

## USING VISUAL SPATIAL SEARCH INTERFACE FOR WWW APPLICATIONS

XIAOFANG ZHOU<sup>1,2</sup>, JOSEPH D. YATES<sup>2</sup>, and GUIHAI CHEN<sup>1</sup>

<sup>1</sup>State Key Laboratory of Novel Software Technology, Nanjing University, China

<sup>2</sup>School of Computer Science & Electrical Engineering, University of Queensland, Australia

(Received 30 September 2000; in final revised form 19 February 2001)

**Abstract** — Most Internet search engines are keyword-based. They are not efficient for the queries where geographical location is important, such as finding hotels within an area or close to a place of interest. A natural interface for spatial searching is a map, which can be used not only to display locations of search results but also to assist forming search conditions. A map-based search engine requires a well-designed visual interface that is intuitive to use yet flexible and expressive enough to support various types of spatial queries as well as aspatial queries. Similar to hyperlinks for text and images in an HTML page, spatial objects in a map should support hyperlinks. Such an interface needs to be scalable with the size of the geographical regions and the number of websites it covers. In spite of handling typically a very large amount of spatial data, a map-based search interface should meet the expectation of fast response time for interactive applications. In this paper we discuss general requirements and the design for a new map-based web search interface, focusing on integration with the WWW and visual spatial query interface. A number of current and future research issues are discussed, and a prototype for the University of Queensland is presented.  
© 2001 Elsevier Science Ltd. All rights reserved

*Key words:* WWW, Spatial Search Interface, Map-Based Search

### 1. INTRODUCTION

The growth of the Internet over the last decade has led to an explosive growth of information availability. However, this information is relatively useless unless it is easy to find by the average Internet user. An important principle of the Internet is to allow users to find the information they are seeking in the shortest time, with the least trouble using the most natural means. By typing in a few keywords, many Internet search engines, such as AltaVista, can quickly find the URLs of most documents containing these words. These websites can be from anywhere in the world. However, when the user is interested in the information from a particular area, this type of keyword-based search is not very effective. A query using locality or landmark names could return much irrelevant data as the same names are frequently used in many places and are also used for other purposes. On the other hand, such a query could also miss a lot of relevant data because a place can be known by different names and place names often have a hierarchical containment structure (therefore, those names at a higher level of the hierarchy may also contain relevant information, but cannot be found by keyword-based search engines). An alternative way for this type of locality-based search is by category-based browsing. For example, currently the most efficient way to find hotels in a city on the Internet is probably to visit the websites of travel companies. However, these specialised sites may not serve all of the user's needs. For example, once they have booked a hotel, they might want to see what restaurants and bookshops are in that area and rank them in terms of the distance from the hotel. Today's web search and browsing tools can hardly support these types of requests.

There is a clear need for a map-based web search where the user is presented with *an interactive map* and *a spatial search interface*. The user can use the map to locate a region of interest and form the search condition (for example, to find only those URLs containing a set of keywords, but related only to a user defined region on the map). The map can also be used to display the locations of the search results, and more importantly, to use the locations of some search results to form new queries. To demonstrate distinctive features of the map-based web search tool proposed in this paper, we will consider the following scenario.

Maria is travelling overseas to attend a conference. She has been notified of the conference venue and decides she would like to stay at a hotel within walking distance of the venue. She does not like any of the three hotels recommended by the conference organiser and decides to find a hotel by herself on the Internet. She has little knowledge of the city. Maria visits the website for the venue and then links to a

map-based navigator to automatically display the map of the surrounding one kilometre of the venue. She decides that she is prepared to walk two kilometres to the conference and so zooms the map out to display the map within two kilometres of the venue. She then specifies a simple query stating that she is looking for any four star hotel within the bounds of the map. A number of different hotels appear on the map. Maria selects each of them one at a time to view their details and web pages, then decides to book her stay at one of them using her credit card. She also asks to display all Chinese and Thai restaurants within 500 meters from that hotel, and goes to their websites to view prices and menus. Finally, she prints out a map of the area, customised to her needs.

While there is an increasing number of websites using a map for navigation, most of them are very primitive. Typically, they use simple raster maps, which do not support queries using spatial conditions. Some web search systems, such as the Yellow Pages service at <http://www.yellowpages.com.au>, can display the location of search results, nevertheless they do not support spatial queries. In other words, most maps used in existing systems on the Internet are not an integral part of web searching.

There are a number of issues to be addressed before a web-enabled map interface can be considered an effective and efficient web navigation tool. Firstly, a simple, intuitive search interface needs to be designed, through which an average user could easily search the web using the map. In addition to supporting various queries using spatial conditions, such an interface should also support aspatial queries. Typically, spatial queries are issued using an SQL-like language [11]. It can be too difficult to require the user to understand the semantic meaning of spatial predicates and learn how to use them to write a spatial query. It is certainly not very convenient if a user is forced to use different interfaces for spatial and aspatial queries. There are many attempts to design simple, visual spatial query interfaces [1, 2, 6, 7]. However, no work has been reported specifically for searching the web using spatial attributes. Secondly, the map-based web search interface should be integrated with other websites seamlessly. To a minimum extent, this means that various bits of information can be associated through hyperlinks to spatial objects easily. At the same time, a mechanism should be provided to relate to spatial objects those entities, which in general do not have a location attribute (e.g. a conference related to its venue, a person related to her office, or a book related to its location in a library). Using such a system, one could perform a query such as "Where is John Smith?". After his office is located and displayed on a map, one could obtain further information surrounding the office, such as road names, building photos, parking places and parking conditions, by viewing the map and clicking objects of interest on the map. Thirdly, such a system should be scalable. On one hand, this means that the performance of a map-based web search tool should not degrade when the geographical region it covers or the websites it can search and link to increase. On the other hand, this also requires manageability of a very large amount of information, such as associations between spatial objects and their related hyperlinks.

