

# UC Berkeley

## Working Papers

### Title

Database Environment for Fast Real-Time Simulation of Urban Traffic Networks with ATMIS

### Permalink

<https://escholarship.org/uc/item/76k4c6nf>

### Authors

Jayakrishnan, R.  
Sheu, Phillip  
Wang, Taehyung  
et al.

### Publication Date

2000-03-01

**This paper has been mechanically scanned. Some errors may have been inadvertently introduced.**

CALIFORNIA PATH PROGRAM  
INSTITUTE OF TRANSPORTATION STUDIES  
UNIVERSITY OF CALIFORNIA, BERKELEY

## **Database Environment for Fast Real-time Simulation of Urban Traffic Networks with ATMIS**

**R. Jayakrishnan, Phillip Sheu,  
Taehyung Wang, MinHua Xu**  
*University of California, Irvine*

**California PATH Working Paper  
UCB-ITS-PWP-2000-4**

This work was performed as part of the California **PATH** Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department Transportation, Federal Highway Administration.

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the State of California. This report does not constitute a standard, specification, or regulation.

Report for MOU 227

March 2000

ISSN 1055-1417

**Database Environment for Fast Real-time Simulation of Urban Traffic  
Networks with ATMIS**

Partners in Advanced Transit and Highways (PATH), CALTRANS  
Research Project MOU-227

September 1998

**R. Jayakrishnan**

Department of Civil and Environmental Engineering  
University of California, Irvine, CA

**Philip Sheu**

Department of Electrical and Computer Engineering  
University of California, Irvine, CA

**Taehyung Wang**

Department of Electrical and Computer Engineering  
University of California, Irvine, CA

**MingHua Xu**

Department of Electrical and Computer Engineering  
University of California, Irvine, CA

## **Abstract**

This project develops the environment for using the ATMIS simulation software developed under previous PATH projects (MOUs 39, 84 and 170) for real-time traffic simulation and scenario analysis with feedback from the **real** urban network. DYNASMART simulation model was developed during the MOU-39 research, and was augmented with a graphical user-interface and display during the MOU-84 research. However, **for real-time use** in realistically large networks, faster parallel processing is required, which is the focus of the current project, MOU-170. **Based** on the early results from **MOU-170** research, it **has** become apparent that the full potential of such a simulation framework cannot be realized without a database management system that handles the data for the multiple processes during the parallel simulation, **as well as** the network feedback data for real-time updates of the simulation data and parameters. An object-oriented database environment is proposed here. Partially on-line data and off-line data from the Anaheim ATMS research testbed will be used to replicate the real network feedback during the research. The model **will** be developed with **as** much compatibility with other PATH projects (MOU-141, MOU-169) **as** possible. It is expected that the model will complete the environment for simulation-based analysis of ATMIS in real-time, which will be required for the field operational tests envisaged in the near future by Caltrans, PATH, and FHWA

Key words: Traffic Simulation, Databases, Object-relational Databases, Dynasmart

# CONTENTS

Abstract

Chapter 1 INTRODUCTION	1
1.1. Overview and Summary of the Research Project	
1.2. Motivation and Background	
Chapter 2 METHODOLOGY	4
2.1. The Simulation Framework (DYNASMART)	
2.2. <b>Project</b> Overview and Relationship to PATH MOU-170	
2.3. Limitations of Current Approaches	
2.4. Methodology of the Object-Relational Database System	
Chapter 3 OBJECT-RELATIONAL DATABASE DESIGN FOR DYNASMART	11
3.1. Introduction to the Fundamental Design Concepts (Relational Databases)	
3.1.1. Primary and Foreign Keys	
3.1.2. Queries	
3.1.3. Structured Query Language	
3.1.4. Referential Integrity	
3.1.5. Normalization	
3.2. Object-Relational Database Design for DYNASMART	
Chapter 4 FRONT-END DATABASE QUERY SYSTEM – <i>Netcompose</i>	24
4.1. Overview of <i>Netcompose</i>	
4.2. <i>Netcompose</i> Network Architecture	
4.3. Active Rule Processing in <i>Netcompose</i>	
4.4. Related Issues on Rules in Database Query Systems	
4.5. Examples of the use of <i>Netcompose</i> to Query Traffic data	

<b>Chapter 5 SOFTWARE INTEGRATION</b>	<b>42</b>
<b>5.1. MS-Access Database System on PCs</b>	
<b>5.2. CORBA/C++ Software at the Caltrans-UCI ATMIS Research Testbed</b>	
<b>5.2.1. Prerequisites for building CORBA clients</b>	
<b>5.2.2. Building and executing CORBA clients</b>	
<b>5.2.3. Description of CORBA clients</b>	
<b>5.2.3.1 LDS</b>	
<b>5.2.3.2 VDS</b>	
<b>5.2.3.3 RMS</b>	
<b>5.2.3.4 CCTV</b>	
<b>5.2.3.5 CMS</b>	
<b>5.2.3.6 Decoded LDS</b>	
<b>5.2.3.7 DecodedLDS and RMS</b>	
<b>Chapter 6 RESEARCH CONCLUSION</b>	<b>47</b>
<b>ACKNOWLEDGEMENT</b>	<b>49</b>
<b>REFERENCES</b>	<b>50</b>

## LIST OF FIGURES

- Figure 1.1. Schematic Diagram of the Proposed Database Management
- Figure 2.1. DYNASMART Object-relational Database Design
- Figure 4.1. Architecture of *Netcompose*
- Figure 4.2. *Netcompose* and Internet/Intranet
- Figure 4.3. A rule network
- Figure 4.4. A merged rule network
- Figure 4.5. User selection object(s) from Object Chooser.
- Figure 4.6. Initialization of simulator from the database query system.
- Figure 4.7. Posted Query on selected vehicles(s)
- Figure 4.8. Query Result (at vehicle generation point)
- Figure 4.9. Query Result (Specific vehicle)
- Figure 4.10. Visualization of vehicle positions (3-D view)



# Chapter 1

## INTRODUCTION

### 1.1. Overview and Summary of the Research Project

This project develops the environment for using the ATMIS simulation software developed under previous PATH projects (MOUs 39, 84 and 170) for real-time traffic simulation and scenario analysis with feedback from the real urban network. DYNASMART simulation model was developed during the MOU-39 research, and was augmented with a graphical user-interface and display during the MOU-84 research. However, for real-time use in realistically large networks, faster parallel processing is required, which is the focus of the current project, MOU-170. Based on the early results from MOU-170 research, it has become apparent that the full potential of such a simulation framework cannot be realized without a database management system that handles the data for the multiple processes during the parallel simulation, as well as the network feedback data for real-time updates of the simulation data and parameters. An object-oriented database environment is designed in this research. The model will complete the environment for simulation-based analysis of ATMIS in real-time at the Caltrans-UCI Research Testbed, and will be useful for the projects and operational tests envisaged in the near future by Caltrans, PATH, and FHWA.

### 1.2. Motivation and Background

The motivation for this project developed during a meeting in 1995 with Mr. Joseph Palen of the California Department of Transportation in connection with the research on parallel processing for fast traffic simulation (PATH MOU-170) which was then ongoing at UC Irvine. The MOU 170 research was exploratory in nature, and developed parallel codes of traffic flow models, with specific focus on the DYNASMART model developed for ATMIS simulations. The model in the sequential form required considerable computation time. As an example, the Anaheim network of about 1000 links runs up to 4 or 5 times faster than real-time, which reduces the applicability of the model to online analyses of multiple scenarios. For real-time applications the simulations for a near horizon (say, about 30

minutes) needs to be done say in a few seconds, **so** that alternative scenarios can be studied before any ATMIS advisory strategies such as route diversions are developed. The research had been successful in showing the potential of parallel processing to achieve such speed-up for real-time applications. However, one of the bottlenecks in achieving such parallel computing efficiency **is** the data management. This was recognized as a primary difficulty which needs to be addressed in further research.

Furthermore, the UCI-Caltrans ATMIS research testbed has several other components needed for online analyses, such as freeway control modules, incident detection algorithms, traffic management expert systems, network optimization algorithms, etc., which are deployed on a distributed computational platform using the CORBA architecture. Before these algorithms can be tested in the real-world, a simulation platform that replicates the real-world in the laboratory is needed, and DYNASMART is a part of the simulation workbench in the testbed. Once again, the studies requires careful data management in the simulation which would facilitate the flexibility of providing timely simulation capabilities of any of these modules, which may be incorporated into the various designs of ATMIS proposed to be studied in the testbed. A data management scheme that is essentially hard-coded inside the simulation program as Fortran or C arrays would be very cumbersome for this purpose and a object-oriented database with the simulation is considered a requirement.

Another significant aspect in real-time operations is the possibility of using on-line data **from** the road network to update the simulation parameters and data. For instance, if real-time counts on the streets are available, it can be used to update the dynamic trip tables used by the simulation, **as** the simulation proceeds. Here the database manager is required to ensure the integrity of the data, **as well as** to ensure that the data is shared by the parallel processors in a correct manner. Models for dynamic updating of O-D demand have been recently developed, but environments where these can be used for real-time simulations **are** still to be developed.

Another significant objective is to prepare for the upcoming field operational tests (FOT) and demonstrations in the near future. One such FOT would be for Dynamic Traffic Assignment (DTA) which is expected to commence within the next two years, **as** planned by the Federal

Highway Administration. DYNASMART has already been used as part of pilot off-line frameworks in this regard, but it was recognized that real-time data management will be a requisite capability for practical implementation of such frameworks. The research conducted here would be significant in helping Caltrans, PATH and UC Irvine in being able to develop future projects where the past efforts can be successfully brought to practical fruition. This is also significant in terms of the further productive use of the real-time ATMS research testbed at **Anaheim, California**, for field operational tests.

Finally, it has been the view of Caltrans that a simulation program such as DYNASMART cannot be used for practical applications, till it is properly validated and calibrated with real-life data. The database system will provide the perfect platform to accomplish this. The intention is that when coupled with real data, calibration/validation routines can be developed and added as processes in this environment, to self-tune the software with real data. This is a very significant benefit for the validity and more importantly, the transferability of the simulation software.

The literature in the area of traffic simulation models is well-known, and the UCI researchers have described this in their earlier reports, as well as in recent papers (Jayakrishnan et al., 1994). Suffices to say that none of the other well-known ATMS simulation packages such as INTEGRATION, NEMIS and THOREAU have flexible object-oriented data base platforms that achieve the above-mentioned objectives. The existing research literature in the area of object oriented databases is extremely vast, and it does not appear necessary to discuss the literature here. Research on the application of object-relational oriented databases for real-time traffic simulation is in its infancy, however. The earlier PATH research on developing such databases for IVHS (MOU-141) is possibly one of the most significant recent research on this topic.

