

CMSC 828D  
Amol Deshpande

*class project*

***SourceSoup***

Connecting Information from many Different Sources

Georg Apitz

December, 13th 2005

## 1 Introduction

The amount of data available increases steadily and is provided from very diverse sources. Traditional databases store information about sales of a supermarket, its inventory, salaries and so on. With sensor networks other kinds of data are provided in huge quantities as well, temperature readings in and outside of buildings, smoke detectors return current readings on the smoke situation, GPS systems and RFID tags that track inventory, equipment or people can constantly deliver information about the location of the tracked items, satellites provide information about weather, and traffic is monitored by cameras and other sensors. This list goes on and on, what we see is a vast amount of data of all sorts that is available, but no architecture that ties different information sources together. Of course there are specific solutions for supermarkets or emergency dispatches.

Looking at the problem of these data sources one approach is to let the user specify which data sources they want to combine and provide the means to do so. In this project we describe an architecture called *SourceSoup* that allows users to specify the information sources they want to access and provides a unified user-centric way of accessing all the data.

We describe the architecture using the example of a fleet control system that provides means to deliver an efficient on-time fleet management and allows for computation in the query context instead of just simple queries.

## 2 Contribution

The main contributions of this paper are: describe an architecture to access different data-sources like databases and data-streams as well as to use the information gathered through queries as source in the system. Based on this provide a way to pose a single query to access these sources and return a single answer. Extending the idea of tangible interfaces to moving objects databases. This is described on the example of traffic related data.

## 3 Motivation

Even though several systems are available that provide the possibility to compute routes from a start to a desired destination like [18, 1] or commercial available tools that are known as navigation systems for cars etc., they are usually specialized on just route finding and do not take other information like weather and traffic information into account. But this information is available through radio, internet and telex. Furthermore, companies have access to huge databases with fleet information of vehicles they own and operate. These information can be vehicle weight, measurements, speed- and cargo-limitations etc. All this information is readily available in its parts, but oftentimes the real use and benefits would be widely improved if one was able to pose a single query that takes information from all the available sources and returns a user-friendly visualization of the result.

This is not just limited to the described traffic scenario but can be seen as a general problem of providing access to many readily available data sources and allow a unified view and the

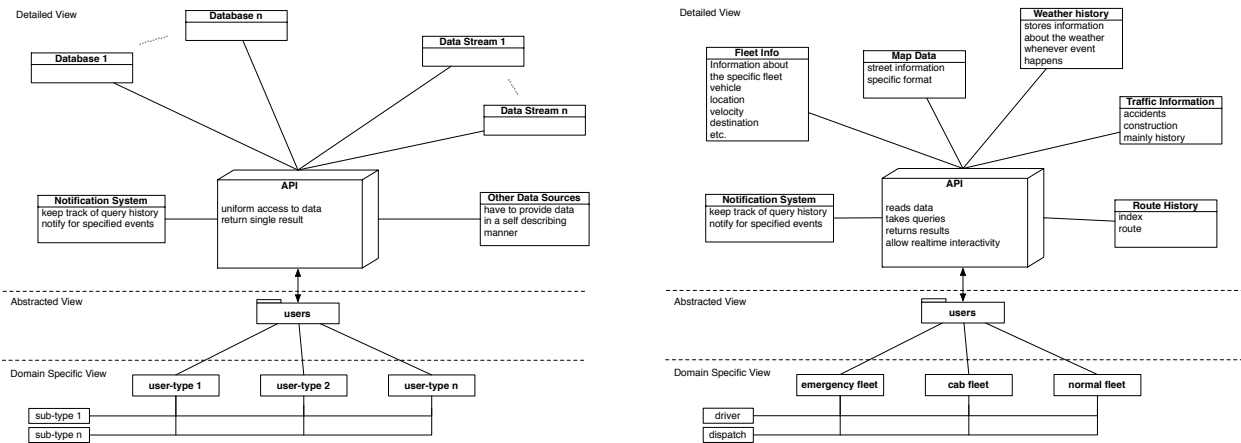
possibility to query them with a single query.

