A Graphical Query Language for Mobile Information Systems*

Ya-Hui Chang
Department of Computer Science
National Taiwan Ocean University
No. 2 Peining Rd., Keelung, 202, Taiwan
email address: yahui@cs.ntou.edu.tw

Abstract

The advance of the mobile computing environment allows data to be accessed in any place at any time, but currently only simple and ad-hoc queries are supported. People are eager for mobile information systems with more functionality and powerful querying facilities. In this paper, a graphical query language called MoSQL is proposed to be the basis of general mobile information systems. It provides a uniform way for users to access alphanumerical data and to query current or future location information, based on an *icon-based* interface. The interface is particularly suitable for the mobile environment, since it is easily operated by *clicking* or *dragging* the mouse. An example and the underlying theoretical framework will be presented in this paper to demonstrate the functionality of MoSQL.

1 Introduction

Mobile computing has been one of the most fast growing areas recently. Mobile phones along with other applications have become necessities to most people. They not only provide the basic function for oral communication, but also allow data to be accessed in any place at any time. For example, mobile users could retrieve information such as the locations of nearest restaurants via WAP. In such kind of applications, selected data items are pre-classified and listed in a navigational hierarchy. Users might encounter difficulty to retrieve the desired information, and only plain text data are presented. GIS systems equipped with GPS and graphical interfaces could show the instant location information on the map. However, still only simple and ad-hoc queries are supported.

An advanced mobile information system should provide more functionality and be able to process general and complicated queries. Consider the police officers, who change locations frequently to patrol their magistracy. When catching a suspect, a police officer might issue *location-unrelated queries* to retrieve the physical characteristics, such as the height and weight, of the suspect. He may also need help to catch the suspect and pose the query: *'Tell me where the policeman nearest to me is now.'* This kind of queries, called *location-related queries*, either directly retrieve location information, or impose constraints on location data. He might want to further estimate the future location of a colleague in ten minutes. Therefore, a general mobile information system should possess the following characteristics:

- allowing past, current, and future location information to be queried
- providing an integrated approach for users to query both location and alphanumerical types of data
- presenting a user-friendly interface

This paper discusses the issues for constructing advanced mobile information systems which could meet the above requirement. The main contribution of this paper, is to propose a graphical query language called MoSQL, as a basis for general mobile information systems. This language has an *icon-based* interface. That is, the same class of data represented in the database system corresponds to the same icon. All the ordinary, spatial, and temporal properties are associated with that icon, and can be similarly invoked via it. Therefore, MoSQL provides a uniform way for users to access alphanumerical data and to query current or future location information. The interface is also particularly suitable in a mobile environment, since it is easily operated mainly by *clicking* or *dragging* the mouse, requiring little keyboard strokes. The theoretical framework underlying the interface will be discussed to demonstrate the complete functionality of MoSQL.

*This work was partially supported by the Republic of China National Science Council under Contract No. NSC 88-2213-E-019-005.

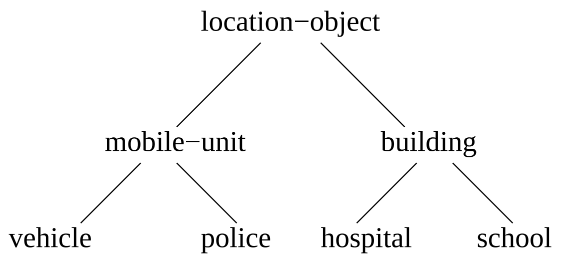


Figure 1: Classification of location objects

The remaining of this paper is organized as follows: Section 2 defines the object model for data in the mobile information system. The MoSQL query language is then presented, with the graphical user interface in Section 3, and the theoretical framework in Section 4. Finally, related research results are compared in Section 5 and conclusions are given in Section 6.

2 The Object Model

In a mobile information system, there will be location-unrelated data and location-related data. We adopt an object model to represent these information. Particularly, objects with location data are named *location objects.* They can be further classified into *mobile units* and *buildings.* Mobile units, or called *moving objects*, refer to those objects which change location constantly, such as vehicles or the police. Their location data should be presented with time stamps to identify when the particular location data is valid. On the other hand, the locations of buildings, or called *stationary objects*, are presumed permanent, so temporal data are not required. This classification is depicted as a hierarchy in Figure 1.