## Chapter 2

### METHODOLOGY

In this section, we start with a brief description on the traffic simulation program. Then we describe the earlier research in parallel processing for traffic simulation, and follow it with a description of the research approach in this project and its methodological details. The details of designing a relational-database are provided in Chapter 3, which is then **followed** by the object-relational database we design in this project.

#### 2.1 The Simulation Framework (DYNASMART)

The ATMIS simulation program, DYNASMART (Dynamic Network Assignment Simulation Model for Advanced Road Telematics), was developed at UC Irvine and the University of Texas, Austin with support from the PATH program and FHWA. The simulation program is macroscopic in nature, but keeps track of individual vehicles and models the behavior **of** the drivers in response to ATIS, **as well as** the operation of the traffic control devices. The extensive capabilities of the model include:

- 1) Macroscopic modelling of traffic flow dynamics such **as** congestion formation and shock wave propagation. Tracking of locations of individual drivers.
- 2) Modelling of different traffic control strategies (freeways, surface streets, signalized intersections, **r a m p** entry/exit etc)
- 3) Modelling of prescriptive/compulsory guidance **as well as** non-prescriptive guidance with trip time information on alternative routes.
- 4) Modelling of various aspects **of** the controller such **as** infrequent updates of the network route information database.
- 5) Modelling of individual drivers' response to information in the case of descriptive guidance based on a set of paths rather than a single shortest path. Random assignment **of** driver behavioral characteristics. Flexibility to incorporate alternative behavioral rules.

- 6) Modelling of capacity-reducing incidents at any time, anywhere in the network.
- 7) Modelling of cases with only a fraction **of** the vehicles equipped for information.
- 8) Capability to carry out simulations based on externally **specified** dynamic equilibrium paths for drivers not equipped to receive information.
- 9) Several levels of output statistics for the system, for individual drivers **as well as** for groups of drivers (equipped drivers, unequipped drivers, drivers on certain O-D pairs etc). Statistics include average trip times, distances, average **speeds** and a variety of route switching statistics.
- 10) A user-friendly graphical interface that allows efficient editing of the network data and run-time display of the simulation.

More details on DYNASMART are available in Jayakrishnan et al (1990, 1993b, 1994). The basic traffic flow modelling approach, called macro-particle simulation, is described in Chang et al (1985). Even though the original model was written for sequential processors (mostly Sun work stations), the model is currently being rewritten for parallel computation on other platforms, **as** described in the next section.

## **2.2 Project Overview and Relationship to PATH MOU-170**

The objective of the earlier project (MOU-170) was to parallelize the DYNASMART traffic simulator in order to achieve near-realtime performance. The original DYNASMART program is written in FORTRAN and runs on a variety **of** platforms, primarily, UNIX. To port the original serial version of the DYNASMART program to parallel computers, we have made changes to **some** portions of the code without altering any semantic of the program. Basically the core of DYNASMART consists of two large loops for each simulation time step. In the first loop, new vehicles **are** generated for each **link of** the transportation network; in the meantime every vehicle already on any given link advances by the **same** distance, based on the **link's speed** during that time step (which in turn, depends on the **link density**). **We call** this loop the **FIRST LINK LOOP**. Moving each vehicle within a **link** is accomplished by a **small loop inside of** the FIRST LINK LOOP called the VEHICLE LOOP. In the second loop, called the SECOND LINK LOOP, each

vehicle that would cross any boundary between two links is taken care of. All vehicles crossing boundaries will be moved to the following link based on capacity restrictions and signal timings. **Since** a vehicle never appears in more than one link simultaneously, these three loops theoretically can be run in parallel.

However, there are still some data dependencies that need to be removed before the loop-level parallelism can be exploited. The data dependency problem is dealt with using either directives provided by the parallel FORTRAN compiler or by creating new data structures that avoid dependencies (*see* Wolfe and Banerjee, 1987). In the following discussion we point to our experience with the FORTRAN utilities on the Convex SPP1000 parallel machine. The LOOP\_PRIVATE directive was used to create multiple copies of a variable under the **same** name so that parallel loops can access them at the **same** time. After removing all potential data conflicts, the next step was to use the LOOP\_PARALLEL directive to force loops to be executed in parallel. Instances of a loop would be dispatched to different processors in forms of threads regardless of data dependencies. This is handled very carefully since any data dependency not being taken care of could create a hazard and crash the program. After careful investigation and extensive testing, we have eliminated all data dependencies that would otherwise handicap the parallelization of the program.

We have also tested another directive called CRITICAL\_SECTION. This directive will guard a segment of code (called a critical section) so that at any time only one thread can run it. This mechanism of serialization helps when dependencies are hard to remove. However, the performance of a program using CRITICAL\_SECTION is expected to be worse than other methods of dependency removal because the accesses to a critical section are serialized. This directive was only used for very limited occasions.

Another way to exploit parallelism is to have the machine automatically explore data dependencies and eliminate any data conflicts. This is achieved through the compiler directive PREFER\_LOOP. When encountering this directive the compiler tries to resolve all data dependencies by itself. If it succeeds, the program is translated into a parallelized version. Otherwise, it will keep the program

**serial.** According to our experiments, this directive never really succeeded. It can only deal with loops of very **small sizes** therefore was not able to **handle** the loop **sizes** in our program. We have not used this directive in our project.

### 23. Limitations of Current Approaches

In **DYNASMART**, some computational resources are unavoidably allocated to **parallel FOR** loops to perform **some** unproductive computations. For example, if there are **1000 links** and only **one** link needs to be computed, the program will loop over each link to make the only one useful computation. Even though the execution jumps to the next iteration very **soon** if there is nothing for to compute in the current iteration, certain wasted computation still occurs. Furthermore, if the computation load for each iteration of **a loop** varies significantly, it is hard to balance the workloads among all the processors. Even though we find it unusual for one processor to be extremely busy while the others are idle, we have to find ways to reduce any resource wastage and make the workload **as balanced as possible**.

As we mentioned before, data sharing has an important impact on the performance of **a program** running in parallel mode. Basically every process will contend for the memory resources. **As** the number of processes and data size grow, the simplistic memory management provided by the **DYNASMART will** not be sufficient. To be practical, we have to **look** for a mechanism which can accommodate large **size of** data and is intrinsically shared-based.

Another limitation is memory. Since the current program uses arrays extensively, the *size* of input cannot grow beyond a certain point. This prevents the program from being scaled to larger problems. Even though we can use dynamically allocated memory to replace these arrays, the **size** of the **main** memory is still limited.

## 2.4. Methodology of the Object-Relational Database System

**Based** on the above observations, we propose to incorporate an object oriented database (OODB) into **DYNASMART** to support its parallel/distributed execution. With such an environment, a number of concurrent processes can run simultaneously with a high degree of data sharing. **See** Figure 1.1 for a schematic diagram. With parallel database search, the synchronous parallel **FOR** loops that iterate on every object (i.e., link and vehicle) can be replaced by a number **of** processes driven by discrete events. For example, an event could be a vehicle entering a link or leaving a link. In this way, not **all** vehicles need to be processed every time step in each loop. The workload balancing and idle loop problems can thus be efficiently resolved.

In our database version of **DYNASMART** there would be a process serving **as** the vehicle generator. Links in the transportation network will be grouped into processes based on the number of processors available. Each process **has** internal data structures to store its states. Processes are awoken only if some triggering event arrives. This **minimizes** the overhead **of** process management.

Vehicles, on the other hand, are pure data and maintained by the database. Later **on** each vehicle is manipulated by the link processes during its lifetime and will be taken out of the system once it reaches its destination. The database will retrieve sets of data/information in response to the requests made by the processes. For example, the database can retrieve **all** vehicles that will cross link boundaries and direct them to the downstream **links**. A portion of computation resources is to be allocated to the database to facilitate parallel search. Further evaluation will be done during the research project to ensure that excessive computational overhead will not be introduced into the simulation due to vehicle processing, possibly resulting in refinements in this approach. Experience from past use **of** the object oriented databases in simulation is that large number **of** objects can in *some* cases slow down the simulation.

We note here that depending on how exactly the database is used for various purposes, it may be **more** theoretically rigorous to call it an “object-relational database”. The distinction arises based



essentially on whether the database is used for primarily storing the object-data, **as** and when the simulation reads and writes to the database, or whether it is used during the simulations to handle all object data. **Our** experience is that it is better to let the simulation's own internal data arrays handle many of the large number **of** objects such **as** vehicles, but to let the database handle other objects, such **as** networks, paths, links, etc, which are much **less** dynamic in nature. From this standpoint, **our** designs, when deployed act often **as** object-relational databases than **an** object-oriented database. For the purpose of the discussions in this report, however, we use both terms interchangeably.

**As** far **as** input/output is concerned, **we** can use the database to hold pre-generated input data to increase the efficiency of the simulator. Also, traffic performance information gathered during each time step, like congestion level and traffic loads, can be held in the database for further processing. **This** is especially helpful when the number of links and the *size* of data are large.

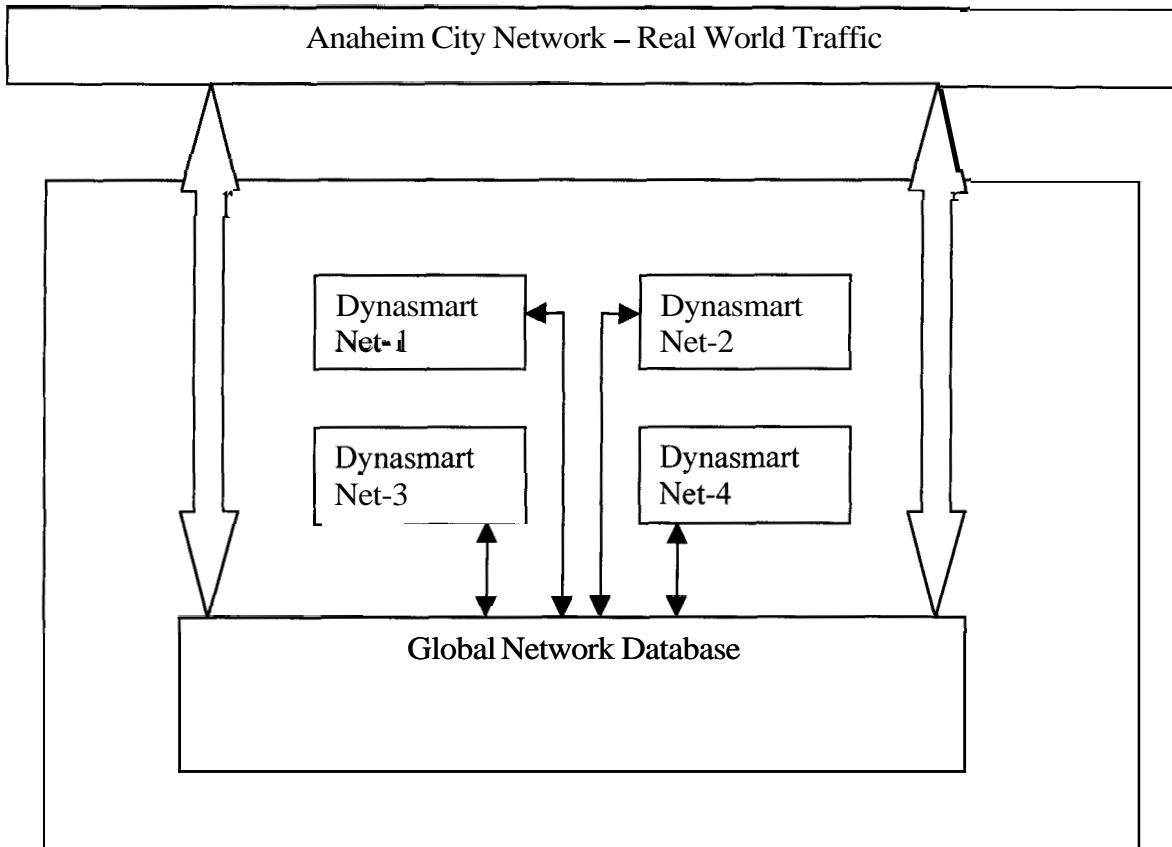


Figure 1.1. Schematic Diagram of the Proposed Database Management System for Fast Real-time Parallel Simulation of Traffic