## 4 Previous Work

This work builds upon several other research work. Sistla et al. [2] propose a data model for representing moving objects in database systems (MOST) and describe a query language (FTL) that can be used to query their data model as well as an algorithm that can be applied to process FTL queries. Another early work in moving objects databases is the work by Wolfson [12] who describes the basic research problems for moving objects databases and then describes the problem of estimating the trip cost for a moving object. In a later paper Wolfson et al. [11] describes several more aspects of moving objects databases including the handling of uncertainty about the acquired data. Tastets [15] gives a good overview over basic aspects of moving objects databases and describes the difference between traditional database systems that represent objects and assume that their value is constant unless a specified manipulation takes place. In moving objects databases not the object changes but its location, in other words just one attribute of the object, namely its location, changes. Since traditional systems are not built to handle changes in attributes, moving objects databases have to account for that. Several aspects are important here and have been researched in the past, such as conceptual modeling [17], data models and languages [14, 9] and various access methods [4, 13]. The Control-Project at Berkeley [8] introduces a way that lets users interactively work with a query and see query results while they are returned and not just after the query processing is completed. This has the big advantage that a user can specify a query and look at the results as they come in and if he or she realizes that the results are not the intended ones the query can be adapted or in another case that the desired results were returned first the user can stop the query. Su et al. [7] investigate logical properties of moving objects in connection with queries over such objects using tools from differential geometry and propose an abstract model as an example. In a vision paper Wolfson [10] draws a picture of the current status of moving objects databases and what challenges still have to be overcome to achieve good results in this area. The Logic of Motion paper [5] describes fundamental logical elements behind databases for moving objects and lays out the basics for querying moving objects.

## 5 General Architecture

In this section we describe the general architecture that is necessary to provide uniform access to different data sources and then describe details of the parts and how they interact using the before mentioned fleet management scenario. The architecture shown in figure 1 shows the general three layered approach. The *Detailed View* contains the different data sources, these can be databases, data-streams or other data sources that have a self-describing format like XML. Furthermore, a notification system is contained here which keeps track of the queries that are posed to the system and can trigger notifications if specific events happen. For example the fact that ambulance dispatchers are asking for ambulances in a specific region contains the information that ambulances are needed in that area. Another query: *Where were ambulances*



**Figure 1:** The left side shows the general architecture of *SourceSoup*. On the right the architecture applied to a fleet management scenario is shown.

*heading to in the last 10 minutes?* could build on these previous queries and reintegrate this information back into the system.

The core part of the *Detailed View* is the API which allows the user to specify a single query and get a single answer from all the underlying sources in combination with the use of the notification system.

It is important to note that the "data sources" do not all have to exist, some of them might be created while the system is used to simplify future queries. Details about that are described in the following sections.

The *Abstract view* contains a generic user that wants to use the system. From this the *Domain Specific View* is derived where special user-types with sub-types can be specified.

### 5.1 Fleet Information

For the fleet management system the actual fleet information is available as database, this part is domain specific, but in general supports 4 basic types of properties. These are vehicle, location, velocity and destination. Other domain specific properties like *vehicle type*, *number of passengers possible* or *cargo volume* can be added for certain user types. When the system is queried the information from the fleet part is used as needed.

Then fleet information from the University of Maryland Department of Transportation could be used to create a fleet data base and to get current locations of the vehicles, the recently introduced GPS tracking system for all UMD shuttles could be utilized for that.

Here we see, when a different user wants to use the system he can just provide a different database with other fleet information and specify the additional information available in it and then can start to use the system.

## 5.2 Street Information

The street information part contains information about available routes in general and detailed information about streets (lanes, type of street), height and width restrictions on streets, weight restrictions on bridges and traffic directionality on specific streets (one way, etc.). This information is used as the "map" of the system to provide the routes for the different fleets and alternative routes. For a system like the MITSIMLab, which can be used to compute routes, a specific format is necessary which is very cumbersome to create. For example, for each road, each lane has to be explicitly specified with information if cars can change into other lanes or make turns and much more details.

## 5.3 Weather Information

In the weather information part a twofold approach is used, the current weather conditions are pulled from a continuously available stream that provides up-to-date weather information<sup>1</sup>. A tight coupling with the traffic information system allows to receive any events from that part and only in the case of events to store weather information. This is done to have an efficient system, continuous weather information is not needed if nothing unusual occurred on the traffic side.

## 5.4 Traffic Information

The traffic information system again is an approach that queries an on-time service that provides traffic information like accidents, traffic jams, high traffic volume and construction and provides it to the system. Our system monitors that and makes entries in the data set for the traffic information to have a future reference to look at events in the past.

