environmental information systems of the state of Baden-Württemberg, Germany.

The work of the Environmental Information Systems department at FAW is currently arranged into the following closely interrelated topics:

- Telematics and hypermedia for environmental protection.
- Geographical information systems for environmental protection.
- Sustainable development in the information society.

The development and use of geographical information systems is a core skill of the Environmental Information Systems department at FAW. It is currently working on behalf of the Baden-Württemberg Ministry of Environment and Transportation and the state water authorities to design, implement and roll out a GIS- based toolbox for use at the water authorities.

There is a link between document management and Geographic Information Systems (GIS), which becomes more and more obvious as many of the data acquired in industry and public services, feature a geo-spatial relationship. Geo-portals can assist a user to identify relevant topics in existing

documents by intuitively navigating through maps. On the other hand, in current search engines geo-spatial relationships only play a minor role and are only available on a textual basis if any.

In Germany, for all authorities on behalf of environmental affairs it is mandatory to provide overall information to the public. For instance, there is a federal report "Environmental Data" which is already digitally available and hence ready for access through a geo-portal. Especially environmental documents feature a strong geographical relationship, as they are often limited to a very specific area. This may be any political entity for which a certain bill or law is valid. It can also be an ecologically defined area, such as a biotope or a retention area, for which data are acquired and then presented in a document. When a user e.g. searches for regulations on waste disposal and soil protection for a specific state or area, he right now has to enter both the theme and the geo-relationship as texts into a conventional search engines. The result would then be references to documents containing exactly these keywords – even if the documents are only valid for a specific county in that state. Using a geo-portal, however, allows a user to pre-select the required map-layer, then to select the state or area he is looking for, and then to start a database query for getting titles and descriptions of documents referring to that location, including hyperlinks. Systems allowing an additional thematic focus are even more comfortable and target-oriented. So a first prototype was developed at FAW within the project PADDLE (Personal Adaptable Digital Library Environment) /SCHWARTZ 2000/. This provides more intuitive access to documents, which refer to geographic locations or areas by a geo-portal instead a conventional search engine.

## **1.2 Goals of the work**

The goal of this work is to access the environmental documents over the Internet-based information systems, which has the same functionality as the GIS software.

As the GIS community sees an expanding need for Internet extensions, SVG will play an important role in providing powerful yet economical solutions. In order to demonstrate the capabilities of this new XML grammar, GeoTechnologies, Inc has developed a number of sample Internet GIS applications.

My thesis project is research on the feasibility, mechanisms and potential of SVG and then implement an appropriate SVG based geo-portal. The portal shall provide access to documents via thematic tree and map functionality as well. Additionally, correlation analysis shall be performed and visualized by SVG techniques.

---

I concentrate on further development of the Web Information System and also research on the feasibility, mechanisms and potential of SVG and then implementation of an SVG based geo-portal. The Information System should have following functionalities:

- Fast, efficient, interactive and dynamic system.
- Having a friendly user interface with interactive maps and reference regions, navigation aid for the user with the functionalities calling of documents over space reference, different possibilities of accessing the document, visualization of the sample places in the map and alternatives, visualization of document types, graphical information presentation. That is a system capable of assembling, storing, manipulating, and displaying geographically referenced information, i.e. data identified according to their locations.
- Providing access to documents via thematic tree and map functionality so that user can access the documents over an interactive map and /or by tree functionalities.
- Easy accessing data what user want and their graphical representation that means to provide a scientific investigation.
- Correlation analysis shall be performed and visualized by SVG techniques.

### **1.3 Requirements of the System:**

The general system requirements are:

- Computer running Arc View GIS 3.x (for the export script).
- Geographic data in any format supported by Arc View.
- A web server with MySQL and PHP.

To view the resulting files, an Internet Explorer 4.0+ web browser with the Adobe SVG Viewer plug-in.

## **CHAPTER 2**

### **2. Technical bases**

#### **2.1 XML**

SVG is based on XML (extensible Markup language), therefore the fundamental concepts of XML should be discussed briefly first. XML is a standard language, which is developed by the World Wide Web Consortium (W3C). The emergence of XML has lead to the development of powerful new technologies to overcome the limitations that exist with HTML, but it has also provided the framework for developing open documents /data formats.

The SVG specification describes one such data format for providing 2-dimensional vector graphics on the Web. Although SVG is not yet a standard, it has already attracted considerable attention from the Web design community and support from software vendors who would welcome a well-supported vector graphics to include in their Web Page. The fact that this vector graphics standard will also be an XML format makes it even more powerful. Apart from the obvious advantage of being non-proprietary and platform independent, it may be styled using style sheets, processed by any XML parser and just like XML, it can be generated dynamically in a number of different ways.

XML is the universal standard for structured web documents, for maximal independences of networks and platforms. Thanks to XML, documents are fit for exchange and for applications of all kinds. It is extensible; therefore it is the foundation of all further "dialects" (specializations), like SMIL (multimedia), SVG (vector graphics), MathML (special mathematical symbols and formats), X3D (3D graphics), XHTML (successor of HTML), XFORM (form generation), GML (Geography Markup Language), and so on. Extensions may be effectuated by professional associations or even by individuals, as long as the particular DTD is included. Some extensions are in use are expertise, standardized and authorized by W3C.

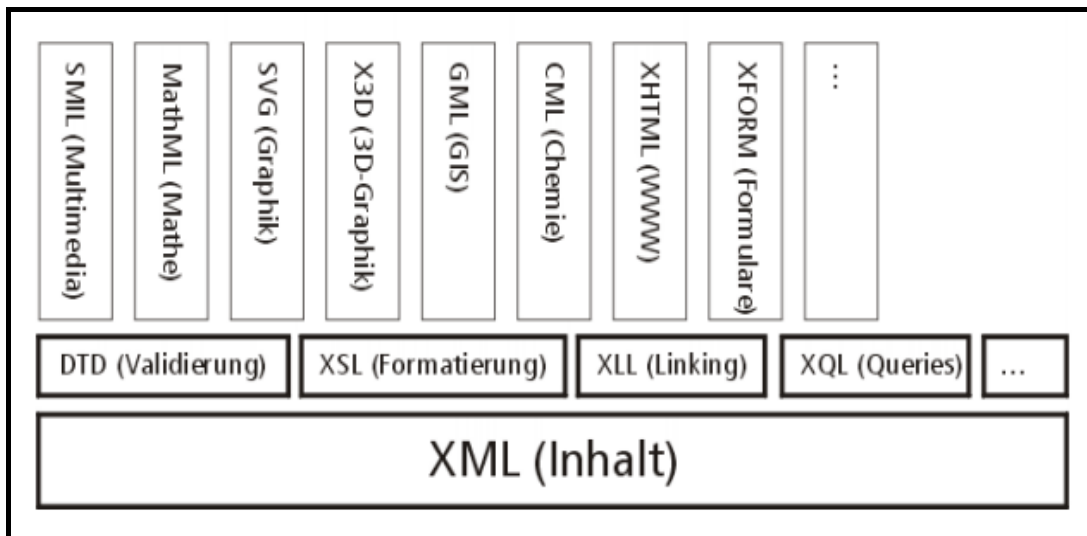


Fig.1: XML architecture and dialects [12]

The actual data is disposed in a XML file, which is composed roughly out of entities (formulated in "tags") and attributes. Examination of structure and syntax occurs in DTD (Document Type Definition). The DTD defines data types, namespaces (authorized tags), hierarchy and interlocking. Compulsory or optional elements and their possible attributes are equally defined. Parsers that meanwhile are available in all programming languages, as well as in the latest browser versions, in order to validate XML files and to detect errors, use dTDs.

Further XML related technologies are XSL (XstyleSheets, successor of CSS - responsible content formatting), XLL (XlinkingLanguage, responsible for interlinking elements and media), and XQL (XqueryLanguage, enables structured queries of XML data). This list is being constantly expanded and made relevant.

This means, vectors should equally be described in an XML compatible manner. Unfortunately, this applies only to a small minority of the above formats. Elements of web sites are rendered animated and interactive, once they may be accessed properly, using a scripting language, with JavaScript in pole position. A prerequisite is a browser capable of interpreting the given language.



## 2.2 DOCUMENT OBJECT MODEL (DOM)

Most WWW vector formats mentioned above are usually represented in the browser by means of a plug in, which means we are dealing with an additional program, which has not originally been an integral part of browser software. Furthermore, often we encounter proprietary, poorly documented data formats, which can be generated only by exporting from a certain graphical program. That is why they are opposed to the Internet principle of open source. More disadvantages are posed by the mostly binary formats, since they can't be edited and corrected any longer outside the generating software. (Almost) any Internet map project requires editing, however, because interactions and display mechanisms need to be adapted "manually", and later manipulation may as well turn necessary, according to new data, analysis, selections and change of symbols.

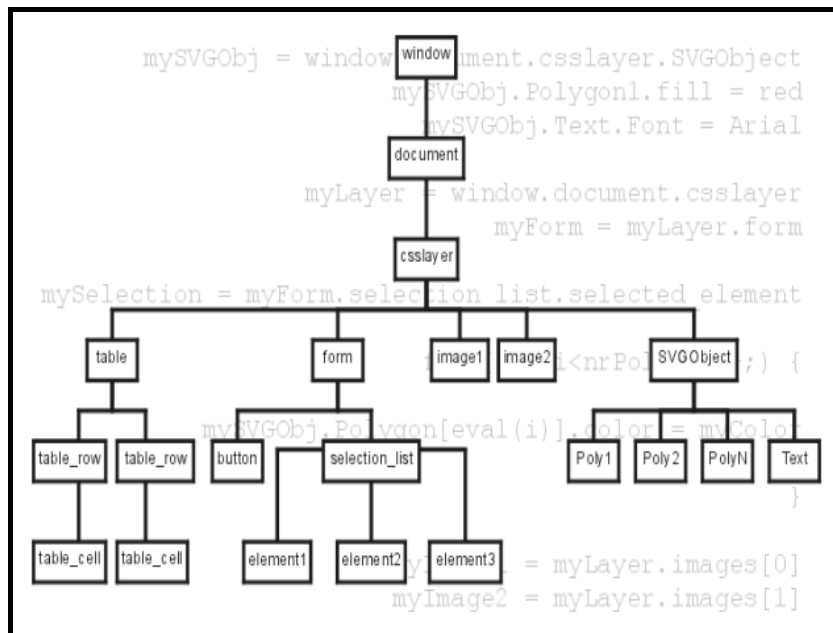


Fig.2: Example for a simple DOM object hierarchy [12]

"Interaction" requires that individual elements of a web page, including the vector objects to be represented, are responsive. None of the above proprietary approaches meet this principle. In order to reach individual objects, a clear object hierarchy is required, the uppermost member of

which is the web page. All objects within a given page may be reached via this structure. Within a conventional web page the following hierarchies may apply (simplified):

This object hierarchy is called Document Object Model (DOM) see in Fig.2. It represents the cornerstone for efficient work involving elements of a web page, as is the case with cartographic contents. A new vector standard therefore needs to go along with DOM. In principle, plugins can be integrated into DOM. The hierarchy underneath a plugin, however, is usually responsible for the good functioning of the plugin, and has not been created to facilitate communication of the plugin with its entire environment. This point, exceedingly important for interaction, frequently causes friction.

In practical terms, the DOM makes it easier to create Web pages with features that are more animated and functional. With scripts, authors can make Web pages process forms, launch pop-up menus and windows, and execute style changes on the fly. Without the DOM, authors would have to code separately for each scripting language to interact with each different browser. The DOM is meant to create a "write once, run everywhere" standard for Web page scripting, allows to access XML and thereby SVG and HTML documents. Thus SVG document can be represented as Objects. The elements of these Objects can be accessed from various scripting languages and which allows changing the attributes. The Document Object Model (DOM) for SVG, which includes the full XML DOM, allows for straightforward and efficient vector graphics animation via scripting.

In the available case SVG documents cannot call directly, they are called by embedding into HTML by an embedded element `<embed>`. The SVG documents can be accessed by following syntax:

```
Var doc = document. [ 'Name' ]. getSvgdocument( );
```

Here 'name' is designated to SVG document, which should identify the document as attribute in `<embed>`.

For accessing the elements of SVG document there are two methods:

```
GetElementById( );
```

```
GetElementByTagName( );
```

These are to be addressed to the Object, which represents the SVG document.

For example, `doc.getElementById('myRect')` returns rectangle which was specified with Id `myRect` within the document `doc`.

The Method `doc.getElementsByTagName('rect');` return's all rectangles of the document type 'rect'. The following Syntax show's how to access the attributes of the rectangles of type rect:

```
Var x = rect.getAttribute('width');
```

This function determines the width of the rectangle and the value is assigned to the variable `x`. And also one can set the attributes to the rectangle by using `set. attribute ('attribute name', 'value');` i.e.. One can set a value to a particular attribute.

### **2.3 JavaScript**

Netscape introduced JavaScript, a scripting language that is validated, compiled and executed on the fly, in the beginning of the 90's. Meanwhile it had been accepted by webmasters and programmers, and implemented in all of the major web-browsers. The syntax is closely related to C++ and Java. Though it is a language relatively easy to learn, it is a quite complex and powerful programming language. One of the not well known but existing features is object-orientation, including inheritance and encapsulation. One should not mix up JavaScript and Java –these are two completely different implementations. We use JavaScript is used for client-side dynamic content creation, interactivity, event handling and animation, mainly through the DOM.

### **2.4 Scalable Vector Graphics (SVG)**

A quantum leap in Web graphics is happening right now. A new technology defined by the W3C called SVG (Scalable Vector Graphics) is bringing rich, compelling, interactive, high-resolution graphics to the Web. This technology is particularly attractive to GIS developers and users.

Today, most mapping systems employ two approaches when delivering interactive maps on the web. The first approach is the familiar Java applet. The second approach involves generating map images on the server and delivering them to the user in either GIF or JPEG image format. Java applets perform well concerning interactivity, but have suffered from browser compatibility and firewall issues. Image maps are compatible with web tools and environments, but interaction

with the map such as zooming, panning, layer control and thematic shading necessitate a round trip to the server in order to re-render the image.

There are some more major drawbacks to the use of images for representing 2D graphics in Web pages. They are:

- **Image size:** The size of an image is defined by its width, height and the number of bits allocated to each pixel in the image. The effectiveness of compression depends on the particular compression technique used and the type of image, but for line drawings there is usually a large amount of information to transfer across the Internet. The transfer of images is a major bottleneck when accessing Web sites. It is also not possible to interact with the image without sending a new image.
- **Fixed resolution:** Once the image has been defined at a specific resolution, that is the only resolution available. Zooming into the image will not reveal more detail. In order to obtain more detail, the image has to be regenerated at a higher resolution.
- **Binary format:** Image formats are typically binary formats, which can make it difficult to embed metadata in the image format and for other components of Web technology to access metadata about the image.
- **Minimal animation:** The GIF format allows several images to be defined in one image file (so-called "animated GIFs"), but each image is a static image. To do better than this requires the use of a video format.
- **Hyper linking:** Web pages depend on hyper linking. To introduce hyper linking into images requires the use of image maps defined as part of the enclosing HTML page. An image map essentially associates hyperlinks with particular geometrical (polygonal) regions of the image. Image maps only allow links from regions of the image, not from particular components of the structure of the objects from which the image is derived.
- **Transformation:** there is increasing interest in the use of transformation techniques (based on XSLT, for example) to generate presentations of information from descriptions in higher level XML languages, for example a graphical presentation of stock market data or a tabular presentation of the same data. Since image formats are not XML-based, it is not possible to use technology such as XSLT for this purpose.

Today, SVG solves all of these problems. SVG is an open, HTTP compatible standard that allows fully interactive mapping applications - without the need for applets or a round trip to the server every time the map presentation is tweaked.

### 2.4.1 what is SVG?

The Scalable Vector Graphics (SVG) format is a new XML grammar for defining vector-based 2D graphics for the Web and other applications. The World Wide Web Consortium (W3C), the non-profit, created SVG industry-wide, open-standards consortium that created HTML and XML.

Scalable Vector Graphics (SVG) is an XML language for describing two-dimensional graphics. SVG allows for three types of graphic objects: vector graphic shapes (e.g., circles, rectangles, and paths consisting of straight lines and curves), images and text. These graphical objects can be grouped, styled, transformed and composited into previously rendered objects. The display and manipulation features include nested transformations, clipping paths, alpha masks, filter effects and template objects.

In order to view SVG documents in the Web browser, one needs to install a viewer. There are several viewers available but Adobe's SVG Viewer is probably the most commonly used and can be download from the Adobe web site.

SVG allows for three types of graphic objects:

1. Vector graphic shapes (e.g., circles, rectangles, and paths consisting of straight lines and curves).



```
<Circle id="staeid_0" cx="40" cy="40" r="20"
Style="fill:orange;opacity:1;stroke-
width:1.5;stroke:red">
```



```
<rectid="coun_1"style="fill:antiquewhite;stro
ke:#000000;stroke-width:0.13" x="10" y="15"
width="25"height="20"/>
```



```
<path id="Krei25" d="m 20 180 1 30 220 60
240 40 240 90
90"Style="fill:#ffffcc;stroke:#ff6633;stroke-
width:100;opacity:1">
```

2. Images.

3. Text.

SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively (i.e., by embedding SVG animation elements in SVG document) or via scripting.

#### **2.4.2 Client-Side Application**

Sophisticated client-side applications of SVG are made possible by use of a browser-based scripting language such as JavaScript, which accesses the SVG Document Object Model (DOM). The SVG DOM provides complete access to all elements, attributes and properties of the SVG document, after it is loaded into the browser. A rich set of event handlers such as onmouseover and onclick can be assigned to any SVG graphical object, thus bringing a very high level of interactivity to GIS applications that use SVG. The Interactivity has been lifted out as an important characteristic of this Web-Information System. With the techniques presented here, changes in the SVG documents can be accomplished exclusively at the client side. This means that no data between Client and server has to be sent, which leads to clear and better period of reply. If one uses Raster diagrams to reach an interactive behavior, even for slight changes of a map a complete new map has to be sent by the server to the Client, so there is loss of time and memory of the system.

CGI is a server-side technology. For a task at hand, such as form validation, a CGI doesn't have to do all the work involved in the process. The work could be shared with client-side technologies such as JavaScript. Moving some of the processing from the server-side to the client-side by supplementing CGIs with client-side technologies has various benefits:

- The browser, which spends most of its time idle, waiting for incoming requests. By moving some of the work onto the browser reduces the amount of work servers do, and hence the load on the server.
- If used for validation, it eliminates wasted CGI calls due to invalid input from the user. Such user-centric validation can save enormous time. The CGI itself can then be smaller. Besides validation, JavaScript could also be used for light calculations on the client-side.

All of this adds up to a fully interactive, browser-based mapping system. Once a map rendered in SVG is downloaded to the browser all types of manipulation are possible - without the need for applets or having to go back to the server for a new image.

Deploying SVG documents in the web or GIS applications is very simple. One need to make sure that the web server is set-up to support the MIME type for SVG, which is "image/svg+xml". It is recommended that SVG files have the extension ".svg" on all platforms. It is recommended that gzip-compressed SVG files have the extension ".svgz" on all platforms. (SVG supports compression, with no loss of information. And since it is loss-less, it is almost always used in delivery of SVG documents).

Finally, the SVG document is included in the HTML of the web page like this:

```
< embed src="name-of-svg-document.svgz" type="image/svg+xml"
Pluginspace=http://www.adobe.com/svg/viewer/install/height="420"
width="480" id="demo">
```

### 2.4.3 **EVENT HANDLERS**

The combination of SVG, DOM (Document Object Model), javascript and events allows the production of highly interactive web-maps.

In order to have certain pieces of code executed on clicking or moving the mouse around, one need to have a mechanism so that the code could be automatically informed of mouse activities. SVG provides event listeners. An event listener is always being focused on what is going on in SVG, and when something noteworthy happens it will tell. In this case, one needs to listen to three different events (all-mouse related):

- **mousedown** (when pushing a mouse button),
- **mousemove** (when moving the mouse), and
- **mouseup** (when releasing a mouse click).

These are the events, and the event listeners are attributes with an "on" prefix. If we want to execute an initialization script when we encounter a mousedown event on our shape, we could just write:

```
onmousedown="some_function()" .
```

An example in SVG:

```
<g id="target" onmousedown="initDrag()">
```

Then all we have to do is implement all we need done for initialization purposes in the `initDrag()` function .

Similarly mousemove and mouseup events:

```
<g id="background" onmousemove="drag()" onmouseup="endDrag()"
style="pointer-events: none;">
```

## 2.4.4 SVG DOCUMENT STRUCTURE

There are a couple of ways SVG can be defined in a Web document: as a standalone SVG page, as an embedded element, or it can be utilized in an XHTML document with a namespace declaration. Let us begin by taking a look at the **standalone** example:

### 2.4.4.1 Standalone SVG

1. `<?xml version="1.0" standalone="no"?>`
2. `<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"`  
`"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">`
3. `<svg width="300" height="300" x="0" y="0">`
4. `<!--Your SVG content goes here -->`
5. `</svg>`

- **Line 1:** Since SVG is an application of XML it must include the initial XML declaration `<?xml version="1.0" standalone="no"?>`
- **Line 2:** SVG must be identified by a standard set of rules. These rules are stored in a separate document called a **Document Type Declaration** or **DTD** and is utilized to validate the accuracy of the SVG document structure. The purpose of a DTD is to describe in precise terms the language and syntax allowed in SVG.
- **Line 3:** The `<svg>` tag denotes to the browser that this is a SVG document. The canvas of the SVG document is defined by the width and height properties. For example, increasing width and height to 500 respectively increases the size of the canvas where the content will be contained. Not defining the width and the height properties cause the SVG canvas to fill the browser dimensions. The `x` and `y` properties denote where the canvas will be placed in the browser window. The `x` property equates to the top position of the browser and the `y`



property equates to the left position of the browser. One can also use percentages to achieve a liquid page design with SVG.

- **Line 4:** All the SVG content is placed between the `<svg>` `</svg>` tags. This will become increasingly clearer as we work through some examples.
- **Line 5:** Since SVG is an application of XML, all tags must be closed. The `</svg>` tag closes the document.

This method is useful for providing standalone examples. There is a provision for Meta tags in the SVG specifications, but since SVG is XML based most of the popular search engines will not pick up a standalone SVG page. RDF enabled search engines will however pick up SVG. Nonetheless, for most of us, being listed by search engines is important and to overcome this problem we can use a combination of HTML/XHTML and SVG.

