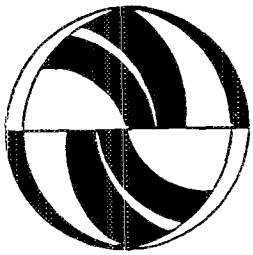
Title
Developing an Object-Oriented Testbed for Modeling Transportation Networks

Authors
Kwan, Mei-Po
Speigle, Jon M.
Golledge, Reginald G.

Publication Date
1997



Developing an Object-Oriented Testbed for Modeling Transportation Networks

Mei-Po Kwan
Jon M. Speigle
Reginald G. Golledge

Reprint
UCTC No. 409

The University of California
Transportation Center

University of California
Berkeley, CA 94720

The University of California Transportation Center

The University of California Transportation Center (UCTC) is one of ten regional units mandated by Congress and established in Fall 1988 to support research, education, and training in surface transportation. The UC Center serves federal Region IX and is supported by matching grants from the U.S. Department of Transportation, the California Department of Transportation (Caltrans), and the University.

Based on the Berkeley Campus, UCTC draws upon existing capabilities and resources of the Institutes of Transportation Studies at Berkeley, Davis, Irvine, and Los Angeles; the Institute of Urban and Regional Development at Berkeley; and several academic departments at the Berkeley, Davis, Irvine, and Los Angeles campuses. Faculty and students on other University of California campuses may participate in Center activities. Researchers at other universities within the region also have opportunities to collaborate with UC faculty on selected studies.

UCTC's educational and research programs are focused on strategic planning for improving metropolitan accessibility, with emphasis on the special conditions in Region IX. Particular attention is directed to strategies for using transportation as an instrument of economic development, while also accommodating to the region's persistent expansion and while maintaining and enhancing the quality of life there.

The Center distributes reports on its research in working papers, monographs, and in reprints of published articles. It also publishes *Access*, a magazine presenting summaries of selected studies. For a list of publications in print, write to the address below.

University of California Transportation Center

108 Naval Architecture Building
Berkeley, California 94720
Tel: 510/643-7378
FAX: 510/643-5456

Developing an Object-Oriented Testbed for Modeling Transportation Networks

Mei-Po Kwan
Department of Geography
Ohio State University
Columbus, OH 43210-1361

Jon M. Speigle
Department of Psychology
University of California, Santa Barbara
Santa Barbara, CA 93106

Reginald G. Golledge
Department of Geography
and
Research Unit in Spatial Cognition and Choice
University of California, Santa Barbara
Santa Barbara, CA 93106

Reprinted from
Santa Barbara Geographical Press
(1997)

UCTC No. 409

The University of California Transportation Center
University of California at Berkeley

Abstract

The objective of the paper is to discuss the development of an alternative representation of the transportation network using object-oriented GIS. This representation is important for the supply side of transportation planning and modeling. Object-orientation provides a way of solving the problem in a planar network for routing. It can facilitate the calculation of detailed network characteristics using properties such as inheritance and polymorphism. This representation is also closer to human perception of a transportation network. It is argued that by using an object-oriented GIS we can facilitate path selection using different criteria. We experiment with the design of the object-oriented system by developing an object-oriented representation of a transportation network and incorporating different path selection algorithms based on various behavioral assumptions. It is especially useful in the design for a versatile ATIS.

Introduction:

Transportation Science is host to a variety of theories concerning (among others) network structure, routing algorithms, traveler activity patterns, mode choice, demand forecasting, vehicle or traffic assignment, trip allocation, and traveler behavior. It has an expressed goal of increasing accessibility for all groups of people with regard to the environments in which they live and interact. A significant component of these goals is to further develop Intelligent Transportation Systems (ITS) through multi-level and multi-modal research and testing. This includes contributing to research on transportation system architecture, technology development, policy formation, and operational tests of various systems.

ITS objectives aim at utilizing advanced communication and transportation technologies to achieve traffic efficiency and safety. There are different components of ITS, including Advanced Traveler Information Systems (ATIS), Automated Highway Systems (AHS), Advanced Traffic Management Systems (ATMS), Advanced Vehicle Control Systems

(AVCS) and Advanced Public Transportation Systems (APTS). Development of a coherent system for ITS depends on our ability to deal with a vast amount of data about the locations of places, as well as with the complex representation of the transportation network linking those places, and the incorporation of both of these into a geographic database. The system therefore, can be constructed based upon the foundation of an integrated and comprehensive Geographic Information System (GIS).

To achieve the ITS aim of using advanced information technology and processing to improve traffic efficiency, both static and dynamic information is needed.
In addition a representation of the real world environment is required, with all the streets and their properties, and with information about location of related objects being required. Vehicle routing and navigation is then based on this network representation. Further, dynamic traffic information updating in a short time interval must be included for accurate traffic forecasting. With recent advances in technologies, fast location and temporal updating are possible. Global Positioning Systems (GPS) can accurately fix and trace the location of a vehicle to within several meters. Movement detectors can provide current traffic counts. The central question is how to handle this fast temporal change and update of vehicular location and volume within an efficient database system.

