Precursor Systems Analyses of Automated Highway Systems

RESOURCE MATERIALS

HiVal: A Simulation and Decision Support System for AHS Concepts Analysis

U.S. Department of Transportation
Federal Highway Administration
Publication No. FHWA-RD-95-104
February 1995

FOREWORD

This report was a product of the Federal Highway Administration's Automated Highway System (AHS) Precursor Systems Analyses (PSA) studies. The AHS Program is part of the larger Department of Transportation (DOT) Intelligent Transportation Systems (ITS) Program and is a multi-year, multi-phase effort to develop the next major upgrade of our nation's vehicle-highway system.

The PSA studies were part of an initial Analysis Phase of the AHS Program and were initiated to identify the high level issues and risks associated with automated highway systems. Fifteen interdisciplinary contractor teams were selected to conduct these studies. The studies were structured around the following 16 activity areas:

(A) Urban and Rural AHS Comparison, (B) Automated Check-In, (C) Automated Check-Out, (D) Lateral and Longitudinal Control Analysis, (E) Malfunction Management and Analysis, (F) Commercial and Transit AHS Analysis, (G) Comparable Systems Analysis, (H) AHS Roadway Deployment Analysis, (I) Impact of AHS on Surrounding Non-AHS Roadways, (J) AHS Entry/Exit Implementation, (K) AHS Roadway Operational Analysis, (L) Vehicle Operational Analysis, (M) Alternative Propulsion Systems Impact, (N) AHS Safety Issues, (O) Institutional and Societal Aspects, and (P) Preliminary Cost/Benefit Factors Analysis.

To provide diverse perspectives, each of these 16 activity areas was studied by at least three of the contractor teams. Also, two of the contractor teams studied all 16 activity areas to provide a synergistic approach to their analyses. The combination of the individual activity studies and additional study topics resulted in a total of 69 studies. Individual reports, such as this one, have been prepared for each of these studies. In addition, each of the eight contractor teams that studied more than one activity area produced a report that summarized all their findings.

Lyle Saxton
Director, Office of Safety and Traffic Operations Research and Development

NOTICE

TABLEOFCONTENTS

TABLE OF CONTENTS (Cont.)

LIST OF TABLES

LIST OF FIGURES

1. EXECUTIVE SUMMARY

At the start of the AHS (Automated Highway Systems) PSA (Precursor Systems Analysis) program, the need for computational tools that can

- Provide integrated, system-level analysis for alternative AHS system concept evaluation that would incorporate and build on the PSA analysis of issues and risks

- Preserve and make accessible the software and database products of PSA and other AHS researchers to support future elements of the National AHS research agenda, such as the NAHSC (National Automated Highway System Consortia)

was recognized. In support of these needs, a prototype integrated modeling, simulation, and decision support testbed, called HiVal, has been developed. HiVal provides simulation, decision support, and database tools for evaluating alternative AHS concepts. HiVal accommodates a range of user expertise and objectives, ranging from high-level, aggregate AHS performance metrics and tradeoffs, to low-level, detailed simulation of individual AHS subsystem elements. HiVal is a computing environment that integrates a variety of simulations, models, and databases from both PSA activities and the broader AHS research community. More than a dozen simulations and models have been incorporated into the prototype system, as illustrated in Figure 1-1. HiVal uses a modular, distributed client-server computing architecture. Modern workstation technology (Unix & X-Motif, DOS/MS Windows, DCE, TCP/IP) is used throughout to support a wide variety of modeling and simulation needs, and allow for continued system growth. All of HiVal's basic interfaces, protocols, and control software uses COTS (Commercial Off The Shelf) technology.

The *first* important facet of HiVal is its ability to support AHS studies and analysis today using the functionality and software modules now contained within the system. However, the HiVal development concept recognizes that AHS simulation, modeling, and decision support elements are continually evolving and improving. Any "static" software system would quickly be rendered obsolete by the development of new analysis tools built after the system is completed. The HiVal system addresses the problem in the *second* important facet of the system: its flexible, expandable computing infrastructure. Using a distributed, client-server architecture, HiVal's infrastructure facilities the integration of new subsystem simulations, databases, and decision support