#### **2.4.4.2 Embedded SVG**

SVG can be embedded within a HTML or XHTML document by using the following structure:

```
1. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2. <html>
3. <head>
4. <title>Object and Embed</title>
5. </head>
6. <body>
7.<object data="test.svg"width="500" height="500" type="image/svg+xml">
8.<embed src="test.svg" width="500" height="500"type="image/svg+xml" />
9. </object>
10. </body>
11. </html>
```

The document is a straightforward HTML document. The important tags in the above example are the **object** and **embed** tags. If strictly adherence to standards based coding is needed only use the **object** tag, but using this tag *only* causes the SVG file to not appear in Netscape version browsers. As a consequence it is best to use both the **object** and **embed** tags or just the `embed` tag. Lines 7 through 9 contain the appropriate **object** and **embed** tags. It is important to note that the `object` tag uses the `data` property to specify the url of the SVG document while the `embed` tag employs the `src` property.

One of the benefits of utilizing this method is that it is able to combine the advantages of HTML and XHTML with SVG.

#### **2.4.5 SVG FEATURES**

Some of the important Features of SVG are:

- **Plain text format** -- SVG files can be read and modified by a range of tools, and are usually much smaller and more compressible than comparable JPEG or GIF images. Additionally since it is based on XML and is entirely text-based maps created in SVG are open for search engines to index.
- **Scalable** -- Unlike bitmapped GIF and JPEG formats, SVG is a vector format, which means SVG images can be printed with high quality at any resolution, without the "staircase" effects seen when printing bitmapped images.
- **Panning/Zooming** -- One can pan and zoom very quickly in an SVG image and not see any degradation of the image, and the system does not need to go back to the server for a new image.
- **Searchable and selectable text** -- Unlike in bitmapped images, text in SVG text is selectable and searchable. For example, One can search for specific text strings, like city names in a map.
- **Styles.** SVG increases the ability to apply different styles. With vector graphics, one can apply different styles (as opposed to raster graphical images which are essentially "frozen"). For example, the text in diagrams can be set to a particular font or color. Cascading Style Sheets (CSS) may be used to alter the look of the graphics, for example, to control the font and positioning of text, color and other aspects of a presentation.
- **Interaction** -- SVG enables dynamic and interactive graphics far more sophisticated than bitmapped solutions. Map features may be made selectable and interactive from within the browser. The user may theme a map, select objects, and manipulates layers - all without a network hop back to the server.
- **Animation** -- SVG enables animation from within language definition itself, and through the combination of JavaScript and the SVG DOM.

- **Open standard** -- SVG is an open recommendation developed by a cross-industry consortium. Unlike some other graphics formats and map file formats, SVG is not proprietary.
- **True XML** -- As an XML grammar, SVG offers all the advantages of XML
- **Open Source Solution** -- SVG can be applied to GIS projects, using entirely open source tools, thus reducing the cost of implementation.

As an XML grammar, SVG offers all the advantages of XML:

- Interoperability
- Internationalization (Unicode support)
- Wide tool support
- Easy manipulation through standard APIs, such as the Document Object Model (DOM) API
- Easy transformation through XML Stylesheet Language Transformation (XSLT).

#### **2.4.6 THE ADVANTAGES OF AN XML-BASED VECTOR GRAPHICS LANGUAGE**

- XML offers greater structural control and sophistication than HTML. SVG is entirely XML-based, a fact that offers many advantages to developers and users alike. As Web sites grow more complex, offering more interactivity and visual sophistication, Web designers require languages more powerful than HTML.
- Extensible Markup Language (XML) has garnered widespread, enthusiastic support from Web industry leaders and developers alike.
- Unlike HTML, XML completely separates content from its presentation. By describing information in terms of structured data types, user applications can process the same XML page as display or data - not just display, as with HTML. For example, an XML-tagged phone number could be dialed on a cell phone, but rendered as an SVG image of the business it belongs to on a neighborhood map depending on the device and application processing the file. This type of flexibility is light-years ahead of HTML.

#### **2.4.7 THE BENEFITS OF A TEXT-BASED VECTOR FORMAT FOR THE WEB**

- SVG and XML are entirely text-based, a fact that offers many advantages to developers and users alike: An SVG image is composed of XML-based commands entered as text, which can be incorporated directly in an HTML or XML page. It's lean, smart code.
- As a text-based format, text within SVG images can be indexed by search engines or searched within a browser by users.
- SVG can also be created on the fly, by any scripting language - such as JavaScript, Perl, or Java - with data from any source from a relational database to an ASCII file, an ideal way to create high quality charts and graphs that update dynamically and allow users zoom, pan, or even do ad hoc reports.
- SVG fully supports the DOM (Document Object Model) and therefore is fully scriptable. SVG images or portions of an SVG image can react to clicks and mouse movement, triggering changes to the graphic itself or to other objects on the same page such as HTML text and other graphics.
- SVG works well across platforms, output resolutions, color spaces, and a range of bandwidths.

#### **2.4.8 Dynamic theme generation:**

The user can "re-theme" the map as often as they like - and never have to go back to the server for a new image. The user can change data sets, change colors, change buckets, and change from count to range algorithms right in the safety of their own browser.

- 1. Layer control:** Layers are controlled from within the browser. Streets, labels, and boundaries may be toggled on and off through the JavaScript controls included on the page.
- 2. Pan and Zoom:** The map view port may be changed dynamically, a new image is never required and the quality of the pan and zoom is outstanding.
- 3. Text Placement and Sizing:** Labels may be moved and sized by the user dynamically.

### 2.4.9 Coordinate Systems and Transformations

SVG is using a Cartesian coordinate system, the outset of which is located in the upper left corner of the drawing level. If using SVG for map representation, one has to invert the y-axis. Either put a minus in front of the y-value, or create a group around the objects to be transformed and assign an inversion to it by a matrix operation. With the help of this 3x3 matrix operation, the elementary geometrical transformations (translate, scale, rotate, cut) are applied singularly or combined, to groups of ones choice, or to the whole SVG realm. Transformations may be interlocked as one pleases, which means one can get any number of local coordinate systems.

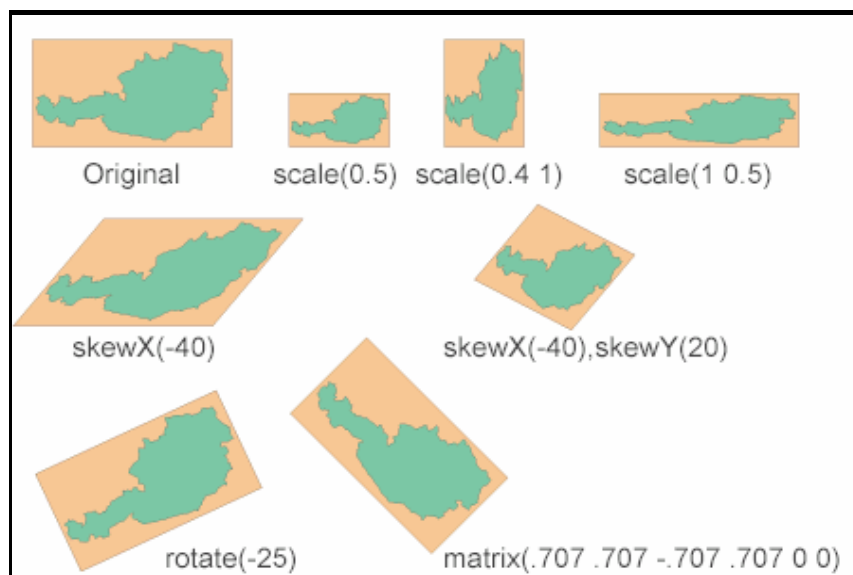


Fig 3.[15]Coordinate System Transformation

Each element and group may be transformed: translated, resized, rotated and skewed. Transformations may be nested or defined using transformation matrices. This way, it is possible to do all affined transformations.

The above example **Fig 3** deals with the same graphics: one object - defined once - with different transformations.

- In the first row scaling with one or two independent factors is used. The latter leads to distortion in one dimension.
- For the maps in the second row skewing with one or two independent factors. One has to use decimal degrees is used.

- In the last row rotations are used. The last example shows an alternative syntax: all statements can be written by using a transformation matrix. Combinations can be accumulated using the rules of matrix calculations. Graphics software exports SVG-code in this manner, resulting in shorter code fragments - however manual interpretation is getting more difficult by using this approach.

To run a representation, one also needs a definition of the viewbox within a web page, or the visible area (view port) of an SVG file respectively. The viewport may be pre-defined in the file, or it may also be modified in an interactive way, by zooming or panning without user action. For example, you may link a survey map on canvas #1 with a larger map section in another canvas, in order to make it easier to choose a section and to zoom in on it - without losing overview. Viewboxes allow the use of true-size coordinates, since conversion parameters for the monitor coordinates are jointly set. Distinction needs to be made between the "viewport-coordinate-system" and the "user space coordinate system" (monitor or printer coordinates, for instance). Viewboxes may be interlocked as one pleases.

#### **2.4.10 Clipping and Masking**

All SVG objects may be clipped and masked; respectively they may serve as a "clipping-path", which includes raster images and text also. Masks can be inserted along the margin of a map for instance, in order to exclude areas. Just as is the case with a path, here too, in case of complex clipping polygons the rule for inlaying and/or cross-sectional polygons needs to be indicated.



*Fig 4 Original image*

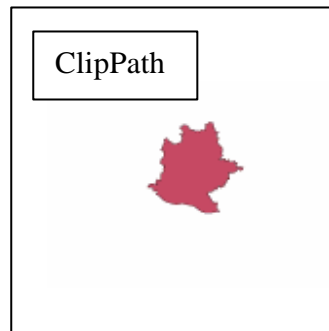


Fig .5 Clipped Region of Stuttgart

A **clipped** region. A closed path from the "Baden-Württemberg" Map in *Fig4* and *Fig5*. The rest of the map outside the clipping-path is never shown, even with sequential loading. Clipping doesn't allow transparency (only value 1 or 0). The background shines through.



Fig.6 Masking

The same closed path as above is colored red and used as a **mask** in *Fig6*. Opacity is 20 percent. Every object, which means also text or a raster image, can work as a mask.

#### **2.4.11 Basic geometric elements**

Basic SVG-geometry includes rectangles, circles, ellipses, lines, polylines and polygons. Of particular interest is the path-element. Its syntax is related to PostScript or PDF: through commands such as "moveTo" or "lineTo" the path is drawn up. In order to suffice to all demands, cubical and square bezier curves and elliptic curve elements may also form a path. It is possible to have holes in a path-object and to combine disjunct path-elements to one single path. For this purpose it is important to specify the right rule for complex polygons using holes: even-odd or nonzero-winding. One can use either absolute or relative coordinates - using relative coordinates often helps to keep file-sizes small. Path elements can also be used for generating pie-slice diagrams, as it is often the case in thematic maps.

#### **2.4.12 Text**

SVG allows for just about any parameters in generating text elements, such as: font family, font type, size, position, width, direction, inclination, etc. - actually all the parameters known by DTP and cartography. An exciting option for cartographs should be the possibility to align text along path elements. Unfortunately, SVG in its present specification does not know continuous text. In order to properly represent non-latin or self-made fonts as well (e.g. for symbols or linear elements, like edges of embankments), SVG allows for external font description files, or for replacement of missing Unicode characters by supplementary graphics or symbols. Entire fonts may also be embedded into an SVG file.

Regarding text formatting, all CSS rules apply, just like with HTML4. Individual characters can be wrought in their orientation. Direction can be chosen, in view of some non-latin alphabets, and within a row of characters both directions, as well as vertical rows, are represented correctly. Like HTML4, SVG supports Unicode, the international standard for coding all characters of this world.

An outstanding feature of SVG text implementation is that text elements are indexable by search engines, since the whole graphics is based on XML. Within the plugin, you can also search for text and highlight it. If the found text is situated outside the present viewport, the latter is adapted accordingly. Automatically centering does not occur, in order to minimize irritating viewport changes.



### **2.4.13 Colors, Fill Samples, Color Courses, Transparencies, Line Types**

Color values are defined in RGB standard. This is a particular color domain for the Internet with parameters specific to display devices. As with HTML4, values are determined in hexadecimals. Each and every filling, even lines and texts, may take on a transparency value. Lines may carry the following attributes: thickness, line end type, vertex markers, broken line mode, broken line offset, etc. On both ends, markers (such as arrows) may be allocated. As a special feature, markers may also be allocated to every vertex, which is helpful with minor angle changes.

As to color courses, linear and radial courses are supported, with any number of interval and stop values. The spreading method, too, can be controlled partially. A trail may be assigned even to opacity. Patterns may be raster and vector based. Those patterns are tiled. A given tile has to be defined once and is repeated according to tile size. Offset values can be defined as well.

### **2.4.14 Interaction, Scripting**

Basic interaction features of an SVG viewer are zooming, panning, return to original view, and a display print option. Adobe's viewer implemented an additional text search function, an anti aliasing switch option, a copy feature, and a source code viewer. They are planning an option to choose a mouse cursor from a set, resp. to add a self-made cursor image. Together with possible events, simple key functions may be implemented, e.g. in order to access certain cartographical applications, such as distance and surface measurement.

Hyper links, as with HTML, may be used to refer to other files, or to other elements within an SVG document. You can also refer to pre-defined "view-elements".

In SVG, three event categories are at disposition: mouse events, keyboard events, and state change events (concerning display and SVG file loading state). Event handling occurs analogous to HTML4. Since all possible events may be combined, almost everything is possible. Complex applications may be realized combining scripts and Java applets. By JavaScript, the developer is able to fully access SVG's and every browser feature's DOM. Here it must be remarked, however, that many of those functions are not yet ready, because not all DOM domains within Adobe's SVG plugin are implemented so far.

Cartographical applications range from simple switching on and off of elements and layers, changing graphical attributes, reacting to mouse events, such as displaying object data when mouse-over,

linked windows (combining various views, e.g. overview and main map), interactive moving, scaling and rotating elements (e.g. didactical puzzles), or small applications, such as a user option to digitize and to save the data on the server in view of further editing and storage. Finally, with the help of server CGI scripts, or Java applets, and/or Java servlets, databases can be linked as well. Countless applications may be envisioned, that are bound to enhance cartographer's creativity in future web projects.

#### **2.4.15 Metadata and Extensibility**

Metadata serves to deposit information on the document in a structured way. Data concerning authorship, date of publication, version, title, brief description, etc. is inserted. Those may be embedded into an SVG file, just as external code (e.g. JavaScript)

As with VRML, SVG offers the possibility to re-use down the track in a given file elements described earlier, which can be identified explicitly by ID. Thus one may pre-define more or less complicated graphical objects and groups, in order to re-use them in a modified way, e.g. after transformation. In cartography, this feature allows placement of symbols or pictograms.

Extensibility is one of the most important features of SVG, as is the case with any DOM/XML compatible specification. Since SVG itself is defined in XML, any other standard, which is equally defined in XML, such as MathML, XHTML, SMIL, and many others, can be embedded and accessed in SVG. For example, you could deposit statistical data of polygons, which are part of a thematic map, in XML. Then a JavaScript function would read and process those values in an appropriate way, e.g. as a map diagram to be represented in SVG. Next to this, as an optional service, values could be displayed in a HTML table.

SVG is a powerful technology when applied to GIS applications. Mapping represents a perfect application for SVG technology. Maps are by nature vector layered representations of 2D space. The SVG specification embodies the same layering concepts that are fundamental to GIS to the presentation of that data.

SVG enables maps that are informative and interactive -without the problems inherent in other technologies such as Java applets. SVG empowers distributed GIS processing, by off-loading the interactive aspects of GIS from the server to the client and combines this interactivity with very high quality output capabilities.

SVG is not a replacement for all the capabilities of a full GIS system. Rather, it represents a new way to present quality geographic information to any user on the web.

## **2.5 SERVER SIDE TECHNIC**

### **2.5.1 WEB SERVER:**

For the structure of the Web-Information System the employment of a Web Server is necessary. The usual task of a Web Server is to convey Internet side request by the Client in Internet Browser over the URL to the Client. Beyond that it should be able to implement programs, those server-laterally, thus on the same computer as the server is installed. A server should allows one to easily set up password-protected pages with enormous numbers of authorized users, without bogging down the server. For the Web-Information System the 'HTTP (web) Server Used is Apache Web Server.

### **Apache Web Server**

Why Apache?

Since, Apache fulfills the above conditions. We installed Apache web server was installed because.

- It is a powerful, flexible, HTTP/1.1 compliant web server
- Implements the latest protocols, including HTTP/1.1 (RFC2616)
- Is highly configurable and extensible with third-party modules
- Can be customized by writing 'modules' using the Apache module API
- Provides full source code and comes with an unrestrictive license
- Runs on Windows NT/9x, Netware 5.x and above, OS/2, and most versions of Unix, as well as several other operating systems
- Is actively being developed
- Encourages user feedback through new ideas, bug reports and patches
- Implements many frequently requested features, including:

### **DBM databases for authentication**

Allow one to easily set up password-protected pages with enormous numbers of authorized users, without bogging down the server.

### **Customized responses to errors and problems**

Allows you to set up files, or even CGI scripts, which are returned by the server in response to errors and problems, e.g. setup a script to intercept **500 Server Errors** and perform on-the-fly diagnostics for both users and yourself.

### **Multiple Directory Index directives**

Allows one to say **DirectoryIndex index.html index.cgi**, which instructs the server to either send back `index.html` or run `index.cgi` when a directory URL is requested, whichever it finds in the directory.

### **Unlimited flexible URL rewriting and aliasing**

Apache one no fixed limit on the numbers of Aliases and Redirects, which may be declared in the config files. In addition, a powerful rewriting engine can be used to solve most URL manipulation problems.

### **Content negotiation**

The ability to automatically serve clients of varying sophistication and HTML level compliance, with documents which offer the best representation of information that the client is capable of accepting.

### **Virtual Hosts**

A much requested feature, sometimes known as multi-homed servers. This allows the server to distinguish between requests made to different IP addresses or names (mapped to the same machine). Apache also offers dynamically configurable mass-virtual hosting.

### **Configurable Reliable Piped Logs**

One can configure Apache to generate logs in the format that one wants. In addition, on most Unix architectures, Apache can send log files to a pipe, allowing for log rotation, hit filtering, real-time splitting of multiple hosts into separate logs, and asynchronous DNS resolving on the fly.

## **2.6 Server side Scripts, Java Servlets and CGI**

Server side Scripts may be written in any scripting or programming language. Popular languages include Perl, PHP, Python, C and Java. They are a great way to use databases or any external scriptable application to deliver dynamic content to the client. Only the processed output is sent and not the original data. One of the advantages when using server side-scripts, is the relative independence of a clients browser capabilities and version, because the interactive part is provided by the server. On the other hand the drawbacks are longer time delays, when doing requests, and the heavy load produced on the server-side. The architecture of server-side techniques is quite open and extensible.

## **2.7 COMMON GATEWAY INTERFACE**

The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon **retrieves** is **static**, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is **executed** in real-time, so that it can output **dynamic** information. The common gateway interface (CGI) is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user. When the user requests a Web page (for example, by clicking on a highlighted word or entering a Web site address), the server sends back the requested page. However, when a user fills out a form on a Web page and sends it in, it usually needs to be processed by an application program. The Web server typically passes the form information to a small application program that processes the data and may send back a confirmation message. This method or convention for passing data back and forth between the server and the application is called the common gateway interface (CGI). It is part of the Web's Hypertext Transfer Protocol.

In simple, applications of HTML communication runs off relatively on one side between Server and Client, the Client has however no possibility to change the contents of the Documents. CGI permits to implement programs Over a Web Server. CGI programs can return a myriad of document types. They can send back an image to the client, and HTML document, a plaintext document, or perhaps even an audio clip. They can also return references to other documents.

The client must know what kind of document one is sending it so it can present it accordingly. In order for the client to know this, your CGI program must tell the server what type of document it is returning. Thus this technology permits a substantially higher flexibility and interaction compared with Client-Side technology.

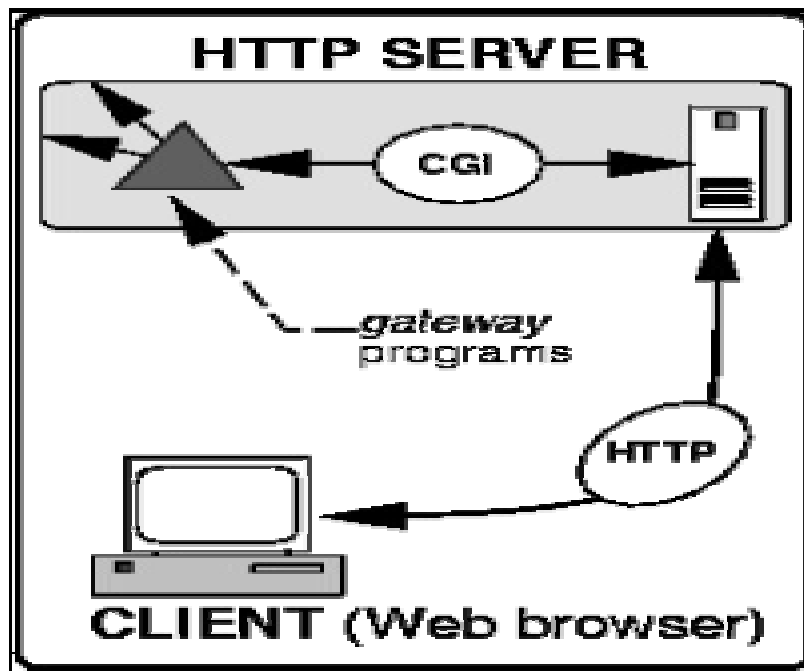


Fig7. [16] Flow of Data to Gateway Programs

This Figure7 illustrates the flow of data when a user accesses a CGI program. The solid line shows to data flow using HTTP and CGI. HTTP transfers data from the client to the HTTP server, and back again. The CGI mechanisms control the flow of data from the server to the gateway program (shown as the prism) and back again. These are called *gateway* programs because they generally act as gateways between the World Wide Web and server-side resources such as databases, feedback forms, clickable image maps, and so on.