Some preliminary research results on the design of a web enabled map-based navigator are reported by the authors in [13]. In this paper, we provide a more powerful visual spatial interface to search the Internet. We discuss solutions for the key problems in the context of a generic vector map-based web search tool. A spatial database system is used to support efficient storage and manipulation of spatial data, as well as mappings between spatial objects and their attributes (such as hyperlinks). The adoption of the modern database management technology that supports efficient spatial query processing is the key to achieve scalability. A simple and intuitive spatial search interface is designed to allow a novice user to search the web using spatial conditions with ease. This interface is also expressive enough to issue most types of spatial queries, including spatial join queries. The same interface also supports, in a very natural way, conventional queries without a spatial component. Performance issues in using vector spatial data for Internet applications are also discussed. To examine the soundness of our design and demonstrate that a map-based search tool is a valuable addition to the range of Internet searching and browsing tools available, a prototype was developed for the University of Queensland. The prototype is optimal for the domain of a university, however, the major software components, such as the map viewer and backend database design and processing, can be reused for other applications. The whole system can be easily modified and extended for use with other organisations and eventually a significant portion of the entire Internet.

The rest of this paper is structured as follows. We discuss the design issues of a map-based web navigation tool in Section 2 and Section 3, focusing on the overall system design and the visual spatial query interface design respectively. The design and experiences for the development of a prototype for the University of Queensland is presented in Section 4. Some current and future research issues on this new web search paradigm are discussed in Section 5. We conclude this paper in Section 6.

## 2. SYSTEM ARCHITECTURE

In this section, we discuss different design alternatives for a generic map-based web search tool. Below is a set of criteria to guide our design.

- The tool should be able to handle different spatial data sets, from street maps, land conservation to other types of geographical data, without the need to modify major components of the tool. This would allow for easy distribution of the system specifically for organisations that want to adopt a map-based navigator to their website.
- The costs involved in query processing and data transfer must be kept to a minimum to ensure suitability for an Internet environment. Ideally, the amount of time required is independent of the total amount of data stored in the database. Also, when the amount of spatial data for a request is too large, some spatial simplification and generalization are needed [10,14].
- The costs involved in processing on the client side must be reasonable for a standard personal computer.
- The tool must be machine and platform independent to allow for worldwide use on the Internet.

### 2.1. Spatial Data

First, we discuss the data structure for the spatial data used to draw the map. As mentioned before, most maps used on the Internet nowadays are raster images. One problem with using raster maps is that it is cumbersome to associate hyperlinks with objects on a raster map, in particular at a fine level (e.g. for each of a large set of buildings). Moreover, high quality images usually come with a very large size. These maps are slow to download, in particular over low bandwidth links such as over a phone line and modem. This performance problem is amplified by the fact that adding/removing a layer or zooming/panning on a raster map can only be done by requesting a new image from the server. Other issues with raster images are: 1) Zooming to a low depth will pixilate the image; 2) Generally higher processing required to analyse the image to perform spatial queries; 3) Limited accuracy depending on the quality and nature of the image.

Vector data, such as points (e.g. cities and post offices), lines (e.g. roads and rivers) and polygons (e.g. city boundaries, suburbs and land parcels), are more suitable for this type of Internet application, especially when the application runs as a Java applet on the client side. The most prominent advantage of using vector data is that it becomes possible to perform some spatial processing on the client side, such as zooming in and out, and turning on and off layers. New data is downloaded from the server only when it is not available in the client's cache (for example, when a new layer is required). With vector data, the map image does not pixilate when zooming in. Grouping objects into layers and associating hyperlinks with spatial objects is a difficult task for raster maps but becomes trivial for vector data. In addition, vector data can be small and compress extremely well. There are a number of Internet-based Geographical Information System (GIS) applications that use vector spatial data [3]. In this paper we will use vector spatial data for the map used for navigating the web. This approach assumes the availability of high quality and reasonably complete vector spatial data. This is not a major issue as many government agencies and large utility and mapping organisations have accumulated highly accurate spatial data over the last two decades, which is comprehensive in terms of geographical coverage and thematic layers (e.g. roads, land parcels, shops, schools etc.). With advances in remote sensing and computer-aided digitising, the task of obtaining vector spatial data has become much easier than before.

Compared with using raster data, there are two problems associated with using vector data that need to be dealt with carefully, namely, the overhead for rendering data on the client side and potentially a very large amount of data needed for a map covering a large area. These problems will be discussed later in this paper.

### 2.2. System Design

As part of the user interface, there is a vector map viewer for the user to define spatial condition of a query as well as to display search results on a map. Many operations, from standard graphics operations such as zoom and pan, to more complex spatial selection and queries, can be issued within the map

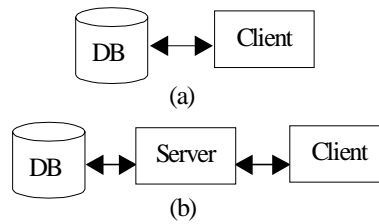


Fig. 1: The Two-Tier and Three-Tier Architectures

viewer. There are a number of advantages, as explained before, for the map viewer to be run on the client machine. It can be an application that requires installation on the client machine or runs in a web browser as a Java applet. If the client component is designed to be an application, the client software needs to be downloaded and installed which is inconvenient if using a machine with installation capabilities removed (i.e. a library terminal). This approach also makes it more difficult to achieve seamless integration with other tools and sites on the Internet. On the other hand, if the client is web browser embedded, the client is forced to be lightweight. We prefer to embed the client in a web page and treat it as any other web page, because this allows easy integration of the new search tool with the Internet. Another reason is that it is important for a search engine to be used by as many users as possible with minimum effort (i.e. not to install any software locally).