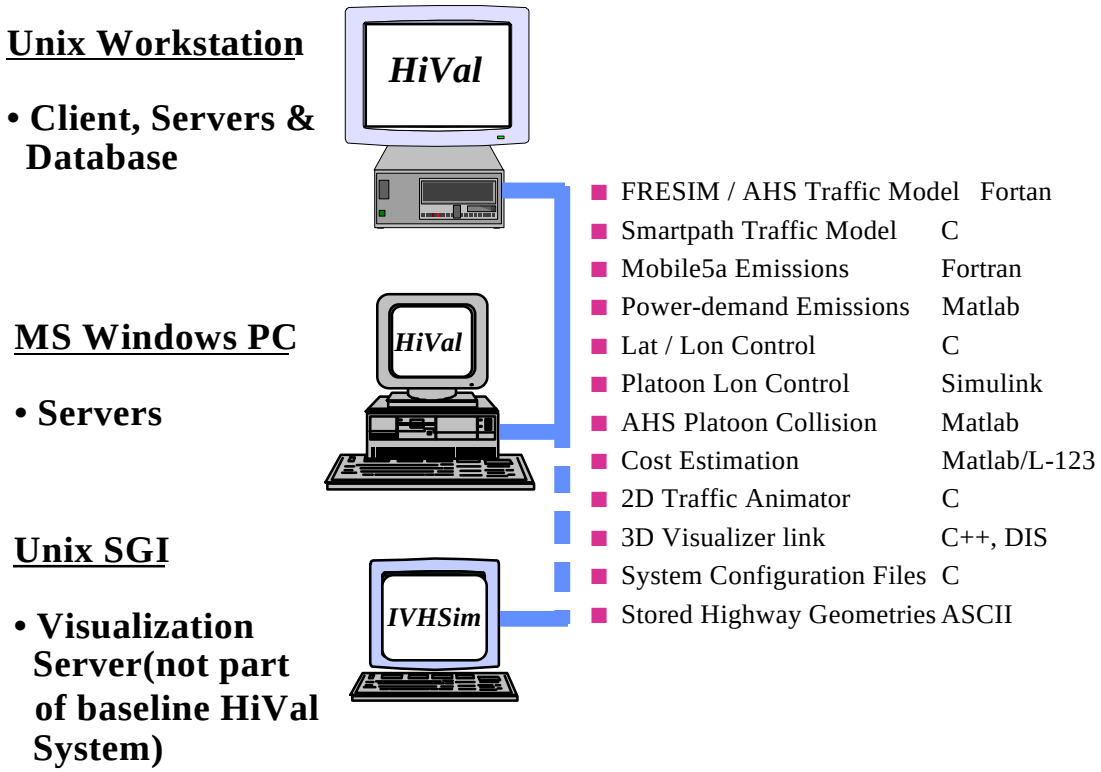

Figure 1-1 The Prototype HiVal System

modules. New elements can be added and combined with existing useful elements to *allow the software to develop incrementally* and remain at the AHS analysis state-of-the art.

While new models to capture emerging aspects of AHS modeling will still need to be developed (e.g. an improved operational effectiveness model), the HiVal architecture allows them to fit within its existing computing infrastructure. By taking advantage of the HiVal infrastructure, it will not be necessary to build a new system to use new models. HiVal integration infrastructure works on three levels:

- ***Hardware level:*** TCP/IP ethernet LAN provides compatible low-level communication protocols among different hardware platforms (e.g. PCs, workstations, supercomputers

- ***Software level*** : Software wrappers allow existing (legacy) simulation and modeling code, written in a variety of languages, to be seamlessly integrated. Wrappers generally eliminate the need to rewrite existing code

- ***Analytic level*** : User-defined pre- and post-processing functions assure that subsystem models and simulations are correctly connected to allow them to function together. This assures that elements that were not explicitly designed to work together are linked so that their inputs and outputs are compatible, and ensures that models and simulations are connected in ways that are consistent with the theoretical assumptions of each element.

In addition to providing a framework for operational effectiveness, environmental, safety, and cost analysis, the HiVal prototype has software “hooks” that can be used to provide sophisticated 3-D interactive AHS concept visualization. These connections make use of DIS (Distributed Interactive Simulation) technology, originally developed for US military applications. HiVal’s current uni-directional DIS connections permit the AHS community to take advantage of powerful 3-D visualization and animation tools available today within a large existing DIS software infrastructure. In addition, expanded DIS connections can be used to directly link HiVal’s traffic modules to human-operated driving simulators for concept evaluation and human factors studies. Figure 1-2 gives an example of a DIS-based visualization of AHS platoons simulated by FHWA’s FRESIM/AHS traffic simulator. These DIS connections to HiVal and visualization tools were developed under an activity that was coordinated with, but separate from, the AHS PSA research project.