## Chapter 3

### OBJECT-RELATIONAL DATABASE DESIGN FOR DYNASMART

In this chapter, we describe our design of the database. Section 3.1 discusses the design concepts. The discussion in section 3.1 is based on a pure relational database, without “objects”, for ease of explanation. The concepts are similar in object-relational databases and regular relational databases. We follow this in section 3.2 with our design of the database for DYNASMART.

#### 3.1. Introduction to the Fundamental Design Concepts (Relational Databases)

The discussion in this section draws heavily from Joe Garrick’s Database design primer (**Error! Bookmark not defined.**). We use the same simple example as in that reference to explain some of the primary design issues involved, rather than explain it on the basis of the object-relational database developed in this project (section 3.2) which is considerably more complex.

The relational database model has become the de-facto standard for the design of databases both large and small. While the concepts involved are not terribly complex, it can be difficult at first to get a handle on the concept.

The simplest model for a database is a “flat file”, which has only a single table which includes fields for each element to be stored. Nearly everyone has worked with flat file databases, at least in the form of spreadsheets. The problem with flat files is that they waste storage space and are problematic to maintain. Let’s consider the classic example of a customer order entry system. Assume that the data has to be managed for a company with customers, each of whom will be placing multiple orders. In addition, each order can have one or more items.

The data to be for each component of the application:

## Customers

- Customer Number
- Company Name
- Address
- City, State, ZIP Code
- Phone Number

## Orders

- Order Number
- Order Date
- PO Number

## Order Line Items

- Item Number
- Description
- Quantity
- Price

A flat file to represent this data would cause serious problems. Each time an order is placed, there is a need to repeat the customer information, including the Customer Number, Company Name, etc. What's worse is that for each item, we not only need to repeat the order information such as the Order Number and Order Date, but we **also** need **to** continue repeating the customer information as well. Consider the case of a customer who has placed two orders, each with four line items. To maintain this tiny amount of information, we need to enter the Customer Number and Company Name eight times. If the company should send you a change of address, the number of records to be updates is equal to the sum of product of orders and order line items. Obviously this will quickly become unacceptable in terms of both the effort required to maintain the data and the likelihood that at some point there will be data entry errors and the customer address will be inconsistent between records.

The solution to this problem is to use a relational model for the data. It simply means that in this example each order entered is related to a customer record, and each line item is related

to an order record. A relational database management system (RDBMS) is then a piece of software that manages groups of records which are related to one another. Let's take our flat file and break it up into three tables: Customers, Orders, and OrderDetails. The fields are just as they are shown above, with a few additions. To the Orders table a Customer Number field is added, and to the OrderDetails table an Order Number field is added. Here's the list again with the required additional fields and modified field names.

#### Customers

CustID

CustName

CustAddress

CustCity

CustState

CustZIP

CustPhone

#### Orders

OrdID

OrdCustID

OrdDate

OrdPONumber

#### OrderDetails

ODID

ODOrdID

ODDescription

ODQty

ODPrice

Note that some clever naming schemes are used here to avoid confusion while programming the database.

What is done here, besides the name change, is to add some new fields to the Orders and OrderDetails tables. Each have key fields used to provide a link to the associated Customers and Orders records, respectively. These additional fields are called foreign keys.

### 3.1.1. Primary and Foreign Keys

A key is simply a field which can be used to identify a record. In some cases, key fields are a part of the data that is being stored or derived from that data, but they are just as often an arbitrary value. For the Customers table above, one could use the company name as a key, but if there are ever two companies with the same name, the system would be broken. One could **also** use some derivation of the company name in an effort to preserve enough of the name to make it easy for users to derive the name based on the key, but that often breaks down when the tables become large. Simply using an arbitrary whole number may be the best solution. One can also completely hide the use of the numbers from the end users, or expose the data.

There are two types of key fields: primary keys and foreign keys. A primary key is a field that uniquely identifies a record in a table. **No** two records can have the same value for a primary key. Each value in a primary key will identify one and only one record. A foreign key represents the value of primary key for a related table. Foreign keys are the cornerstone of relational databases. In the Orders table, the OrdCustID field would hold the value **of** the CustID field for the customer who placed the order. By doing this, we can attach the information for the customer record to the order by storing only the one value

### 3.1.2. Queries

Rather than repeating the Customers table data for each Orders table record, we simply record a customer number in the OrdCustID field. By doing this, we can change the information in the Customers table record and have that change be reflected in every order placed by the customer. This is accomplished by using queries to reassemble the data. One of the inherent problems of any type of data management system is that ultimately the human users of the system will only be able to view data in two dimensions, which in the end become

rows and columns in a table either on the screen or on paper. While people can conceptualize objects in three dimensions, its very difficult to represent detail data in anything other than a flat table. This is accomplished using queries. A query is simply a view of data which represents the data from one or more tables.

### 3.1.3. Structured Query Language

Queries are built in a relational database using Structured Query Language, or SQL. This is the standard language for relational databases and includes the capability of manipulating both the structure of a database and its data. In its most common form, SQL is used to create a simple SELECT query. For the earlier example, an SQL for a query to look at customer orders could be:

```
SELECT CustName, CustCity, CustState, OrdDate  
FROM Customers INNER JOIN Orders ON  
Customer.CustID = Orders.OrdCustID;
```

This query starts with the SELECT keyword. Most of the are normally SELECT queries. Following the SELECT keyword is the list of fields. Next comes the FROM keyword. This is used to indicate where the data is coming from. In this case, its coming from the Customers table and the Orders table. The key to this query is the INNER JOIN. There are two basic types of joins which can be done between tables: inner joins and outer joins. **An** inner join will return records for which only the matching fields in both tables are equal. **An** outer join will return all the records from one table, and only the matching records from the other table. Outer joins are further divided into left joins and right joins. The left or right specifies which side of the join returns all records. The balance of the example query specifies which fields are used to join the table. In this case we are matching the CustID field from Customers to the OrdCustID field (the foreign key) in Orders. Note that this is a specific kind of SQL. Each **RDBMS** has its own particular dialect of SQL, but they all have similar characteristics.

#### 3.1.4. Referential Integrity

Consider what happens while manipulating the records involved in the order entry system. In the example above, editing the customer information can be done at will without any ill effects, but deleting a customer can cause problems. **If** the customer has orders, the orders will be orphaned. Clearly there cannot be an order placed by a non-existent customer, *so* there has to be a means in place to enforce that for each order, there is a corresponding customer. This is the basis of enforcing referential integrity. There are two ways that you can enforce the validity of the data in this situation. One is by cascading deletions through the related tables, the other is by preventing deletions when related records exist. Database applications have several choices available for enforcing referential integrity, but if possible, it is better to let the database engine handle this. The latest advanced database engines allow the use of “declarative referential integrity”, i.e., a relationship is specified between tables at design time, indicating if updates and deletes will cascade through related tables. If cascading updates are enabled, changes to the primary key in a table are propagated through related tables. If cascading deletes are enabled, deletions from a table are propagated through related tables.

#### 3.1.5. Normalization

Normalization is a subject that is extensively and often confusingly addressed in database literature. In a nutshell, its simply the process of distilling the structure of the database to the point where the repeating groups of data are moved into separate tables. In the above example, we have normalized customers and orders by creating a separate table for the orders. Careful normalization is needed to design a database to be efficient and reliable, and at times one may need to sacrifice normalization to practicality. When taken to extremes, there is indeed a performance penalty for excessive normalization. The problem is often the added overhead of additional joins, when the queries look for many details at once.