This kind of information is already available for example traffic information for Baltimore are available online, <http://www.traffic.com/Baltimore-Traffic/Baltimore-Traffic-Reports.html> and are updated very frequently.

## 5.5 Route History

This data sources is one that gets created by the system while it is in use.

Keeping a clever history of the system can make all the difference for efficient retrieval of information. If we have long and static routes we can just reuse information from the past to output an new route instead of re-computing the route. On the other hand for local highly erratic routes like the ones for emergency vehicles storing the routes does not seem to be promising instead storing of patterns seems to be more appropriate which could be seen as sub-routes. If we look at possible routes for these type of vehicles it is unlikely that exact same routes will be travelled over and over again, instead parts of these routes will be travelled again and again. Thus, having information about these sub-routes can assist the system in more efficiently building complete routes while still incorporating all the data from the different

---

<sup>1</sup>Current weather information with a high update rate are available from <http://www.weather.gov/>.

sources. Also, information about traffic accidents and following traffic patterns ( congestion on road X, no traffic at road Y etc.) should be stored in the history for later access to assist the system in making decisions about which route to suggest.

To actually test the influence of the history in the performance of the system the next step here would be to run a performance study in two parts, where in one part no history is used and in the other history information is used. This way the influence of the history information in the time and space requirements can be computed. It is likely that the history over traffic data and especially about patterns that occurred after accidents or road construction can improve the performance. On the other hand history information about static routes vehicles take over and over again might not be so helpful in improving the performance.

## 5.6 Notification System

In the notification system the user can specify events that should trigger a notification. Possible scenarios here would be:  $E_1$ =*The number of vehicles passing point X in the last 10 mins is greater X.* or  $E_2$ =*More than X ambulances were dispatched to area A in the last Y minutes.* Furthermore the notification system keeps track of the queries issued to the system for possible reuse and also to reintegrate that information back into the system as described with the ambulance scenario before.

Once an event is trigger there can be several ways the notification happens. One would be to just output it to the user on the display, but other notifications such as RSS feeds or emails could be used. Imagine the case of  $E_1$ , this information could be interesting for other queries that involve the specified point and thus if fed back into the system, as RSS feed for example, these new queries could take this information directly into account.

## 5.7 API

Basically all the connecting work is done in this part, if we look at the underlying single part most of them just represent traditional relational data sets or some information source that provides information like weather and traffic information. The crucial part of our system is combining these different data sets and give the user a unified view on the data so that he is able to access all the information at once instead of querying the different data sets separately. This can have several advantages. For example, if a user would query the street information to get a route for a vehicle and after getting the route would query the traffic information system about the traffic on that route, he would already have the complete route and then get the traffic information. Thus any problems that are reported by the traffic information system would lead to a complete re-computation of the route or at least an afterwards adaption. With our system and its tight coupling problems like this can be avoided since all parts are continuously involved and therefore different aspects can be worked into the route finding that are not just based on the map information but influenced by traffic and weather information as well as information about routes taken in the past.

An interesting feature that can be added here is to "pre-process" the results based on user specified criteria. For example for the route computation the idea of weighting can be

introduced. Depending on the request for a route and who issued the request different weights can be applied. In the simplest case we can say that we always want the route with the most certainty for the emergency vehicles, while it is possible to make some compromises with respect to certainty for cab and normal fleet routes for the sake of a possibly faster arrival at the destination. This could also be seen as a filter that is applied to the results that get returned from the underlying sources before it gets pushed up to the user.

The API is intended to be realized as a Java application that wraps the user query and issues the necessary queries to the different sources and return the different results as one combined result.

### 5.7.1 Accessing the Data

As mentioned before most the information about the fleets, the map data and the history is stored in traditional relational data bases and the traffic and weather information is assumed to be provided as a data stream. The fleet, map and history information can be accessed using SQL, for the weather and traffic streams using CQL or an adaption of it is intended and the current location/velocity of vehicles that are registered in the fleet database is provided as a data-stream.

## 5.8 Specific Users

In the fleet management we have three different general user types. First, fleet management for consistent routes, e.g. long distance truck deliveries. Second, we consider the case where routes are somewhat erratic, such as with cabs or short distance deliveries. And, third we consider the case of emergency vehicles like ambulances or firefighters.