A CGI program can be written in any language that allows it to be executed on the system, such as:

- C/C++
- Fortran
- PERL
- TCL

- Any Unix shell
- Visual Basic
- Apple Script

In this Web-Information System PERL is used as CGI program because it is freely available software and has problem free-installation. Perl's process, file, and text manipulation facilities make it particularly well-suited for tasks involving quick prototyping, system utilities, software tools, system management tasks, database access, graphical programming, networking, and world wide web programming. It can execute Database queries. Since PERL from many reasons for it's performance it is having very limited number of pre-defined functions, In order to make Database queries we need to install two modules, they are DBI (Database Interface, a common interface for talking to different databases.) and DBD-ODBC (Database Driver, allows the DBI interface to communicate with a specific database server.), which can be downloaded freely from <http://www.activestate.com/PPMPackages/zips/5xx-builds-only/>. Install these modules should be installed after installing ActivePerl, which can also be downloaded from ([http://www.activestate.com/products/Download/Get.plex? Id = ActivePerl&a=e](http://www.activestate.com/products/Download/Get.plex?Id=ActivePerl&a=e))

These modules can be accessed over the Database interface ODBC (Open Database Connectivity).

## **2.8 Standards and Open Source Technology**

It is important that cartographers stick to official and open standards, when presenting cartographic information on the Internet. Open standards guarantee a longer life cycle and an easier change to other software products, both on client and server side. There are several standardization organizations, f.e. ISO, ANSI, ECMA, etc. Most relevant for web- Publishers and Internet software developers are the recommendations worked out and published by the W3 ORG (WWW consortium) - a consortium consisting of universities, institutions and companies. Standards, published so far, include several XML and XML-related base-standards, web graphics, multimedia, payment, security, accessibility,

Internationalization, etc. For almost all web-techniques, both client and server, one can find open-source solutions. Well-established open source projects not only save time and money but also are in most cases better tested and documented. Recent studies have shown that security-

related patches are usually quicker released for open-source software than for commercial one. Some open source packages such as Apache (Web server), Tomcat (Java Servlets), Linux and FreeBSD (Operating-systems), Perl, PHP, Python (Scripting Languages), Progress and MySQL (databases), bind (DNS-Server), Gnu-Compilers, Routers, etc. are widely used and build the base of the Internet infrastructure.

## **2.9 SVG and GML**

The Geography Markup Language (GML) is an XML encoding for the transport and storage of geographic information, including both the spatial and non-spatial properties of geographic features. It was developed by OpenGIS Consortium (OGC) and is different to the SVG standards. It exclusively concerns with storage and transportation of geographical data. The basic element for the representation of geographical contents is <feature> element. Within this element both spatial and geometrical data is stored. GML is concerned with the representation of the geographic data content.

Of course we can also use GML to make maps. This might be accomplished by developing a rendering tool to interpret GML data, however, this would go against the GML approach to standardization, and to the separation of content and presentation. To make a map from GML we need only to style the GML elements into a form, which can be interpreted for graphical display in a web browser. Potential graphical display formats include W3C Scalable Vector Graphics (SVG), the Microsoft Vector Markup Language (VML), and the X3D. A map styler is thus used to locate GML elements and interpret them using particular graphical styles.

GML can play a part in geographic data exchange in a number of ways:

- As an "offline" exchange format (downloadable files or on CD)
- As an "online" format for retrieving and maybe even updating spatial data. Another OpenGIS initiative is important here: the Web Feature Server specification. The Web server outputs GML as response to a client request.

We chose XSLT for the transformation of GML into SVG. Some components of the GML viewer were hard coded: the rendering order of the map layers and the styling parameters (color, fill, line width). For the "identify" operation the original GML data is queried with the help of a unique id.



The next step will be to make a viewer that will consist of a generic 'core' in combination with some interactive modules that make it possible for users to change the rendering order of the map layers, and to change the classification and/or styling attributes for feature sets.

It proved not "trivial" to write transformation software that can transform GML into SVG regardless of its data structure. This is because of the fact that feature and attribute names are not fixed, but vary from organization to organization. Also the hierarchical structure (nesting of features and feature collections) is not the same for every GML implementation.

The XML application schema(s) that describe the structure of the GML play a crucial role in this respect: the transformation software will have to read the implementation specific XML Schema document or documents first, before it can 'understand' and transform the GML data.

## **CHAPTER 3**

### **Installation of Web-Information System**

In context to my thesis I have used Web-Information system that was developed by FAW Research institute, ULM.

#### **3.1 Basic requirements**

The basic requirements for installing Web-Information system are:

1. Apache Web Server.
2. ActivePerl
3. Perl Modules DBI and DBD-ODBC
4. Microsoft Internet Explorer
5. Adobe SVG Viewer.

Minimum disc space required to install Web-Information System is 70MB.

#### **3.1.1 Apache Web Server**

- Download the Apache Web Server from <http://www.apache.org/dist/httpd/binaries/win32/>

From that website search for the file “apache\_1.3.27-win32-x86-no\_src.exe”

- a) Install these files on C drive for example: C:\Sudhir-Thesis\Apache Group\Apache
- b) Select the sub folder “conf” from the Apache folder and open the file “httpd.conf” with a text editor. Scroll to the end of “httpd.conf” and insert a new line

**Include** C:/Sudhir-Thesis/Wis/conf/wis-apache.conf

**Note:** If necessary change the Drive name if you are installing Apache in different drive.

- c) Open C:/**Sudhir-Thesis/Wis/conf/wis-apache.conf** with the editor. If necessary change the drive name.
- d) Open C:/**Sudhir-Thesis/Wis/cgi-bin/.htaccess** file with an editor, if necessary change the drive names.

### **3.1.2 Adobe SVG Viewer**

**Installing SVG-Plugin:** To view SVG on Internet Browser we need to have SVG-Plugin, which can be downloaded from

<http://www.adobe.com/svg/viewer/install/main.html>

The minimum System requirements for installing Adobe SVG Viewer are:

#### **1. Windows**

- a. Windows 95, 98, SE, 2000, ME, XP, or NT 4.0 Service Pack 4 and up.
- b. Netscape Navigator or Communicator versions 4.0 through 4.75, or Internet Explorer 4.0 or higher. Netscape 6 is not supported.
- c. 13 MB of hard disk space.
- d. 32 MB of RAM recommended.

#### **2. Macintosh**

- a. System 8.6 through 9.2, or 10.1 (not 10.0 through 10.0.4)
- b. Netscape Navigator or Communicator versions 4.5 through 4.78, or Internet Explorer 5.0 or higher (un-scripted SVG only). Netscape 6 is not supported.
- c. 10 MB of hard disk space
- d. 48 MB of RAM recommended.

Install SVGView.exe file.

### **3.1.3 ActivePerl**

ActivePerl is Active State's quality-assured distribution of Perl, available for Linux, Solaris, and Windows. Download ActivePerl from

<http://www.activestate.com/Products/ActivePerl/>

ActivePerl contains the Perl language, the Perl Package Manager, (for installing CPAN packages), and complete online help.

The Windows package provides additional features to take advantage of that platform.

After downloading ActivePerl from activestate website install **ActivePerl-5.6.1.631-MSWin32-x86.mis** on to a separate folder in e.g., 'C:\Sudhir-Thesis\' drive

E.g. C:\Sudhir-Thesis\Perl.

### 3.1.4 Perl Modules DBI and DBD-ODBC

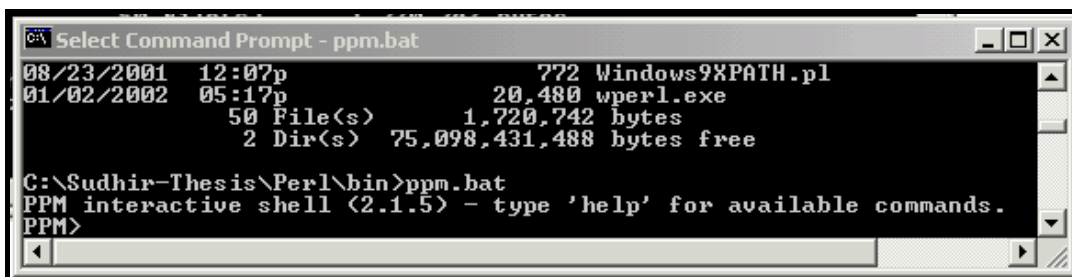
Since PERL from many reasons for its performance it is having very limited number of pre-defined functions, In order to make Database queries we need to install two modules they are DBI (Database Interface, a common interface for talking to different databases.) and DBD-ODBC (Database Driver, allows the DBI interface to communicate with a specific database server.), which can be downloaded freely from <http://www.activestate.com/PPMPackages/zips/5xx-builds-only/>.

Steps for Installing DBI and DBD-ODBC

1. Go to MSDOS-Prompt change to Perl-subdirectory to "C:\Sudhir-Thesis\Perl"

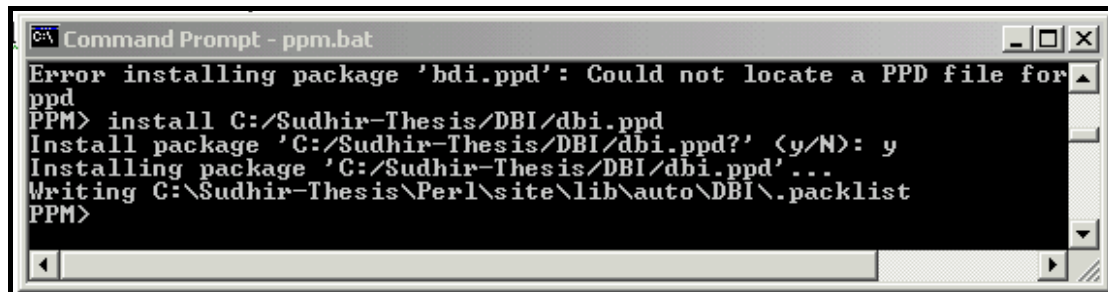


2. Execute file "ppm.bat"



3. Install "DBI-Subdirectory"

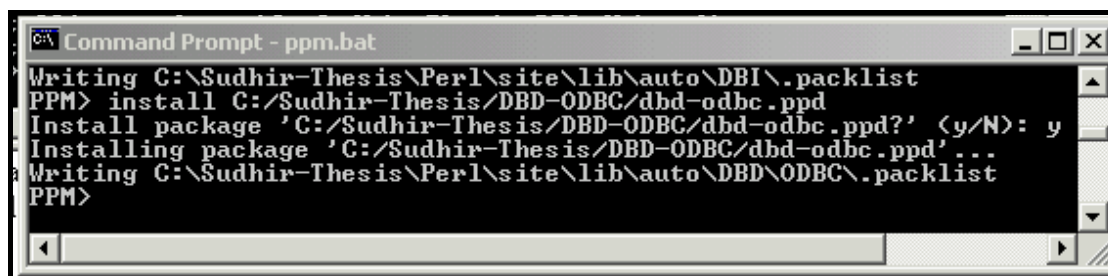
-Install C:/Sudhir-Thesis/DBI/dbi.ppd



```
Command Prompt - ppm.bat
Error installing package 'bdi.ppd': Could not locate a PPD file for
ppd
PPM> install C:/Sudhir-Thesis/DBI/dbi.ppd
Install package 'C:/Sudhir-Thesis/DBI/dbi.ppd?' (y/N): y
Installing package 'C:/Sudhir-Thesis/DBI/dbi.ppd'...
Writing C:/Sudhir-Thesis/Perl/site/lib/auto/DBI/.packlist
PPM>
```

4. Similarly install “DBD-ODBC Subdirectory”

-Install C:/Sudhir-Thesis/DBD-ODBC/dbd-odbc.ppd



```
Command Prompt - ppm.bat
Writing C:/Sudhir-Thesis/Perl/site/lib/auto/DBI/.packlist
PPM> install C:/Sudhir-Thesis/DBD-ODBC/dbd-odbc.ppd
Install package 'C:/Sudhir-Thesis/DBD-ODBC/dbd-odbc.ppd?' (y/N): y
Installing package 'C:/Sudhir-Thesis/DBD-ODBC/dbd-odbc.ppd'...
Writing C:/Sudhir-Thesis/Perl/site/lib/auto/DBD/ODBC/.packlist
PPM>
```

5. Close PPM by giving quit command and close MSDOS-Prompt window

### 3.1.5 Creating an ODBC-Connection

- Create a Folder database in “C:\Sudhir-Thesis\Wis\” copy wis-database.mdb and Geoportalz.mdb into that Folder .
- Change to “system control” and open “ODBC-databases”
- Click on “System-DSN” and “Add...” an “Microsoft-Access-Driver”
- Give Data source name as wis-database.mdb and select the database wis-database.mdb from “C:\Sudhir-Thesis\Wis\database” similarly do for Geoportalz.mdb.

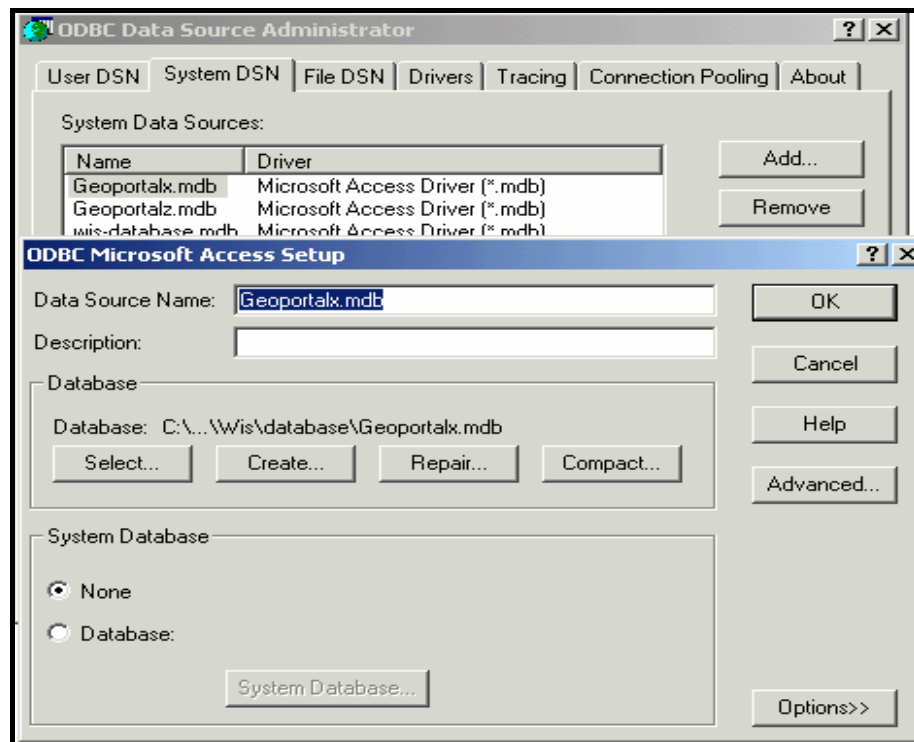


Fig.ODBC Connection

1. Create a New Folder htdocs in "C:\Sudhir-Thesis\Wis" save all .SVG ,javascript file and html files in htdocs.

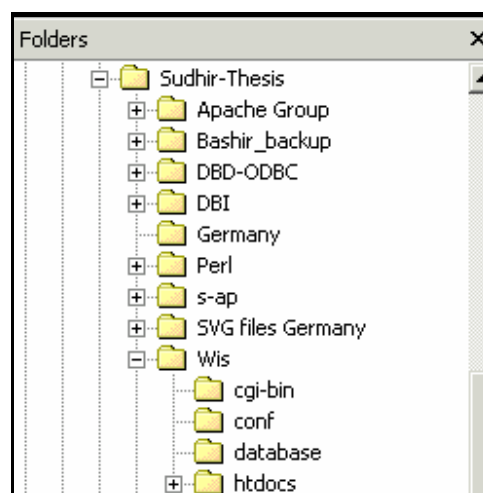
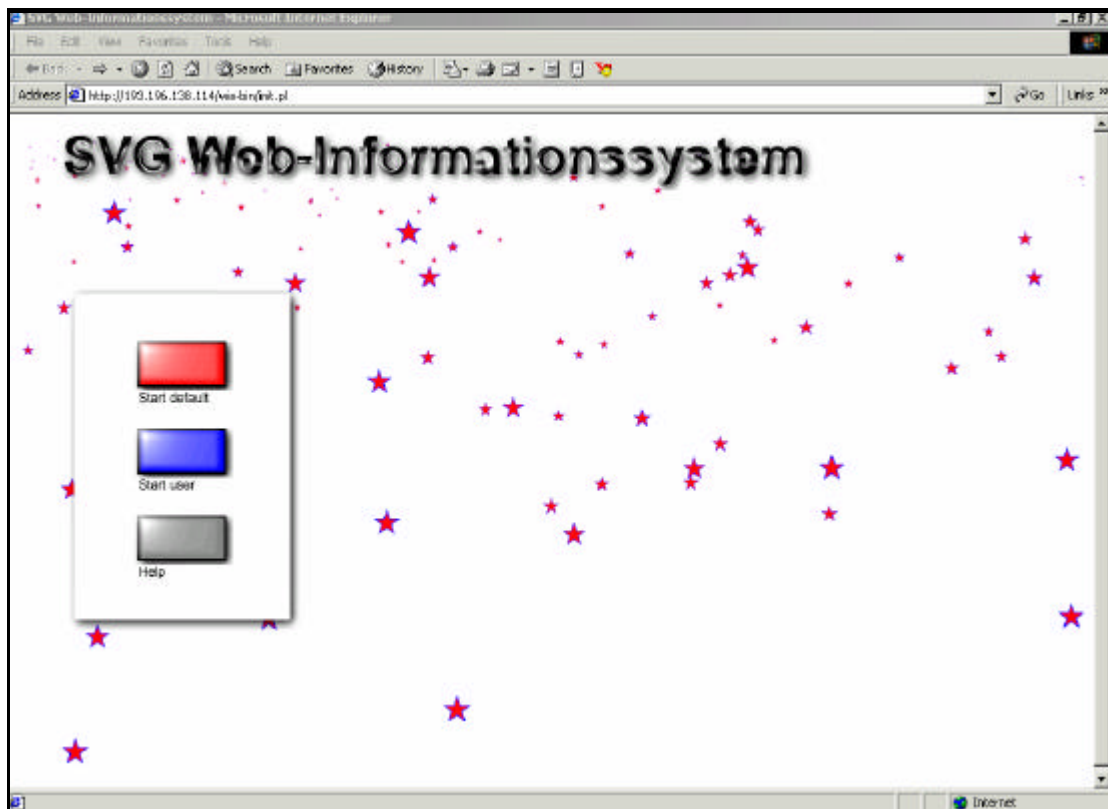


Fig. Directory structure

2. Create a Folder cgi-bin and Store all .pl files in that Folder.

**Note:** Be careful in setting the path in each perl file otherwise it will create a lot of problems

- Start the “Apache Web server” (start-programs-Apache Web-Server-Start Apache)
- Start Microsoft Internet Explorer.
- Enter: <http://193.196.138.114/wis-bin/init.pl>

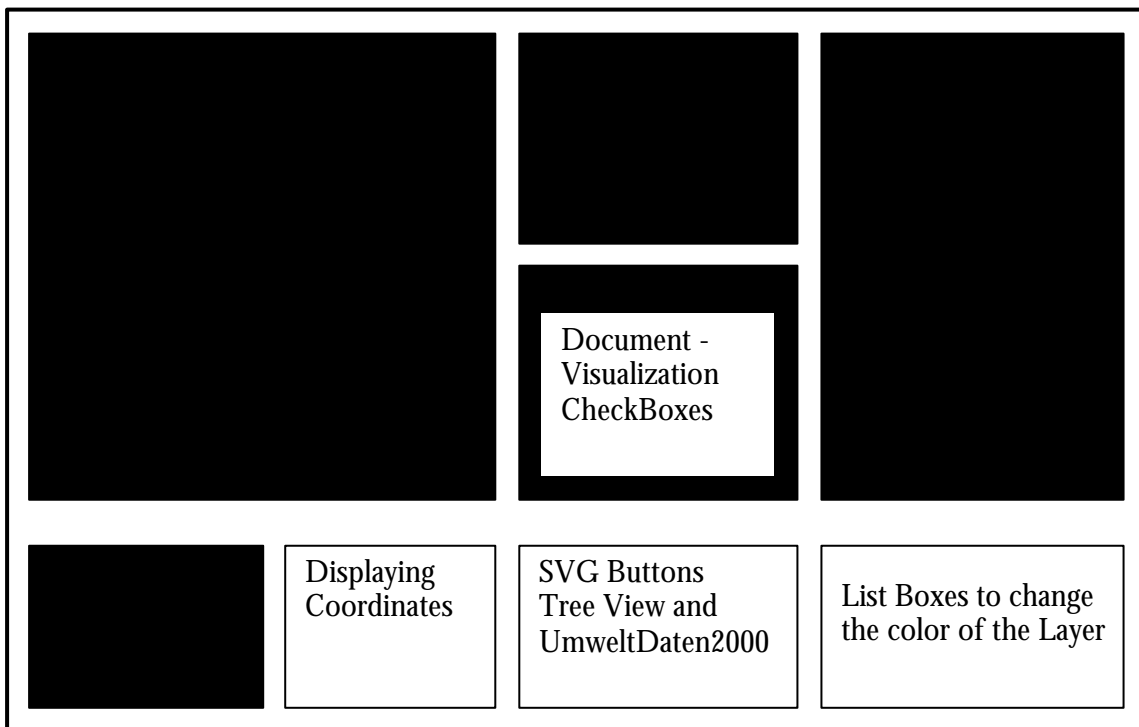


*Fig. Shows the Home page of SVG Web-Information system.*

### **3.2 GUI - Graphical User Interface**

The whole webpage is filled by a "scalable" SVG with 100% width and height in the outer SVG element. This means that the whole user-interface is "scalable" to a certain extent. Of course the

text-size would be too small if the user resizes to a very small window. The outer SVG's coordinate system uses screen coordinates with a 4:3 ratio. The main map and a small linked reference map for navigation purposes, where the user can zoom and pan by selecting the zoom values in the List Box. An additional section beside the main map serves as an "info panel" to show additional data like selecting the Region boundaries, displaying names of the region, Document visualization.