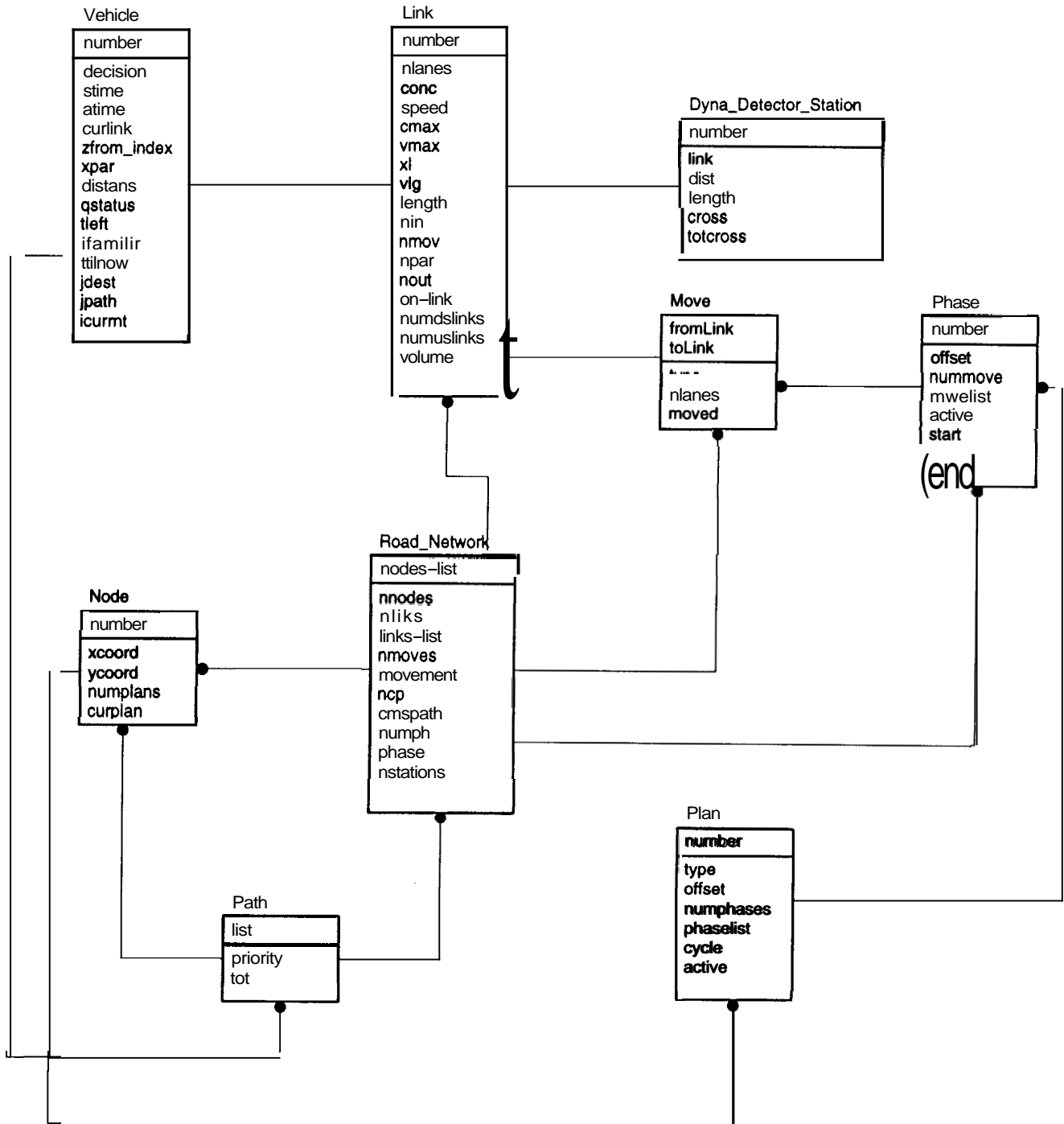
### 3.2. Object-Relational Database Design for DYNASMART

In this section we describe the Object-relational design developed to manage the data objects in the DYNASMART simulator. The design is developed such that only minimal modifications will be needed in the DYNASMART program code. For this reason, the **main** objects (which are essentially tables, **as** explained in the relational database primer in section 3.1) are as in the existing simulator code. The primary objects are,

- 1) Vehicle Object
- 2) Link Object
- 3) Node Object
- 4) Network Object
- 5) Path Object
- 6) Detector Station Object
- 7) Move
- 8) Phase
- 9) Plan

Each of these objects include several “data items” which are the variables associated with each object. The vehicle, **Link** and Node objects are self-explanatory. The Network Object is needed to handle the distributed operation of DYNASMART on multiple networks, i.e., the sub-networks being simulated by different DYNASMART processes. In a single simulator setup, this would be a single Network object, and is obviously easier to implement than in a distributed simulator case. In the distributed simulator case, the database acts **as** the global database for the complete network, and thus each subnetwork becomes a separate Network Object in the database. The plan, move and phase objects deal with the signalization at each node, and are split into various tables, according to the scheme used in the design of DYNASMART. Figure 3.1 in the next page shows the “relations” between these objects which is the key aspect behind the ‘object-relational database’. This database design easy to implement in Microsoft Access. How the MS-Access operates with the Simulator processes is explained in Chapter 5.

# Dynasmart Database



Dynasmart DB Objects

**Object Name: Dyna\_Detector\_Station**

Variable Name	Type	Description
Number	Integer	station number
Link	Integer	<b>link number on which station resides</b>
Dist	Integer	Location downstream end of detector (from upstream node of link)
Length	Integer	Effective length of detector
Cross	Integer	Number of vehicles that <b>crossed in</b> previous interval
Totcross	Integer	<b>total number of vehicle which have</b> crossed detector

**Object Name: Road-Network**

Variable Name	Type	Description
nnodes	Integer	number of nodes
nodes-list	String	list of nodes in the network
nlinks	Integer	number of links
links-list	String	list <b>of</b> links in the network
nmoves	Integer	number of link to link movements
movement	String	list of movements in the network
n <del>cp</del>	Integer	total number of cms paths in the network
cmspath	String	list of all the cms paths in the network
numph	Integer	total number of phases
phase	String	list of all phase in the network
nstations	Integer	number <b>of</b> detector stations

**Object Name: Link**

Variable Name	Type	Description
Number	Integer	the number of the link
Nlanes	Integer	number of lanes on the (main section of the link)
Conc	Float	current concentration
Speed	Float	current speed (mi/min)
Cmax	Float	max concentration on link
Vmax	Float	max (freeflow) speed on link
x l	Float	lane-miles on link
Vlg	Float	volume to be generated during current timestep
Length	Integer	The length of the link
Nin	Integer	The number of vehicles that entered this link during the timestep
Nmov	Integer	The number of vehicles that have left the link during the timestep
Npar	Integer	number of vehicles currently on link
Nout	Integer	number of vehicles which left the network from link
on-link	String	List of vehicles on the link
numdlinks	Integer	Number of links connecting downstream from this link
Numuslinks	Integer	Number of links connecting upstream form this link
Volume	Integer	

**Object Name: Node**

Variable Name	Type	Description
Number	Integer	number of node
Xcoord	Integer	x coordinate
Ycoord	Integer	y coordinate
Numplans	Integer	number of signal(control) plans available for node
Curplan	Integer	the current plan number used with node

**Object Name: Path**

Variable Name	Type	Description
List	String	node listing in path
Priority	Double	path choice rating variable
Tot	Integer	total number of vehicles which have selected path

**Object Name: Phase**

Variable Name	Type	Description
Number	Integer	phase number
Offset	Integer	
Nummove	Integer	number of moves, each approach
Movelist	String	list of all the movements in the phase
Active	Integer	
Start	Integer	
End	Integer	

**Object Name: Plan**

Variable Name	Type	Description
number	Integer	
type	Integer	
offset	Integer	
numphases	Integer	
phaseslist	String	list of all phases in the plan
cycle	Integer	
active	Integer	

**Object Name: Move**

Variable Name	Type	Description
fromlink	Integer	movement from link number
tolink	Integer	movement to link number
type	Integer	movement type
capacity	Integer	movement capacity available during current timestep ( number of vehicles)
nlanes	Integer	approximate number of lanes serving movement
moved	Integer	number of vehicles in the queue for this movement after the current timestep

**Object Name: Vehicle**

Variable Name	Type	Description
number	Integer	vehicle number
decision	Integer	number of routing decision vehicle has made
stime	Float	vehicles start time from origin
atime	Float	vehicles arrival time at destination
curlink	Integer	vehicles current link
zfrom_index	Integer	Index of vehicles origin zone
xpar	Float	vehicles distance from the downstream node on its current link
distan	Float	distance vehicle has travelled
qstatur	Integer	
tleft	Float	amount of timestep remaining after a vehicle reaches the end of its link
ifamiliar	Integer	vehicles familiarity level with network
ttilow	Float	time vehicle has spent travelling
jdest	Integer	number of vehicle destination
jpath	String	enumeration of vehicles path
icurrnt	String	list of vehicles location in its path

## Chapter 4

### FRONT-END DATABASE QUERY SYSTEM - *NetCompose*

In this chapter we describe the query system developed for the object-relational database of DYNASMART. While the database can be operational for purely computational purposes and flexibility to add/delete/compare data items, the user could substantially benefit from queries to the database while the simulation is running. For instance, if a given network is being simulated, and the real-world data from actual freeway detectors is coming to the lab (as at the UCI-Caltrans ATMIS research testbed), the analyst can visualize the simulation's performance in comparison with the real data. Alternatively, if the user wants to find more complicated items such as say the positions of all vehicles in the network which departed during certain time periods, a query front-end to the database is essential. Indeed, the benefits from an object-relational database is its ability to work with such structured queries, and a query system is developed in this research for this purpose. We call it *NetCompose*, as it evolved from the *Compose* database system developed in the electrical engineering department at UCI for studies with biological databases.

Section 4.1 describes the *NetCompose* database query system. This is followed by a brief section on the Network implementation architecture of *NetCompose*. Section 4.3 describes the core research development that makes the query system flexible and powerful, i.e., the active rule processing schemes, which are state-of-the-art in its theoretical foundations. This is followed by section 4.4 which discusses some related research issues, and the chapter concludes with several screen shots of the front-end system, which bring out the user-friendly and flexible nature of the system.

#### 4.1. Overview of *NetCompose*

*NetCompose* is our next-generation database query tool that is organized hierarchically and possess a logic and structure easily applied. It runs on top of any relational database (such as



Sybase, Oracle, MicroSoft SQL Server, and Informix) and provides an intelligent and complete object-relational interface to the user. Unlike the traditional approach, which is completely table driven, queries in *NetCompose* are structured along the lines of natural language and sentences. Objects (nouns) are identified, described (adjectives, predicates) and acted upon (verbs). Queries are composed by naive users based on simple multiple hierarchical choices without knowing any low level concepts such as “**join**” and “selection”. Suppose for example the investigator wishes to identify the vehicles which are located in the area which is defined by two-dimensional points and whose speeds are less than 50 miles. The query would consist of a noun (vehicles), two predicates (within a certain area, and having less than 50 miles speed) and a verb (find).

The *NetCompose* database management system controls accesses/changes of information based on integrity constraints (i.e., rules that have to be verified whenever the contents of the database are changed; e.g., **No** vehicle should run A if vehicle arrives to destination), triggers (i.e., rules that define the actions to be taken when certain conditions are satisfied; e.g., Let **me** know whenever vehicles with behavior pattern A also presents behavior pattern **B**), and security rules which are again constructed based on the semantic building blocks. In addition it allows the user to define complex building blocks or rules in terms of simpler ones without any knowledge of programming. Consequently even the most complicated queries or logic can be composed naturally and easily for non-expert users and managers.

A *NetCompose* database has a three-layer architecture, as shown in Figure 4.1. At the lowest level, the relation layer, information is stored as relations (tables). On top of relations, in the object layer objects are formed according to their structures. The top layer, the knowledge layer, stores the object schema, the object vocabulary (i.e., the basic predicates and deductive laws, i.e., rules that define higher-order predicates in terms of more primitive ones), integrity constraints, and triggers that can be expanded dynamically; they are stored internally in mathematical logic that is transparent **to** the user. It is the top layer that provides the object-oriented intelligent user interface and control. Conceptually, a query is first processed by the knowledge layer, which expands higher-level predicates into lower-level ones, checks the

impacts of the query to the database, and takes appropriate actions if certain conditions are satisfied (in the case of triggers) or informing the user whenever any data inconsistency is detected (in the case of integrity constraints).