In all cases we have two kinds of users, on the one hand there is the dispatcher who has a view over all the available vehicles in the fleet and on the other hand there are drivers that have a view of there routes and possible alternatives as well as the possibility to enter information from where they are. This could be a congestion that was not reported by the traffic system yet or changes in cargo or defects.

To accommodate these types of users we propose a different front end for each of them so that they have access to the information they need.

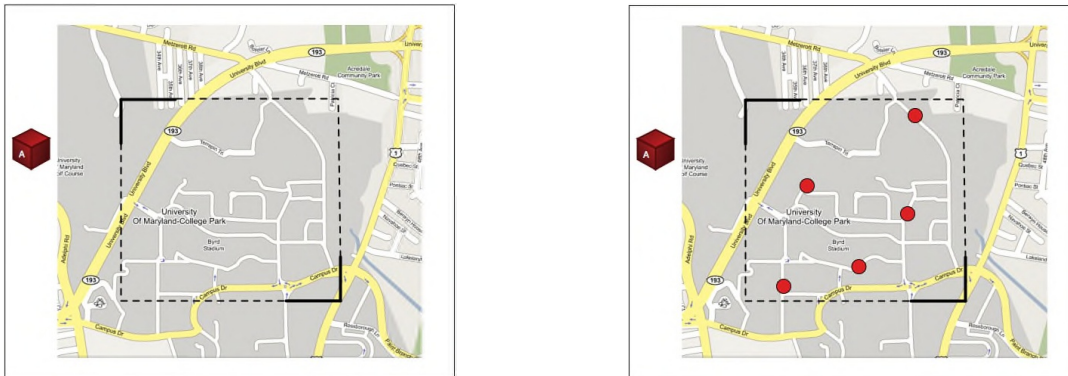
## 6 Graphical and Tangible User Interface for the Fleet Management

In the *Abstract View* a general Graphical and Tangible User Interface (GTUI) is specified, which can then be refined for the different user types. The description her focuses on the *Abstract View* and mainly on the dispatcher for the fleet information.

The Graphical and Tangible User Interface at the dispatch (GTUI) is used to enter and display queries as well as the results. The user can specify queries at the GTUI and edit them by specifying the objects involved in the query and the area or time over which the query is issued. The objects are represented as "real" objects that the user can place on an interactive display, this is similar to [6] and [3]. The user can specify if the object he is interacting with

represents just one vehicle or a group of vehicles by selecting the database entry of the vehicle or the vehicle group and linking them to the object he is operating on with. The area of operation is specified using a finger to draw crop marks on the display around the region of interest or specifying that the query is over the whole search space.

The results are presented starting as soon as they are partially available by displaying estimated locations or partial/complete suggested routes on the interactive display. As soon as the user sees results he has the option to activate them and interactively change them. For example if a new route is suggested, the users can activate this route by tapping on it with a finger and with a double tap "attaching" the point where his finger touches the route to his finger and then move it to a different position, a second double tap deactivates the route. This changes are applied immediately to the result.



**Figure 2:** Sketch of the GTUI, on the left the user specifies the cube A and associates it with all Shuttle buses and specifies a region for which to display the query result. On the right the query result shows the Shuttles for that region.

For example a query: *Show me all Shuttle buses of the University of Maryland that are in region X Y in 10 minutes.* could be issued to the system as follows: The users uses the tangible cube A to specify the objects he is interested in, for that to happen he creates a link between the database entries of the Shuttle buses and the cube by selecting all Shuttle buses in the data base on the graphical user interface<sup>2</sup>. Then he specifies the condition, in this case: "property" location with the "constraint" in 10 minutes using again the graphical user interface, the "property" is selected from a list of possible properties of the object as well as the "constraint", which is then described in detail using a slider that represents the time. After that the area of interest is specified at the display. The display is a large size display create with rear projection on a table top. The user first can specify a general area of interest using a list with entries like company, county, organizations, states and then zoom into the area of interest on the map. Once the area of interest is visible the user marks it using crop marks shown on the left of Figure 2, the dotted lines just complete the rectangle for better feedback. After the user has specified the rectangle the shuttle within are shown, by changing the size

<sup>2</sup>The entries are displayed as a list and checkboxes at the beginning allow for the selection of single objects and there is an option to select all objects.

and position of the rectangle the user can get different results. Or the user could just move the rectangle like in a panning motion and the content would automatically update.