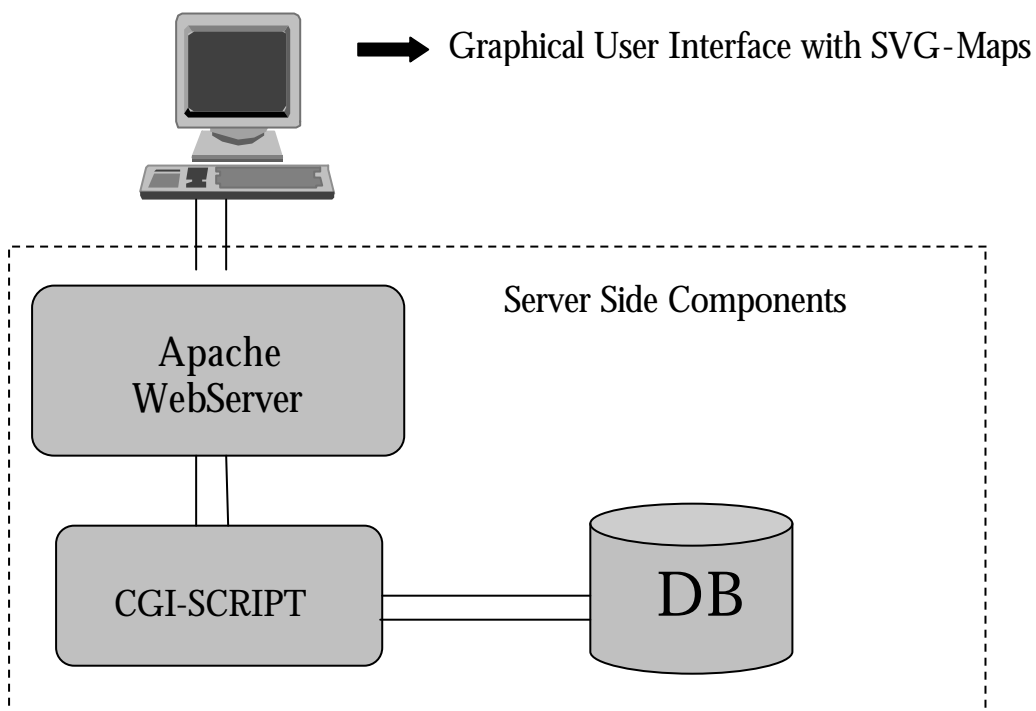
*Fig. GUI of Web-Information System*

### 3.3 System Architecture

The System architecture consists of two components Client side component and server side components. Client side consists of Graphical user interface with SVG Maps and Server side consists of an HTTP apache web server and MS ACCESS database, which are used as a background of this



System (see Fig. System Architecture). When the user performs a query, request is sent to the server, which is connected to the database via CGI-SCRIPT, it will retrieve the result from the database and sends the response back to the client in the form of an HTML page.



**Fig. System Architecture**

## **CHAPTER 4**

### **Implementation and Design of Web Information System**

#### **4.1 Generating SVG map:**

In this thesis work I got ESRI shape files of Baden-Württemberg. It has to be converted to SVG, and for this purpose Demo tools are available which can convert Shape files to SVG. These shape files can be converted into Scalable Vector Graphics (SVG) format can be used on the Web using tools such as SVG mapper and Mapview. SVG mapper tool was downloaded from <http://www.svgmapper.com> and Map view tool from <http://www.mapview.de>. Both tools offer an optically equivalent conversion of Shape Files. The only difference between these two tools is the transmission of the coordinates.

##### **4.1.1 Steps to Install SVGMapper**

SVGMapper is an extension for ArcView GIS (3.x) for exporting the View into SVG (Scalable Vector Graphics) format (which is very exciting new XML-based language for displaying vector graphics on the WWW). This is an evaluation version with some options disabled and limited in use.

### **SOFTWARE REQUIREMENTS**

For installing SVG Mapper the minimum software requirements are:

- ArcView GIS (3.x) for using the above mentioned extension
- Adobe SVG Viewer (version 2.0 or higher, free plugin - download at ([www.adobe.com/svg](http://www.adobe.com/svg)))
- WWW browser (IE 4-6 or NN 4.0-4.77)

#### **Installation:**

- Unzip the package into directory of your own choice.
- Copy file SVGMAPPER.AVX into ArcView's extension directory (..\ArcView\Ext32\).
- Start ArcView and load the extension (Menu File - Extensions and click on check box next to SVGMapper 1.3.9 as shown in the Fig)

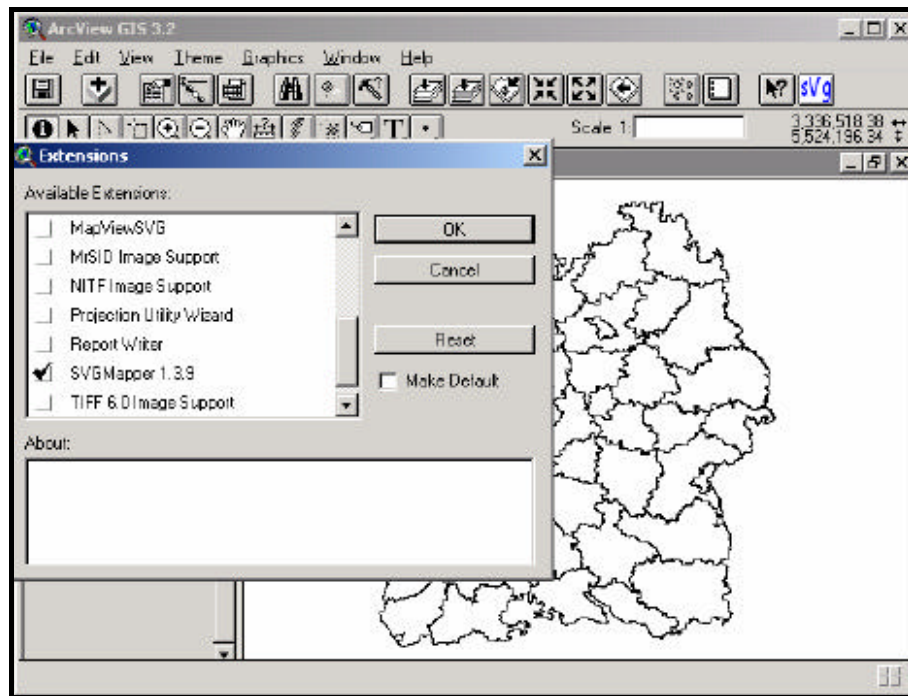


Fig. Adding SVGMapper as an extension to the ARCVIEW

- Open your View and click on SVG icon at the end of button bar.
- After you select and configure your options click on START.
- Save all SVG file in one Folder.

#### 4.1.2 Features of SVGMapper:

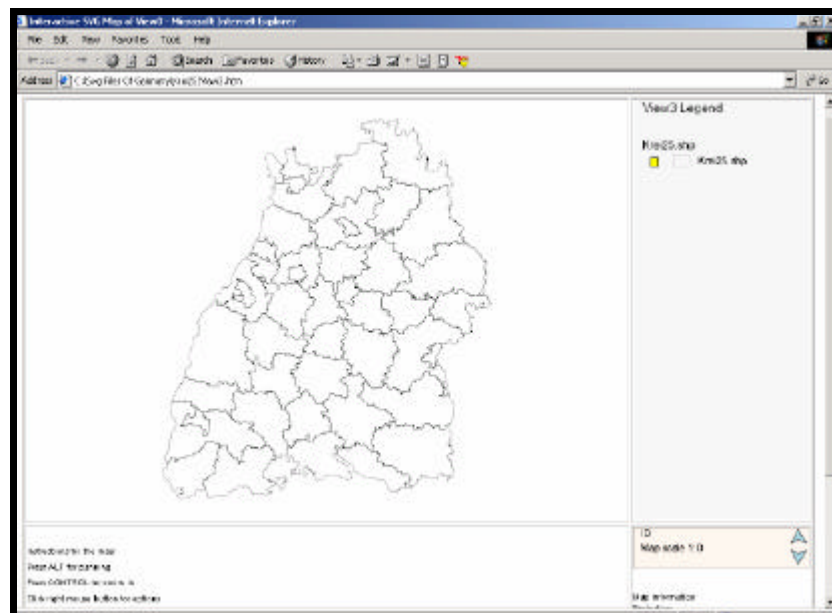
Some of the important features of this SVGMapper are

- Vector graphics in web browser (zooming, panning, searching text, printing and viewing in high resolution - possible with Adobe SVG Viewer).
- SVG format - XML based future standard of web graphics (Each element has its own ID which is useful if one wants to use dynamic SVG. Each layer or thematic classification in the View is a group of elements in SVG.)
- Export of attribute data to html and then display in new window using hyperlink function.

- Interactive map (in HTML) with layer visibility control, legend, hyperlink function and dynamic map scale.
- Classification of data.
- Small file sizes (compressed using GZIP, not included in SVGMapper package).

SVGMapper will support

- Polygons (color, outline color, width)
- Polylines (dash array-broken line, cap style, join style)
- Points (as font characters, circles or use of SVG symbols)
- Text (font and styling, also text-rotation)
- Raster layers (as JPEG image and writes that to SVG as a separate group).



*Fig* SVG Map generated from SVGMapper

The above map is generated by using SVG Mapper with limited functionalities, for this map I have added additional functionalities like OverviewMap, Adding new layers to the existing map,

Interactively Changing the colors of the layers on the map Adding Scale Bar and Coordinate System, changing the layers of the map interactively. The most important thing is to access documents over the map by using Event Handlers like onClick and accessing documents via Tree Structure.

#### **4.2 Showing coordinates on mouse-move:**

This uses different JavaScript functions to display coordinate data. A group called "groupShowCoords" which is scaled and translated to be at a fixed place while the rest of the graphics is zoom/pannable exists. The function InitMap( ) initializes values for coordinate calculation and gets references to SVG objects on "Mouse over effects". The function showcoords(evt) does the actual coordinate calculation and the display of the data in a svg-text-element. The function resetcoords( ) does reset offset and pixel sizes for the coordinate calculation. It uses global variables. The function resetcoords( ) is also called when a zoom and scroll event happens. The source code for these is given below.

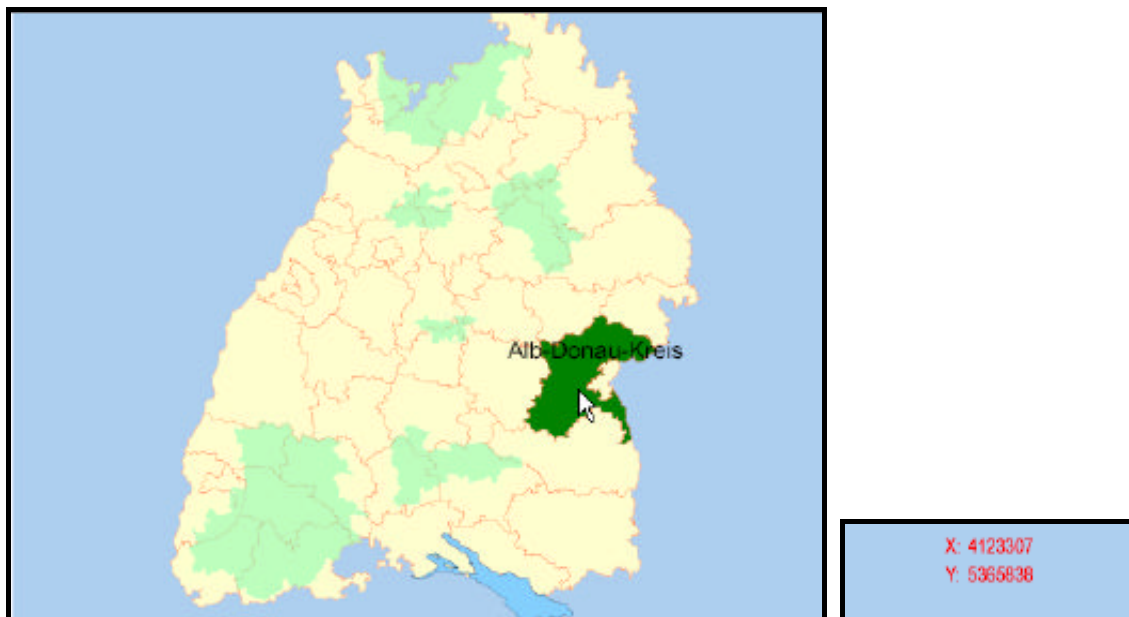


Fig. Showing coordinates on mouse-move

#### **Source Code:**

```
//global variables  
var svgObjCity;
```

---

```

var svgObjX;
var svgObjY;
var svgSVGObj;
var svgDoc;
var svgCoordsGroup;
var ulXCorner;
var ulYCorner;
var origPixSize;
var pixSize;
var offsetX;
var offsetY;
var yMinusVal;
var refCoordRect;
var ratioY;
var pixHeight;

function initMap(evt) {
    // Retrieve the SVG document object:
    var directTarget = evt.getTarget();
    if( directTarget.getNodeType() != 9 ) // if not DOCUMENT_NODE
        svgDoc = directTarget.getOwnerDocument();
    else
        svgDoc = directTarget;

    //get reference to text-Element
    svgObjX = svgDoc.getElementById("coordx");
    svgObjY = svgDoc.getElementById("coordy");
    //get reference to text within text-Element
    svgObjX = svgObjX.getFirstChild();
    svgObjY = svgObjY.getFirstChild();
    svgSVGObj = svgDoc.getDocumentElement();
    //reference to coordinate box (for later scaling and translating)
    refCoordRect = svgDoc.getElementById("groupShowCoords");
    //initialize coordinates, pixsize etc.
    var viewBox = new String(svgSVGObj.getAttribute("viewBox"));
    var viewboxes = viewBox.split(' ');
    ulXCorner = viewboxes[0];
    ulYCorner = viewboxes[1];
    var width = viewboxes[2];
    var height = viewboxes[3];
    var pixWidth = svgSVGObj.getAttribute("width");
    pixHeight = svgSVGObj.getAttribute("height");
    origPixSize = width / pixWidth;
    //a value to subtract Y-Values, because of inverted y-axis
    yMinusVal = 302000;

    //determine ratio for coordinate box placement along y-axis
    translateY = getTranslate("groupShowCoords","y");
    ratioY = translateY / height;

    //call resetCoords();
    resetCoords();
}

```

First we initialize some global variables that should be available throughout all functions. We get references to the SVG root-element and the two text-elements where we want to display the coordinate data. The function `InitMap()` is called from the html-file with the event `onLoad`. To

determine the starting-viewbox the values from the ViewBox-attribute is extracted using a split-function. To determine a pixel-size the width of the viewBox is divided with the number of pixels of the plugins' width, assuming that the same ratio exists for both axes. The value is stored in the variable origpixSize.

```
function resetCoords() {
    //get current zoom and pan values
    var scale = svgSVGObj.getCurrentScale();
    var trans = svgSVGObj.getCurrentTranslate();
    var transx = trans.getX();
    var transy = trans.getY();
    //reset offset-values and pixSize according to current scale and
    translate
    pixSize = origPixSize / scale;
    offsetX = parseFloat(ulXCorner) - transx * pixSize;
    offsetY = parseFloat(ulYCorner) - transy * pixSize;

    //to determine y-position, x-position is always the same ...
    var height = pixHeight * pixSize;
    var newScale = 1 / parseFloat(scale);
    var newTranslateX = offsetX;
    //position always relative to bottom of viewBox
    var newTranslateY = offsetY + height * ratioY;
    newtransform = "translate(" + newTranslateX + " " + newTranslateY + ")
    " + "scale(" + newScale + ")";

    //reset position and scale for the showCoordsgroup so it always
    stays at the same place
    refCoordRect.setAttribute('transform', newtransform);
}
```

The function resetCoords() is called from the functions initMap(), and from the onzoom and onscroll-events to reset offsetValues (getCurrentTranslate()) and pixel Sizes (getCurrentScale()) whenever a zoom and pan-event occurs. These values are important to calculate the real-world coordinates, since the reported values by the method evt.getClientX() only represent pixel-values.

To actually calculate the coordinates and display them in a text-object we call the function showCoords(evt) that gets a reference to the mouse-move event as a parameter to determine the actual mouse-position. The setData()-method displays the coordinates that are calculated using the scale- and offset-values.

```
function showCoords(evt) {
    //to show coordinates
    svgObjX.setData(Math.round(offsetX+evt.getClientX() * pixSize))
    svgObjY.setData(Math.round(yMinusVal-evt.getClientY()*pixSize-
    offsetX))
}
```

Finally the events `onmousemove`, `onzoom` and `onscroll` need to be registered in order to execute the javascript-functions as described in the `svg-code-snippet` below. This is done in the `svg-root-element`.

```
<svg xml:space="preserve" width="450" height="325" viewBox="480000 0
360000,260000" id="svgAll" onmousemove="showCoords(evt)" onzoom="zooming()"
onscroll="scrolling()">
```

Not to forget, there is one more function that needs to be determined i.e., a current translate value of an object. It gets a reference to the object required, does some pattern-matching and extracts the translate part for either x or y:

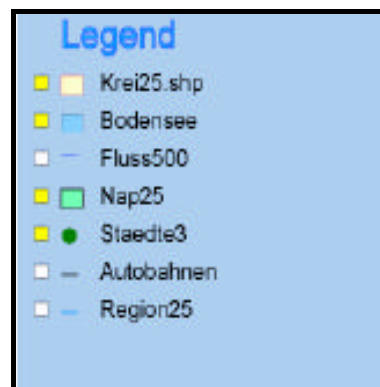
```
function getTranslate(myElement,xOrY) {
    //get reference to element
    element = svgDoc.getElementById(myElement);

    //first get transform value of coordinate box
    var curTransform = element.getAttribute("transform");
    curTransform = new String(curTransform); //Wert in ein String
    umwandeln
    //no fear from Regular expressions ... just copy it, I copied it
    either ...
    var          translateRegExp=/translate\((([-+]?[d+]) (\s*[\s,]\s*) ([-+]?[d+]) \) \s*/;
    //This part extracts the translation-part from the overall transform-
    string
    if (curTransform.length != 0)
    {
        var result = curTransform.match(translateRegExp);
        if (result == null || result.index == -1)
        {
            var oldTranslateX = 0;
            var oldTranslateY = 0;
        }
        else
        {
            var oldTranslateX = result[1];
            var oldTranslateY = result[3];
        }
    }
    if (xOrY == "x") {
        return oldTranslateX;
    }
    if (xOrY == "y") {
        return oldTranslateY;
    }
}
```



### 4.3 Legend

Layers can be turned on and off under the GIS Functions section by checking or unchecking the box next to the layer name. This includes current map layer styles and also theme styles. An integrated SVG legend will help users get a clear picture of what the SVG maps means. Generates legend graphic as SVG file for a particular layer. Takes parameter LayerID (to determine which layer the legend is for). On initial map load Init ( ), Legend will load only those data layers set to display at the initial (base) scale.



*Fig Legend*

Clicking the Yellow checkbox next to a layer in the legend will display or hide that layer in the map. When the map is loaded on the Web it uses JavaScript functions to display the layers. First some global variables are initialized that should be available throughout all functions. Get the reference to the SVG legend root-element. The function Init ( ) is called from the html-file with the event onLoad. From where one will get the reference to the legend and as well as for the main SVG document.

```
svgMainDoc = document.svg_main.getSVGDocument();
svgLegDoc = document.legend.getSVGDocument();
```

After getting reference of the legend user can display or hide the layers by checking on the checkboxes next to the layer that will call the function

function onoff(cbx,theme) , which takes two parameters one is ID of the rectangular box(Yellow box) and the other one is Layer ID.

```
function onoff(cbx,theme)
{
```

```

mainobj = svgMainDoc.getElementById(theme);
mainstyle = mainobj.getStyle();
mainvis = mainstyle.getPropertyValue('visibility');
legobj = svgLegDoc1.getElementById(cbx);
legstyle = legobj.getStyle();
legstylefill = legstyle.getPropertyValue('fill');
if (legstylefill == 'yellow')
{
  legstyle.setProperty('fill', 'white');
  mainstyle.setProperty('visibility', 'hidden');
}
else
{
  legstyle.setProperty('fill', 'yellow');
  mainstyle.setProperty('visibility', 'visible');
}

```

#### 4.4 Overview Map

The navigation section has tools to move around the map. To zoom in on an area you have to select Zoom Values in the List Box and the software will zoom-in automatically for you. The Zoom out is also possible, zooming out to a scale pre-specified by the software. Panning is done on the Overview map. Users simply drag the rectangle on the overview map to change the view on the main map. Users should be careful not to confuse where it says “Drag Light Green Rectangle to Change map in the Main map”. This statement is referring to pan the map on the overview map.