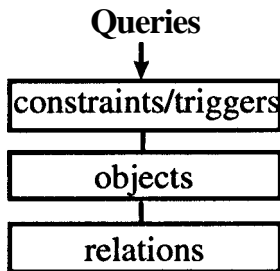


Figure 4.1. Architecture of *NetCompose*

As discussed, on top of the objects, *NetCompose* provides a high level query language in which object-oriented queries are expressed in the following form:

*declaration of object variables (inform  $\alpha$  object var: domain)*

*object-oriented operation(s)*

*where object-oriented predicates*

Note that an object-oriented action can be a set-oriented operation such as adding an element, deleting an element, or replacing an element; it may be any operation that is associated with the object involved. As a simple example, the query of finding those cases which present similar pattern of behaviors with vehicle “v00001” is expressed formally in *NetCompose* as follows:

*range of va is vehicle*

*range of vb is vehicle*

*retrieve (va)*

where  $vb.id = "v00001"$  and  $similar\_in\_behavior(va,vb)$

Including procedural methods is problematic to conventional relational query optimization techniques. For a large database, an optimal nested-loop algorithm **is** inefficient when the number of variables involved in a query is large. Realizing this, a non-linear search approach based on query decomposition is taken in the database. In this research, query processing is optimized based on an extended query decomposition algorithm we have developed to minimize the number **of** function calls. Specifically, before an object-oriented predicate is evaluated, all the inputs are instantiated or dissected because the values of input arguments are needed for evaluation.

#### 4.2. *NetCompose* Network Architecture

Figure 4.2 shows how the *NetCompose* technology fits in the Internet/Intranet architecture.

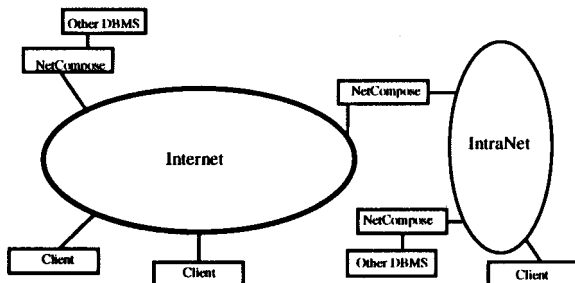


Figure 4.2. *NetCompose* and Internet/Intranet

### 4.3 Active Rule Processing in *NetCompose*

In *NetCompose*, the database server continuously monitors the triggers and integrity constraints. The constraints and triggers, which are expressed as logical rules and organized into a set of networks, are evaluated based on incoming events.

In *NetCompose*, five types of primitive events are defined:

**Message Events:** Point in time when a message is arriving at an active object and the point in time when the object has finished executing the appropriate method requested by the message.

**Value Events:** Point in time when the value of an object is being modified

**Time Events:** Absolute points in time (e.g., 22:00:00, Feb. 28, 1995), periodically reappearing events (e.g., every hour), or relative to occurring events (e.g., one min after event *E1*).

**Transaction Events:** Defined by the beginning or termination of (user-defined) transactions.

**Abstract Events:** Events defined by users and applications according to their specific semantics.

The following composite events can be specified given two (composite or primitive) events *E1* and *E2*:

**(*E1*|*E2*):** Occurs when either *E1* or *E2* occurs.

**(*E1*,*E2*):** Occurs when *E1* and *E2* occur, regardless of the order.

**(*E1*;*E2*):** Occurs when *E1* and afterward *E2* occurs.

**(~*E1*):** Occurs when *E1* does not occur in a specified (named) transaction or in a predefined time interval.

In general, processing of production rules or integrity constraints can become a serious performance bottleneck if we evaluate each rule from scratch every time the contents of the database are changed. To solve this problem, we take an incremental approach which

compiles each rule into a rule network. When a rule is first evaluated, all intermediate results are saved instead of being purged. Subsequently, whenever the database is changed, the changes are input to the network to trigger other changes in the rule network. Thus only the part of the rule network that is affected by the database change needs to be re-computed. In addition, heuristic rules are employed to merge rules which have common sub-expression to avoid duplicated efforts. The network approach is similar to the RETE algorithm ([1][7]) but is more general in treating logical formulas, events, and structured objects.

In our approach, integrity constraints are converted into production rules as well so that both integrity constraints and triggers can be treated in the same way. Specifically, given a set of constraints  $\{[E1]F1 \rightarrow R1, \dots, [En]Fn \rightarrow Rn\}$ , where  $Ei$  is a formula describing events and both  $Fi$  and  $Ri$  are formulas describing the database state, each constraint  $Fi \rightarrow Ri$  is converted into the form  $Fi \wedge \neg Ri \rightarrow \text{warning}()$ .

Thus, in *NetCompose*, rules are in general expressed in the following form:

$$L1 \wedge L2 \wedge \dots \wedge Ln \rightarrow R$$

where  $L1, L2, \dots, Ln, R$  are predicates which can be in one of the two forms:

*class\_name(X)*, where *class-name* is the name of a class and  $X$  is an variable. This predicate asserts that  $X$  is an instance of the *classclass-name*, **or**

*predicate\_name(arg1, ..., argN)*, where each argument can be a variable (which are expressed in the form  $X$  or  $X.attribute.attribute\dots$ ) **or** a constant (integer, float, **or** a string in the form of "text"). This asserts that when applying the predicate *predicate-name* on  $arg1, \dots, argN$ , the result is true.

Following are some example rules:

$rectangle(obj_0) \wedge rectangle(obj_1) \wedge seq(obj_0, "A") \wedge intersect(obj_0, obj_1) \rightarrow rotate(obj_1)$   
 $patient(obj_0) \wedge patient(obj_1) \wedge patient(obj_2) \wedge similar(obj_0, obj_1) \wedge similar(obj_0, obj_2) \wedge ieq(obj_0.age, 65) \rightarrow plot\_profile(obj_2, 10, "mytext")$

Now we define a rule network as a directed graph that consists of a set of nodes, where each node corresponds to a predicate. There are three types of nodes: source nodes, operational nodes, and terminal nodes. Nodes whose corresponding predicates are of the form *class\_name(arguments)* are "source nodes". Source nodes do not have any input arcs. A source node interfaces with the database and generates a set of objects of the same class for further processing. A node which does not have any output arcs are "terminal nodes"; it consumes its input objects and do not generate any output objects. Finally, a node which is not a source node or a terminal node is called an "operational" node. In general, if an operational node *N* has a set of input arcs from the nodes *N1*, ..., *Nk* and has an output arc going to the node *Nc*, it means the object sets produced by the *Ni*'s are input to *N* so that *N* can perform some computation and produce a set of objects which will be taken by *Nc*.

Now given a production rule of the form  $[E1]L1 \wedge L2 \wedge \dots \wedge Ln \rightarrow R$ , we can construct a rule network as follows:

Let  $A = \{L1, \dots, Ln\}$

For each *Li* of the form *class\_name(obj\_x)*, create a source node *Nx*.  $A = A - \{Li\}$ .

Let  $Ax = \{Li \mid Li \in A \text{ whose arguments have one and only one variable } obj\_x\}$ . Create an operational node *Nx'* and create an arc from *Nx* to *Nx'*.  $A = A - Ax$ .

Let  $Axy = \{Li \mid Li \in A \text{ whose arguments have two and only two variables } obj\_x \text{ and } obj\_y\}$ . Create an operational node *Nxy* and create an arc from *Nx'* to *Nxy* and an arc from *Ny'* to *Nxy*.  $A = A - Axy$ .

Let  $Axyz = \{Li \mid Li \in A \text{ whose arguments have three and only three variables } obj\_x, obj\_y, \text{ and } obj\_z\}$ . Create an operational node *Nxyz* and create an arc from (*Nx'*, *Nyz*), (*Ny'*, *Nxz*),

$(Nz', Nxy)$ , or  $(Nx', Ny', Nz')$ , whichever is applicable, to  $Nxyz$ . If two or more candidates exist, then the choice is arbitrary.  $A = A - Axyz$ .

Repeat the above process for predicates involving **4** or more variables with the **same** principle until  $A$  becomes an empty set.

Create a terminal node  $Nt$ . Create an arc from each node created in steps **2-6** which do not have any output arc. The terminal node  $Nt$  performs the computations involved in  $R$ .

Store in each node its corresponding predicate and the associated arguments.

The rule network functions as follows:

In general, if **an** operational node  $N$  has a set of input arcs from the nodes  $N1, \dots, Nk$  and has an output arc going to the node  $Nc$ , it means that the object sets produced by the  $Ni$ 's are input to  $N$  **so** that  $N$  can perform **some** computation and produce a set **of** objects which will be taken by  $Nc$ .

Each source node corresponding to a class class-name retrieves from the database the set of objects of class-name and sends a copy of the set to each operational node connected to it via **an** arc.

For each operational node, perform the operations involved in the corresponding predicates whenever the data set on each of its input arc is ready. **Assume** the operation node has  $n$  input arcs whose corresponding object sets are  $A1, \dots, An$ , respectively, the result produced by the operational node will be a subset of  $A1 \times A2 \times \dots \times An$ .

The terminal node performs the operations involved in its corresponding predicate whenever all data sets on the input arcs are ready.

Since the nodes of a rule network form a partial order, we can associate a logical formula with each node as follows:

Each source node is associated with its corresponding predicate. Assume the operation node has  $n$  input arcs which are connected to nodes  $A1, \dots, An$ , whose associated formulas are  $L1, \dots, Ln$ , respectively. Also assume that the operation node was created based on  $m$  predicates  $P1, \dots, Pm$ . Then the operation node is associated with the formula  $f = L1 \wedge L2 \wedge \dots \wedge Ln \wedge P1 \wedge \dots \wedge Pm$ .

As an example, given the following production rule, the resulting rule network is shown in Figure 4.3, where the source nodes are shaded and the terminal node is designated by an oval.

On E1:  $\text{class\_1}(X) \wedge \text{class\_2}(Y) \wedge p1(X,Y) \wedge p2(Y) \rightarrow R(X)$

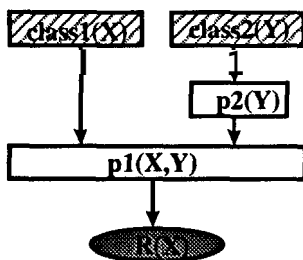


Figure 4.3. A rule network

To avoid duplicated efforts, two networks can be merged into one according to **some** heuristics. The following are some example heuristic rules:

The networks corresponding to two identical rules can be merged into one. Consider two production rules  $[E1] F1 \rightarrow R1$  whose corresponding network is  $N1$  and  $[E2] F1 \wedge F2 \rightarrow R2$  whose corresponding network is  $N2$ . In this case  $N2$  can be built as shown in Figure 4.4.