Thus, the successful development of ITS depends on the capability of incorporating a vast amount of information about the location of facilities which generate travel, with as realistic a representation as possible of elements of the transportation network in which travel occurs. Such a system can be based on an innovative and comprehensive Geographic Information System (GIS). Whereas current ITS primarily use simplified transportation networks as their basis, using GIS allows us to provide a more realistic representation of elements of the network and the ways that people perceive them. We can represent the network by defining roads or street hierarchies and by storing

environmental data as layers which can be overlain, aggregated, or decomposed at will. Storing the transportation network as a hierarchy facilitates the calculation of different paths through the network and allows the introduction of different path selection criteria, and the ability to handle overlapping modal use of network elements (e.g. cars, busses, and freight carriers). Since a long-run aim of ITS is to develop a real time multi-strategy travel decision support system over a multi-modal network, it is necessary to develop a data host and system model that is flexible, comprehensive, and realistic.

GIS have the potential to handle human movement in space and time. Existing GIS routines have been used to perform basic network actions such as finding a shortest path, solving a traveling salesperson problem, and recently, handling location-allocation problems. Applications commonly available in existing GIS software like TRANSCAD and ARC/INFO NETWORK have largely been operated in data models that rely on a simple planar link-node representation of network structure. However, this structure does not satisfy the various requirements of today's traffic management. In this paper, therefore, we illustrate a way of developing a testbed of a transportation network that could provide a realistic base on which to graft many ITS components.

I. System Review

A. System Overview:

Our system is implemented in C++, under the Windows NT operating system, and has a user-extensible interpreted front-end. The interpreted environment is comprised of variables, classes, and functions. The user is able to create text files which define classes and named sequences of statements which manipulate one and two-dimensional variables. The variables may be either homogeneous arrays of a primitive type (i.e., integer, floating point number, character, or string) or arrays of class instances. The arrays

of instances may be either all of the same type (homogeneous) or of different types (heterogeneous). The operations supported by a given array depends, accordingly, on the content type. For the numeric types, basic matrix operations are defined as well as a number of standard mathematical functions. For the string type, the operations include concatenation, comparison and formatted assignment. For the instance arrays, the method functions may be called and the attributes may be accessed.

Figure 1

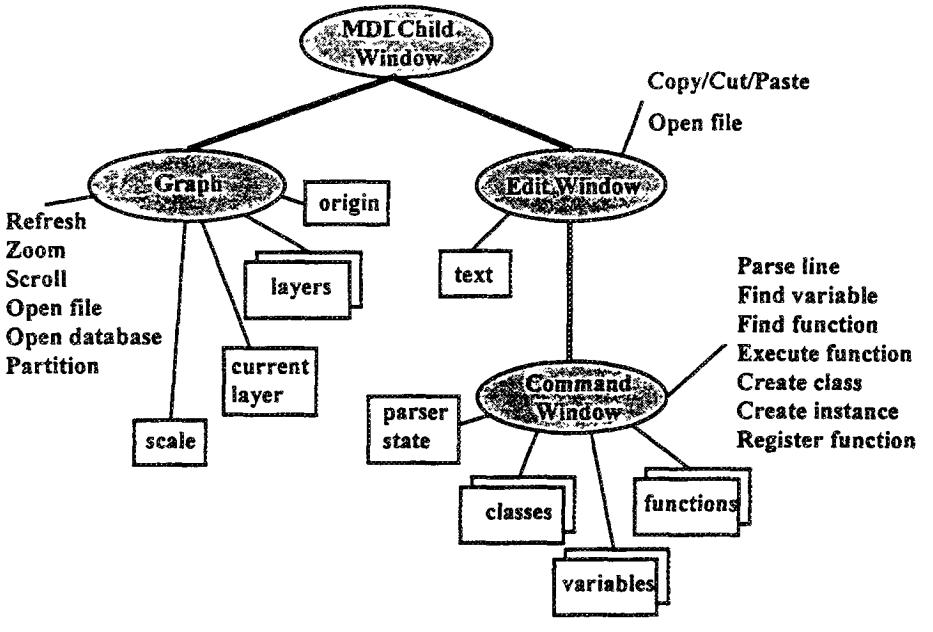


Figure 1. Interface components. The Edit and Graph classes derive from a common, MDI child class. The Command Window extents the functionality of the Edit class by adding the parser.

The interpreter supports complex data types by means of aggregation and inheritance. Each type corresponds to a class definition and a set of "metadata". The metadata characterizes the types and names of attributes possessed by all instances of the class, the names and input/output signatures of methods supported by the instances, and the inter-

class, parent/child relations. The metadata also includes the set of all instances of the class, or class extents.

Classes are defined in one of two ways. The first method is for the user to create a text file which when interpreted creates the class. A class "road" is declared as follows. "Road" has two single-precision floating point attributes and a member function called "Cost". The members may be referred to as "road::name", where "name" refers to either a method or attribute name.

```
class "road" {
        float       s;
        float       t;
        function    Cost;
}
```

The syntax for deriving a class "highway" from "road" is as follows. A derived class "inherits" the attributes and methods of its parent class. "Highway" has two integer attributes as well as the floating point attributes and Cost method of class "road".

```
class highway : road {
        int         x;
        int         y;
        function    Cost;
}
```

If a method defined in the derived class has the same name as a method of the parent class, then it "overrides" or "hides" the method of the parent class. This feature, combined with the capability to store heterogeneous arrays of instances, allows for a method name to evaluate to a different function call depending on the type of the instance. In the following, classes 'road' and 'highway' are instantiated, the instances are stored in a vector, and their cost method is invoked:

```
'Road'          road(1);
'Highway'       highway(2);
v=[road,highway];
x=v.Cost()
```

For the instance of Road, Road::Cost is called; for Highway, Highway::Cost is called. This is referred to as polymorphism.