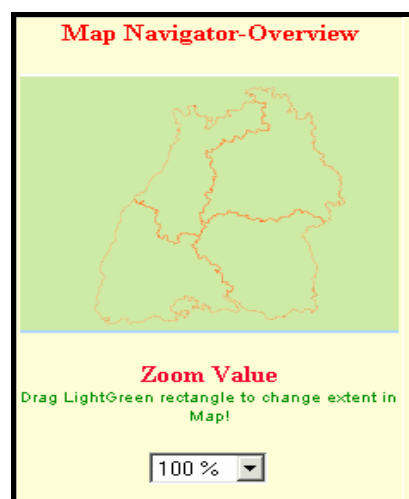


Fig. Overview Map

Over the Overview Map a Light Green rectangle is attached which is an SVG defined in Overview.svg with rect id="rectForPlace" where it defines the X and Y-coordinates, as well as width

and height of that rectangle. When the user selects Zoom values from the list box it will call the Function `zoomIt()` in the JavaScript from where the values of draggable rectangle are obtained.

```
xulcorner = parseFloat(svgRect.getAttribute("x"));
yulcorner = parseFloat(svgRect.getAttribute("y"));
width = parseFloat(svgRect.getAttribute("width"));
height = parseFloat(svgRect.getAttribute("height"));
```

Initially define the `allwidth` and `allheight` as global variables, by using some calculation get the X and Y-coordinates width and height of the rectangle and as well as the coordinates of the main Map.

```
xcenter = xulcorner + width / 2;
ycenter = yulcorner + height / 2;
xnulcorner = xcenter - allWidth / 2 * (100/zoomVal);
ynulcorner = ycenter - allHeight / 2 * (100/zoomVal);
nWidth = allWidth * (100/zoomVal);
nHeight = allHeight * (100/zoomVal);
```

Then set the values of draggable rectangle

```
svgRect.setAttribute("x",xnulcorner);
svgRect.setAttribute("y",ynulcorner);
svgRect.setAttribute("width",nWidth);
svgRect.setAttribute("height",nHeight);
```

The change of Coordinates in the Overview Map window are registered and converted into the coordinates of the map. Then set viewport of main map from the above obtained values

```
newViewport = xnulcorner + " " + ynulcorner + " " + nWidth + " " + nHeight;

svgMainViewport.setAttribute("viewBox",newViewport);
```

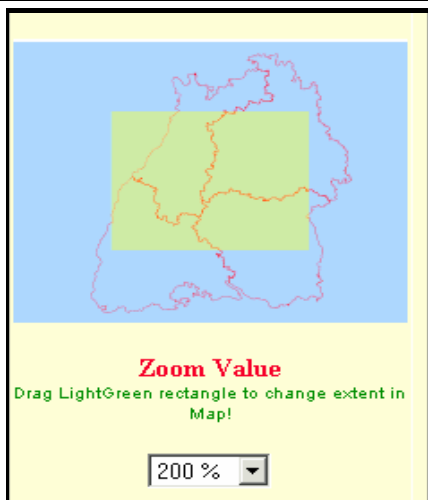


Fig. Selecting Zoom Value

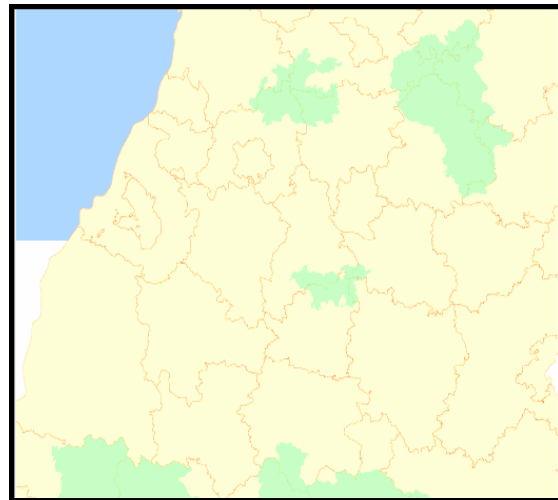
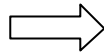


Fig. View in Main Map

#### 4.5 Panning

Panning can also be performed by dragging the light green rectangular box over the Overview map. When the user drags the rectangular box the change of Coordinates in the Overview Map window are registered and converted into the coordinates of the main map. During this process three functions were called

- **beginPan(evt)**
- **doPan(evt)**
- **endPan(evt)**

The first function gets the coordinates of the rectangular box.

First get the reference to the element of the Overview Map.

```
svgoverviewdoc = svgOverview.getSVGDocument();
```

Then get the reference to the rectangular Box

```
svgRect = svgoverviewdoc.getElementById("rectForPlace");
width = parseFloat(svgRect.getAttribute("width"));
height = parseFloat(svgRect.getAttribute("height"));
evtX = parseFloat(evt.getClientX());
evtY = parseFloat(evt.getClientY());
rectUlxCorner = parseFloat(svgRect.getAttribute("x"));
rectUlyCorner = parseFloat(svgRect.getAttribute("y"));
```

The second Function gets the final coordinates of the rectangular box when the user moves the box over the Overview map.

```

    evtX = newEvtX;
    evtY = newEvtY;
    rectUlxCorner=parseFloat(svgRect.getAttribute("x"));
    rectUlyCorner = parseFloat(svgRect.getAttribute("y"));
  
```

The third function sets the final viewport of main map.

```

svgMainViewport.setAttribute("viewBox",newViewport);
  
```

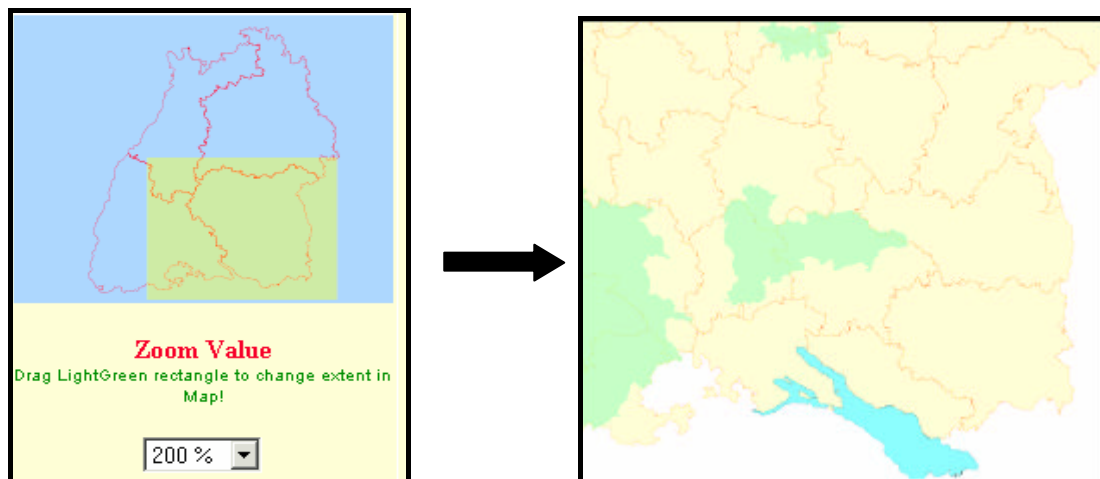


Fig. Panning (Drag the rectangular box)

fig. View in Main Map

## 4.6 Accessing Documents

The most important part in this thesis work is to allow the user to access documents over an interactive map and also via Tree View.

### 4.6.1 Via Tree View

The navigation interface is one of the most important pieces of any application. Correctly designed, it lets users find the information they need, easily and quickly.

With a tree menu control, also called tree view, the information is displayed in a hierarchical order, with the home topic at the top and the subordinated items underneath. Beginner users like to use a tree of folders because it is easy to learn; experienced users like it because it is efficient in the number of clicks involved. Web-based applications can only gain from applying the tree menu to their navigation systems.

Thus this tree view structure acts as a Gazetteer; it includes all documents belonging to a particular Region, Sub-regions, Counties and Cities.

#### **4.6.1.1 Main features of Tree View:**

The main features of this Tree structure are:

- Unlimited hierarchical levels.
- Fast performance even for trees with thousands of nodes.
- Does not require Java or cookies.
- Frame-based or frame-less layouts.
- Optional and customizable folder and document icons.
- Option to wrap node text into multiple lines.
- Option to highlight selected node.
- Expand/collapse does not require slow trip to server.
- Cross-browser: supports all major browsers/versions/platforms.
- Quick configuration through online visual tool.
- Optional server-side browsing of files and directories.
- Optional server-side connection to databases.
- Optional tree-state persistency across page loads.
- Open/close state is tracked separately for multiple trees.
- Modular design: make only small, localized changes to your page.

#### **4.6.1.2 How to set up a Tree structure**

To setup the tree Structure manually and explore all the possible layout and customization options, one has to learn more about the contents of the configuration files (demoFramesetNodes.js).

Inside the configuration file and before the definition of the nodes, it is possible to change layout options and other settings of the tree by assigning values to "environment variables". The variables, their purpose, and the possible values are listed in the table below.

Variable name	Description	Possible values	Default value
BUILDALL	<p>Fast performance with large or very large trees is obtained by deferring the construction of nodes until the user actually expands the folder that contains them.</p> <p>By default the BUILDALL variable has the value 0. Setting this variable to 1 will make the script build all the items when the page is loaded, even those that will not be visible because they are inside closed folders.</p>	<p>0 -&gt; defer the construction of hidden items</p> <p>1 -&gt; write HTML for all items on page load</p>	0
HIGHLIGHT	<p>Highlight the text of a folder or "document" node when the node is selected. Typically this option would reverse the foreground and background colors of the clicked node, but any colors may be used. To be able to reload the tree and keep the highlight of the selected node, use the PERSERVESTATE variable.</p> <p>Only nodes that actually load pages will be highlighted on selection. It is also possible to configure a node that opens pages to not be highlighted. The maySelect member exists both for documents and folders:</p> <p>d=insDoc(.. d.maySelect = false</p>	<p>0 -&gt; do not highlight node</p> <p>1 -&gt; apply highlight</p>	0
HIGHLIGHT_COLOR	String with the color used for the	May be any HTML-valid	'White'

	highlighted text.	color; for example: 'white' or '#FFFF80'.	
HIGHLIGHT_BG	String with color used for the background of the highlighted text.	Any HTMLvalid color.	'Blue'
ICONPATH	There are cases where the gif files have to be separated from the remaining Tree view files and then put them in one directory just for images. For those cases use this variable to specify the location of the directory. Note: the value of this string should either be empty or terminates with a forward slash (/).	Any valid URL that points to a directory (as opposed to a file.) Examples: 'images/', 'http://www.x.com/y/', etc.	" (Empty string)
PERSERVESTATE	<p>The user changes the state of the tree in two ways: by opening and contracting folders, and by highlighting nodes (if HIGHLIGHT=1). Unless that state information is stored, reloading the tree will revert it to its original state. With this variable set to one, Treeview JavaScript stores the appropriate information in cookies to make it available across page loads. With the variable set to 0 the script uses no cookies.</p> <p>Two typical scenarios that require the reload of the tree are: pages with a frameless layout (see USEFRAMES), and database driven applications.</p> <p>If the structure of the tree may change between reloads (a new folder is added in a database-driven application, for</p>	<p>0 -&gt; no storage of "state" in cookies 1 -&gt; preserve state upon reload</p>	0



	example) the use of the PERSESTATE will require the nodes to have a constant means of identification. Without persistent ids the script may try to use the wrong id and show an error.		
STARTALLOPEN	Specify if, upon loading the page, the tree should be displayed with all folders expanded. Note that this may be convenient for small or medium sized trees but defeats the optimizations introduced for very large tree (trees with hundreds or thousands of nodes.) Naturally, this option should also not be used together with PERSESTATE.	0 -> show only one level 1 -> expand all.	0
USEFRAMES	Specify whether the tree will be displayed inside a frame or in a regular frameless page. If a frameless layout is intended, this variable should be explicitly reset because special DHTML needs to be generated in those cases.	0 -> frameless layout 1 -> tree is in its own frame	1
USEICONS	Setting this variable to zero removes the icons from all entries in the tree. Note, however, that for folder the little +/- control button is not removed, only the icon with the folder image.	0 -> remove icons 1 -> display icons	1
USETEXTLINKS	When an URL is associated with a node (folder or link) the icon of the node becomes clickable. This variable controls if the text of each node should	0 -> only the icons are clickable 1 -> the text is clickable too.	0

	also be clickable (be an hyperlink) or not.		
WRAPTEXT	Specify whether the text in each entry is allowed to wrap around into more than one line.	0 -> single line 1 -> multi-line	0

(Source: TreeView <http://www.treeview.net/treemenu/instructions.asp>)

### **4.6.1.3 Workflow**

#### **Hierarchical construction of nodes**

The configuration files (demoFramesetNodes.js) have an initial section where some global variables are set (described above) and then a section for the actual creation of folders and links. Here we describe those tree-construction commands.

1. Create the root folder in demoFramesetNodes.js. Use this command:

```
foldersTree = gFld( "Baden-Württemberg", " " )
```

The function gFld takes a name (Baden-Württemberg) and an optional URL, and returns the folder. Any name can be given to the root folder; note how even some formatting can be used. The URL must either be just a file name (for example: demoFramesetRightFrame.html).

2. Place folders inside other folders by using the function 'insFld ([parent folder], [child folder])'.

For example:

```
aux1 = insFld (foldersTree, gFld ( "Stuttgart (RB)", 'query1.pl? 8010000000 ' ))
```

3. Creating a document link use the function 'gLnk'. It takes three arguments:

- **Target:**

'R' opens in the right frame (a frame named basefrm),

'B' opens in a new window,

'T' opens in the current browser window, replacing the frameset if one exists,

'S' opens in the current frame, replacing the tree.

Note: the function is case sensitive; use uppercase.

- **Title:** Text to be displayed in the tree.

- **Link:** URL of the document (may be absolute or relative.) Do not enter other information in this string. Adding a target parameter or an event handler will not work.

### Example:

```
Aux1 = insFld(aux2, gFld("Main-Tauber-Kreis", 'query1.pl?8010101000'))

      insDoc(aux3, gLnk("R", "Bad Mergentheim", "query1.pl?8010101001 "))
```

4. To place the document inside the tree use the function 'insDoc([parentfolder], [document link])'.

Note: If one want to rename some of them one will have to take in account the file dependencies: one may want to rename demoFramesetRightFrame.html but then one will need to update both demoFramesetNodes.js and demoFrameset.html. The file demoFrameset.html itself can be renamed without any problems with dependencies. Do not rename any other file.

### The optional link of the folder icon

In terms of how you want the script to behave to the user, one may not want the action of clicking on the folder to open a page (on the right frame or on a separate page.) In this case one wants to click on the folder to be similar to a click on the '+' sign.

The way one does this in the configuration file is by giving a special value to the second argument of the gFld function. In the example included with in the download, this is how the whole statement looks:

```
aux1 = insFld (foldersTree, gFld ("Baden-Wurttemberg", "javascript:parent.op0"))
```

The special argument used is javascript:parent.op(), which is a call to an empty function. If one just gives the empty string as the argument the folder may not be "clickable" for older browsers.

### Changing the icons in the tree

In the configuration file one can indicate what icon should be used with a particular folder or "doc". This showcases this functionality (it works with frameless layouts too.) Check the last folder and the document it contains.

If one is replacing the icon of a folder, one will have to provide two icons: one for when it's open, and another for when it's closed. In the configuration file demoFramelessNodes.js included with the free download one can see how this is done:

```
aux1 = insFld(foldersTree, gFld("Other Icons", "..."))
      aux1.iconSrc = ICONPATH + "diffFolder.gif"
      aux1.iconSrcClosed = ICONPATH + "diffFolder.gif"
```

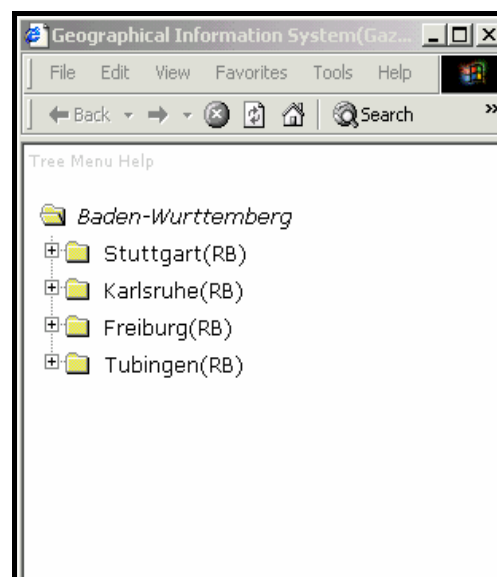
ICONPATH is one of the Treeview JavaScript "environment variables" (see section "Options for layout, look-and-feel, and more".) If one is not setting this variable it may be omitted from the above statements.

If one wants to have the icon for open and closed folders to be the same one can just assign the same pathname to both the iconSrc and iconSrcClosed attributes.

In the case of the "doc" node, only the iconSrc is applicable. Here's the code from the demoFramelessNodes.js example:

```
docAux = insDoc(aux1, gLnk("B", "D/L Treeview", "..."))  
docAux.iconSrc = ICONPATH + "diffDoc.gif"
```

When the user clicks on the Tree View SVG button in the Web-information System a Pop-up window will open where the Baden-Württemberg main folder with four sub-folders were created ex: Stuttgart, Karlsruhe, Freiburg, and Tübingen.



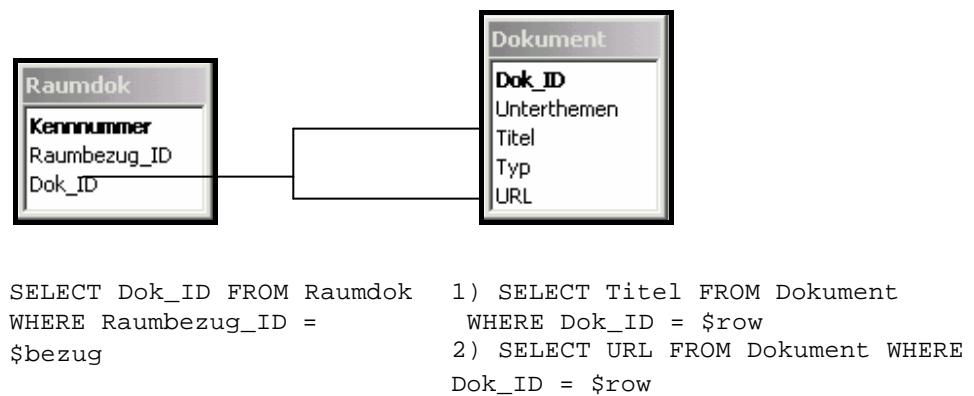
*Fig Tree View of Baden-Württemberg*

#### **4.6.1.4 Accessing the documents of the regions:**

In order to access the documents available in Stuttgart region user should click on Stuttgart Region Folder, where the id is passed as a second parameter, which acts as URL

```
aux1 = insFld (foldersTree, gFld ("Stuttgart (RB)", 'query1.pl? 8010000000 '))
```

It will call **quer1.pl** file where this perl file is connected to the database via DBI:ODBC connection, by taking id of Stuttgart some SQL queries are performed as shown below and it will retrieve the documents available for that particular Folder. A pop up window is opened showing the number of Documents available and also the corresponding URL will be displayed as well as Sub-Region Folders belonging to this region are also opened.



*Fig. Relationship Diagram*

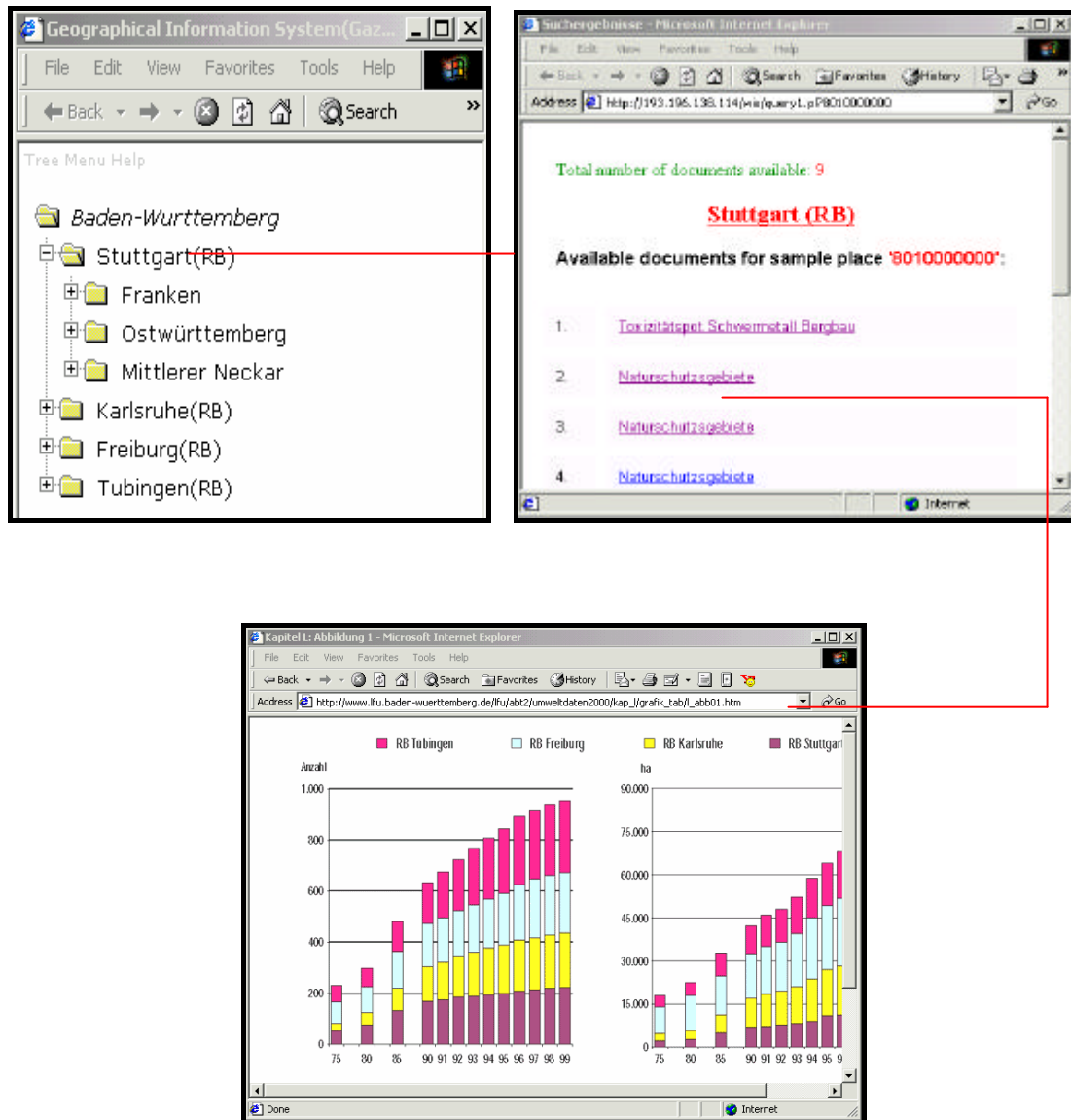
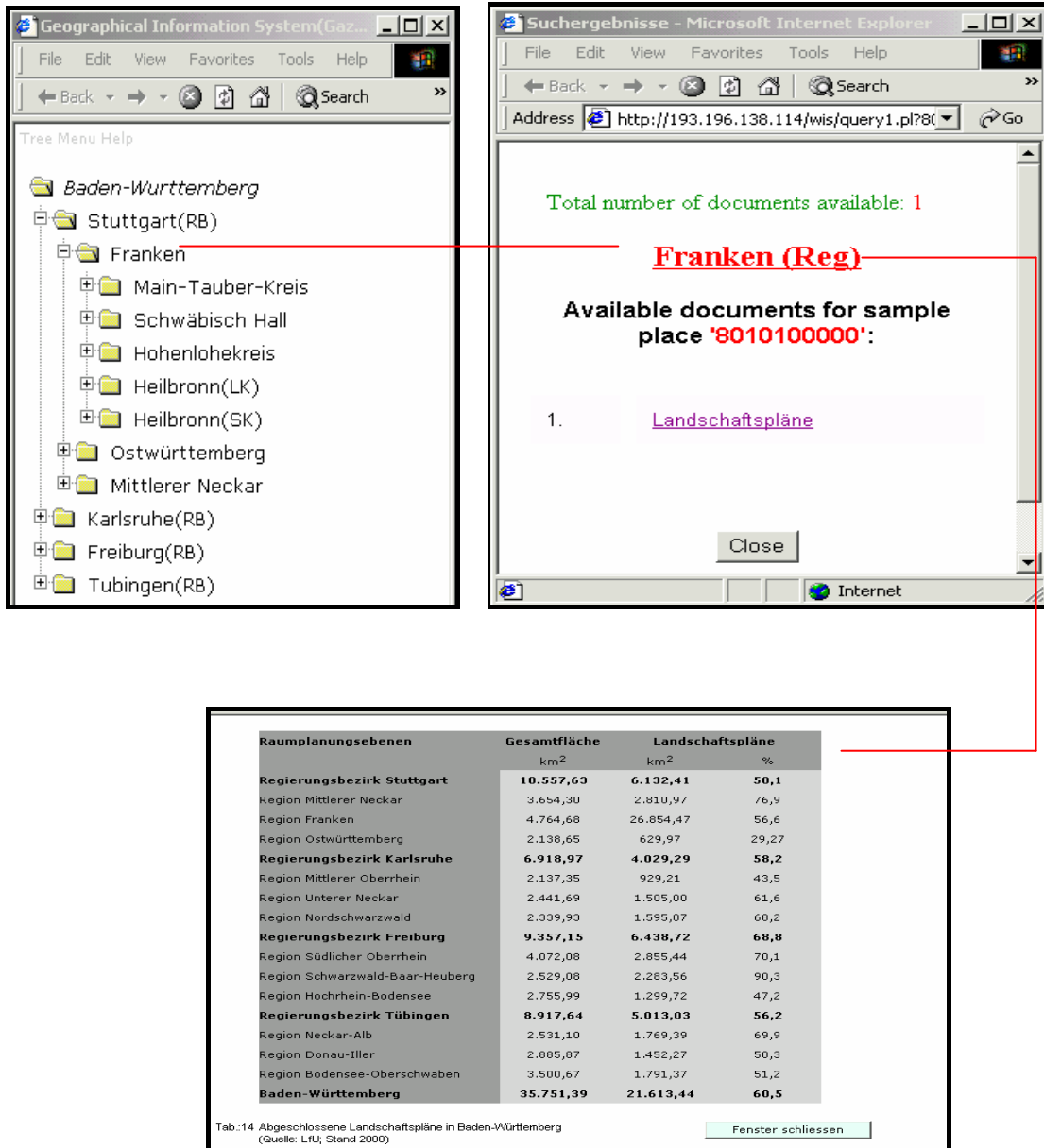