There are two types of architectures commonly used in Internet-enabled GIS and database applications: the two-tier (Figure 1(a)) and the three-tier (Figure 1(b)) client-server architectures. The major difference is whether the client talks directly to the database system or via a middle layer (i.e. the server). The two-tier approach allows the client to be generic at the expense of a permanently fixed data structure in the database server or alternatively a flexible database structure at the expense of a non-generic client. Neither of these is satisfactory – the database structure must be flexible while allowing for a generic client. The three-tier solution on the other hand does allow for the client to be generic as well as allowing for flexibility in the database server structure and DBMS product as there is a middle tier acting as a mediator between the two. For this reason the three-tier option is required to maintain its generic nature.

In either case the client has the ability to be lightweight, leaving a large portion of the processing to be done on the server side while permitting the client to perform simple tasks such as zooming in and out, panning and identification of selected objects. This means that the client can be run on almost any machine, as the speed of it is not the limiting factor.

Now that we have identified the fact that a three-tier architecture is desired we will briefly discuss the tasks performed by the individual components.

The task of the database server is to store all the spatial and aspatial data required for the system to run as well as performing much of the spatial processing. It needs also to store some meta data which the client requires to maintain its generic nature.

The server process (running on a web server) will be designed to process the client's request, retrieve data from the database server and then return the data to the client in an appropriate format. This is the only component that directly interacts with the database server (allowing for the client to be generic).

The client process (running in a web browser) is the only component that the user will visibly see. This component requires care in terms of design of the user interface and the spatial query language (see Section 3). The client must be written in a programming language that can be used within a web browser on any machine by any user connected to the Internet.

The bottleneck of the system will, in general, be caused by the speed of the Internet link between the server and the client. For spatial Internet applications, however, the interaction between the server and the backend spatial database can also be very significant as far as processing time is concerned. We will discuss the later further in Section 5.

### 2.3. Database Design

A map-based web search tool can potentially use a very large amount of spatial and aspatial data. For example, the spatial data for the land information (land parcels, roads, rivers etc) of a medium city can be

|                 |   |
|-----------------|---|
| <i>id</i>       | a number that uniquely identifies the spatial object. |
| <i>name</i>     | the object name (such as road name).                  |
| <i>layer</i>    | the thematic layer of the object (e.g. road, river).  |
| <i>size</i>     | number of points in the geometry attribute.           |
| <i>geometry</i> | a sequence of points defining the object.             |
| <i>mbr</i>      | the minimum bounding rectangle of the object.         |

Table 1: Schema of Spatial Table

from 200 MBs to a few GBs depends on data resolution. Each spatial object can be associated with many aspatial attributes, such as street names and all organizations in buildings. The only way to support efficient management of such huge amount of complex data is to use a modern DBMS. Such a DBMS should support spatial data types so that many critical tasks, such as spatial data indexing and spatial operations [9], can be performed. This will greatly simplify the complexity of the design and implementation of a map-based web search tool while speeding up query time.

In this subsection, we discuss new tables needed for supporting vector spatial data, and for supporting associations between spatial data and other information, including Web sites. The schema of the spatial table is shown in Table 1. Every spatial object has an *id* that is used for associating other information with the object. For example, when the user clicks an object on the client side, its *id* will be sent back to the server for requesting additional information. The minimum bounding rectangle (MBR) of a spatial object is an approximation of the object, commonly used in spatial databases to improve the efficiency of spatial operations [9].

There are a number of aspatial tables. Three of them are worth mentioning at this point. There are:

```
LAYER(layer, parent_layer)
LINK(spatial_object_id, type, value)
MAPPING(object, type, spatial_object_id)
```

The table `LAYER` defines a hierarchical structure of layers. While the *layer* attribute in the spatial table above groups spatial objects of the type, this table allows more flexible groups at different semantic levels. For example, a tuple (*creek*, *water\_system*) in this table says that creek is an instance of water system. More examples can be found in Figure 2. The table `LINK` associates various information with spatial objects. For example, (*78*, *HTML*, "*www.csee.uq.edu.au*") means that object 78 has an HTML link *www.csee.uq.edu.au*. Because the linkage between an object and an information source is recorded in a database table, multiple links can be associated with an object (see Section 4 for an example). This is different from the traditional approach that one object can only be associated with one hyperlink. There is also another table, `MAPPING`, which is used to map arbitrary entities (such as conferences, people, and books) to a location (i.e., a spatial object). For example, ("*John Smith*", *PERSON\_OFFICE*, *78*) and ("*John Smith*", *PERSON\_HOME*, *1145*) map John Smith to spatial object 78 in the context *PERSON\_OFFICE*, and to object 1145 in the context of *PERSON\_HOME*. Through these two tables, associations between spatial objects and their linked information are established by adding proper tuples, rather than hard-coded. This makes it is easier to establish and manage such links.

### 3. VISUAL SPATIAL QUERY INTERFACE

A visual spatial language is a term used to differentiate with a command spatial query language. A spatial query language is the language through which users communicate with a spatial information system by formulating instructions to retrieve and represent spatial data. The success of a spatial information system is largely dependent on this query language, as it is the only mechanism that can be used to access the information system. For this reason it needs serious attention especially now that non-specialist users are utilising spatial information systems. In spite of much research work in this area [1, 2,

6, 7], this is still an active research area with many open issues, in particular for answering new challenges arising from new applications such as web-based spatial applications. In this section, we will look at this aspect of spatial information systems in the light of the domain we are working with.