Since the interaction at the drivers front end is very limited a small screen display would be sufficient, that shows the current rout and provides a way of interaction with the system to enter basic information as mentioned in the previous chapter.

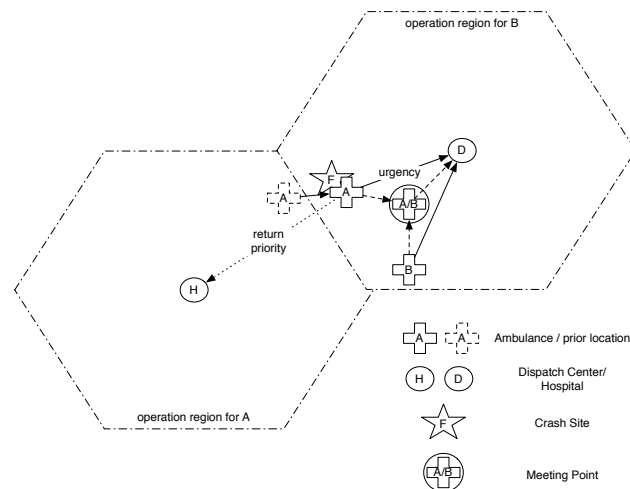
The GUI is intended to be realized in Java as well, building on the Piccolo<sup>3</sup> toolkit that allows to create zoom-able interfaces. The display on the dispatch side can be build using a rear projecting technique and something like Smartskin<sup>4</sup> to allow for the interactivity.

## 7 More than answering Queries

Usually, all that is expected from an interaction with a database system, even if several databases are involved, is to pose a query and get a result based on the data. With *SourceSoup* the API-part can actually do complex computation which are completely hidden from the user and then return a result to the query. In the following we are going to describe a special scenario that illustrates this feature.

### 7.1 Patient Transfer Scenario

In a fast moving and constraint environment emergency vehicles, such as ambulances, may need to transfer patients and courier services, such as FedEx trucks, may need to transfer cargo. The problem of finding a meeting point between two moving objects with the same destination as in the before mentioned cases provides an interesting database challenge.



**Figure 3:** A case scenario for the meeting point computation.

<sup>3</sup><http://www.cs.umd.edu/hcil/piccolo/>

<sup>4</sup><http://www.csl.sony.co.jp/person/rekimoto/smartskin/>

Imagine a scenario where ambulances operate in a grid like area and have on dispatch center. (see Figure 3). In a case where ambulance A picks up a patient from a crash site (F) and is heading for hospital (D) which is also the dispatch center of ambulance B. If we assume that ambulance B is heading for its dispatch center at the same time as A is heading there, they both have the same destination, albeit with different priorities. In the interest of efficiency in the scheduling of the ambulances it is important that they do not leave their designated operation areas for too long a time. So naturally B wants to get to D as fast as possible since that is its dispatch center and A wants to get back to H, its dispatch center.

For B the priorities are clear defined such as its main goal is to get to D as fast as possible. For B the priorities are twofold, they are described as weights in the graph that connects back to B's dispatch center (dotted line in the figure) as *return priority* and *urgency* on the graph to D. Assume that the ambulances have to transfer the patient A picked up at F from one ambulance into the other then the two cases are as follows:

1. The first case is that the patient has to be at the hospital (D) as fast as possible and thus the *urgency* is weighted higher than *return priority*. In this case the goal is to find a meeting point the minimizes the time for the patient to reach D, accommodates a transfer between A and B and also keeps in mind not to get A too far a way from H.
2. In the other case the *return priority* is higher and the patient has a lower *urgency*, thus the transfer should happen as soon as possible to minimize the distance between A and its dispatch center.