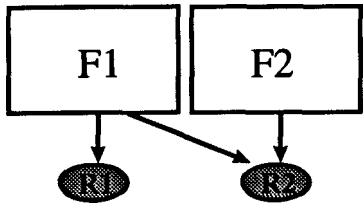


Figure 4.4. A merged rule network

When several rules are merged into one network, for each rule of the form  $[E] F \rightarrow R$ , where  $E$  is an event qualifier, we associate each node involved in making the formula  $F$  the event qualifier  $E$ . Consequently, given a set of rules, each node of the merged rule network may be associated with several event qualifiers. These qualifiers play an important role when the rule network is evaluated. Once a merged network is built, it is evaluated based **on** the incoming events. Specifically, when an event arrives, all the nodes which are associated with the event **become** active (Note that these nodes can be found by checking the event modifiers associated with each node). All the active nodes **of** the network forms a sub-network and the sub-net is evaluated.

#### 4.4. Related Issues on Rules in Database Query Systems

The idea of incorporating rules into a database system has exist as integrity constraints and triggers **as** early as in CODASYL, in the form of ON conditions. More recently, the idea of combining rules and data has received much serious consideration. The term “active database” has been used frequently in referencing such database. For example, rules has been built into POSTGRES [10]: there is no difference between constraints and triggers; all are implemented **as** a single rules mechanism. In addition, POSTGRES allows queries be stored **as a** data field **so** that it is evaluated whenever the field is retrieved. In HiPAC [4], the concept of Event-Condition-Action (ECA) rules was proposed. When an event occurs, the condition is evaluated; if the condition is satisfied, the action is executed. It can be shown that ECA rules **can** be used to realize integrity constraints, alters, and other facilities. Rules have

also been included in the context of object-oriented databases. In Starburst, for example, rules can be used to enforce integrity constraints and to trigger consequent actions.

On the other hand, there has been growing interest in building large production systems that run in database environments. The motivation for these are two. First, expert systems have made an entry into the commercial world. This has brought forth the need for knowledge sharing and knowledge persistence, These are features found in current databases. Secondly, many emerging database applications have shown the need for **some** kind of rule-based reasoning. **This** is one of the principle features of expert systems. Production systems is a commonly used paradigm for the implementation of expert systems. The confluence of needs from the areas of AI and database has made the study of database productions very important. The research conducted in this project has thus relevance to other areas, as well.

Traditionally production systems have been used in AI, where data are stored in main memory. Various needs, as mentioned above, have lead production systems designers to use databases for data storage. We refer to these **as** database production systems (DPS). Commercial DBMS's do not have the necessary mechanisms **to** provide full support for such systems. Views can be used in lieu of rules, but only in a limited way. Work has been reported for more powerful mechanisms to handle a large class of rules [2] [3]. However, the focus has been on retrieval, especially evaluating recursive predicates, and proposed approaches do not handle updates as in systems like OPS5 and HEARSAY-11. [5] [6] [9] [11] have addressed this issue, and much attention has been placed on parallelizing the evaluation of production systems (see, e.g., [8]). To our knowledge, little effort has been made for production systems that work on objects or distributed evaluation **of** production systems.

Next we provide some screen shots to demonstrate the graphical front end developed for *Netcompose*, which facilitate its use in conjunction with DYNASMART.

## 4.5. Examples of the use of *NetCompose* to Query Traffic Data

Figure 4.5. User selection object(s) from ObjectChooser.

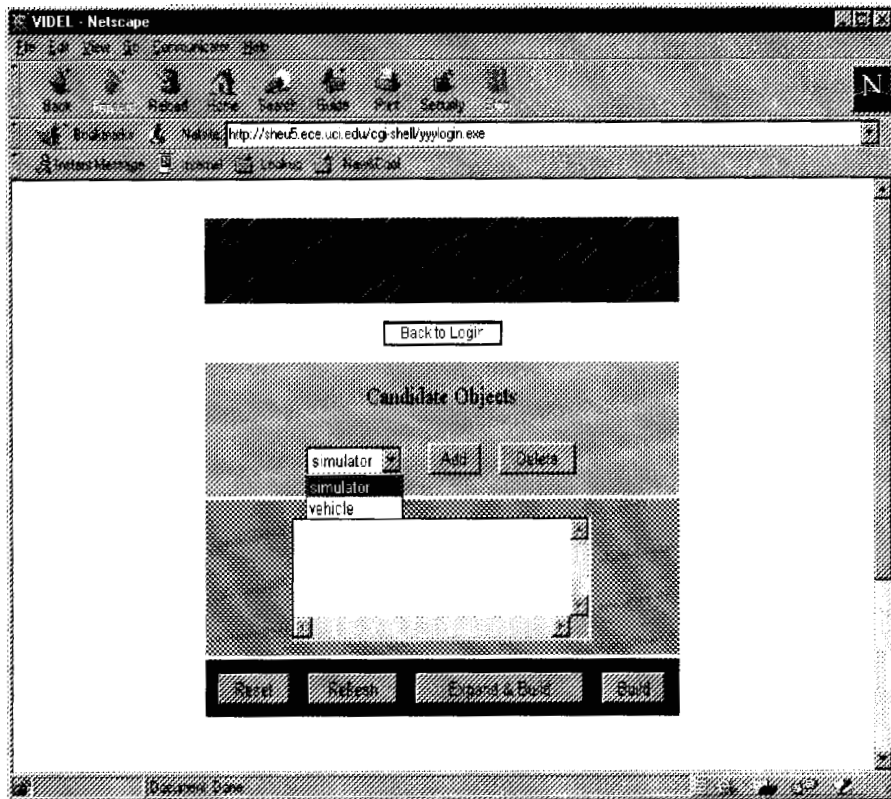


Figure 4.6. Initialization of simulator from the database query system.

Object Expander - Netscape

Back Forward Home Stop Reload Print

http://sheu.eca.uic.edu/cgi-bin/xyexpand.exe

Internet Message Name Lookup Newsgroup

[Back to Object Chooser]

**Query Builder legends**

Name Mark Function Relationship Attribute Expression

---

**CONTROL** [Objects] [Actions] [Relations]

[Reset] [Add Object] [Submit]

---

**OBJECTS** [Control] [Actions] [Relations]

simulator [simulator] [dynamsoft]

---

**ACTIONS** [Control] [Objects] [Relations]

show object [simulator]

simulate [simulator] time step [5]

initialize simulation [simulator]

append object [simulator] [doe@dba/sheu/sheu]  including all component objects

delete object [simulator]  including all component objects

report [Report]

---

<input type="checkbox"/> Summary <input type="checkbox"/> Print (default) <input type="checkbox"/> Bar Graph <input type="checkbox"/> Pie Chart	Grouped by	COUNT	SUM
	<input type="checkbox"/> simulator simulatorid	<input type="checkbox"/> simulator simulatorid	<input type="checkbox"/>
	AVG	MAX	MIN
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Select		Ordered by	
<input type="checkbox"/> List <input type="checkbox"/> simulator simulatorid		<input type="checkbox"/> simulator simulatorid	

---

**RELATIONS** [Control] [Objects] [Actions]

Relational operators

<input type="checkbox"/> > <input type="checkbox"/> < <input type="checkbox"/> >= <input type="checkbox"/> <= <input type="checkbox"/> != <input type="checkbox"/> =	<input type="checkbox"/> [simulator simulatorid] OR <input type="checkbox"/> [value] (enter value)
---	--

negate [Add] [Reset]

Figure 4.7. Posted Query on selected vehicle(s)

Object Expander - Netscape

Back Forward Home Stop Back Print Search

http://ns.ece.uci.edu/cgi-bin/objexpander.exe

Internet Messages Internal Links Broadcast

**Back to Object Chooser**

**Query Builder legends**  
 Noun Verb Function Relationship Adjective Property

**CONTROL** Objects Actions Relations

Reset Build Query Submit

**OBJECTS** Control Actions Relations

vehicles

The vehicles are located in the area defined by X-max  and Y-min  Y-max

X-max  and Y-min  Y-max

vehicle\_id  xrc  yrc

time\_step  stage  destination

currentlink  duration  move\_speed

resettable  link  taken\_off

**ACTIONS** Control Objects Relations

show object

visualize

append object  to   including all component objects

delete object   including all component objects

report   including all component objects

Summary

Grouped by	COUNT	SUM
vehicle.vehicle_id	vehicle.vehicle_id	vehicle.xrc
vehicle.yrc	vehicle.yrc	vehicle.yrc
vehicle.time_step		vehicle.time_step
AVG	MAX	MIN
vehicle.xrc	vehicle.yrc	vehicle.xrc
vehicle.yrc	vehicle.yrc	vehicle.yrc
vehicle.time_step	vehicle.time_step	vehicle.time_step

Pivot default

Bar Graph

Pie Chart

List

Select:

vehicle.vehicle_id	vehicle.vehicle_id
vehicle.xrc	vehicle.xrc
vehicle.yrc	vehicle.yrc

Ordered by:

vehicle.vehicle_id	vehicle.vehicle_id
vehicle.xrc	vehicle.xrc
vehicle.yrc	vehicle.yrc

**RELATIONS** Control Objects Actions

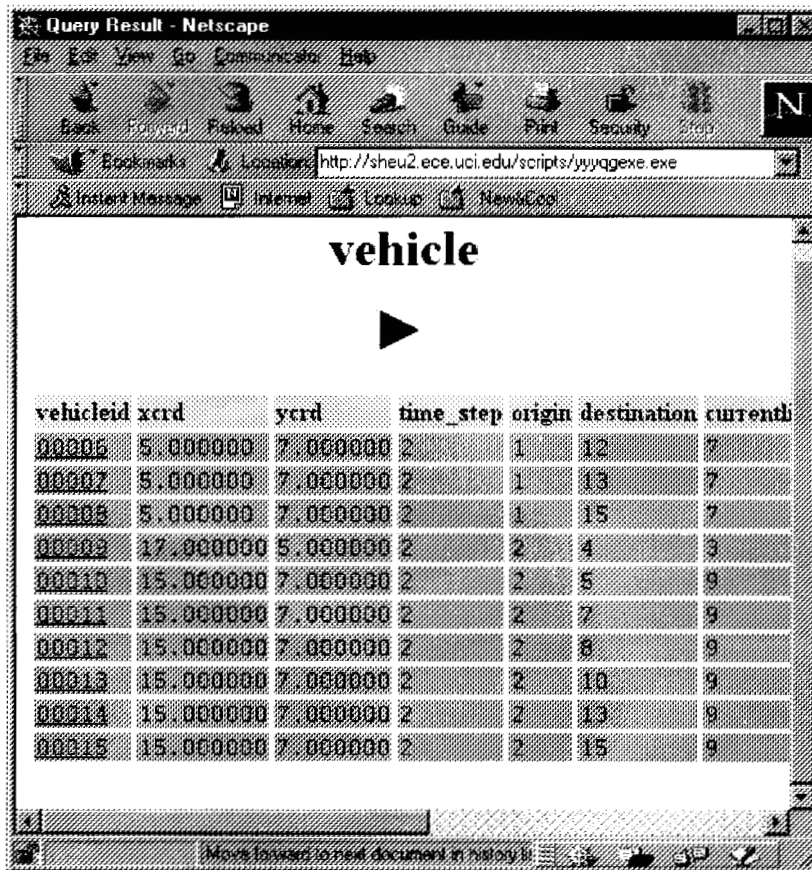
Relational operators

OR

(enter value)

negate

Figure 4.8. Query Result (at vehicle generation point)