When we build an application in the mobile computing environment, all the relevant information might be stored in different places. Take the police information system for examples. The location data are managed by the mobile service provider. The duties performed by each policeman will be managed by the police headquarter. Therefore, each real-world object usually possesses two identities. We will refer a policeman as a location object (or specifically a moving object) in the mobile environment, but as an *application object* in the application context.

Several possible constraints exist for the objects under such environment. The *heterogeneous constraint* refers to the semantic conflicts or other kinds of heterogeneity for the data presented in multidatabases with different formats. There are also the *spatial constraint*, *temporal constraint* and *normal constraint.* The first two constraints restrict the positions of *location objects* at a certain time. The last one refers to restriction on attributes of the *application object*, as supported in the traditional database system.

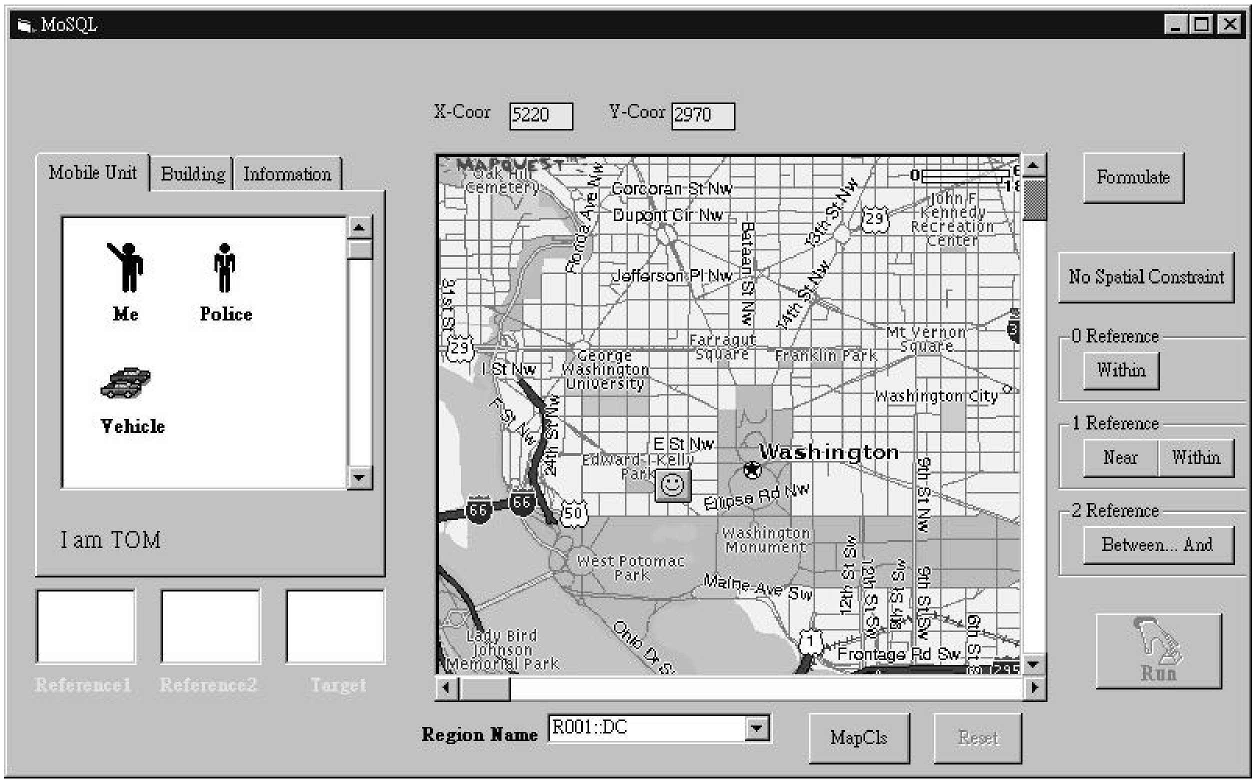
There is a distinction among location objects in a particular query. If their location data retrieved from databases are used to form spatial constraints, these objects will be called *reference objects.* For the objects whose location data are to be displayed to the users, they will be called the *target objects.* More generally, *target objects* refer to those objects whose properties, either location-related, or location-unrelated, are to be retrieved.

3 The User Interface

The graphical user interface as seen in Figure 2 provides a uniform way for users to pose *location-unrelated queries* and *location-related queries.* To cope with the portable characteristic of mobile environments, the interface could be fully operated by the mouse, without needing the keyboard. Three basic actions for using the mouse include clicking using the left button (*left click*), clicking using the right button (*right click*), and dragging by continuously pressing the left button of the mouse (*dragging*). Their functions might differ under different contexts, as will be explained later.