Fig Accessing Stuttgart region documents Via Tree Structure

#### **4.6.1.5 Accessing the documents of Sub-Regions:**

User can also access to the documents available in these Sub-Region Folders by clicking on the Sub-Region Folder (example: Franken), it will call **quer1.pl** file where this perl file is connected to the database via DBI:ODBC connection. It will retrieve the documents available for that particular Folder. A pop up window will be opened displaying the number of Documents available for that

particular Sub-Region and also its corresponding URL being displayed as well as Sub-Folders of counties (Kreis) belonging to this Sub-Region being opened,



Raumplanungsebenen	Gesamtfläche		Landschaftspläne	
	km <sup>2</sup>	km <sup>2</sup>	km <sup>2</sup>	%
<b>Regierungsbezirk Stuttgart</b>	<b>10.557,63</b>	<b>6.132,41</b>		<b>58,1</b>
Region Mittlerer Neckar	3.654,30	2.810,97		76,9
Region Franken	4.764,68	26.854,47		56,6
Region Ostwürttemberg	2.138,65	629,97		29,27
<b>Regierungsbezirk Karlsruhe</b>	<b>6.918,97</b>	<b>4.029,29</b>		<b>58,2</b>
Region Mittlerer Oberrhein	2.137,35	929,21		43,5
Region Unterer Neckar	2.441,69	1.505,00		61,6
Region Nordschwarzwald	2.339,93	1.595,07		68,2
<b>Regierungsbezirk Freiburg</b>	<b>9.357,15</b>	<b>6.438,72</b>		<b>68,8</b>
Region Südlicher Oberrhein	4.072,08	2.855,44		70,1
Region Schwarzwald-Baar-Heuberg	2.529,08	2.283,56		90,3
Region Hochrhein-Bodensee	2.755,99	1.299,72		47,2
<b>Regierungsbezirk Tübingen</b>	<b>8.917,64</b>	<b>5.013,03</b>		<b>56,2</b>
Region Neckar-Alb	2.531,10	1.769,39		69,9
Region Donau-Iller	2.885,87	1.452,27		50,3
Region Bodensee-Oberschwaben	3.500,67	1.791,37		51,2
<b>Baden-Württemberg</b>	<b>35.751,39</b>	<b>21.613,44</b>		<b>60,5</b>

Tab.:14 Abgeschlossene Landschaftspläne in Baden-Württemberg  
(Quelle: LfU, Stand 2000)

Fenster schließen

Fig Accessing Franken documents Via Tree Structure.

#### 4.6.1.6 Accessing the documents of Counties (Kreis):

User can also access to the documents available in these counties (Kreis) by clicking on the counties Folder (example: Main-Tauber-Kreis), which will call **quer1.pl** file where this perl file is connected to the database via DBI:ODBC connection. This will retrieve the documents available for the particular Folder. A pop up window will be opened with Number of Documents available for that particular county and also the corresponding URL being displayed as well as the Documents of important cities belonging to this county being opened.

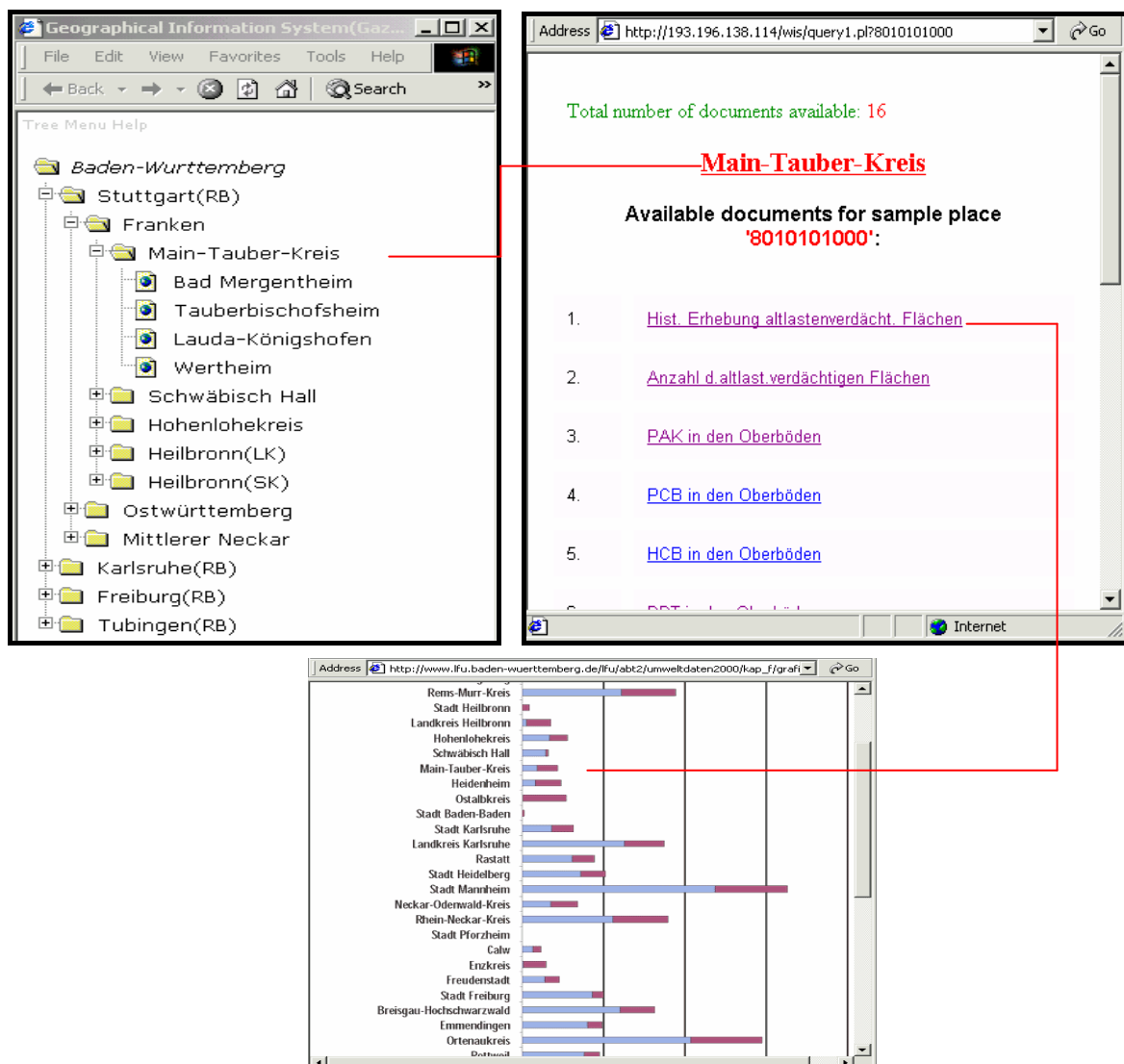


Fig Accessing Main-Tauber-Kreis county documents Via Tree Structure



#### 4.6.1.7 Accessing the documents of Cities:

Finally user can also access to the documents available for cities by clicking on the cities (example: Bad Mergentheim), it will call **quer1.pl** file where this perl file is connected to the database via DBI:ODBC connection and it will retrieve the documents available for the particular city. A pop up window will open with Number of Documents available for a particular city and also the corresponding URL will be displayed.

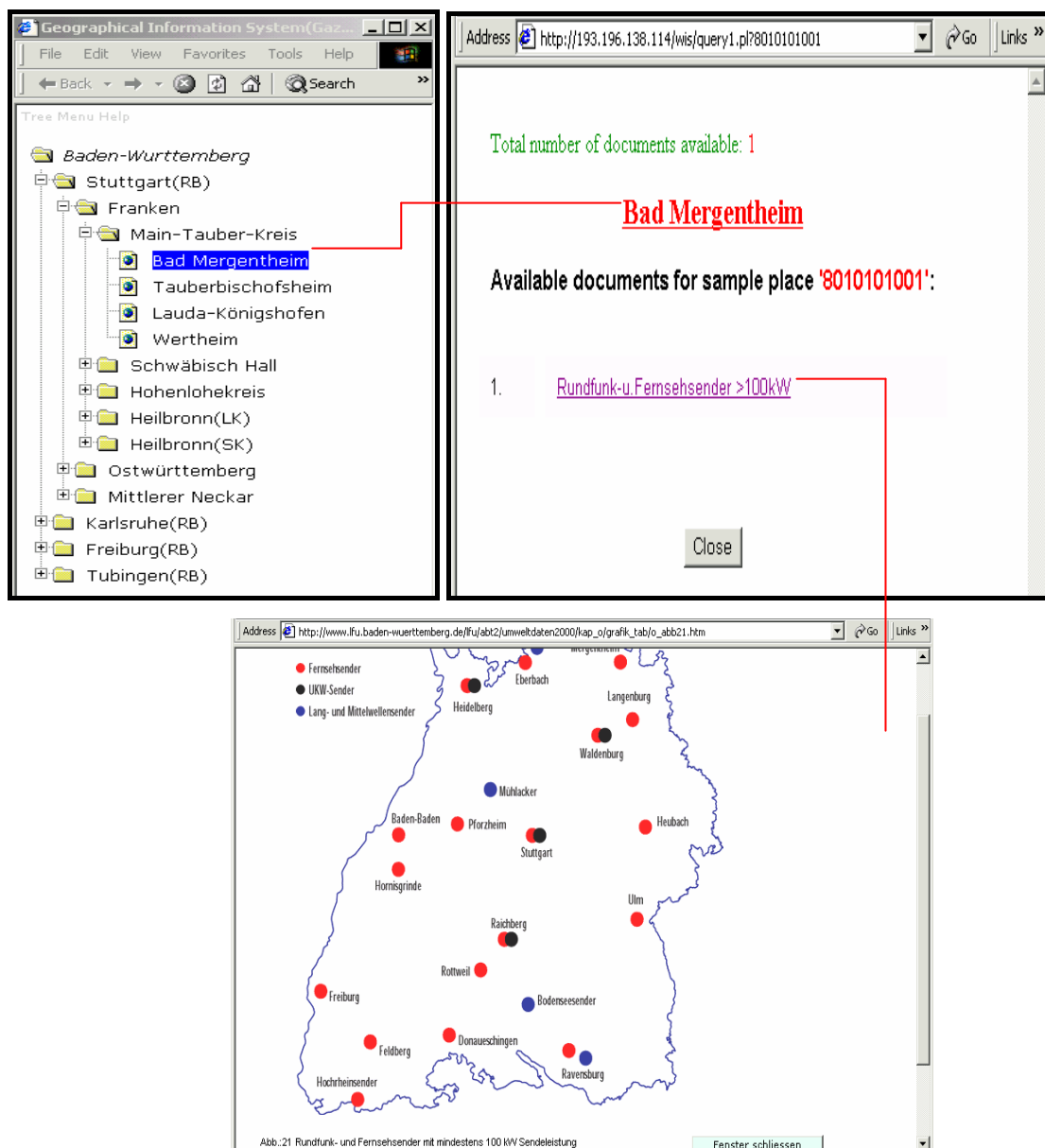
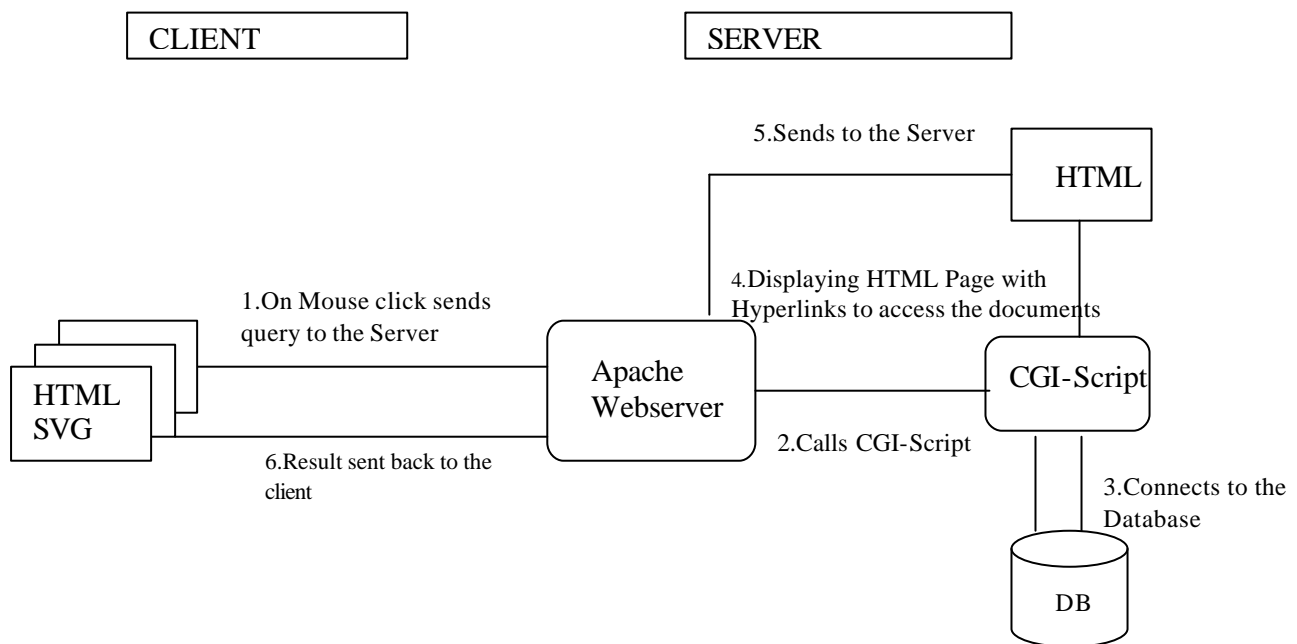


Fig. Accessing Bad Mergentheim documents Via Tree Structure

#### **4.6.2 Accessing Documents By Clicking Over The Interactive Map**

User can access the documents by Clicking over the interactive Map. One can access documents according to the Region, or Counties (Kreis), or Cities .For that purpose first one has to check the checkbox to the right side of the Map. When this is done user can approximately know how many documents are available for each region by seeing the color of the region. A legend will be displayed showing the number of documents available for each color, thus the user can easily know which region is having more number of documents and which region is having less number of documents.

##### **4.6.2.1 Overview of the Process**



##### **4.6.2.2 Accessing Documents of the Regions**

Baden-Wurttemberg is divided into four regions Stuttgart, Karlsruhe, Freiburg and Tübingen.

Each region is again divided into three Sub-Regions. The documents available for these Sub-Regions should include the documents of Sub-Region, Counties, and Cities. For that purpose I created an additional column Region\_Id in the table Raumbezug in the Database. Where I defined a particular Id for each Sub-Region as shown in Fig,

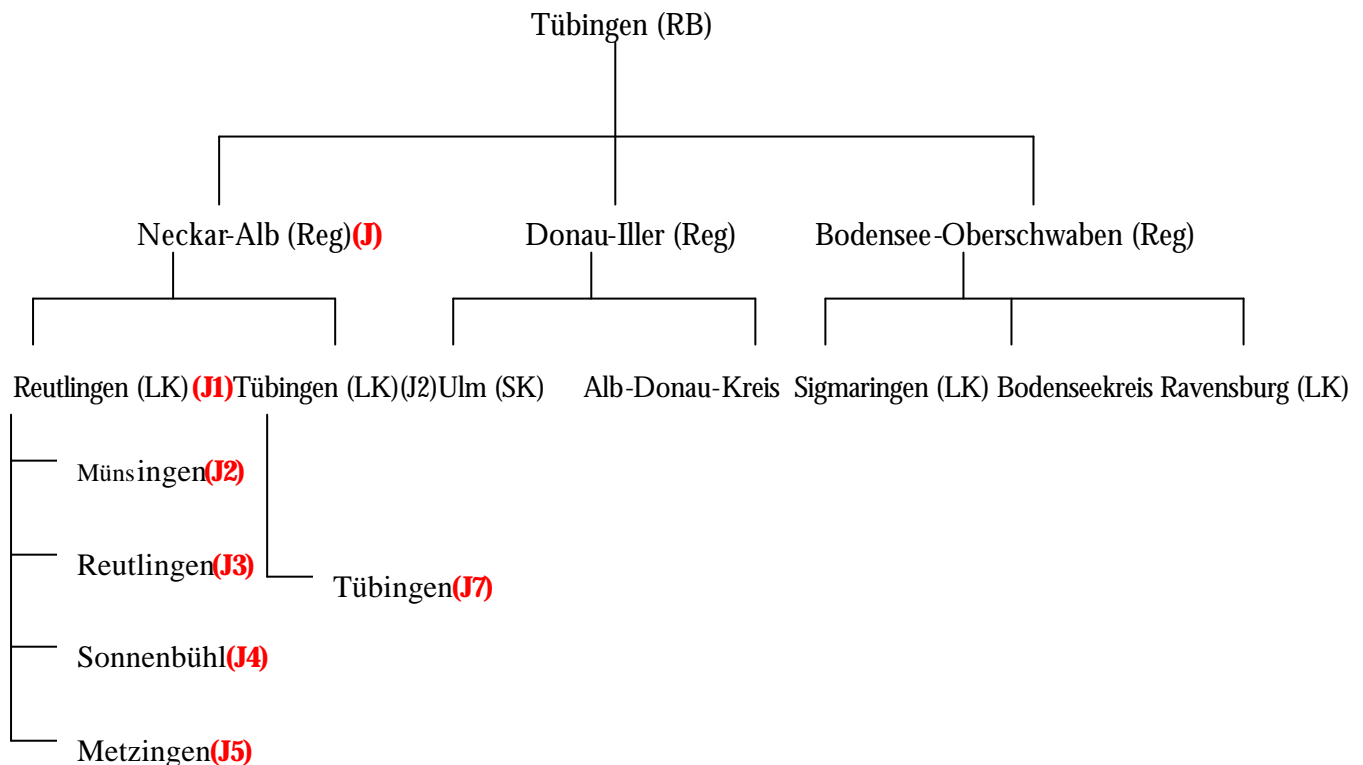


Fig. Defining Id's in the Database

Then this Id is passed to mouse event Onclick() when the user clicked on a particular Sub-Region. It will be connected to **query4.pl** file where this perl file is connected to the database via DBI:ODBC connection and it will retrieve all the documents available for that particular region (relationship diagram below shows how the documents are retrieved), it will include the documents of that Sub-Region as well as the documents of Counties and Cities belonging to the same region and a Pop up window will be opened with all URL'S of those documents.