The screenshot shows a Netscape browser window titled "Query Result - Netscape". The address bar contains the URL "http://sheu2.ece.uci.edu/scripts/jyqgexe.exe". The main content area displays the word "vehicle" in a large font, followed by a right-pointing triangle. Below this is a table with the following data:

vehicleid	xcrd	ycrd	time_step	origin	destination	currenth
00006	5.000000	7.000000	2	1	12	7
00007	5.000000	7.000000	2	1	13	7
00008	5.000000	7.000000	2	1	15	7
00009	17.000000	5.000000	2	2	4	3
00010	15.000000	7.000000	2	2	5	9
00011	15.000000	7.000000	2	2	7	9
00012	15.000000	7.000000	2	2	8	9
00013	15.000000	7.000000	2	2	10	9
00014	15.000000	7.000000	2	2	13	9
00015	15.000000	7.000000	2	2	15	9

Figure 4.9. Query Result (Specific vehicle)

Output HTML - Netscape

Back Forward Home Search Stop Print Security

Backmarks Netscape | U2.ecs.ucl.edu/User/dba/onimi/vehicle00001\_11961.html

Instant Message | News | Look Up | Mail/Cal

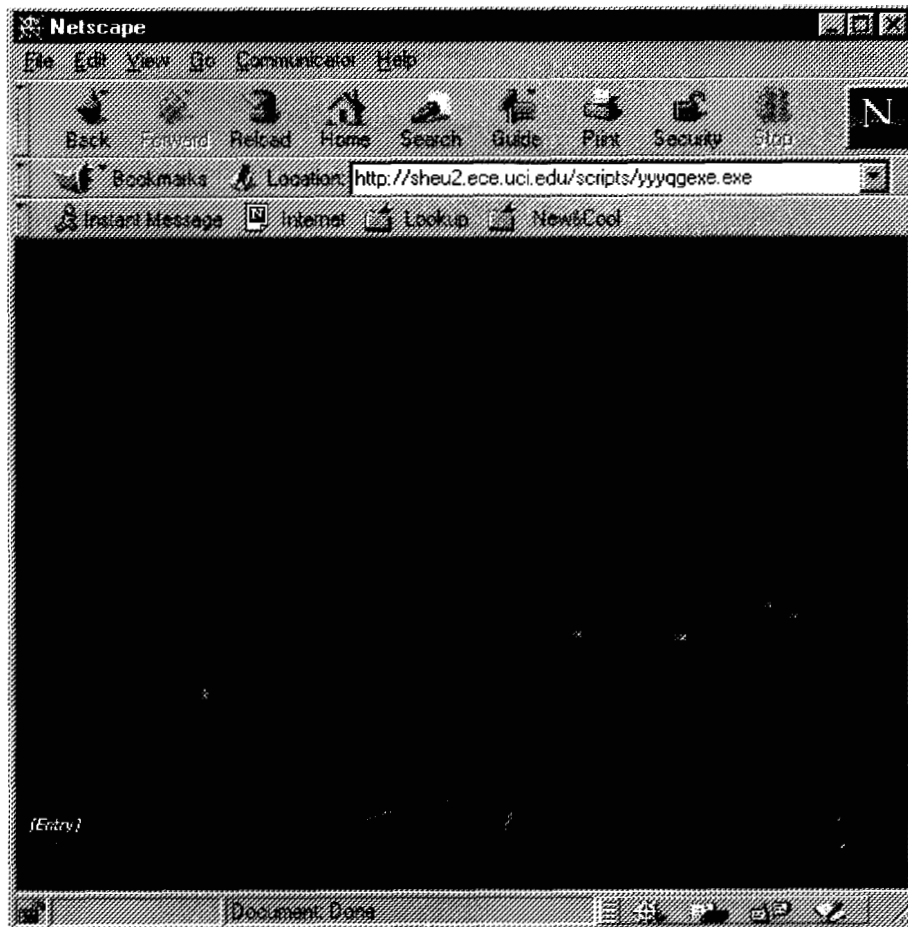
## vehicle properties

database	object	primary key	attribute	value
dba	vehicle	00001	year	8.000000
			weight	5.000000
			time (days)	3
			origin	1
			destination	2
			location	1
			location ref	2
			max speed	1.000000
			max number	1
			links	1
			links ref	0

Revise  
 Drop  including all component objects  
 Save in Server   including all component objects

Browser - Done

Figure 4.10. Visualization of vehicle positions (3-D view, Network **links** not shown)





The visualization capabilities of *NetCompose* are significant. The graphical visualization front end is currently written in Java and can show the vehicle positions in the network. In fact, the java program is general enough for it to be easily modified to show other details such as locations of signals, freeway detectors, etc. The Figure 4.10 above shows the vehicles positions in the network in a 3-dimensional figure and it is easy to pan, **zoom**, and change the viewpoint in the network. This is a very significant capability from the user's perspective, as it can show the areas in the network where congestion develops, as well as other details such as entry ramp queues, etc. It is also reasonably easy to make modifications to include other details such as intersections, however, several graphical "primitives" will need to be created for such purposes. The above picture does not show the links in the network, for instance, as we have not yet created the basic link icons.

If the database is expanded in the future to include further data items in addition to the DYNASMART data items, then the above-shown visualization capabilities can be extended to show other items, as long as they are items with **x** and **y** coordinates on the network.

## Chapter 5

### SOFTWARE INTEGRATION

In this chapter, we describe the essential details of the Distributed CORBA platform in the testbed, which is used to bring in the real-time data, and make it available for comparisons with the simulation data stored in the object-relational database.

#### 5.1. MS-Access Database System on PCs

The object relational database is designed for implementation on a PC platform running Windows 95 or Windows-NT, with MS-Access database. The research has originally focussed on the database implementation on the ORACLE platform, but found that the overhead computations involved did not justify its use. The MS-Access database communicates through ODBC that is available as standard under Windows-95 for communication with the database. The ODBC provides the ability to send data from a process on PC, communicating with a DYNASMART process (or processes) running on UNIX systems.

The communication between multiple machines (running various subnetwork DYNASMARTs) to the PC process (which in turn communicates through ODBC to the database) can be easily set up in the Caltrans-UCI ATMIS research testbed using the CORBA platform for inter-processor communication and process distribution. The following sections describe the CORBA platform that exists in the testbed. This is **also** important, **as** other modules in the Testbed may in the future utilize an extended form of the Object-relational database developed in this project, since the *NetCompose* query system can then be valuable **as** a single front-end to the whole testbed network traffic analysis. We do not describe the details of the systems programming needed to set up MS-Access and ODBC, as they are internal to the database architecture. We do, however, describe the CORBA platform in the testbed, **as** it is significant in future uses of the database as explained above.

## 5.2. CORBA/C++ Software at the Caltrans-UCIATMIS Research Testbed

### 5.2.1. Prerequisities for building CORBA clients

The following software components must be installed on a PC connected to the private LAN at UCI TMC before client programs which communicate with CORBA objects at D12 can be built at UCI; (1) Windows NT **4.0** (2) Microsoft Visual C++ 97 (3) Iona Technologies' Orbix v2.3c for Windows NT/95, and **(4)** Iona Technologies' OrbixNames software. Software component **(4)** is freely downloadable at <http://www.iona.com/>.

### 5.2.2. Building and executing CORBA clients

*Makefiles* are provided in each of the CORBA client directories. Open the MSDOS window under Windows NT and simply type *nmake* in the appropriate directory. This command reads the *Makefile*, invokes the Microsoft Visual C++ compiler and compiles all the CORBA client source files in the directory. After compilation, the makefile directs the Microsoft Visual C++ linker to link the generated object modules with the *Orbix* and *OrbixNames* libraries.

Once a ".EXE" file is generated, it is ready for execution. Note that the PC has to have proper communication facilities set up *so* that the CORBA clients may invoke the CORBA objects at D12 TMC. If communication facilities are not set up properly, the CORBA client execution will fail. Note that in order to execute just the CORBA client software, the Orbix daemon *orbixd* need not be executed in the local client node.

### 5.2.3. Description of CORBA clients

All the CORBA clients currently available at UCI are based on the original CORBA clients provided to UCI by NET via the D12 website <http://tctdb03/download>. The NET document *Testbed Intertie And Intranet* provided to UCI by NET explains the basic functionality of

various CORBA objects available at the D12 TMC server. The reader is strongly recommended to read through that manual before continuing any further. Note that while the client source files available at the D12 web site are HP-UX 10.20 version, they have been adapted to a Windows NT/Microsoft VC++ environment here at UCI.