### 3.1. Past Approaches

In this subsection, we discuss a variety of approaches that have been presented to date. The most fundamental of all spatial query languages is SQL or at least extensions of SQL. SQL dialects are the basis of most other spatial query languages. SQL in general is very popular, however, we know that typical end users of a conventional information system do not type in SQL queries to the database. Instead the user is usually presented with some type of form to fill in; abstracting the query to something the user is familiar with. For this same reason it is impossible to expect end users of a spatial information system to type in spatial SQL queries to the database because spatial queries are far more complex to formulate. With this thought in mind, our task is to create an intuitive mechanism by which these end users can easily interact with the system. With the advent of Graphical User Interfaces (GUI) and pointing devices it seems foolish not to make use of their many benefits. However, while doing so, we must not neglect the usefulness of lexical input and output. This is precisely what researchers have aimed at doing.

**Command Line Interface** One exception to this is the standard command line interface, which is not often used purely by itself these days. It is at one extreme of the spectrum of the types of languages available. With this approach, users type in expressions in a formal language in order to formulate a query. One such example of a formal language is in fact SQL. Although this type of interface can be powerful, it is often extremely difficult to use because of the necessity of the user to know all the available commands, their usage, as well as the data structures (at least sometimes). One of the additions to command line interfaces is the ability to augment the user's lexical input with mouse input. This allows users to select objects on the map which are to be referred to in the query. An example of this can be seen in [7]. Despite the relatively limited user-friendliness of these systems it was a vast improvement over the plain command line interface.

**Spatial Query-by-Example** A more intuitive approach to spatial query languages that makes use of GUIs and pointing devices, is a spatial version of query-by-example (QBE) [1, 2]. With this approach, the user is presented with a working space on the screen where they can formulate queries graphically. The user has available a number of spatial operations and object types to work with represented by icons on the screen. The user drags and drops various icons across to the working space to formulate an example of the query graphically. Additional constraints can be added to the icons on the working space to provide restrictions on certain objects.

**Spatial Query-by-Sketch** A very similar approach to spatial QBE is spatial-query-by-sketch [6]. The overall concepts are the same but the means by which the user interacts with the computer is somewhat different. Instead of dragging and dropping icons from one part of the screen to another the user actually draws on the screen using either a mouse or a touch sensitive screen and pen. The end result is much the same as that in the spatial QBE, the major difference is the input mechanism. One would also expect that the query resolution time for this approach would be higher than in spatial QBE as the system must decipher the user's drawing.

**Forms** In many conventional information systems, the user can select the type of query they wish to perform from a list of prefabricated queries. The query that they select appears on the screen as a form fill-in, where all they have to do is to fill in the blanks. This technique has been applied to spatial information systems as described in [12]. One problem with this is that usually the forms are static resulting in a large number of prefabricated forms needed in the system (one for each query type). Some spatial information systems should allow users to formulate complicated queries and ad hoc queries. By modifying this form fill-in approach slightly we can remove the need for so many forms and provide just a single form that advances the user through the query formulation process step by step. This form "wizard" will dynamically change the next step to be taken on the form based on all the input for the query to that point in time. This concept is similar to that of the wizards used in Windows 95.

### 3.2. Criteria for a Good Query Interface

When developing a spatial query language it is vital to know exactly what characteristics it should have and the criteria it should fulfill. Here we have identified a list of requirements which, in general need to be satisfied by spatial query languages. Note that this list has been partially adapted from [5] and extended upon based on our own research. For additional requirements see [5].

- The coexistence of lexical and graphical data is needed so that users can simultaneously view the rendered map along with associated aspatial information.
- Query formulation needs to be natural for the user while allowing the specification of complex queries. To achieve this the query language needs to be abstracted from the real workings of the system.
- Progressive formulation of queries needs to be supported to reduce the overall complexity of the query formulation process.
- The need for the users to learn a new language needs to be removed (or at least reduced as much as possible) encouraging the use of intuitive interfaces.
- Attention needs to be given to both the specification of queries *and* the rendering of the results because the effectiveness of spatial information systems is dependent on how the users request information *and* in which form it is presented to them.
- The results of queries need to be placed in a context so that the user can make sense of them in light of the larger picture.
- Efficient utilisation of screen space is always an issue and needs to be considered.

We will present the design of a user interface, which follows these guidelines in the next section.

### 3.3. A Visual Spatial Query Interface

Next we will examine typical spatial queries a web user may ask, and design a simple, intuitive, visual query interface which can support spatial as well as aspatial queries.

Below is a list of queries that can be asked using a map-based interface.

- Q1: “What is *this* object?”
- Q2: “Where is the Computer Science building?”
- Q3: “Display all information associated with *this* object”
- Q4: “Link *this* object to the Geography Department’s website so I can browse it”
- Q5: “Show all refectories within 200 m of the General Purpose South Building”
- Q6: “Show me all the buildings in the currently viewable portion of the map”
- Q7: “Where is John Smith’s office?”
- Q8: “Show me all the buildings which do not have a Red Permit parking zone within 100 m”

More details about spatial query classification can be found in [8, 9]. Above queries are derived from a user survey and filtered by restricting to queries using basic spatial operations. More complex queries, such as “how to get to the Library from the Bookshop”, are not considered.

Those queries with aspatial input but require displaying results with a map (e.g., Q2 and Q7, in spite of that John Smith as a person does not normally has a spatial location attached. See Section 2.3) are not the focus of this paper. These aspatial queries are widely used in many currently available raster image based web applications (although efficient and effective display of resultant map is not a trivial issue, as one can see in Section 5). For the queries with spatial input, we group them into three categories:

- 1) *Point Queries*, where requested types of spatial objects (and their related attributes as per query) covering a point (i.e., where the mouse is clicked) are returned.
- 2) *Spatial Selection Queries*, which specify a spatial relationship (e.g., overlap, within distance etc) and a specific spatial object. The resultant spatial objects of a spatial selection query should satisfy the spatial relationship with the given spatial object. The spatial object here can be specified using the name of an object, or by identifying the object on the map. The latter is particularly useful to form progressive spatial queries, when a new spatial query is built on the results of previous queries. Note that a window query (e.g., Q6) is a special case of a spatial selection query.
- 3) *Spatial Join Queries*, which are similar to spatial selection queries except that the spatial objects used in query conditions are not specific instances, but a type of spatial object.