	Raumbezug_ID	Name	Region_Id
+	8030303002	Tuttlingen	I9
+	8030303003	Denkingen	I10
+	8040000000	Tübingen (RB)	
+	8040100000	Neckar-Alb (Reg)	J
+	8040101000	Reutlingen (LK)	J1
+	8040101001	Münsingen	J2
+	8040101002	Reutlingen	J3
+	8040101003	Sonnenbühl	J4
+	8040101004	Metzingen	J5
+	8040102000	Tübingen (LK)	J6
+	8040102001	Tübingen	J7
+	8040103000	Zollernalbkreis	J8
+	8040103001	Balingen	J9
+	8040103002	Hechingen	J10
+	8040200000	Donau-Iller (Reg)	K
+	8040201000	Alb-Donau-Kreis	K1
+	8040201001	Erbach	K2
+	8040201002	Laichingen	K3
+	8040201003	Ehingen (Donau)	K4
+	8040202000	Ulm (SK)	K5
+	8040202001	Ulm	K6
+	8040203000	Biberach (LK)	K7
+	8040203001	Biberach a. d. Riß	K8

Fig. Database table showing how the ids were defined

### Relationship Diagram

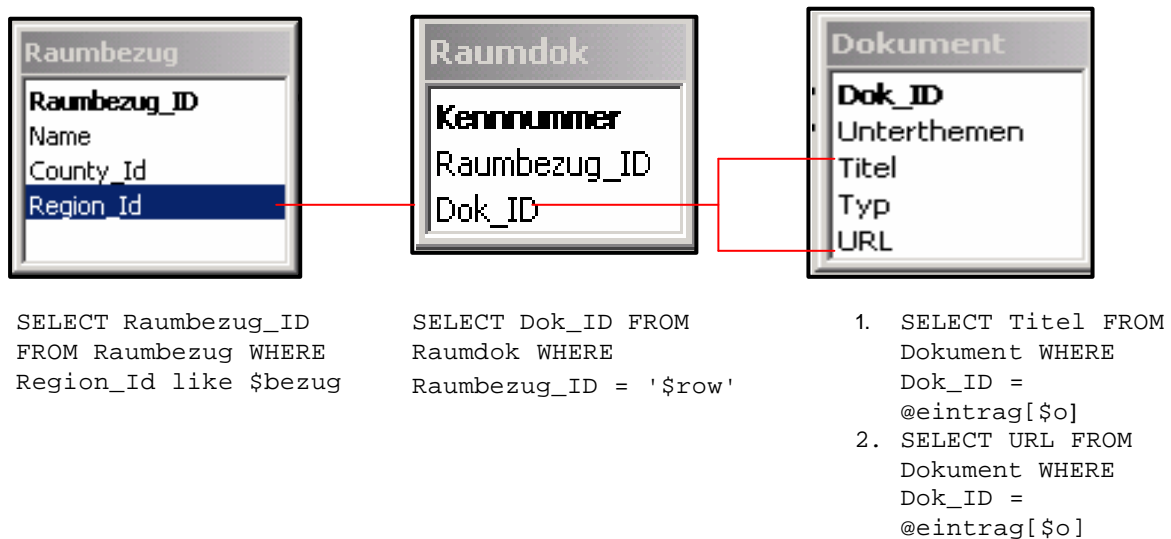


Fig Relationship Diagram

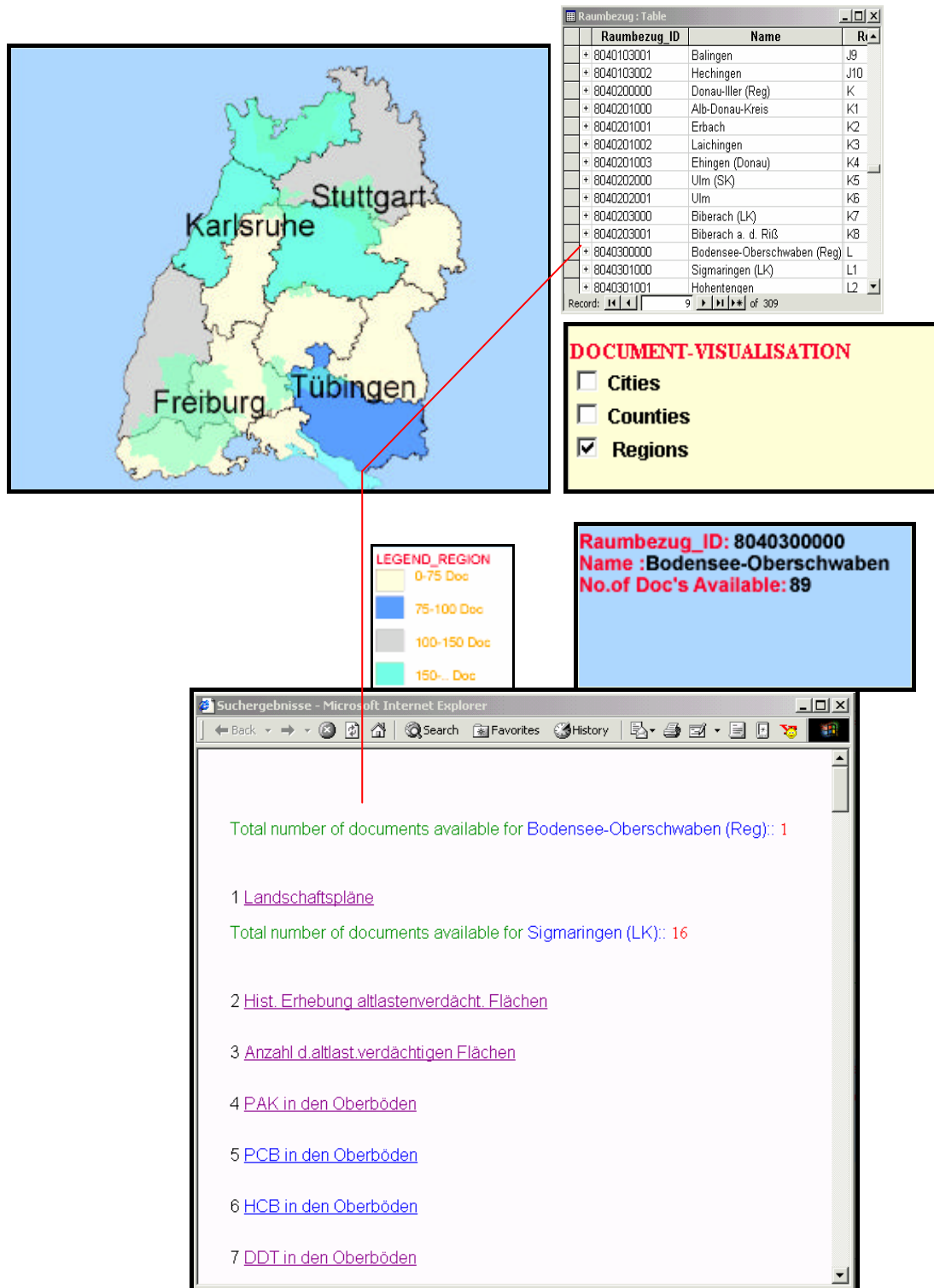


Fig. Showing how to retrieve documents on Mouse click

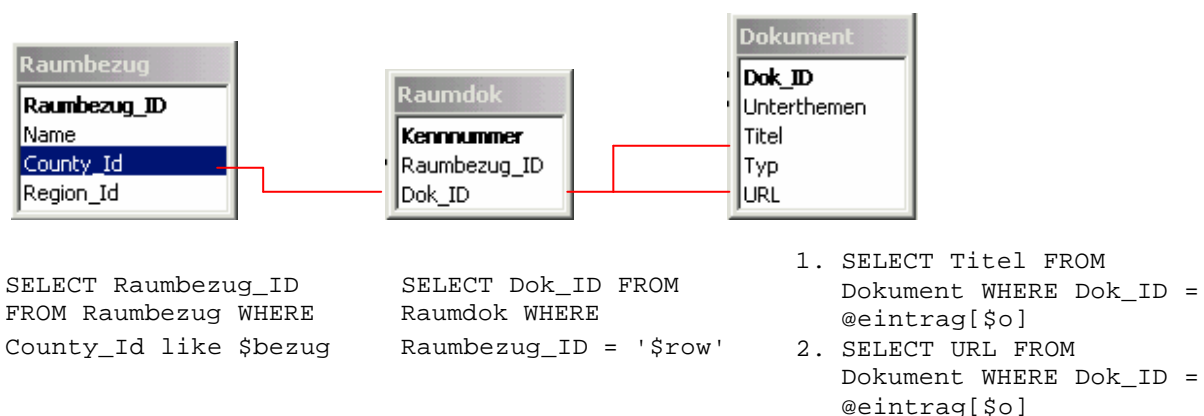
### 4.6.2.3 Accessing Documents of the Counties

To access the documents of the counties, User has to check the checkbox to the right side of the Map. When the checkbox is checked user can approximately know how many documents are available for each region by seeing the color of the Counties (Kreis), a legend will be displayed showing the number of documents available for each color, thus the user can easily know which County (Kreis) is having more number of documents and which County (Kreis) is having less number of documents.

If the user wants to access the documents of a particular County one has to click on that county. The documents available for these Counties should include the documents of County, and Cities belonging to that county. For that purpose an additional column County\_Id in the table Raumbezug in the Database was created. Where a particular Id for each County was defined.

Then Id of that particular county is passed to mouse event Onclick(). It will be connected to query3.pl file where this perl file is connected to the database via DBI:ODBC connection and it will retrieve all the documents available for that particular region (relationship diagram below shows how the documents are retrieved), it will include the documents of that County as well as the documents of the Cities belonging to the same County and a Pop up window will be opened with all URL'S of those documents shown in the Fig...

#### Relationship Diagram



*Fig Relationship Diagram*

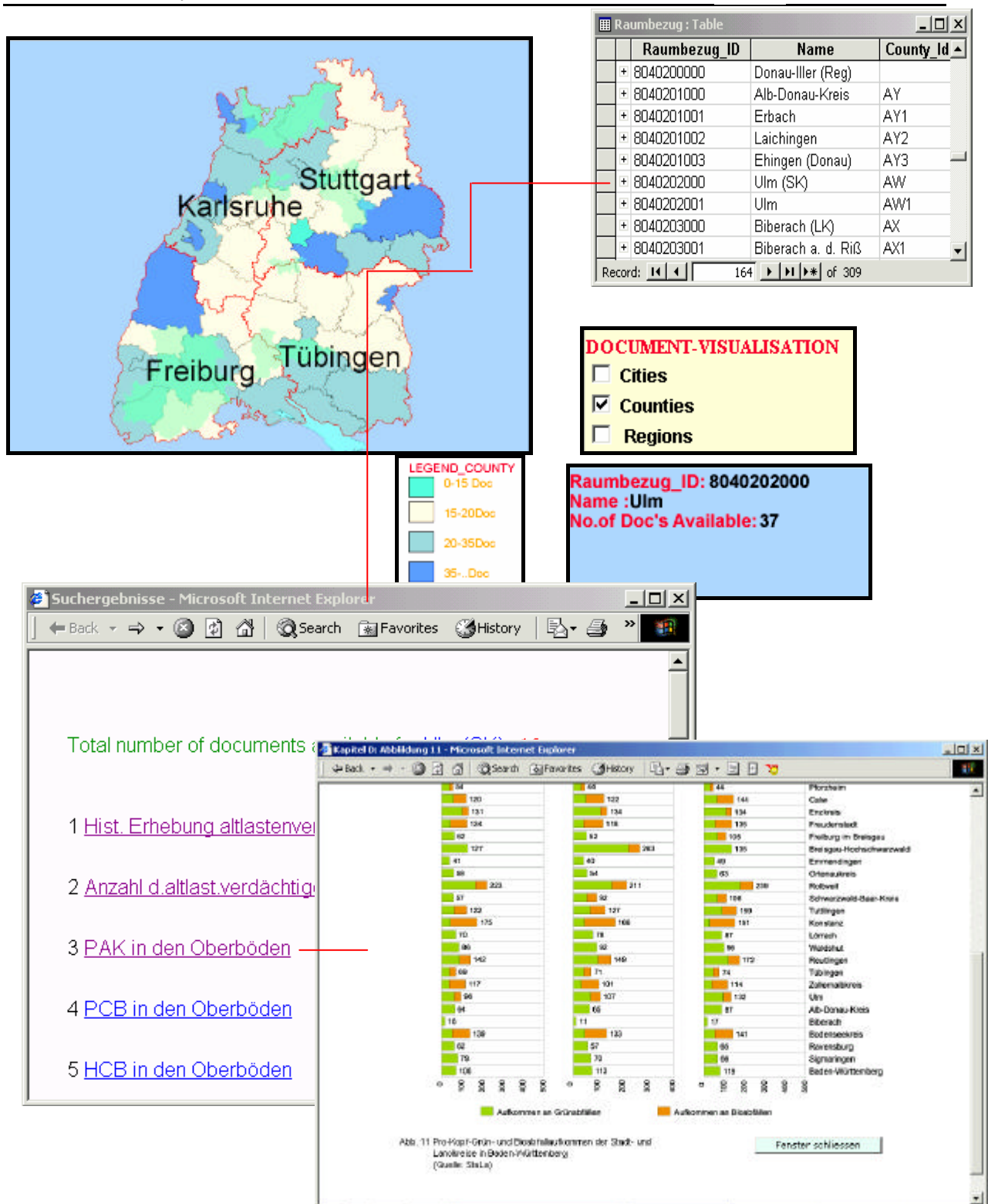


Fig. Showing how to retrieve documents of ULM on Mouse click

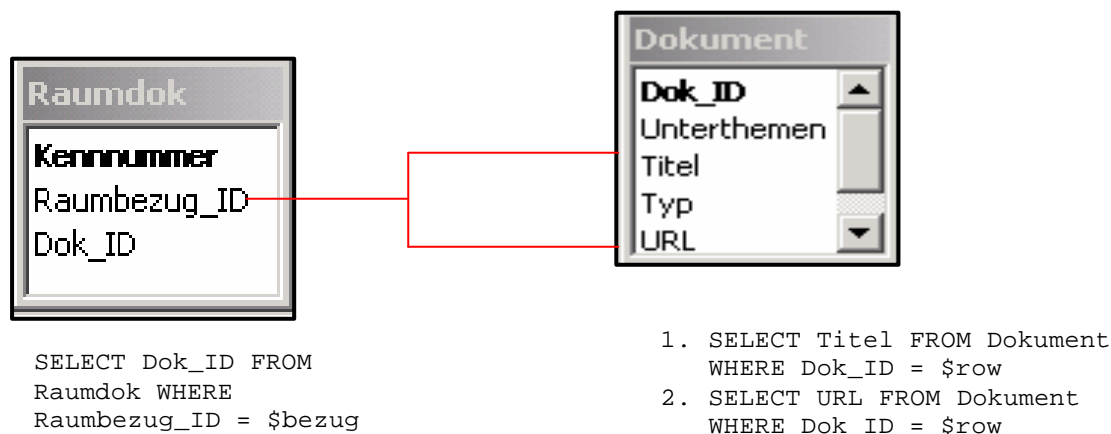
#### 4.6.2.4 Accessing Documents of Cities

To access the documents of the Cities, User has to check the checkbox of Cities to the right side of the Map when the checkbox is checked user can approximately know how many documents are available for each City by seeing the color of the circle representing city, a legend will be displayed showing the number of documents available for each color, thus the user can easily know which City is having more number of documents and which City is having less number of documents.

If user wants to access the documents of a particular City one has to click on that City.

Then Id of that particular City is passed to mouse event Onclick(). It will be connected to query1.pl file where this perl file establishes the connection to the database via DBI:ODBC connection and it will retrieve all the documents available for that particular region (relationship diagram shown below how the documents are retrieved), and a Pop up window will be opened with all URL'S of those documents shown in the Fig...

#### Relationship Diagram



*Fig* Relationship Diagram



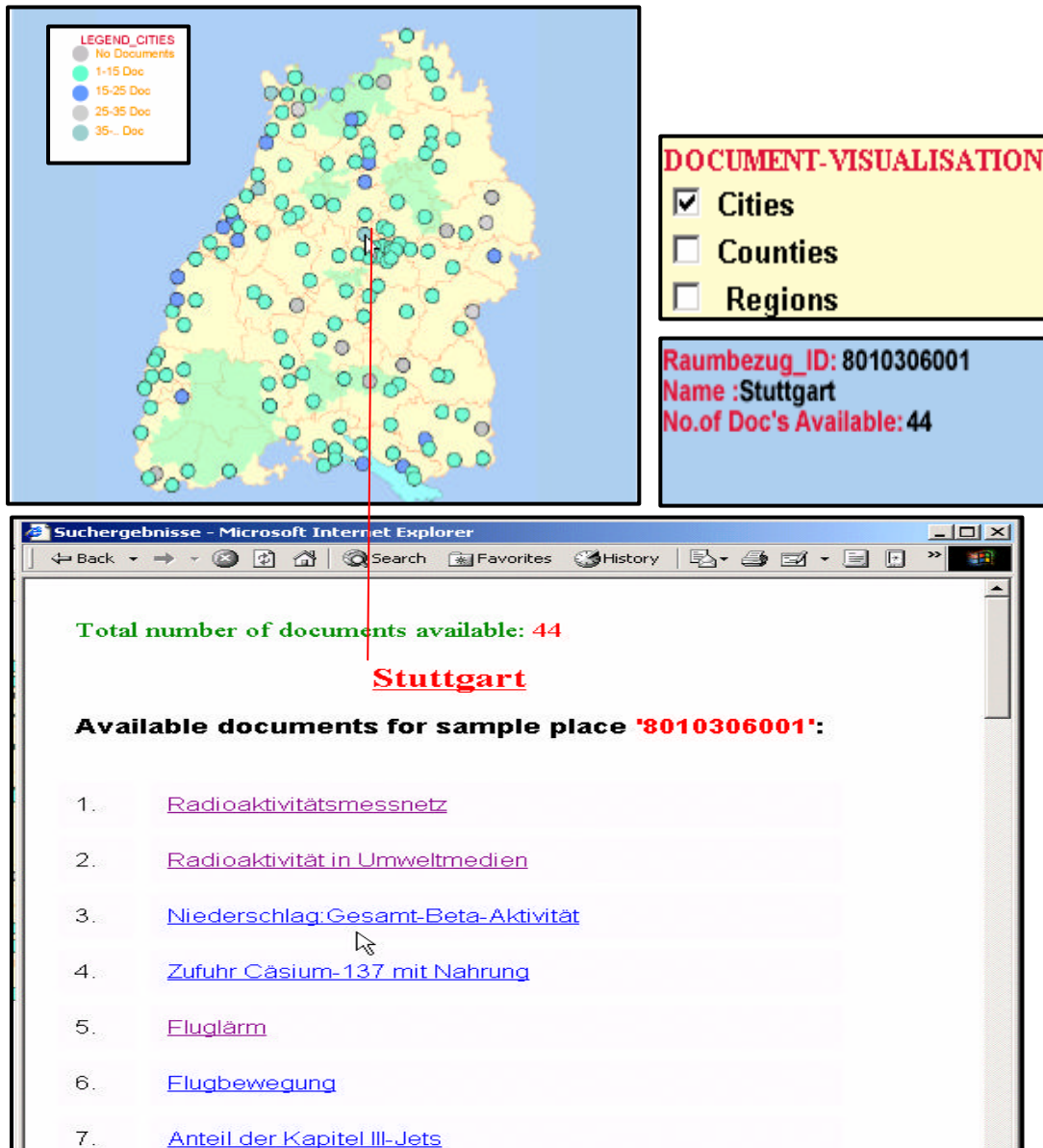


Fig. Showing how to retrieve documents of Stuttgart on Mouse click

#### **4.7 Changing Colors of the Layers over the Map**

One can interactively change the color of the layers by selecting the appropriate color and layer in the list box as shown in Fig. When the user selects color and layer in the list box it will call ColorIt()

function in the java script, where one will get the reference to the legend as well as for the main SVG document and also values selected in the layer list box and color list box.

```

if (document.all) {
    colorVal = ColorVal.ColorSelect.value;
}
else {
    colorVal =
document.Color.document.ColorVal.ColorSelect.options[document.Color.document.C
olor.ColorVal.options.selectedIndex].value;
}

if (document.all) {
    colorVal1 = ColorVal.LayerSelect.value;
}
else {
    colorVal1 =
document.Color.document.ColorVal.LayerSelect.options[document.Color.document.C
olor.LayerSelect.options.selectedIndex].value;
}

```



Fig. Selecting colors and layers from the List Box

Get ElementId and Style property of the Main SVG document and also the Style properties of the Legend. Then set the Color selected by the user to the main SVG document and also to the Legend:

```

svgObj=svgdoc.getElementById(colorVal1);
svgStyle=svgObj.getStyle();
svgtest = svgStyle.getPropertyValue('fill');
if (svgtest == 'none')
{
    svgStyle.setProperty('stroke',colorVal);
    legobj = svgLegDoc1.getElementById(colorVal1);
    legstyle = legobj.getStyle();
    legstyle.setProperty('stroke',colorVal);
}
else
{
    svgStyle.setProperty('fill',colorVal);
    legobj = svgLegDoc1.getElementById(colorVal1);
    legstyle = legobj.getStyle();
    legstyle.setProperty('fill',colorVal);
}

```

#### 4.8 Scale Bar

User can visualize the scale bar, whenever zooming in or zooming out the scale will be changed just like in any GIS software. Here it uses different Java script functions to display scale. When ever the user selects the zoom value in the Zoomvalue list box it will call ZoomIt() function where it will call another function zooming1(scalefact) where this function will update the scale bar. The parameter scalefact is defined manually for each zoomvalue. This scalefact is passed as a parameter in zooming1( ) function. Inside this function first initialize some variables like origscale and massEnd, set default standard scale value to origscale and massEnd as shown below.

```
origscale = 9520;           //Default-Maßstab in Standardeinstellungen
massEnd = (9520/100)*4;     //Rechter Wert in Maßstabsleiste
```

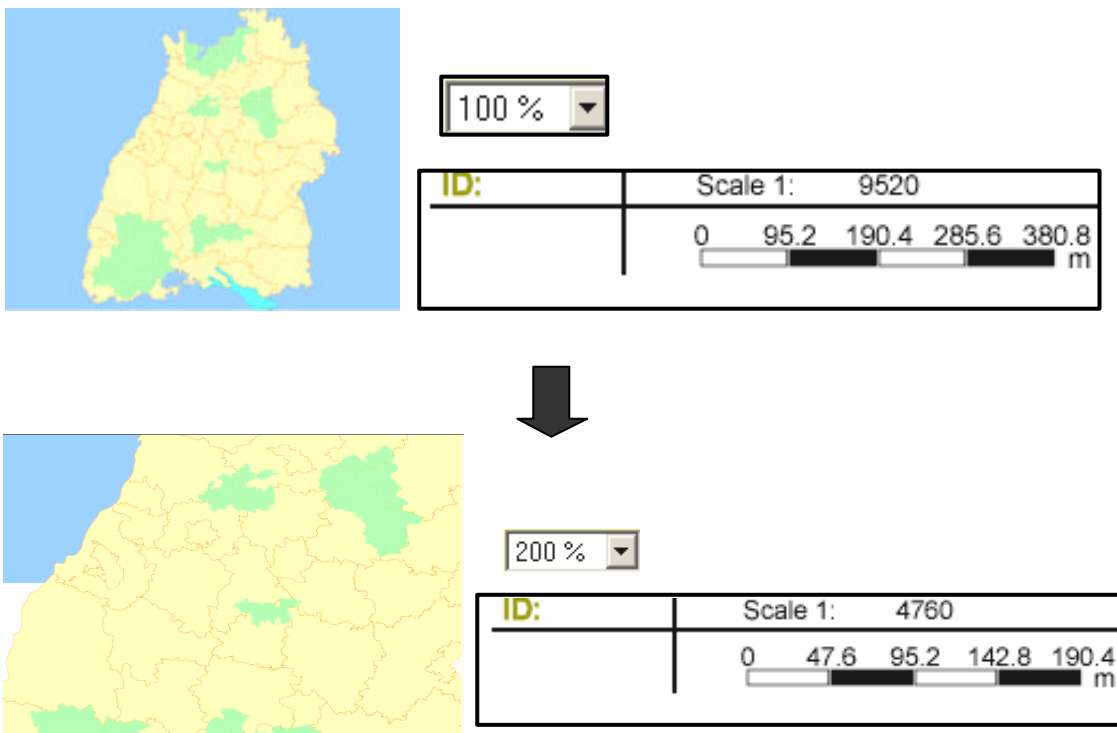


Fig. Shows the change in the scale bar when zoomed in

then get the reference to the main SVG document and the text-element, and get the current scale of that main SVG document. Calculate the map scale by dividing the origscale by scalefactor and set the obtained map scale to the text-element 'idObj'.

```
function zooming1(scalefact){
```

---

```

//Aktualisierung der Maßstabsanzeigen
scalefactor= svgdoc.getDocumentElement().getCurrentScale();
scalefactor=scalefact + 1;
    mapscale = Math.round(origscale / scalefactor);
    textscale.setData(mapscale);
    changeScalebar1(scalefactor);
}

```

#### **4.9 Generalization of The Map**

**Definition:** Removal of detail from a data layer to make processing or visualization easier at smaller scales.

Generalization is perhaps the most intellectually challenging task for cartographers. It has proved to be very difficult to automate.