The interface is mainly divided into three portions. Data stored in the system are shown on the left side. The interface is *icon-based,* where the objects in the same class correspond to the same icon. Their properties are directly associated with that icon and could be retrieved by clicking it. The icons are classified into three groups. Groups *mobile unit* and *building* correspond to the second level of Figure 1. The last group *information* is used to collect all location-unrelated data. If we *left click* on the tab labeled *Mobile Unit*, those data belong to this group, *e.g.*, the police and the vehicle, will be shown on the space below, which correspond to the leaves of the hierarchy in Figure 1. A special icon Me corresponds to the user who is posing the query.

The lower left of the interface has three small *input windows*, called *Reference1*, *Reference2* and *Target.* Currently we support at most two *reference objects.* They are used along with the spatial operators to form spatial constraints. The data belong to the *mobile unit* group or the *building* group, could be input to these windows, by dragging the appropriate icon into the windows. If we *left click* the input icon, the *constraint window* will pop up for us to specify normal constraints and temporal constraints. For target objects, we could also *right click* the icon to invoke



X-Coor 5220 Y-Coor 2970

I am TOM

Region Name R001::DC

Figure 2: The graphical user interface

another window for specifying the output attributes.

The middle of the interface is a geographic map, which could be used to pose a spatial constraint, *e.g.*, by drawing a circle, and is also used to show the positions of location objects. The spatial operators are shown on the right of the interface, which are classified based on the number of reference objects needed, as will be explained in Section 4. When the button entitled "No Spatial Constraint" is selected, among the three *input windows*, only the window entitled "Target" is enabled. It is designed for users to pose *location-unrelated* queries.

A prototype is built to simulate a mobile police information system. It is a client-server system based on a fixed TCP/IP network. The data are represented in a relational database system. The graphical user interface was first built by Microsoft Visual Basic, and then ported to a web system. Therefore, the users could operate the system through browsers.

We illustrate the usage of the interface by an example. Suppose the user poses the following query: *Please find all the police who are currently located between me and the building named "State Hospital"*. The detailed steps using this graphical user interface are listed as follows:

1. Identifying the spatial operator:

   Left click the button labeled *Between ... And*, and the three *input windows* will be enabled.

2. Identifying the reference objects:

   - Left click the tab labeled *Mobile Unit*. Drag the Me icon into the *Reference1* window.
   - Left click the tab labeled *Building*. Drag the Hospital icon into the *Reference2* window.
   - Left click the window *Reference2* and the *constraint window* will pop up. Restrict the attribute "Name" to be "State Hospital".

3. Identifying the target object and specifying the temporal constraint:

   - Left click the tab labeled *Mobile Unit*. Drag the Police icon into the *Target* window.
   - Left click the *Target* window to invoke the *constraint window*. Specify the temporal constrain to be *Now*.
   - Right click the window *Target*. On the popped up window, select all the attributes for output.

4. Executing the query:

   Left click the Run icon. The qualified police will be shown on the map as red dots based on their current locations.

5. Retrieving location-unrelated data:

   Left click a particular red dot, and we could see the output attribute values.

6. Retrieving location-related information:

   Right click a particular red dot, and we could see its estimated speed. We could further give a period to estimate its future location.

4 The Theoretical Framework

The query posed on the interface in Figure 2 could be transformed into an SQL-like query. We name it **MoSQL** since it is an extension of the standard SQL language, and is designed to cope with the special requirement for a mobile information system. The following subsections present this query language in detail.

4.1 Domain Constructors

Data in MoSQL could be defined with the alphanumeric data types, as in the traditional relational model. MoSQL also supports two spatial domains, *point* and *region*, and one temporal domain, which is *timestamp*. The domain *timestamp* is used to represent the specific time instants when something occurs, *i.e.*, tuples are true at those particular time instants. The *point* domain is used to precisely represent an object's location by its coordinates. An element in the *region* domain is a geographical area. Currently we only consider two basic forms, *i.e.*, *circle* and *window (rectangle)*. Constant constructor functions used to create instances of the spatial domains via the map, are described as follows:

- the point constructor:

  The *point* constructor takes a pair of values (x, y) and creates a point with the X-axis coordinate at x and Y-axis coordinate at y. It corresponds to the action *left clicking* on the map.