Q1 and Q3 are point queries. Q4 is also a point query, but it asks to establish links between spatial objects and other information thus it is a restricted query that can only be used by authorized people. Q5 and Q6 are selection queries. Q5 is also known as a within-buffer query, and Q6 is called a window query. Q8 is a spatial join query between any buildings and any Red Permit parking zones.

The above discussion may not be that straightforward to the user who may have little knowledge about spatial databases or even relational databases. It is vital to hide all these details from the user of the spatial web search interface. Actually, the user should not even be aware of specifying spatial or aspatial conditions. By using the spatial map viewer, it becomes very easy to specify a window or to choose any object, by simply zooming, panning, clicking and dragging on the map. That is, it is not much different from clicking a hyperlink on an HTML page in order to perform a point query or to specify a spatial object for spatial selection query.

Based on the above analysis, now we can propose a visual spatial query interface which is simple yet powerful enough to support the types of queries discussed. It consists of four key components:

- 1) *Keywords*. Recall that our aim is to provide a spatial interface that is compatible with conventional keyword-based search interface. So our new interface should still support keyword based search. We set the default value for the keyword to “everything”, which is reserved word in our system. If the user does not change this default, it means that the query is likely to be specified purely by spatial conditions.
- 2) *Type*. As we use a spatial database as the backend search engine, the data our system searches is better structured. Therefore, we introduce this component to make a query less ambiguous. For example, “John Smith” can be a person, a book, a building name or a road name. Irrelevant results can only be filtered out if object type is specified in a query. This component is also related to layers in spatial databases. The default for this attribute is “all types”, in case the user is not sure how to set this value.
- 3) *Spatial Relationship*. The user can choose, from a drop-down menu, a spatial relationship to be used in the query. Spatial relationships can be inside, contain, adjacent, overlap, disjoint, and within (distance). If within is selected, a distance should be specified. The default is inside.
- 4) *Spatial Objects*. A spatial object can be defined by choosing the entire map (including those not currently viewable), by choosing only the current viewable part, by typing the name of an object in the map viewer (so a search is performed there), or by clicking an object in the map viewer, or by drawing a window on the map viewer, or by specifying a set of spatial objects via invoking a sub-screen containing components 1-2 above (for spatial join queries). The default is the entire map.

Note that the combination of the default values for components 3 and 4 essentially means that that no spatial condition is specified (thus the query is purely aspatial). In this way, the difference between a spatial and aspatial query has been eliminated. Without a spatial map viewer, this is not possible.

In Section 4.2, we give an interface designed for a university campus web search. In that prototype, we further simplify the interface. Spatial relationships other than within-distance and spatial join queries are not supported after balancing query interface complexity and usefulness of these complex spatial queries.

## 4. PROTOTYPE

In order to verify that map-based navigation is a valuable contribution to GIS, spatial database and Internet research areas, we built a prototype demonstrating a number of the above-mentioned concepts. The prototype was based on the domain of the University of Queensland. While a relatively small-scale prototype is discussed here, the whole system is designed to be able to work with significantly larger domains. Several software components, such as the map viewer and backend database design and processing, can be reused for other applications.

### 4.1. Implementation Details

**Database Server** The database server is implemented using Oracle8 Database Server storing the spatial data in vector format. The database consisted of over 1,600 spatial objects or over 20,000 vertices. The size of the spatial and aspatial data in total comes to approximately 450Kb, representing about 2.2 square kilometres (the size of the University of Queensland St Lucia campus) in relatively good detail.

As a result of using vector data it was not necessary to perform all of the spatial operations on the database server. Instead, a portion of these operations, such as selection of objects, zooming, panning and selection of map layers, were performed on the client side.

Indexes are created on the spatial and mapping tables discussed in Section 2.3 wherever it is useful. Spatial objects are indexed on the MBR. With these indexes, the database processing time for most types of selection queries will be efficient and independent of the table size.

**Server** The server is implemented as a Java 1.2 application running on an Apache 1.3.6 for Windows web server. Two-way communication between the server and the client was achieved by utilising Java's implementation of sockets. Communication with the database server was also achieved by means of the Java Database Connectivity – Open Database Connectivity Bridge 1.2 (JDBC-ODBC Bridge) and JDBC 2.0 allowing SQL queries on the database to be issued by the server.

The primary task of the server is to receive requests from the client, translate them into SQL, request for the database server to perform them and then return the results in an appropriate format to the client. The fact that the client does not communicate to the server in the form of SQL queries means that the client is data independent. This allows the database server to change data structures, data models or even product without visibly impacting the client or user. Since the server acts purely as a mediator between the client and the database, it resulted in being lightweight. To improve efficiency the server assists in reducing the data transfer involved between server and client, by making use of some data compression techniques.

**Client** The client process is implemented as a Java 1.2 Applet running in an Internet Explorer 5 web browser with a Java Runtime Environment (JRE) 1.2 Plug-in. It makes use of the Java "Swing" package, which is an extensive GUI toolkit containing a large number of widgets which assisted greatly in developing the GUI. One important aspect of the client is when it performs a request to the server (and in turn to the database) the retrieved information becomes cached at the client side to remove the need for retrieving the information again at another time. This can be done without any need to worry about "dirty" reads because generally speaking the database is read only. Another reason why caching is desirable is so that the client can perform some spatial processing as mentioned before. The client is the core component of the system and obviously is the component that most time was invested in because of both the need for an intuitive user interface and its need to be efficient. The next section focuses more on this component and its functionality.