Figure 1-2 DIS Visualization of FRESIM/AHS Platoons

2. INTRODUCTION

The HiVal prototype has been developed as one of the project in the PSA activity area "other", because of a scope that encompasses the outputs of most of the other activity areas. Building on the PSA issues and risks analysis and products, HiVal provides simulation, decision support, and database tools for evaluating alternative AHS concepts. HiVal development occurred in coordination with studies in the other sixteen activity areas in FHWA's Precursor Systems Analysis program. A proof-of-concept development of the simulation and modeling testbed, called HiVal, is the output of this initial research. An ultimate goal of HiVal is to provide a system which can integrate a variety of inputs from AHS systems analysis activities (both PSA and others, past and future) and provide an AHS-community-wide tool for issues, risks, tradeoff analyses, and analytic decision support as FHWA progresses towards the Congressionally-mandated 1997 demonstration with the NAHSC, and into the next century. The overall HiVal concept is illustrated in Figure 2-1.

As AHS development and evaluation expanded under PSA AHS activities, the number of individual AHS-subsystem-level models, databases, and simulations developed by researchers and available to HiVal increased significantly. In order to evaluate AHS system-level performance metrics (such as throughput, or emissions, or cost) based on the combination of multiple system components, there is a need to integrate individual analyses into a complete system evaluation. In addition, unique or large-scale models, simulations, or databases that are of importance to the entire AHS system analysis community can be made accessible through HiVal over an extended wide area network. The models included in the HiVal system span the range from detailed low-level models (e.g. lat/lon control, platooning scheme, entry/exit implementation), through traffic simulations of selected AHS scenarios (e.g. shared vs. segregated highways; grouped vs. individual vehicles), to system level measures (e.g. safety, environmental, productivity) to be used for analytic decision support. The HiVal system demonstration focused on the integration of existing models to provide system-level analyses and AHS performance metrics (MOEs). Modularity of system components and standardization of simulation and database interfaces are important design requirements for the HiVal system. Figure 2-2 summarizes the major functional elements that make up the HiVal system design.