#### 5.2.3.1 LDS

This version contains the CORBA/C++ source files for receiving the raw LDS data here at UCI from the D12 TMC server. When the client program is executed, it will request the user to enter a freeway name, direction, and two post-mile boundaries. Upon entering this information, the program will attempt to receive the raw LDS data on all post-mile locations between the ones on the freeway chosen by the user. Besides displaying the data on the screen, it is also saved into a local file named "lds.log". (Source File Directory: D:\users\atms\lds)

#### 5.2.3.2 VDS

This version contains the CORBA/C++ source files for receiving the VDS data here at UCI from the D12 TMC server. When the client program is executed, it **will** request the user to enter a freeway name, direction, and two post-mile boundaries. Upon entering this information, the program will attempt to receive the raw LDS data on all post-mile locations between the ones on the freeway chosen by the user. The process will be repeated once roughly every 30 seconds until the user stops the client by pressing the CTRL-C key. Besides displaying the data on the screen, it is also saved into a local file named "vds.log". (Source File Directory: D:\users\atms\vds)

#### 5.2.3.3 RMS

This version contains the CORBA/C++ source files for receiving and sending meter rates to the field. This version is very similar to the **RMS** client version available at the D12 web site

**http://tctdb03/download.** I have merely ported the **HP** version available at the web site to a Windows NT environment. Currently, this client attempts to first read the current metering rates on all post-mile locations on freeway 91W. Next, it attempts to read the current metering rates at all post-mile locations in Orange County. (Source File Directory: D:\users\atms\rms)

#### 5.2.3.4 CCTV

This version contains the CORBA/C++ source files for choosing CCTV cameras at a specific post-mile location. When the client program is executed, it will request the user to enter a freeway name, direction, two post-mile boundaries, and a camera number (either # 1 or # 2). Upon entering this information, the program will attempt to select cameras one by one between the post-mile locations chosen by the user. After selecting a location, the user has the ability to say whether he/she is satisfied with the camera image. If the user is not satisfied, the program will attempt to choose the camera from the next post-mile location which falls within the boundaries chosen by the user. (Source File Directory: D:\users\atms\cctv)

#### 5.2.3.5 CMS

This version contains the CORBA/C++ source files for reading CMS's from the field. This version is very **similar** to the CMS client version available at the D12 web site **http://tctdb03/download.** I have merely ported the **HP** version available at the web site to a Windows NT environment. Currently, this client attempts to first read the current CMS's on all post-mile locations on freeway 91W. Next, it attempts to read the CMS's at all post-mile locations in Orange County. (Source File Directory: D:\users\atms\cms)

### 5.2.3.6 DecodedLDS

This version contains the CORBA/C++ source files for receiving the raw LDS data here at UCI from the D12 TMC server and then decoding it into meaningful lane-by-lane volumes and occupancies. When the client program is executed, it will request the user to enter a freeway name, direction, and two post-mile boundaries. Upon entering this information, the program will attempt to receive the raw LDS data on all post-mile locations between the ones on the freeway chosen by the user. It will then decode the raw byte data into meaningful lane-by-lane volumes and occupancies. Besides displaying the raw data and the decoded data on the screen, it is also saved into a local file named "lds.log" (Source File Directory: D:\users\atms\decoded\_lds)

### 5.2.3.7 Decoded LDS and RMS

This *multithreaded* version contains the CORBA/C++ source files for receiving the raw LDS data here at UCI from the D12 TMC server, decoding it into meaningful lane-by-lane volumes and occupancies, using this decoded data and a simple metering algorithm to calculate the metering rates, and then attempting to send the data back to D12 and subsequently to the field. When the client program is executed, it will request the user to enter a freeway name, direction, and two post-mile boundaries. Upon entering this information, the program will attempt to receive the raw LDS data on all post-mile locations between the ones on the freeway chosen by the user. It will then decode the raw byte data into meaningful lane-by-lane volumes and occupancies. After this, the program attempts to calculate metering rates for these post-mile locations and then attempts to send back the rates to the field via the D12 RMS CORBA server. The raw LDS data and the decoded LDS data are saved into a local file named "lds.log". (Source File Directory: D:\users\atms\decoded\_lds\_and\_rms)

## Chapter 6

### RESEARCH CONCLUSION

This project developed an object-relational database, initially designed for operating with a distributed set of DYNASMART simulators (working on subnetworks), but to be extended in the future to include other modules in the Caltrans-UCI ATMIS research testbed. There are several motivations for developing such a capability, the most significant being the ability to integrate the data management requirements during distributed simulations in a seamless and flexible manner.

The UCI-Caltrans ATMIS research testbed has several other components needed for online analyses, such as freeway control modules, incident detection algorithms, traffic management expert systems, network optimization algorithms, etc., which are deployed on a distributed computational platform using the CORBA architecture. Before these algorithms can be tested in the real-world, a simulation platform that replicates the real-world in the laboratory is needed, and DYNASMART is a part of the simulation workbench in the testbed. Once again, the studies requires careful data management in the simulation which would facilitate the flexibility of providing timely simulation capabilities of any of these modules, which may be incorporated into the various designs of ATMIS proposed to be studied in the testbed.

Another significant aspect benefit from the database is in storing “mirror-data”, i.e., from the real-world as well as from the simulated-network, for easy comparisons on the performance of the simulations, as well as to facilitate the online calibration of the simulator. This is an important part of the “consistency checks” currently being developed in DTA (Dynamic Traffic Assignment) models.

The research conducted here would be significant in helping Caltrans, PATH and UC Irvine in being able to develop future projects where the past efforts can be successfully brought to practical fruition. This is also significant in terms of the further productive use of the real-time ATMS research testbed at **Anaheim**, California, for field operational tests.

Finally, it **has** been the view of Caltrans that a simulation program such as DYNASMART cannot be used for practical applications, till it is properly validated and calibrated with real-life data. The database system will provide the perfect platform to accomplish this. The intention is that when coupled with real data, calibration/validation routines can be developed and **added as** processes in this environment, to self-tune the software with real data. This is a very significant benefit for the validity and more importantly, the transferability of the simulation software.

The main conclusions from the research projects are:

- 1) **An** object-relational database is useful for simulations and for general hybrid simulation/heal-world ATMIS testbeds.
- 2) Several components of the simulation, such as vehicles, **links**, nodes, paths, signal phases, movements, and subnetworks can all be designed as objects in the database with their own associated data values, and with specified relations to other such objects.
- 3) It is possible to design the database in an efficient manner for it to provide easy query capabilities during simulations, using straightforward structured rules.
- 4) The database system implemented on a PC platform using standard MS-Access databases suffices for the purposes of research studies, and based on our experience, the effort and expenses involved in using “higher-strength” databases such as **ORACLE** may not be advisable for most purposes in traffic analysis, especially when very dynamically changing data values are involved.
- 5) The front end query system developed in this project is rather advanced in its capabilities to **use** structured rules to group and extract object and data values from the simulator. This was indeed a significant theoretical and academic research contribution resulting from this research.
- 6) The *Netcompose* database query system can provide answers to complex questions into the state of the simulation or the results after simulation. **An** example would be a question such as, “show the speeds of all vehicles which entered the network during a



given period, at a give time”, or “show the speeds on all **links** of a given path”. Such capabilities extends the practical usefulness of simulation programs significantly.

- 7) Visualization capabilities developed for the database front end are ideal for traffic analysis purposes, **as** vehicle movements and other dynamic data changes in the database can be visualized using Java-based graphical screens.
- 8) The database system is on a PC-based platform communicating through ODBC to outside, and in the case of the Caltrans-UCI research testbed, can communicate to multiple DYNASMARTs or other modules, through the COMA process distribution platform.

This research project, however, does not develop a product that is “completed and delivered”, but rather a product that satisfies the ambitious objectives when the project started. The research is an ongoing effort, and even though this PATH project (MOU-227) researched and developed several components, it needs further testing which **will** be continued **as** part of the Caltrans-UCI ATMIS research testbed plans. Specifically, the researchers wish that the real-world hardware/software connections and the **COMA** platform were fully implemented and understood by the testbed researchers a few months earlier **so** that a more complete evaluation of the performance of the database in real-time simulations could be completed. As such, this report does not report extensive evaluations, which will be completed as part of the Testbed research program and reported in the future.

## **ACKNOWLEDGEMENT**

The researchers wish to that Caltrans and PATH for the generous support for this research. We also wish to thank Prof. Wilfred Recker, Director of the Institute of Transportation Studies at UCI for facilitating the use of the labs, as well as for the constant encouragement.

## REFERENCES

- [1] Anoop Gupta, Charles Forgy, Allen Newell, and Robert Wedig, "Parallel Algorithms and Architectures for Rule-based System," *Proc. ICPP*, 1986.
- [2] Bancilhon, F., and R. Ramakrishnan, "**An** Amateur's Introduction to Recursive Query Processing," *Proc. ACM-SIGMOD International Conference on Management of Data*, Washington, D.C., 1986.
- [3] Chakravarthy, U.S., and J. Minker, "Multiple Query Processing in Deductive Database," *Proc. 12th International Conference on Very Large Databases*, Kyoto, Japan, 1986.
- [4] Dayal, U., Blaustein, B., Buchmann, A., Chakravarthy, U., et. al. "The HiPAC Project: Combining Active Databases and Timing Constraints," *ACM SIGMOD Record*, 17, 1, March, 1988, pp. 51-70.
- [5] Delcambre, L.M.L, J.N. Etheredge, "The Relational Production Language: **A** Production Language for Relational Databases," *Proc. Second International Conference of Expert Database Systems*, 1988.
- [6] Eick, C., J. Liu, P. Werstein, "Integration of Rules into a Knowledge Base Management System," *Proc. First International Conference on Systems Integration*, Morristown, April, 1990
- [7] Forgy, Charles L., "RETE : **A** Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence* 19, September 1982.

- [8] Ishida, T. "Parallel Rule Firing in Production Systems", *IEEE Transactions on Knowledge and Data Engineering*, pp 11-17, March, 1991.
- [9] Raschid, L., T. Sellis, C.-C. Lin, "Exploiting Concurrency in a DBMS Implementation for Production Systems", *Proceedings of the 1st International Symposium on Databases in Parallel and Distributed Systems*, 1988.
- [10] Rowe, L., and Stonebraker, M., "The POSTGRES Data Model," *Proc. VLDB*, 1987, pp. 83-96.
- [11] Sellis, T., et. al., "Implementing Large Production Systems in a DBMS Environment: Concepts and Algorithms," *Proceedings of ACM-SIGMOD*, 1988.
- [12] Chang, G-L, Mahmassani, H. S., and Herman, R. (1985); "A Macroparticle Traffic Simulation Model to Investigate Peak-period Commuter Decision Dynamics", *Transportation Research Record 1005*, pp. 107-120.
- [13] Jayakrishnan, R. and H. S. Mahmassani (1990); "Dynamic Simulation-Assignment Methodology to Evaluate In-vehicle Information Strategies in Urban Traffic Networks", proceedings of the *1990 Winter Simulation Conference*, New Orleans, Dec. 1990.
- [14] Jayakrishnan, R., Cohen, M., Kim, J., Mahmassani, H. and Hu, T-Y. (1993b); "Simulation Framework for the Analysis of Urban Traffic Networks Operating Under Real-time Information", PATH Final Research Report **UCB-ITS-PRR-93-25**.
- [15] Jayakrishnan, R., Mahmassani, H. S., and Hu T-Y (1994); "An Evaluation Tool for Advanced Traffic Information and Management Systems in Urban Networks", *Transportation Research, Part-C*.

- [16] Sheu, P. C-Y. and Yoo, S. B. (1990); "A Knowledge-based Software Environment (KSBE) for Designing Concurrent Processes," *International Journal of Human-Computer Interactions*, Vol. 1, No. 2, pp. 161-185, 1990.
- [17] Wolfe, M., and U. Banerjee (1987); "Data Dependence and Its Application to parallel Processing," *International Journal of Parallel Programming*, Vol. 16, No. 2, 1987, pp. 137-178.
- [18] Yoo, S., Yu. M. and Sheu, P. C-Y. (1993); "Concurrency Control in Deductive Databases and Object Bases," *International Journal of Data and Knowledge Engineering*, Vol. 9, pp. 223-240