### 4.2. GUI and Core Features

In this section we present snap shots of the GUI developed for the prototype and discuss the functions of each of the physical groupings of components on the screen. We begin with the map itself. When the user begins their session with the map-based navigator they are presented with a screen similar to that in Figure 2. On the left-hand side of the screen is the map of the University of Queensland zoomed out so

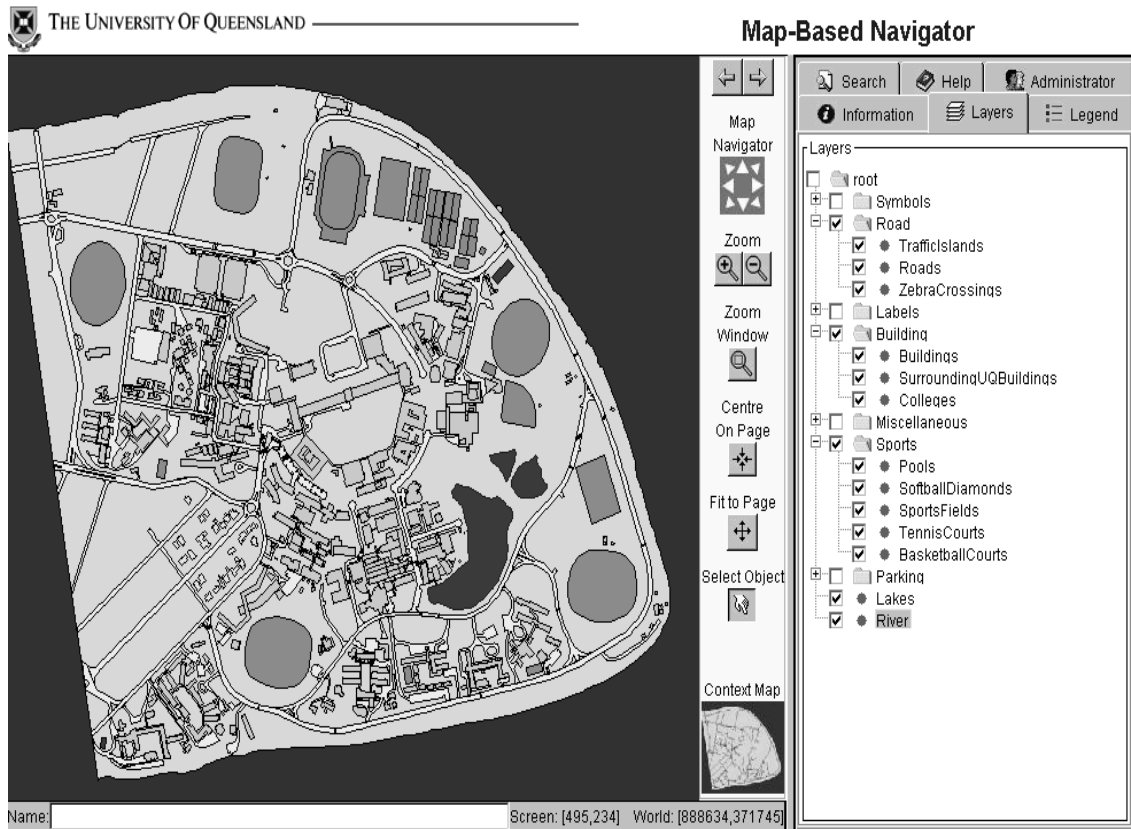


Fig. 2: A Snapshot of the GUI with Layer Hierarchy

that the entire map fits on the panel. This is called the display panel and will be used for graphical output and user interaction with the map throughout the session. This display is synchronised with each of the user's interactions with the system.

In the middle of the screen is a toolbar with a variety of mainly navigation-oriented buttons. They perform fundamental operations that one would expect to be able to perform. Some of these buttons require interaction with the display panel in addition to the selection of the button; others just require selection of the button. From the top to bottom, the functions available on the toolbar are:

- Undo and Redo last action
- Scroll North, South, East and West etc.
- Zoom In and Out
- Zoom In to a given window
- Centre map on the display panel where the user clicks
- Fit the map to the display panel
- Select object on map and request associated spatial data
- Context Map (in our prototype, user is unable to interact with this – they are just able to view it to gain some insight into where on the display panel they are currently zoomed in to)

At the bottom left of the screen is a simple text field which allows users to type in the name of an object of interest for a quick search. The map will be zoomed to a proper level and the objects found will be highlighted on the map.