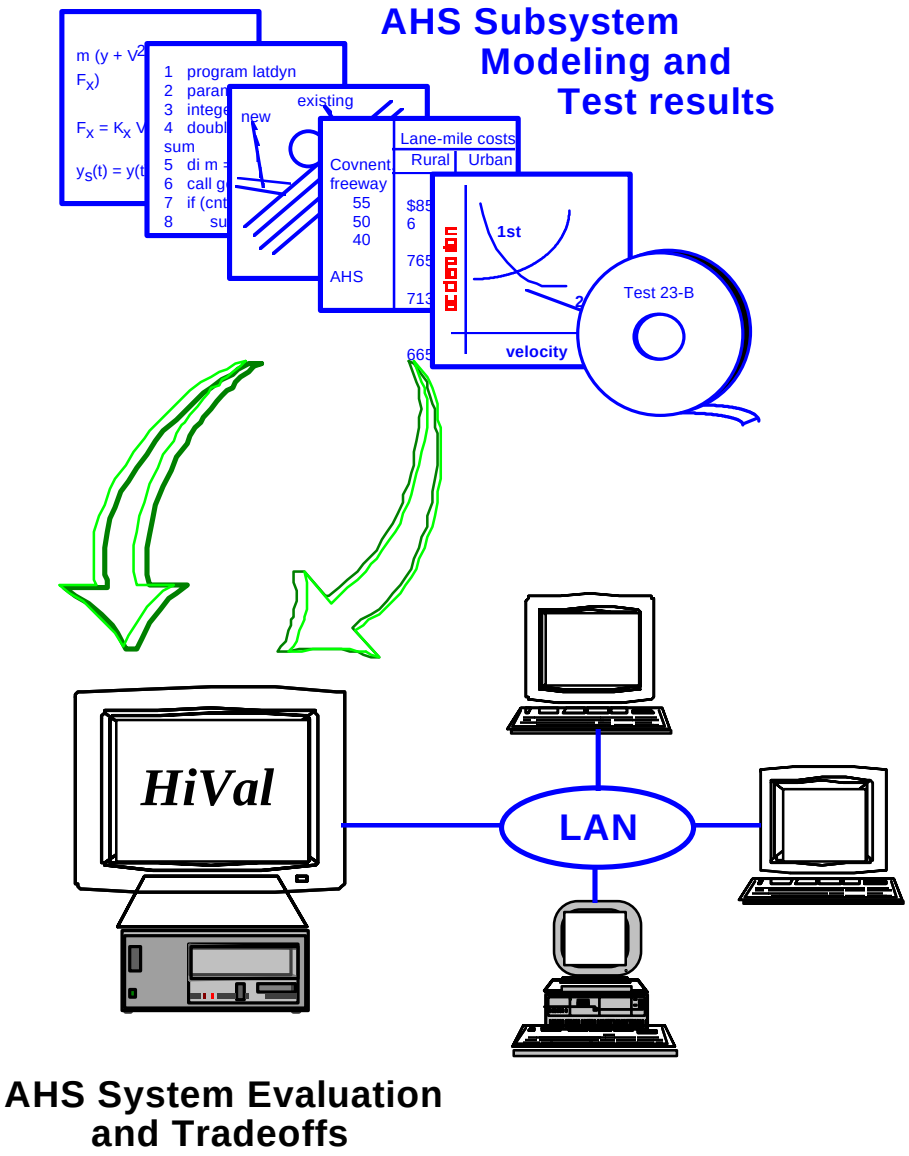

Figure 2-1 HiVal Concept for Integration and Synthesis

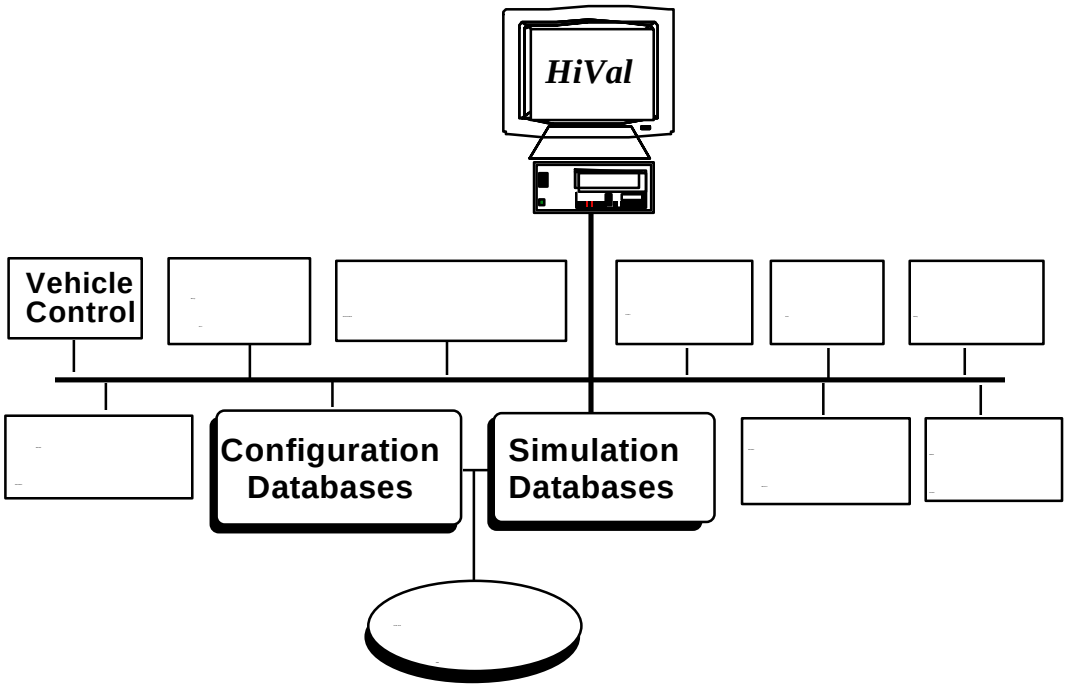

Figure 2-2: The HiVal System for Concept Integration, Evaluation, and Collaboration

3. REPRESENTATIVE SYSTEM CONFIGURATION (RSC)

The scope of this PSA activity did not include development of an independent RSC set. Instead, the thirteen RSC developed in Ref. 1 were used to define the database of system configurations used by HiVal. These RSC were a widely-used, all-encompassing set that evolved during the PSA program. They are defined in terms of infrastructure characteristics, vehicle intelligence, and communications and control schemes. The description of this RSC set was augmented based on related PSA Program work to include additional descriptors in order to help in mapping it to alternative RSC sets under development.