The second method of creating classes is for the metadata to describe a "built-in", C++ class. The metadata in this case describes a mapping from the method names to an index into a dispatch table. The index is used to call actual C++ methods for the class. The distinction between attributes and methods is blurred, as the methods provided in the dispatch table are the only means of accessing the attributes. In some cases, the built-in classes may be instantiated at the command-line. The spatial data types which will be described below fit into this category. Any built-in class which may be instantiated may also be derived from. This is the primary method of adding attributes to the built-in types. In other cases the built-in classes do not allow instantiation (e.g., the "Graph" or "MetaData" classes). Instances of this sort are returned by some routines (e.g., "gr=GetCurrentGraph"). The user may then pass methods to the instances (e.g., "gr.Zoom(2)"). The user interface to the two types of metadata are consistent.

As with to the class definitions, the operations supported by the interpreter come in two flavors: built-in and user-defined. Built-in operations are implemented as C++ routines and perform such tasks as listing and deleting variables, retrieving instances of built-in types (e.g., Graphs or MetaData), and interacting with the file system. The user-defined operations are text files which may contain any number of statements. When the first word of the file is "function" a local environment is created when the function is called (e.g., "z = Square(2)").

```
function [y] = Square(x)
y = x*x;
```

The first line of the file declares the input and output signature of the function. For "Square" the input variable is "x" and the output variable is "y". The newly created environment is initialized with the function inputs arguments. The output variables are culled from the environment when the function is exited. Alternatively, when the first word is not "function", the calling environment is used.

The member functions of user-defined classes are also text files. From outside a member function, the member variables may be accessed and the functions invoked by following an instance's variable name with the method name. When a method is invoked, only the member variables corresponding to the class defining the method are accessible. But from within a method, these other method or attribute names may be referenced directly. For the case of overridden functions, a syntax is defined for explicitly calling the overridden function.

Queries are supported both against a class's extents and against the results of previous queries. In the following example a class is defined, several instances are created, a query is made against the class, and a second query is made against the first query's results.

```
class Road {
        int             v_road;
};

Road a(1);
Road b(2);
Road c(3);
Road d(4);

def=findclass(Road);
res1=def.Find("v_road>=4")
res2=res1.Find("v_road>=3")
```

When a query is made against a named class's extent, it is also made against the extents of any derived classes. The conditional is evaluated for each instance of the base class and then recursively on any derived classes. The instances for which the conditional evaluates to 'true' are accumulated as a heterogeneous instance list. This list may be assigned to a variable and subsequently used as the extent for further queries. The evaluation process sets the environment state as in a method call, to the particular instance against which the query is being evaluated. This means that complex, "path" expressions may be used within the conditional, that is, involving the attributes of fields which are themselves instances (i.e., "road.highway==1").

A query conditional may be composed of any number of sub-conditionals. Each sub-conditional will be a function/method call which evaluates to true or false, a relational expression (i.e., ">", "<", ">=", "<="), or an equality expression (i.e., "==" or "!="). The sub-conditionals will be joined by the logical "and" and "or" operators: && and ||. The precedence of the arithmetic, relational and logical operators is as in C++. The bindings for method calls are determined at run-time. The following example would return the instances of class Road for which the attribute was between 1 and 4:

```
res=def.Find("v_road>1 && v_road<4")
```

B. Built-In, Spatial Types

The spatial type hierarchy is shown in Figure 2. The basic spatial types are the Point, Line, Polyline and Circle. The types specific to a network implementation are the Node and Link. Each of these types is implemented as a C++ class and has a corresponding metadata description. The user is able to instantiate these classes at the command-line, to access the instance's methods and attributes, and to derive from these classes. The classes used to represent a hierarchy of road types are described in the next section. The methods

common to all types derived from spatial object include drawing, topological comparisons, and serialization. Each spatial object possess a "location" (i.e., x, y, and z coordinate) and unique "identifier". Polyline adds a list of vertices (i.e., x, y, and z coordinates). The Link adds attributes for identifying the "junctions" at the endpoints of the list of vertices. The Node adds a list of links to the basic attributes.

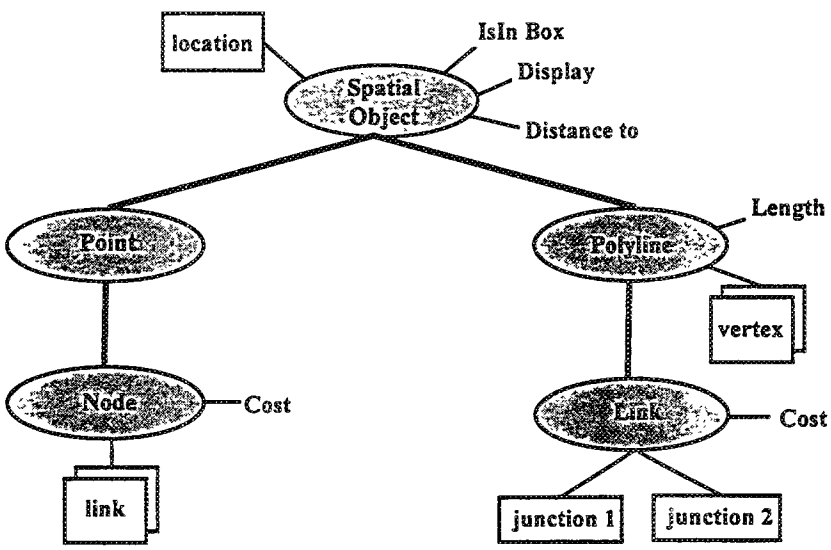


Figure 2. Built-in spatial object hierarchy.