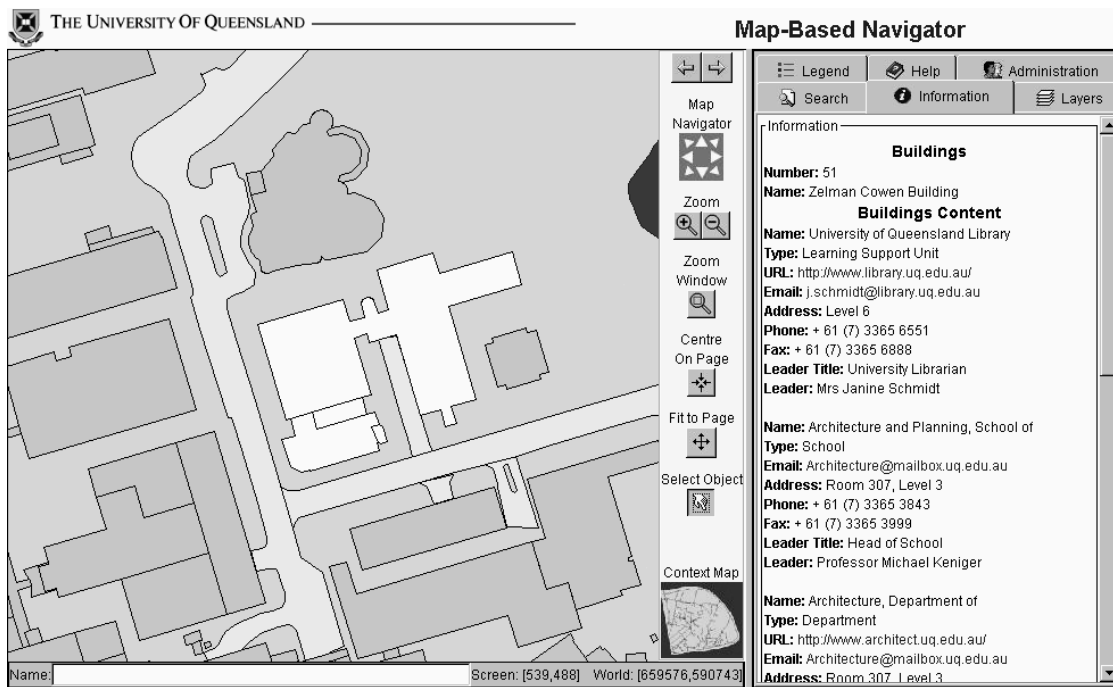


Fig. 3: A Snapshot with a Building Highlighted and Its Associated Information Displayed

On the right side of the screen there is a set of tabbed panels which optimise the use of screen space. An assortment of operations can be performed using them. We will describe several of the more significant tabs.

Still looking at Figure 2, the first tab encountered is called “Layers”. Intuitively, the purpose of this is to allow the user to select and de-select layers of the map. The fact that the tree structure is in the same form as a Windows 95 Explorer tree means that most users would be familiar with this concept. When a layer is selected by clicking a check box, the client determines if the layer has already been downloaded and if so displays the layer on the map. If it has not been downloaded previously then a request is sent off asking for the data for the newly selected layer to be downloaded. Multiple layers can be selected at any one time as well as allowing for any depth of nested layers.

Changing focus to Figure 3, we are presented with another full screen snapshot with the tab labelled “Information” selected, the map zoomed in and an object selected (in white). This screen demonstrates the above three utilities. To be presented with this screen the user has clicked one of the zoom tools and zoomed in, then clicked the “Select Object” tool and selected an object on one of the currently displayed layers. The client determines if the aspatial data associated with the selected object has been downloaded already and if, so displays it in the “Information” tab. If it has not been downloaded previously then a request is made for it. In the tab, the information displayed is that of the object and those things directly related to that object. Some of the information displayed can be URLs and email addresses (coloured blue) which are hyperlinked and can be clicked. Clicking these will in turn open a new web browser window to perform the desired tasks. Note that after the user zooms in, the Context Map is updated to show the current zoom area.

The last of the essential tabs is the one labelled “Search” which is clearly used to perform searches. It is split into two parts – Search and Results, which can be seen in Figure 4. When the user clicks on the tab for the first time they are presented with just the top section (Search). Here the user can formulate a range of search queries by filling in all or none of the sections on the form. There are fields for entering the name of the search goal and/or the type of the search goal. Additionally the area that should be searched can also be set or can be left as the default. The third option available when setting the search area allows the user to formulate a query like “Find all the refectories within 200m of the General Purpose South Building”. This is possible by either filling in the blanks in that option or by selecting an object on the map and dragging the mouse to form a circle around that object to specify the search radius.

| View                                | Name                       | Type     | Details |
|-------------------------------------|----------------------------|----------|---------|
| <input checked="" type="checkbox"/> | General Purpose South ...  | Building | Details |
| <input type="checkbox"/>            | Information Technology ... | Unit     | Details |
| <input checked="" type="checkbox"/> | Axon Building              | Building | Details |
| <input type="checkbox"/>            | Xiaofang Zhou              | Person   | Details |

Fig. 4: The Search Interface

Once the user is satisfied with their search query the “Submit” button is clicked to execute it. Between the client, server and database server the results are generated and the lower section of the screen appears (Results). Here a listing of all the possible matches to the query are displayed. Any number of the results can be checked in the “View” column of the table, which in turn highlights the objects on the map. If more detailed information about any of the results is desired, then the button in the “Details” column can be clicked which brings the “Information” tab to the front displaying the relevant information. At any stage the user can select the “New Search” button under the “Search” tab to reset the search form and begin again.

There are three remaining tabs which we have not discussed, and nor will we in great depth as they do not add much functionality to the system. We have included a “Legend” tab that displays the names of the layers of the map and the associated colour of each layer. A password protected “Administrators” tab exists for DBA type operations, namely updating of aspatial data. The last tab is for “Help”.

## 5. RESEARCH ISSUES

This area of research, namely navigating an information space by means of a geographical map, is still very young. In this section, we discuss a number of interesting research issues important to this area.

### 5.1. Semantic Zoom and Data Generalisation

A user can perform zoom operations on the map viewer. In general, when the user zooms in, the area viewable become smaller, but the objects in the viewable area becomes “clearer” (i.e., with more details available). When the user zooms out, the viewable area increases and the level of object details decrease.

One way to perform zoom operations is called *optical zoom*, where the level of details for all objects are increased or decreased proportionally (sometimes including thickness of lines). At this end, the vector data has certain advantages over raster images as explained before.

However, optical zoom is not really the most appropriate way to perform zoom operations for applications where a map is used for navigational purposes. *Semantic zoom* performs intelligent operations, such as selecting objects or selecting object details. For example, some small objects, such as landmarks, may be too small in size to be seen, but are important enough for the user to orientate and navigate.