In recent years, there has been a drive towards automatic map generalization. As geographical information systems (GIS) have become more prevalent, the issue of generalization has increased in importance. Generalization is perhaps the most intellectually challenging task for cartographers, a proposition supported by the comparatively marginal success of computer algorithms in generalizing maps. Generalization is needed in order to represent information on an appropriate level of detail. As only a restricted amount of data can be represented on a certain level of detail (see Figure 2), one can see only rivers and roads at a scale of 1:2380 all other layers were displayed off for this scale. Different pieces of information have to 'fight' for their representation on a Specific aggregation level. This implies that generalization is an optimization problem, where different goals have to be satisfied simultaneously. It is a difficult procedure, as only a limited amount of data can be visualized, perceived and understood at a time.

At a scale of 1:9520 all layers except autobahn and rivers were displayed (see in figure 1). When the user zoomed into the map some of the layers will be displayed off, for example when the user select a zoom value of 400% (i.e., 1:2380) only the Autobahn and rivers will be seen and rest of the layers will be displayed off (see in figure2).

With respect to the principle of generalization SVG elements representing Autobahn and River objects are created dynamically every time a zoom request induces the adaptation to a new target scale, Figure 2 shows the process's results according to specific zoom levels. The remaining SVG element objects were hidden whenever the user zoomed inside the map. To hide the SVG elements, first get the reference to the corresponding SVG element whenever the zoom value is greater than 200% set the style property, visibility as hidden and when ever the user is zoomed out and when the

zoom value is less than 200% set the style property visibility as visible. Detail explanation is given below:

```
if (document.all) {
ZoomVal = parseFloat (selectZoomVal.myZoomSelect.value); //get the
zoom value from the list box
    }
    else {
zoomVal=parseFloat(document.overviewZoom.document.selectZoomVal.m
yZoomSelect.options[document.overviewZoom.document.selectZoomVal.
myZoomSelect.options.selectedIndex].value);
    }
svgObj4=svgdoc.getElementById('Autobahnen.shp'); //get the reference to the
SVG element Autobahn
svgStyle4 = svgObj4.getStyle(); // get the Style property of Autobahn.
legstyle4 = legobj4.getStyle(); // get the reference to the SVG legend element
Autobahn
legstylefill4 = legstyle4.getPropertyValue('fill'); //get the style
property, fill of the above SVG legend object.
if (zoomVal > 200){
svgStyle4.setProperty('visibility','visible'); // Set the style property
visibility as visible at a zoom level greater than 200.
legstyle4.setProperty('fill','yellow'); // set the legend style property fill as
Yellow
}
else {
svgStyle4.setProperty('visibility','hidden'); // Set the style property
visibility as hidden at a zoom level greater less than 200.
legstyle4.setProperty('fill','yellow'); // set the legend style property fill as
Yellow
}
```

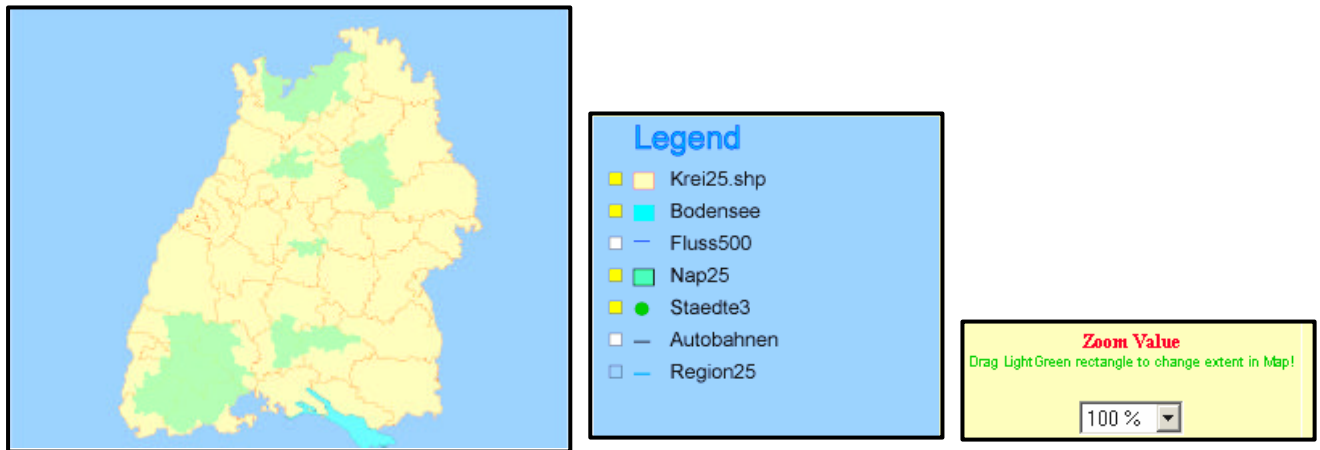


Fig1: All layers except autobahn and rivers are displayed at a scale of 1:9820 (Zoom value=100%)

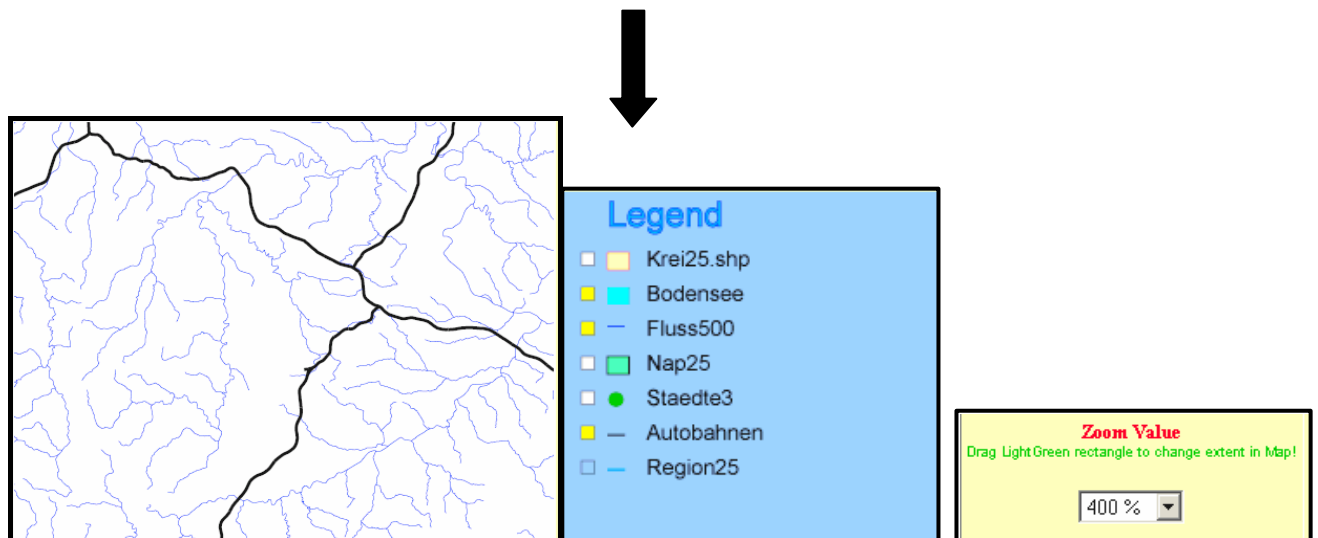


Fig2: Only Roads and rivers are displayed at a scale of 1:2380 (Zoom value=400%)

### **Generalization of Rivers:**

With respect to the principle of on-the-fly generalization SVG elements representing river objects are created dynamically every time a zoom request induces the adaptation to a new target scale, Figure 3 shows the process's results according to specific zoom levels. Compared to the original situation (zoom level 100%) illustrated in the left-hand figure, the other two figures show generalized views of a section of the river network at zoom levels of 50% and 37.5%.

The algorithm is given as follow:

For each iteration

```
{  
For every pixel in the image  
{  
If cell is the same state as its group made by several adjacent neighbor cells  
Keep the state of the cell unchanged  
Else  
Choose the majority cells' value  
}  
}
```

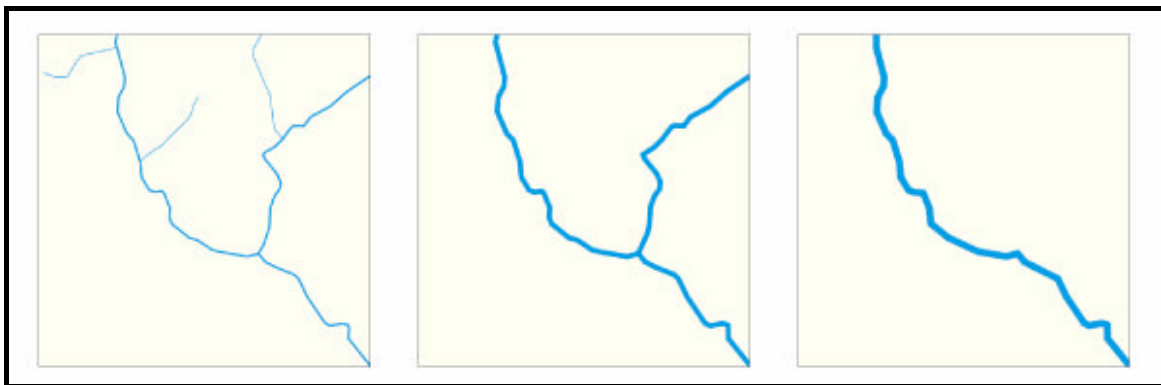
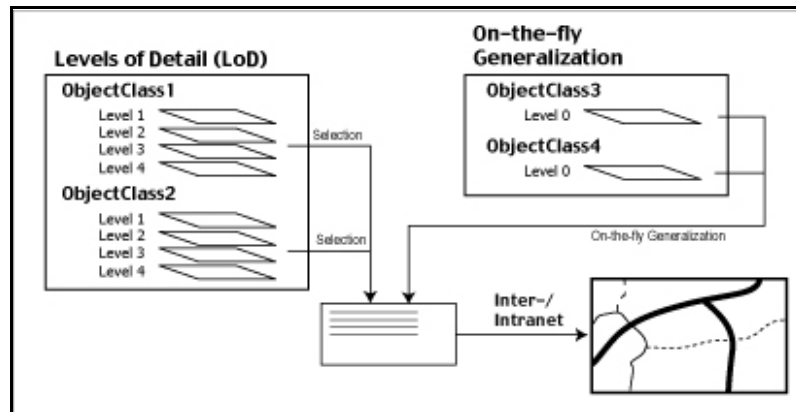


Figure 3: [14] A section of the river feature class and its adaptation to the target zoom level of 100% (left-hand figure), 50 % (figure In the middle) and 37.5 % (right-hand figure)

However, the figure 3 clarifies the way on-the-fly generalization enables adaptive zooming by modifying the content and symbolization of the base dataset: the importance value triggers the omission of objects, i.e. the smaller the zoom level the less river objects are included in the map - cf. Figure 3; the line geometries are simplified by the algorithm and so the river's granularity is reduced at smaller scales; the line-width increases at smaller zoom levels to ensure good perceptibility.

Beyond any doubt cartographic quality lacks in most of the web mapping and web GIS applications that can be found on the web nowadays. The reasons aren't only the technical limitations related to the Internet, e.g. screen resolution, data rate etc., but rather the neglecting of cartographic principles for map generalization. A prototype combining web technologies (SVG, JavaScript) and known well methods from cartography (multi-scale databases, generalization in real time) may be presented. This cartographic Internet service produces visualizing better quality than we are usually used to.



*Fig4:* Creation of maps for the web based on a multi-scale database and generalization in real time (Source: [14] Cecconi and Galanda 2001).

Adaptive zooming terms the adjustment of a map, its contents and the symbolization used to the target scale consequently to a zooming operation (= scale change). In web mapping commonly the concept of levels of detail (LOD) is applied. That is, a certain number of pre-calculated maps that cover the mapped area at different scales is used to adapt to the target scales such an approach implicate three main drawbacks:

- For a wide scale range several LODs are necessary to reach a good adaptation.
- A limitation to the precalculated maps and scales.
- Updates must be propagated through all the scales.

The requirements for on-the-fly generalization are very stringent, because the maps must be generated in real-time, which makes the process extremely time-sensitive. Identify the following characteristics for the on-the-fly generalization:

- A temporary dataset at a reduced scale is generated for visualization purposes;
- The temporary dataset has to meet the user preferences and the technical specifications, i.e. the map specifications;
- The scale is arbitrary and not predefined;
- The generalization process is accomplished automatically, with no possibility of checking the result before publishing.

Next to these characteristics other effects must be respected particularly in the context of the Internet. The performance plays a very important role. Through the generalization process the amount of data, which has to be transferred and displayed, can be optimized for the user's purpose



according to scale and theme. The user wishes to have their own defined map with their own preferences, where they can easily zoom in and out, pan, etc.

On contrary the ideal solution would be to use a single master database and to calculate every arbitrary desired scale on demand by means of automated generalization (on-the-fly generalization). This approach fails too because of the limited time available to react on a zoom request (a limit of 10-15 seconds is assumed) and the complexity of the generalization task. Therefore the combination of the on-the-fly generalization and LOD approach methods are used to implement adaptive zooming (see figure 2) in order to compromise between flexibility in scale and loading time.

LODs should be used for those object classes, which require high computational costs to be generalized automatically, like road networks or buildings. On-the-fly generalization produces on the basis of the database temporary views on run time by applying generalization operators like selection, simplification or label placement. Each zooming request triggers an independent re-generalization process.

The main advantages of SVG compared to SWF lies in SVG's strength as open standard and the better adaptation of it to web cartography's requirements. The most important reasons for using SVG for web mapping in general and for its implementation in the proposed adaptive zooming approach are that SVG:

- Is XML based and compatible to other Web standards;
- Contains a Document Object Model (DOM) that is compatible to the HTML DOM and CSS;
- Enables the use of real world coordinates;
- Supports the import of GIS data and the programming of third party converters;
- Allows interactive and dynamic graphics - SVG objects can be accessed via DOM by any scripting or programming language;
- Complies with high graphical demands.

#### **4.10 Conclusions**

Cartographical, or geographical maps are widely used on the World Wide Web to visualize various kinds of spatial related data. Their purposes are as different as their appearances. With the progress of web techniques, the visualization of maps is becoming more advanced and sophisticated. To attract the users attention and to make their web experience more interesting, the use of interaction has increased with the expansion of www.

Most Web graphics today are in bitmap formats such as GIF, PNG or JPEG. Bitmap graphics must contain information on every single pixel needed to display an image - they're big, slow and dumb. If the user wants to zoom or magnify the image, it will become blurry and crispy. All these formats are compiled and compassed into binary files - which means that they can't be edited and corrected any longer outside the generating software. Up to now, all attempts to deliver high-quality cartography to the web, more or less failed, often due to technical restrictions. Therefore the World Wide Web Consortium (W3C) has brought up a new technology based on XML - eXtended Markup Language. For the first time, SVG, the new Internet vector standard, it opens ways for cartographers to concentrate on content delivery and interactions. This new Web image standard will be radically different to both GIF and PNG as it won't be based on pixels at all but rather on vectors. The two main benefits that vector images offer over bitmaps are small file sizes and resolution - independent scalability that has become absolutely compelling within a Web environment where download times are crucial and viewing platforms vary dramatically.

This results in the user being able to dig deep into an SVG object to obtain more dynamic information - a perfect application for various Internet applications, especially large memory- and processor-consuming interactive maps and GIS.

Cartographical applications range from simple switching on and off of elements and layers changing graphical attributes, reacting to mouse events, such as displaying object data with mouse-over, linked windows, scaling and rotating elements. Finally, with the help of server CGI/Perl scripts, databases can be linked as well. A lot of applications may be implemented, that could enhance cartographer's creativity in future web projects

It is expected that the major Browser's will support SVG in the near future. When they do release versions with native SVG support the market for vector-graphics, animation and interaction will increase rapidly. I think the same goes for vector-maps. As SVG opens new and simpler ways for

cartographers to create attractive, interactive public maps. GML Server-based services is a yet not developed market, which I think, allows further exploration and research. The possibilities to create an entire GIS-software based only on an ordinary web-browser, SVG/XML and scripts would bring us one step further into the future of Internet.

### **Problems In Using SVG**

Of course, SVG is not a "cure-all" for all cartographic problems. One of the biggest problems, which will likely prevent from using SVG, is, that

The source-code cannot be protected. This is not an SVG-specific problem but hits also other documented vector-based internet-file-formats, such as Macromedia Flash. As long as SVG-files are derived from more complex databases this should not be too much of a problem.

One further disadvantage is the current situation of installed SVG-Plugins: Compared to Flash, the number of installed plugins is small. This is likely to change, because the Adobe SVG-plugin is currently bundled with different graphics software, incl. Acrobat-Reader. Several browser vendors or open-source projects are in the state of developing native SVG-support within their browser, so there won't be a need for installing additional plugins.

Finally, there is the problem of missing topology and limitation in graphical symbolization: complex line-styles have to be reassembled from two or more lines. This drawback will perhaps be resolved in further SVG-versions. Topology is not available, unless one writes software to calculate it on the fly.

### **Further possible improvements**

- More classifying-methods: equal steps and manual
- Color choice of all elements via the map legend
- Search function for polygon-names
- Dynamic Generation of Diagrams
- Indicate statistical values such as range, variation, median, average, etc
- Histogram of variables

## **References**

1. FERRAILOLO, J. ET.AL. (2001): Scalable Vector Graphics (SVG) 1.0 Specification,  
<http://www.w3.org/TR/SVG/>
2. The Apache Software Foundation  
<http://www.apache.org/dist/>
3. ADOBE SYSTEMS INCORPORATED (2000): SVG, Scalable Vector Graphics, Release notes, [www.adobe.com/svg/indepth/releasenotes.html](http://www.adobe.com/svg/indepth/releasenotes.html) (2000.06.25)
4. Adobe SVG Viewer Download Area  
<http://www.adobe.com/svg/viewer/install/main.html>
5. WINTER, Andréas M. (2001): »SVG basic shapes«, [http://www.carto.net/papers/svg/shapes\\_e.html](http://www.carto.net/papers/svg/shapes_e.html)
6. NEUMANN, A. (2000): Vienna – Social patterns and structures.  
[www.karto.ethz.ch/~an/cartography/vienna](http://www.karto.ethz.ch/~an/cartography/vienna), Zugriff am 08.09.2002.
7. Winter Andréas, Institut für Geographie und Regionalforschung, Universität Wien  
[www.carto.net/papers/svg](http://www.carto.net/papers/svg)
8. NEUMANN, Andreas and Andréas M. WINTER (2001): SVG web mapping Examples,  
[http://www.carto.net/papers/svg/first\\_e.html](http://www.carto.net/papers/svg/first_e.html)
9. APACHE.ORG (2001): Batik SVG toolkit  
<http://xml.apache.org/batik/index.html>
10. SVG-W3C (2002): Scalable Vector Graphics  
<http://www.w3.org/Graphics/SVG>
11. Treeview, Marcelino Martins: Cross-Browser DHTML tree  
<http://www.treeview.net/treemenu/instructions.asp>
12. SVG main\_papers, Winter André, Institute for Geography and Regional Studies, University of Vienna  
[http://www.carto.net/papers/svg/index\\_e.shtml](http://www.carto.net/papers/svg/index_e.shtml)
13. W3C, Word Wide Web Consortium (2000): Scalable Vector Graphics (SVG) 1.0 Specification, Candidate Recommendation, 02 August 2000,  
<http://www.w3.org/TR/SVG/> (2000.08.03)
14. Generalization for On-demand Web Mapping, Cecconi and Galanda 2001  
<http://www.geo.unizh.ch/gis/research/webmap/gendem/>

15. svg examples basic coordinate\_transformations,( 02-Apr-2003) cart.net

<http://www.carto.net/papers/svg/samples/matrix.shtml>

16. An Introduction to The Common Gateway Interface,

<http://www.utoronto.ca/webdocs/CGI/cgi2.html>

## **Contact Details:**

---

### **In Germany**

**Sudhir Kumar Reddy Maddirala**  
**Dept.of Geomatics,Computer Science and Mathematics**  
**Schellingstr 24, Stuttgart D-70174,**  
**Germany**  
**E-mail:masus110@mars.rz.fht-stuttgart.de**

\*\*\*\*\*

**Allmandring 26 A,52.2**  
**70569 Stuttgart**  
**Germany**  
**Tel:0049-179-7984060**

### **In India**

**17-1-388/C/23,Vinaynagar,**  
**Saidabad, Hyderabad-59**  
**Tel:0091-4024044143**  
**E-mail: sudheer75@yahoo.com**  
**sudheermaddirala@yahoo.com**