The spatial classes and any classes derived from them support spatial indexing of the class's extents. A quadtree of depth N is created for each class. Each quadtree leaf contains a list of pointers into the class extents. The list contains pointers to the instances whose centroid fell within the bounding box for that leaf. We refer to this procedure as "partitioning". Figure 3 shows the hierarchical relation between the basic meta-class type and the specialized, spatial meta-class type.

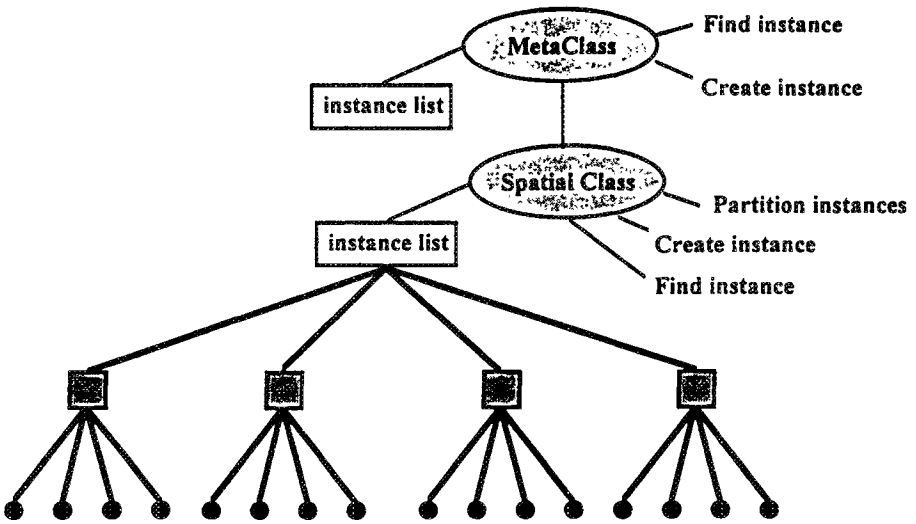


Figure 3. Meta-Class hierarchy.

As noted above, member functions may be utilized within queries as well as value-based relational or equality statements. The topological member functions may therefore be used to select instances. A fully optimized query evaluator would utilize the spatial indexing when available. Such optimizations are operational for simple queries (i.e., queries composed of only a single spatial method call) and are under construction for more complicated, compound queries. The following example illustrates instantiating a built-in spatial class, partitioning, and executing several spatially based queries:

```
Line a([0 0],[1 1]);
Line b([1 0],[2 1]);

def=findclass(Line);
bbox=def.ComputeBBox();
def.Partition(bbox,3);

disp("Testing spatial query:  find a");
res=def.Find("IsInBBox([0 0 1 1])")

disp("Testing spatial query:  find b");
res=def.Find("IsInBBox([1 0 2 1])")

disp("Testing spatial query:  find a and b");
res=def.Find("IsInBBox([0 0 2 1])")
```

C. Road Hierarchy

The spatial classes were extended by deriving user-defined classes. The derived classes added attributes as well as constructor/destructor member functions. As shown in Figure 4, the class "Road" was derived from Link.

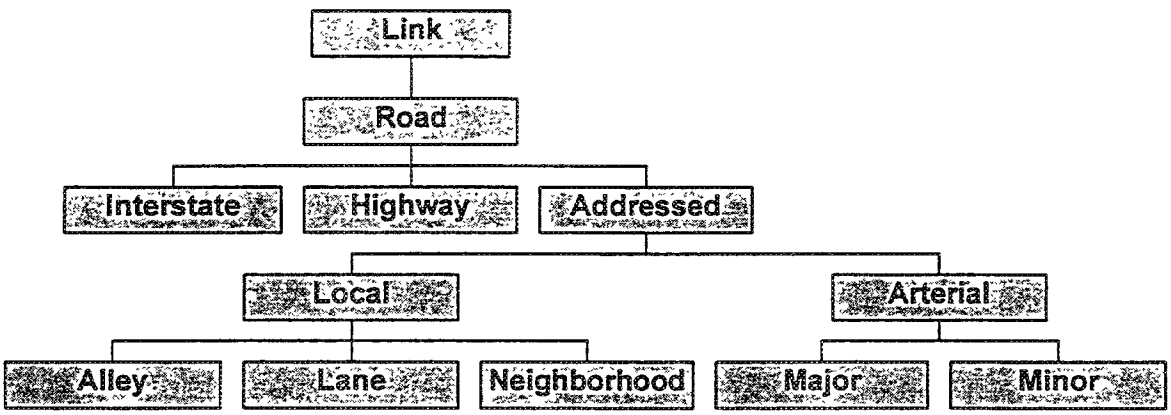


Figure 4. Road hierarchy.