Semantic zoom is supported by data generalisation [10]. Typically, spatial data is stored in the database in the finest level of detail available. When a vector map for a large area is to be used, potential problems may arise because of the large volume of detailed data. This affects data transferring and rendering costs. These problems can be solved by two methods. First, initially only a few layers are displayed on the map, other layers are downloaded and displayed only when the user requests them. Second, spatial data can be generalised such that only those visually significant objects are used [10,14]. Both optimisations rely on a properly designed spatial database with the support of a modern DBMS. Selecting objects and parts of objects from a spatial database to perform cartographical generalisation and semantic zoom is a topic with many open issues. A web-based spatial search interface requires arbitrary level of generalisation. Combined with a variety of selections on layers and the online nature of interaction, this makes the generalisation process more challenging in the web environment.

### 5.2. Automatic and Semiautomatic Data Linking

We have presented a prototype with a small scale of data. Further work in this area could be performed by extending the prototype for use with larger data sets the size of a city or state or even to incorporate navigation of three-dimensional space around office and building space etc. Apart from potential performance problems as discussed in Section 5.1, a key problem is how to link a very large number of websites with an equally large number of spatial objects.

This problem can be approached in two steps. First, a table which maps street addresses to spatial objects (such as buildings) is needed. Such data are available from most city councils in Australia and other countries. Second, most websites do have address information. Therefore, if such address information can be extracted from websites automatically, it is possible to link websites to spatial objects in an automatic or semi-automatic way, avoiding tedious manual work. Some recent work on learning-based web query processing shows a promising way to extract required information from websites [4].

### 5.3. Full Web Integration and Spatial Web Services

The actual integration of map-based navigators with the Internet opens many research areas. The main focus here would be to allow two-way linkage between the navigator and the Internet. That is, the location for a website can be displayed on a map and new queries can be issued from there. At the same time, the users can surf to other websites from the map viewers and return to the map viewer later, with the map context changed following the user's navigation history.

Recognising that it would be impossible to include the entire Internet into a map-based search system, it is necessary to make the spatial search and display functions open to other sources. That is, a spatial map viewer and its search functions need to be provided as a service on the Internet, readily adoptable by other web developers. That requires a different way of thinking than what we have proposed in this paper, although the fundamental ideas discussed in this paper will still be applicable.

## 6. CONCLUSIONS

In this paper we have proposed a new way to search the Internet, which will compliment conventional searching tools. It will provide users with a high performance, powerful spatial search tool integrated with the Internet, allowing users to perform searches based on geography. We discussed some of the characteristics that such a tool should exhibit and also considered various architectures for the system. A simple yet powerful visual spatial query language has been proposed. This language does not only blend spatial and aspatial queries naturally, its spatial query functions are also very easy to use. Some additional focus was placed on the actual spatial query language to be used in any spatial database tool and then we concluded with a discussion of a prototype that we built to demonstrate the tool's usefulness.

*Acknowledgements* — The research reported in this paper is supported partially by a grant from the Australian Research Council and a grant from the Visiting Scholar Foundation of Key Laboratory in University, China.

## REFERENCES

- [1] M.A. Aufaure-Portier. A high level interface language for GIS. *Journal of Visual Languages and Computing*, **2**(2):167-182 (1995).
- [2] D. Calcinelli and M. Mainguenaud. Cigales, a visual query language for a geographical information system: the user interface. *Journal of Visual Languages and Computing*, **5**(2):113-132 (1994).
- [3] W. Collins. Online spatial systems: the geospatially enabled ACT. *The AURISA '98 Conference*, Perth, Australia, Australasian Urban and Regional Information Systems Association Inc. (AURISA) [www.w3c2.com.au/aurisa/aurisa98](http://www.w3c2.com.au/aurisa/aurisa98) (1998).
- [4] Y. Diao, H. Lu, S. Chen, and Z. Tian. Towards learning based web query processing. *Proceedings of the 26th VLDB Conference*, Cairo, Egypt, pp. 371- 328 (2000).
- [5] M.J. Egenhofer. Interaction with GIS via spatial queries. *Journal of Visual Languages and Computing*, **1**(4): 319-413 1990
- [6] M.J. Egenhofer. Spatial-query-by-sketch. *IEEE Symposium on Visual Languages*, pp. 60-67, California (1996).
- [7] M.J. Egenhofer and A.U. Frank. Towards a spatial query language: user interface considerations. *Proceedings of the 14<sup>th</sup> VLDB Conference*, California, pp. 124-133 (1998).
- [8] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, **30**(2):170-231 (1998).
- [9] R.H. Güting. An introduction to spatial database systems. *VLDB Journal*, **3**(4):357-399 (1994).
- [10] R.B. McMaster and K.S. Shea. *Generalization in Cartography*. Association of American Geographers, Washington, D.C. (1992).
- [11] J. Sharma. *Oracle 8i Spatial: Experiences with Extensible Database, an Oracle Technical White Paper*. [www.jlocationsservices.com/company/Oracle/Twpsdo8i\\_wp.pdf](http://www.jlocationsservices.com/company/Oracle/Twpsdo8i_wp.pdf)(1999).
- [12] H.T. Thomas and M.J. Egenhofer. User interfaces for map algebra. *Journal of the Urban Regional Information Systems Association*, **9**(1):44-54 (1997).
- [13] J.D. Yates and X. Zhou. Searching the web using a map. *Proceedings of the 1st International Conference on Web Info. Sys. Eng.*, Hong Kong, pp. 222-229, IEEE Computer Society Press (2000).
- [14] X. Zhou, A. Krumm-Heller, and V. Gaede. Generalisation of spatial data for web presentation. *Proceedings of the 2<sup>nd</sup> Asia Pacific Web Conference*, Hong Kong, pp.115-121, CSREA Press, Georgia, USA (1999).