### 7.1.1 Possible Queries in the Patient Transfer Scenario

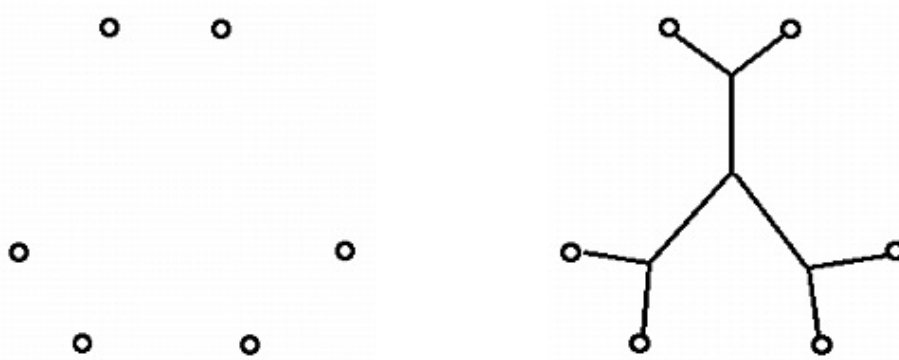
What is the best meeting point if the *urgency* =  $x$  and *return priority* =  $y$ ?

How does the meeting point change if we reduce *urgency* by  $\delta$ ?

How far away from the border of the operation area does vehicle V get?

The challenge is to find the meeting point to satisfy the different weights and at the same time accommodate information about traffic, weather and fastest routes. For the equally weighted case (*return priority* = *urgency*) the problem to find the meeting point is known as a Steiner Tree [16], see Figure 4. Steiner tree often arises in network design and wiring layout problems. Suppose we are given a set of sites that must be connected by wires as cheaply as possible. The minimum Steiner tree describes the way to connect them using the smallest amount of wire. Analogous problems arise in designing networks of water pipes or heating ducts in buildings. Similar considerations also arise in VLSI circuit layout, where we seek to connect a set of sites to (say) ground under constraints such as material cost, signal propagation time, or reducing capacitance.

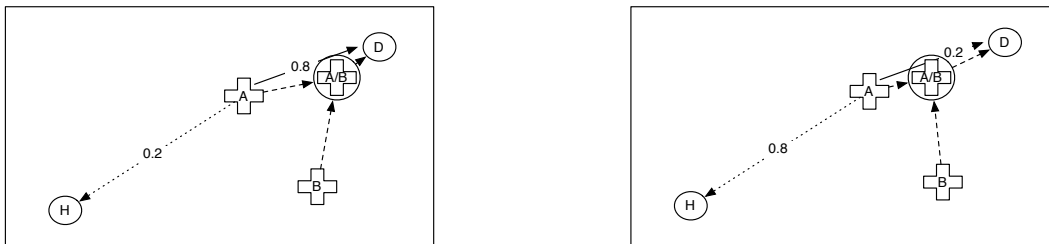
The locations of the ambulances, the hospital and the computed meeting point can be seen as a Steiner tree.



**Figure 4:** An example of a Steiner Tree, left we see the vertices and right we see the smallest tree connecting all vertices.

### 7.1.2 An adapted Steiner Tree approach

The Steiner Tree that was described in the previous work section assumes straight graphs with no weights attached. In order to compute a meeting point for the transfer scenario the Steiner Tree algorithm has to be adapted. This can be done in two steps. In the first step the weight that depends on *urgency* and *return priority* is considered when computing the Steiner tree. Depending on which edge has more weight ( *urgency* or *return priority*) the graph will "lean" towards that edge (see Figure 5).



**Figure 5:** An example of two weighted Steiner Trees, left we see the *return priority* with a higher weight and right we see the *urgency* with a higher weight, the graphs skew accordingly. The actual Steiner tree are the two dotted lines to A/B and the line from there to D. The part from A to A/B (*return priority*) and A/B to D (*urgency*) are the ones with the weights.

In the next step the the actual computation of the routes is done. Since the Steiner and even the weighted Steiner tree only give straight lines which is an over-simplification the actual routes need to be computed as nearest path to the computed weighted Steiner tree. Also, at this point information about traffic, weather and frequently used routes have to be taken into consideration.

## 8 Future Work

Working with the MITSIMLab application as well as from looking at other route computing applications it became obvious that the specification of the underlying street system is very cumbersome. A future project could address this problem by creating an abstract syntax for specifying traffic patterns like streets, lanes, traffic lights, parking spots etc. and restrictions on these patterns like *one way* or *no right turns* which are necessary to qualify illegal actions (i.e. driving in the wrong direction in a one way street). This would allow to specify a complex street scenario in an abstract description and an interface could convert the abstract specification into a system specific representation of the scenario.