The class definitions for Road, Highway and Addressed are as follows. The remainder of the definitions do not add attributes. Their purpose may be viewed as providing a hierarchical classification scheme. Member functions may be added to distinguish the favorability/unfavorability of any of the road classes.

```
class Road : Link {
        string      name;
        int         nLanes;
        int         divider;
        int         paved;
        int         speedLimit;
        int         oneWay;

        function Cost;
};
```

The attributes of class Highway are the "level", which may take values of STATE or INTERSTATE.

```
class Highway : Road {
        int         level;

        function Cost;
};
```

The AddressedRoad contains attributes for characterizing the address ranges along the segment. The member functions determine the address given a position and the inverse process of determining the position given an address.

```
class AddressedRoad : Road {
        int         lowAddress;
        int         highAddress;

        function    PositionToAddress;
        function    AddressToPosition;
};
```

D. Junction Hierarchy

This section is still relatively undeveloped because at this stage, only the approach toward Junctions has been conceptualized. Currently, the Links contain only pointers to Nodes. This will be converted to pointers to Junctions in later phases of the work.

The basic network behavior is represented by class Node, which inherits from Point. The Node class adds a cost function and a list of connected Links. The class Junction inherits from Node and adds the capability to describe a set of turn restrictions. The Cost method of class Junction determines the cost of traversing the junction given a pair of links. Figure 5 shows several user-defined classes which further specialize class Junction. These classes basically override the Cost function for class Junction. The further derivation of this derivation is shown in Figure 6.

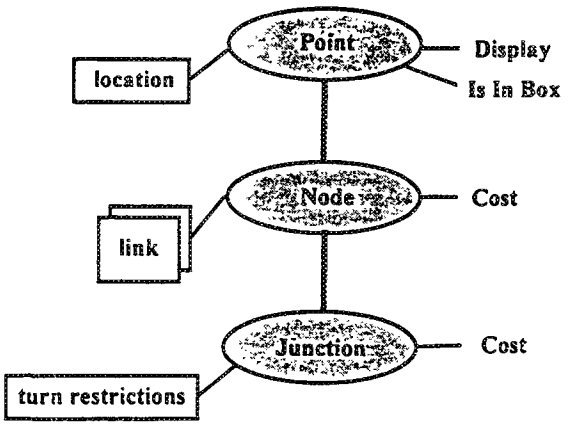


Figure 5. Members of Point, Node and Junction classes.

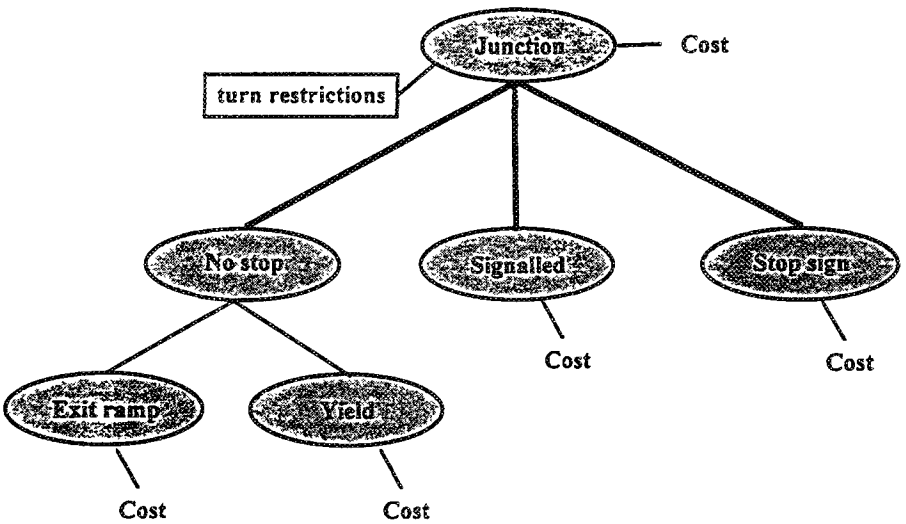


Figure 6. Junction hierarchy.

E. How does OO affect route finding?

First, polymorphism allows us to call different cost functions for different types of road. For instance, the cost function for Link (Link::Cost) might just return the length of the link. The cost function of Road, which derives from Link, might divide the length by the speed limit to have the cost reflect the time needed to traverse the link (i.e., "Link::Cost/speed"). Polymorphism is the OO capability to call different cost functions depending on the type of the instance. Each road type in the Road hierarchy may define

its own cost function. A Djikstra algorithm requires the network topology and the capability to evaluate a cost for each instance. The algorithm does not care about exactly how the cost is assigned. With polymorphism we are able to call different cost functions for different types of roads or we are able to minimize different criteria by modifying the cost functions. The code which implements the network algorithm is "re-used" while only the cost functions are modified.

With our system it is possible to minimize other criteria besides distance or time. Each cost function in the Road hierarchy is based on the output of the previous level. If Highway does not supply a specialized cost function, then its cost will automatically call the Cost function of its parent class. This is "inheritance". But if we wanted highways to be preferred over other types of roads, then Highway::Cost could be set to "Road::Cost*0.1". The cost for a highway would be less than for another type of Road. In this way, it would be possible to use the Djikstra route finding algorithm with the following criteria: shortest path, shortest time, at-the-highest-speed, minimum-traffic-volume, most aesthetic, maximum proportion on a road-class, maximum use of one-way travel (or on divided road).

A similar strategy may be used to specialize cost functions for different types of junctions. We use a modified Djikstra algorithm which allows assigning costs to traversing a node. With this implementation we may minimize criteria such as the number of left turns. The node has associated with it a turn matrix. A specialized, "hate-left-turn" cost function could produce a higher cost for making a left turn. This allows us to incorporate "behavioral" criteria into the route finding process. These behavioral criteria may even be customized for different users. With a junction hierarchy we may deal with the following criteria: minimize left turns, minimize delay at intersection, minimize number of signaled intersections.