- region constructors:

  Two region constructors are supported. The *circle* constructor needs two points as arguments. The first point is the center of the circle. The user needs to *left click* on the map to determine the second point. The distance between the two points will be the radius of the circle.

  The *window* constructor corresponds to the action *dragging* on the map. Suppose a line is stretched from the point (x_1, y_1) to the point (x_2, y_2). It will form the diagonal of a rectangle, and the four endpoints of the rectangle will be (x_1, y_1), (x_2, y_2), (x_1, y_2) and (x_2, y_1).

4.2 Operators

The operators discussed in this subsection are used to formulate spatial constraints. They are classified based on the numbers of reference objects which they need. The location data of the reference objects are retrieved from the database system.

- without reference objects:

  The operator *within* could form a predicate with the form *target within window*, where *target* represents the target object, and *window* represents a region directly specified by the user using the *window* constructor. This predicate will restrict the target objects to be located within the particular window.

- with one reference object:

  The *near* operator forms the predicate *target near reference*. All the objects near to the location of the reference object will be retrieved. The meaning of *nearness* is system-defined.

  The *within* operator is overloaded and could form another predicate like *target within circle*. The circle is created using the *circle* constructor, and is centered at the location of the reference object. If the target object is located within the particular circle, this predicate will return *true*. This operator is more flexible than the *near* operator in the sense that we allow users to determine the size of the region.

- with two reference objects:

  The operator *between* $\cdots$ *and* needs two reference objects, and the predicate is like *target between reference1 and reference2*. Suppose *reference1* is located at (x_1, y_1) and *reference2* is located at (x_2, y_2). This predicate will return *true*, if the location of the target object, (x_3, y_3), satisfies the following inequation:

$$\frac{y_2x_3 + x_1y_3 + y_1x_2 - y_1x_3 - x_2y_3 - x_1y_2}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} < d$$

  To explain, we first use the two points (x_1, y_1) and (x_2, y_2) to form a straight line. Then we calculate the distance between (x_3, y_3) and this line, which forms the left side of the inequation. If the value is less than d, which is a system-defined value, we will determine that the point (x_3, y_3) is located between (x_1, y_1) and (x_2, y_2).

4.3 Functions

Several functions are defined in the MoSQL query language. The values of the functions could be directly displayed to the users, or could be used to form a predicate. They are classified as follows:

- spatial functions:

  The *s-distance* function calculates the spatial distance between two points.

- temporal functions:

  The *t-distance* function returns a temporal distance between two instants from the *timestamp* domain.

- mobile functions:

  The function *speed* returns the speed of a moving object. There are two ways to invoke this function. In the first method, the function is specified as *speed (s-distance, t-distance)*, where a value representing a spatial distance is the first argument, and a value representing a temporal distance is the second argument. This function will use these two values to calculate the speed. In the second method, we could invoke this function with a mobile unit as the argument. The system will estimate the speed of this mobile unit based on its most recent location data represented in the system.

  Given a mobile unit and a temporal distance, the function *location* estimates the future location of the mobile unit.

- heterogeneous functions:

  The *equal* function takes two arguments, where the first argument represents a mobile unit, and the second argument represents an application object. This function will return the value *true*, if the two arguments correspond to the same object in the real world; otherwise, it will return the value *false*.

4.4 Syntax

As described in Section 2, a real-world object, like a policeman, will be referred to as a location object in the mobile environment, but as an application object when discussing his duties. Therefore, the SELECT clause of the MoSQL DML is designed to display the properties of the two identities, including attribute values or the values of functions as previously defined. The syntax of the MoSQL DML is listed as follows:

```
< mosql > ::=
SELECT < property-of-application-object > |
          < property-of-location-object >
FROM < relation-list >
[ VALID < temporal-constraint > ]
WHERE EQUAL ( < location-object >,
                  < application-object > )
          { AND < normal-constraint > }
          [ AND < spatial-constraint > ]
```

The WHERE clause of the MoSQL DML represents the conditions which need to be satisfied by the tuples in the resulting relation. As discussed in Section 2, there could be four types of constraints, and three of them are specified in the WHERE clause. The *equal* function relates a location object to an application object, which belongs to *heterogeneous constraints*. An MoSQL query could have many *normal constraints* on the identity of the application object, but have at most one *spatial constraint* on the identity of the location object.