## 9 Conclusion

With this project we have shown how a multi-layered approach can help to hide the underlying details from the user of a moving objects database system by providing the information of interest on an abstract level. The user can view and edit the information, pose and edit queries and view the results without having to know the complexity of the system. To pose queries over the system we use a combination of graphical and tangible user interface, where users can utilize real objects to interact with the system. This is possible in the case of query posing as well as the query editing. Furthermore we allow for an interaction with the queries with immediate feedback, so that users can analyze results as soon as partial results are available and adjust or stop queries if needed or wanted.

## References

- [1] MITSIMLab. <http://mit.edu/its/mitsimlab.html>.
- [2] Prasad Sistla A., Wolfson Ouri, Chamberlain Sam, and Dao Son. Modeling and Querying Moving Objects. In *ICDE*, pages 422–432, 1997.
- [3] Ullmer B., Ishii H., and Jacob R. Tangible Query Interfaces: Physically Constrained Tokens for Manipulating Database Queries. In *INTERACT'03*, 2003.
- [4] Elias Frenzos. *Indexing Objects Moving on Fixed Networks*. 2003.
- [5] Yaman Fusun, S. Nau Dana, and S. Subrahmanian V. A Logic of Motion. In *International Conference on the Principles of Knowledge Representation and Reasoning (KR-2004)*, pages 85–94. ACM, 2004.
- [6] Ishii Hiroshi and Ullmer Brygg. Tangible bits: towards seamless interfaces between people, bits and atoms. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241, Atlanta, Georgia, United States, 1997. ACM Press New York, NY, USA.
- [7] Su Jianwen, Xu Haiyan, and H. Ibarra Oscar. Moving Objects: Logical Relationships and Queries. In *SSTD '01: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 3–19. Springer-Verlag London, UK, 2001.
- [8] M. Hellerstein Joseph, Avnur Ron, Chou Andy, Hidber Christian, Olston Chris, Raman Vijayshankar, Roth Tali, and J. Haas Peter. Interactive Data Analysis: The Control Project. *Computer*, 32(8):51–59, 1999.

- [9] Forlizzi Luca, Hartmut Güting Ralf, Nardelli Enrico, and Schneider Markus. A data model and data structures for moving objects databases. pages 319–330, 2000.
- [10] Wolfson Ouri. Moving Objects Information Management: The Database Challenge. In NGITS '02: Proceedings of the 5th International Workshop on Next Generation Information Technologies and Systems, pages 75–89. Springer-Verlag London, UK, 2002.
- [11] Wolfson Ouri, Xu Bo, Chamberlain Sam, and Jiang Liqin. Moving Objects Databases: Issues and Solutions. In Statistical and Scientific Database Management, pages 111–122, 1998.
- [12] Wolfson Ouri, Chamberlain S., Dao S., and Jiang L. Location Management in Moving Objects Databases. In Proc. of the Second International Workshop on Satellite-Based Information Services. Budapest, Hungary, 12, 1997.
- [13] Dieter Pfoser and Tryfona Nectaria. Requirements, definitions, and notations for spatiotemporal application environments. In GIS '98: Proceedings of the 6th ACM international symposium on Advances in geographic information systems, pages 124–130, Washington, D.C., United States, 1998. ACM Press New York, NY, USA.
- [14] Hartmut Güting Ralf, H. Böhlen Michael, Erwig Martin, S. Jensen Christian, A. Lorentzos Nikos, Nardelli Enrico, Schneider Markus, and Ramon Rios Viqueira Jose. Spatio-temporal Models and Languages: An Approach Based on Data Types. In Spatio-Temporal Databases: The CHOROCHRONOS Approach, pages 117–176, 2003.
- [15] M. Andrea Rodríguez-Tastets. Moving Objects Databases: Achievements and Challenges. Seminar Paper, 2005.
- [16] Steve S. Skiena. The Algorithm Design Manual. Springer, 1997.
- [17] Nectaria Tryfona, Rosanne Price, and S. Jensen, Christian. Chapter 3: Conceptual Models for Spatio-temporal Applications. In Lecture Notes in Computer Science, pages 79–116. 2003.
- [18] Q. Yang. A Simulation Laboratory for Evaluation of Dynamic Traffic Management Systems. PhD thesis, Massachusetts Institute of Technology, 1997.