Some types of criteria will, however, lie outside the capabilities of the polymorphic Djikstra algorithm. Those types of criteria could not be handled because they require information to be passed to the cost function which is not normally passed by the Djikstra algorithm. For example, a criteria could be to "always proceed in a corridor toward the destination". Evaluating this criteria would require the endpoint and the position of the last point, neither of which Djikstra automatically supplies. Other questionable criteria are "place the longest/shortest leg first", "avoid high accident places", "maximize number of destinations along a single route", or "maximize length of a dominant leg".

F. Versant

The Versant ODBMS is designed as a fault-tolerant, multi-user, and distributed system. Some set of databases may be distributed across a number of machines and may be accessed simultaneously by any number of users. Versant's client-server architecture supports distribution, the capability of which is essential for scalability. Transactions are handled in a robust fashion by using "two-phase commits". The multi-user requirements have resulted in a sophisticated set of object locking and concurrency mechanisms. Users may "checkout" items from a group database to a personal database, with the items being locked until the corresponding "checkin".

Our use of Versant's system does not utilize many of the advanced features. A fully operational ITS system would, however, require the system to be highly distributed and robust. Our use of the distribution capabilities was limited to placing the database on one machine and the client program on another. We also did not explore the issues concerning concurrent access by a large number of users. In the testbed, a single user possessed read/write access to the database. In a fully operational ITS, different classes of users would have different levels of access. "End-users" would have only read access.

Concurrency would be an issue only for the "administrators" because they are likely to be the only users able to modify the database.

Versant provides interfaces for a number of programming languages: Smalltalk, C, and C++. We utilized the C++ interfaces, including the capability to declare classes at run-time.

We selected a particular OO DBMS software - VERSANT - as our primary medium (see also the recommendation in Göllü, 1995). Part of using Versant software as the medium for an object-oriented GIS is the development of the partitioning scheme. This is possible because of Versant's implementation as a client-server database. Our application is the "client" and would request data from the database "server." Of the several ways in which a client's request may be couched, one is to use simple equality/inequality statements on a given class's attribute values. The server will return class instances for which the conditional is met. We believe that this is the highest complexity of queries supported by Versant. More complicated, user-defined selection procedures are not supported yet because such procedures would need to reside on the server side. This is what is meant as "storing class methods on the server." Versant only provides the means to store the data, not the methods. The only way around this inability to conduct complicated queries (such as a bounding box comparison against an entity's position) seems to be to "recompute" these necessary computations. This is why it seems necessary to partition the map within Versant. The need is probably greater than it was for displaying the map.

The methods described above for creating hierarchically defined attributes and a hierarchically structured set of layers were mapped to Versant. Versant's role was to allow for storage of class instances between sessions. In a relational database, this amounts to the storage on disk of an application's tables and their loading when next

required. In much the same way that Versant scans the application's C++ header files to determine the class hierarchy, the code that has been written for creating class hierarchies does the same thing. The class definitions are used to create the "meta-class" information used to describe the memory layout of the attribute fields. The procedures to store class instances to disk were then written based on the meta-class information (i.e., the class fields are stored as a stream of bytes which is exactly what is needed to store the instance to disk). This approach is an interim solution while the partitioning, network and entity-attribute relations are worked out. All of these components seem required prior to storing class instances using Versant. First, the data to be stored in Versant must be instances of C++ classes. Second, using Versant as the storage mediary appears to require storing the map within the database. Storing the map requires partitioning. It is still possible even with a partitioning approach that map storage in Versant will prove prohibitively expensive in terms of space or access time. We will investigate possible solutions to this problem, particularly one that would be to store only the partitioning structure within Versant, store the map separately in some other file format, and then load the map into the partitions at run time.

G. A Real-World System Experiment

1. Conversion of Etak map to Object-Oriented Data Model

[This section is very sketchy and will be elaborated upon.] The road hierarchy was defined and the Etak map of the Santa Barbara area was read into it. There are approximately 28,000 nodes and some 14,000 links in the Etak county map, and 7800 notes and 3900 links in the testbed area.

I have not dealt with converting the Etak map into the junction hierarchy, as this may involve much more user intervention. Making this jump requires the additional user-interface capability of mouse tracking and selecting objects in the graph. The task of

converting nodes to/from different junction classes might also be automated by a set of dialog boxes.

2. "Modified" Djikstra Routing algorithm

A description of the "modified" Djikstra algorithm should be inserted here and how it was implemented using the Road and Junction classes.

Evaluating the Testbed:

Some of the critical features involved in evaluating the worth of any object-oriented data model or object-oriented data structure include its ease of use, the relevance of the attributes defined in the system, whether or not the model deals with real or artificial concepts, the degree to which there is a clear translation between model entities and the actual objects, and whether or not there are acceptable matches between those activities undertaken in the real world and those activities incorporated into the model. Other criteria relate to the number of modules embedded in the system that have to be changed in order to work in a real environment, the number of steps that operation of the system requires, the degree to which one must know and accept a process model of the system, and the time that is required to integrate changes in the system to ensure it is dynamic real time.