The other constraint, *i.e.*, the temporal constraint, is specified in the VALID clause. It will be applied to those relations which define temporal attributes. There are three ways to specify a VALID clause. The first one is to impose constraints on instants. For example, the clause VALID TIMESTAMP '10:00am' will only retrieve the tuple whose timestamp is exactly '10:00am'. The second one is to impose constraints on intervals by using the INTERVAL keyword, such as INTERVAL '5' MINUTES. The third one is to impose constraints on periods, such as VALID PERIOD '[10:00am - 10:05am]'.

5 Related Work

Some constructs of the MoSQL query language are stemmed from the work in spatial databases [4], particularly PSQL [6]. The temporal constructs are adopted from the TSQL2 language [9]. They are modified in order to be more suitable in a mobile environment.

As to the modeling of the mobile unit, the temporal component is directly associated with the spatial component in our system. Such information could be obtained if the mobile service provider updates the location data of mobile units with a timestamp regularly. To reduce the cost of updating, the authors in [8] represent the position of moving objects as a function of time. The Future Temporal Logic (FTL) is proposed to express future queries and continuous queries based on that model. In [10], the authors propose an information cost model to determine when the location of a moving object in the database should be updated. They also describe a three-layer architecture as a platform for developing motion database type of applications [11].

The issue of designing graphical user interfaces has also attracted the attentions of researchers from different areas. Query extensions and corresponding visual representations have been proposed for spatial applications [1]. The author in [3] specifically advocates that a graphical user interface for spatial querying may have three sub-windows: (1) a text window for textual representation of a collection of objects; (2) a graphics window for graphical representation of a collection of objects; (3) a text window for entering queries and display of system messages. On the other hand, for the mobile environment, the authors in [2] advocate a pen-based graphical database language. The interface is easy to use, avoids keyboard use, and is suited to small screens and small memory size of mobile machines. Some researchers propose an iconic query language for the mobile environment, which involves no path specification in composing a query [5]. These characteristics could be found in our graphical user interface.

6 Conclusion

We have presented a graphical query language called MoSQL in this paper. This language provides a uniform way for users to access all kinds of information possibly presented in a mobile environment, including alphanumerical and location data. The graphical user interface is also designed to cope with the characteristic of the mobile environment, so that it could be easily operated by the mouse only. This language has been successfully used as a basis to build a mobile police information prototype.

The current work could be extended in several directions. First, we hope to extend the modeling and expressive power of the MoSQL query language to allow more complex queries being posed, especially in the mobility aspect. We also plan to improve the query processing methods by incorporating special data structures. In [7], the authors propose an R*-tree based indexing technique to support the efficient querying of the current and projected future positions of moving objects. These research results are worth further investigating.

Acknowledgment:

The author is thankful to Dr. Wang-Chien Lee for the constructive suggestions to improve the paper.

References

[1] A. Aiken, J. Chen, M. Stonebraker, and A. Woodruff. Tioga-2: A direct manipulation database visualization environment. In *Proceedings of the 12th International Conference on Data Engineering*, pages 208–217, 1996.

[2] R. Alonso, E. Haber, and H. Korth. A database interface for mobile computers. In *IEEE Globecom '92 Workshop: Networking of Personal Communications Applications*, 1992.

[3] M. Egenhofer. Spatial sql: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6:86–95, 1994.

[4] R.H. Guting. An introduction to spatial database systems. *VLDB Journal*, 3(4):357–399, 1994.

[5] A. Massari, S. Weissman, and P. K. Chrysanthis. Supporting mobile database access through query by icons. *Distributed and Parallel Databases Journal (Special Issue on Databases and Mobility)*, 4(3):249–270, 1996.

[6] N. Roussopoulos, C. Faloutsos, and T. Sellis. An efficient pictorial database system for psql. *IEEE Transactions on Software Engineering*, 14(5):639–650, 1988.

[7] S Saltenis, CS Jensen, ST Leutenegger, and MA Lopez. Indexing the positions of continuously moving objects. *SIGMOD RECORD*, 29(2):331–342, 2000.

[8] A. Prasad Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proceedings of the IEEE International Conference on Data Engineering*, 1997.

[9] Richard T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.

[10] O. Wlfson, A. Prasad sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7:257–287, 1999.

[11] O. Wolfson, B. Xu, and S. Chamberlain. Location prediction and queries for tracking moving objects. In *Proceedings of the IEEE International Conference on Data Engineering*, 2000.