Our project was designed to contribute to the next generation of traffic management technology, particularly in terms of dispensing information to travelers in a pre-planning or en-route phase via an ATIS. ITS generally appears to be moving more towards Object-Oriented data structures and models and we believe our work is in line with these nationwide trends.

We expect that this research will have important significance on both basic and applied levels. We have conceptualized and developed an object-oriented geographic information system from transportation modeling. Our continuing effort will focus on the data modeling issues of a multi-modal network, and the implementation of a multi-strategy travel decision support system built on this object-oriented system. Elsewhere (Kwan, Golledge, & Speigle, 1996) we have discussed the advantages and disadvantages of an object-oriented approach to transportation modeling. Apart from its improved capability of handling the transport network as more of a perceptually accurate system, the object-oriented data model allows us to incorporate hierarchical layering within the basic network that ties to the normal engineering way of interpreting road systems. Polymorphism allows us to perform ITS functions on various classes of objects in the transportation network, an ability that is not easily obtained using existing software systems. The approach also appears likely to substantially decrease the time involved in interacting with the database, particularly by using partitioning and inheritance characteristics. Cost of operation should consequently be reduced. Both these factors are important when considering that the primary aim of an ATIS component of an ITS is to get useful information to travelers in as timely a manner as possible so that on-route decision making can be undertaken. There are many basic research problems relating to the development of workable object-oriented data models for use in transportation planning and this research has examined some of these. We also expect that the data model as developed will greatly facilitate the implementation of ITS by more quickly resolving conflicts with respect to ultimate selection of routes, substituting destinations, changing activity patterns, and rescheduling activities.

References

Booch, G. (1991) _____

Booch, G. (1994). Object-Oriented Analysis and Design with Applications, Benjamin/Cummins, Redwood City, California.

Clementini, E., and Difelice, P. (1994) Object-oriented modeling of geographic data. *Journal of the American Society for Information Science,* 45, 9: 694-704.

Date, C.J. (1985) *An introduction to database systems.* Reading, MA: Edison Wesley.

David, R., Raynal, I., Schorter, G., and Mansart, V. (1993) "GeO2: Why Objects in a Geographical DBMS." The Third Symposium Databases, 264-276.

Davison, P.A. (1986) Inter-relationships between British driver's age, visual attributes, and road accident histories. In A.G. Gaile, et al (Eds.), *Vision in vehicles.* North Holland: Elsevier Science Publications, pp. 23-32.

Dingus, T.A., et al. (1989) Intentional demand requirement of an automobile moving-map navigation system. *Transportation Research A,* 23A, 4: 301-315.

Egenhofer, M.J., and Frank, A.U. (Eds.) (1989) *Object-oriented modeling in GIS: Inheritance and propagation.* Falls Church: American Congress on Surveying and Mapping, American Society of Photogrammetry and Remote Sensing.

Frank, A.U., and Egenhofer, M.J. (1992) Computer cartography for GIS - an object-oriented view on the display transformation. *Computers & Geosciences,* 18, 8: 975-987.

Gahegan, M.N., and Roberts, S.A. (1988) An intelligent, object-oriented geographical information system. *International Journal of Geographical Information Systems,* 2, 101-110.

Göllü, A.O. (1995) Object Management Systems. California PATH Research Report UCB-ITS-PRR-95-19.

Gunther, O., and Lamberts, J. (1994) Object-oriented techniques for the management of geographic and environmental data. *Computer Journal,* 37, 1: 16-25.

Guptill, S. C. (1989) "Speculations on Seamless, Scaleless Cartographic Data Bases." Auto-Carto 9, Ninth International Symposium on Computer-Assisted Cartography, Baltimore, Maryland, 436-443.

Guttman, A. (1984) "R-Trees: A Dynamic Index Structure for Spatial Searching." ACM International Conference on Management of Data.

Haas, L. M., and Cody, W. (1991). "Exploiting Extensible DBMS in Integrated Geographic Information Systems." Advances in Spatial Databases. Report from Second Symposium, SSD91, O. Gunther and H. Schek, eds., New York, Springer-Verlag, Zurich, 423-450.

Har-El, Z., and Kurshan, R.P. (1987) COSPAN Users guide. Murray Hill, NJ: AT&T Bell Laboratories.

Herring, J.R. (1992) Tigris - a data model for an object-oriented geographic information system. *Computers & Geosciences,* 18, 4: 443-452.

Hoare, C.A.R., (1985) *Communicating sequential processes.* Prentice-Hall International.

Inan, K., and Varaiya, P. (1988) *Finitely recursive process models for discrete event systems.* IEEE Transactions, Auto.Control, Volume AC-33, No. 7, pp. 626-639.

Jackson, R. W. (1994). "Object-Oriented Modeling in Regional Science: An Advocacy View." Papers in Regional Science, 73(4), 347-367.

Jonah, B.A., and Dawson, N.E. (1987) Youth and risk: Age differences in risky driving, risk perception and risk utility. *Alcohol, Drugs, and Driving,* 3, 3-4: 13-29.

Kemp, K. (1992) "Environmental Modelling with GIS: A Strategy for Dealing with Spatial Continuity." GIS/LIS Annual Conference, 397-406.

Khoshafian, S. (1993). Object-Oriented Databases, Wiley, New York.

Khoshafian, S., and Abnous, R. (1990). Object-Orientation: Concepts, Languages, Databases, User Interfaces, Wiley, New York.

(NOTE: THE KWAN (1994) REFERENCE WAS NOT CITED IN THE TEXT)

Kwan, M.-P. (1994). "GISICAS: A GIS-Interfaced computational-Process Model for Activity Scheduling in Advanced Traveler Information Systems," , University of California, Santa Barbara.

Kwan, M.-P., Golledge, R.G., and Speigle, J. (1996) A Review of Object-Oriented Approaches for Transportation Modeling.

Lohman, G., Lindsay, B., Pirahesh, H., and Schiefer, K. B. (1991). "Extensions to Starburst: Objects, Types, Functions and Rules." Communications of the Association for Computing Machinery, 34, 94-109.

Medeiros, C., and Pires, F. (1994). "Databases for GIS." SIGMOD Record, 23(1), 107-115.

Milne, P., Milton, S., and Smith, J.L. (1993) Geographical object-oriented databases - a case study. *International Journal of Geographical Information Systems,* 7, 1: 39-55.

Milner, R. (1980) *A calculus of communicating systems.* Springer-Verlag.

Nievergelt, J., Hinterberger, H., and Sevcik, K. (1984) "The Grid-File: An Adaptable Symmetric Multikey File Structure." ACM Trans. on Database Systems.

Noy, I.Y. (1990) *Attention and performance while driving with the auxiliary in-vehicle displays.* Road Safety and Motor Vehicle Regulation - Transport Canada, Ottawa, Report #TP10727(E), December.

Özveren, C.M. (1989) Analysis and control of discrete event dynamic systems: A state space approach. Ph.D. dissertation, MIT.

Praehofer, H., Auernig, F., and Resinger, G. (1993) An environment for DEVS-based multiformalism simulation in common LISL/CLOS. *Discrete Event Dynamic Systems: Theory and Application,* 3, 2: 119-149.

Ptolemy Manual (1995) The almalmagest, Volume 1-4, Version 0.5.2, College of Engineering, UC Berkeley.

Ramadge, P., and Wonham, W. (1987) Supervisor control of a class of discrete event processes. *SIAM Journal of Control Optimization,* 25, 1: 206-230.

Roberts, S.A., and Gahegan, M.N. (1993) An object-oriented geographic information system shell. *Information and Software Technology,* 35, 10: 561-572.

Roberts, S.A., Gahegan, M.N., Hogg, J., and Hoyle, B. (1991) Application of object-oriented databases to geographic information systems. *Information and Software Technology,* 33, 1: 38-46.

Rowe, L. A., and Stonebraker, M. R. (1987) "The Postgres Data Model." The 13th VLDB Conference, San Mateo, California, 83-96.

Rumbaugh, J., et al. (1991) *Object-oriented modeling and design.* Englewood Cliffs, NJ: Prentice-Hall.

Schlaer, S., and Mellor, S. (1988) *Object-oriented systems analysis: Modeling the world in data.* Englewood Cliffs, NJ: Yourdan Press.

Scholl, M., and Viosard, A. (1992) *Object-oriented database systems for geographic applications: An experiment with O2.* Paper presented at the Geographic Database Management Systems Workshop Proceedings, Capri, Italy. May 1991.

Schwetman, H. (1989) CSIM Reference Manual (Revision 13). Micro Electronics and Computer Technology Corporation, 3500 West Valcones Center Drive, Austin, Texas 78759.

Shinar, D. (1978) *Psychology on the road - the human factor in traffic safety.* New York: John Wiley & Sons.

Tomlin, C. D. (1990). Geographic Information Systems and Cartographic Modelling, Prentice-Hall, Englewood Cliffs, New Jersey.

van Oosterom, P., and van den Bos, J. (1989) An object-oriented approach to the design of geographic information systems. *Computers & Graphics,* 13, 4: 409-418.

Walker, J., Alicandri, E., Sedney, C., and Roberts, K. (1990) *In-vehicle navigation devices: Effects on the safety of driver performance.* Report #FHWA-RD-90-053. Office of Safety and Traffic Operations, Research and Development, Federal Highway Administration, McLean, Virginia, May.

Wiegand, N., and Adams, T. M. (1994). "Using Object-Oriented Database Management for Feature-Based Geographic Information Systems." Journal of the Urban and Regional Information Systems Association, 6(1), 21-36.

Williamson, R., and Stucky, J. (1991). "An Object-Oriented Geographical Information System." Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD, R. Gupta and E. Horowitz, eds., Prentice-Hall, Englewood Cliffs, NJ, 297-311.

Worboys, M.F. (1992a) A generic model for planar geographical objects. *International Journal of Geographical Information Systems,* 6, 5: 353-372.

Worboys, M. F. (1992b). "Object-Oriented Models of Spatiotemporal Information." GIS/LIS Proceedings, 2, 825-834.

Worboys, M.F. (1994a) A unified model for spatial and temporal information. *Computer Journal,* 37, 1: 26-34.

Worboys, M.F. (1994b) Object-oriented approaches to geo-referenced information. *International Journal of Geographical Information Systems,* 8, 4: 385-399.

Worboys, M.F., Hearnshaw, H.M., and Maguire, D.J. (1990) Object-oriented data modelling for spatial databases. *International Journal of Geographical Information Systems,* 4, 4: 369-383.

Yourdan, E. (1989) *Modern structural analysis.* Englewood Cliffs, NJ: Yourdan Press.

Zeigler, B. (1984) *Multifaceted modeling and discrete event simulation.* London: Academic Press.