4. TECHNICAL DISCUSSION

A key challenge faced in implementing HiVal is the task of integrating disparate models which operate with varying levels of fidelity for different aspects of an AHS. HiVal takes advantage of defense technology transfer in working these integration problems, in particular the extensive work accomplished in distributed simulation of integrated military systems. The design elements of the overall HiVal system include:

- Using an adaptively-structured hierarchy of linked, functional modules
- Developing an open architecture that allows "plug and play" of results (and eventually full simulations themselves) for each module function

- Specifying required parameters for the input / output links of each module and developing interface and simulation standards for current and future elements

- Defining as possible "transfer functions" that describe simplified functional relationships (e.g. composite MOE) between input and output parameters of modules

- Providing networked, distributed access to simulations and databases, including databases of field tests and demonstrations

- Offering connections for driving simulators and 3-D visualizat ion tools.

The objective of the HiVal proof-of-concept demo wass to prepare a demonstration of a selected subset of the overall HiVal system, focusing on a selected set of models and/or functions. While the demonstration concentrates on selected components (that in fact include most of the major system elements) of the overall system, functional requirements analysis and high-level system design for the complete system have been performed. This approach recognizes the constraints placed on implementation by the fact that many PSA/AHS methods and resources were available only late in the program. By pursuing two parallel paths of developing a thread of HiVal for existing AHS models and databases, while at the same time using the insights gained from this process to begin the high-level design of the overall HiVal system, maximum progress toward HiVal's goals was achieved. The design and implementation work was accomplished in four main task activities: 1) General Functional Requirements Analysis; 2) Standards and Interface Requirements; 3) System Architecture Specification; and 4) Prototype System Development.

4.1 *General Functional Requirements Analysis*

This activity defined the scope of the HiVal system, analyzed the requirements for such a system, and identified modeling and database resources to support these requirements. Consultations with members of the broad AHS community took place to help in identifying simulation requirements, modeling assets, and measures of effectiveness for use in decision support. As part of the AHS PSA program, a report discussing general requirements for simulation and decision support for AHS was prepared late in the program (Ref. 2). Using that framework, functional requirements identified during the HiVal project and capabilities implemented in the HiVal prototype system are described in Tables 4-1 through 4-3. As the charts illustrate, there is a wide range of functional requirements for an AHS simulation and decision support system, and the current HiVal system design meets nearly all these requirements.

TABLE 4-1: General Simulation and Decision Support System Functional Requirements

GENERAL CHARACTERISTICS AND FUNCTIONS	HiVal DESIGN AND IMPLEMENTATION
1. Well-structured approach to concepts evaluation is needed	1. HiVal system has been designed explicitly to support structured, consistent, efficient comparative evaluations
2. Decision support system for evaluation of AHS concepts	2. Provides decision support, MOEs, and evaluation of alternative AHS concepts
3. NAHSC will be on a constrained schedule and budget	3. HiVal prototype has already been designed and built
4. Modify existing simulations and models to meet needs	4. HiVal incorporates existing models that have been modified for AHS (e.g. SMARTPATH, FRESIM/AHS, 2-D and 3-D Animators)
5. Required integration of models may be complex	5. HiVal's client server, remote procedure call linkage paradigm provides an integration method and architecture that has been successfully applied to AHS
6. Use a phased implementation	6. HiVal has implemented a flexible architecture which is easily scalable and facilitates phased development.
7. Will become a design tool to support new concept development and respond rapidly to weakness of a concept and so form and evaluate new concepts	7. HiVal's ability to "plug and play" multiple models of any type (e.g., alternative control models) provides "on-the-fly" reconfiguration to evaluate new concepts. Implemented HiVal architecture also makes it easy to add new models needed to evaluate new concepts

TABLE 4-2: Specific Simulation and Decision Support System Functional Requirements

SPECIFIC CHARACTERISTICS AND FUNCTIONS	HiVal DESIGN AND IMPLEMENTATION
1. Integrated models, simulations, analyses, and prototype test data	1. HiVal architecture is designed to achieve this integration from. Modular client/server architecture allows test data and models to be used interchangeably
2. Ensures that data regarding all alternative concepts are objectively generated and presented in manner to assure fair evaluation	2. HiVal system provides common access to models and databases, and analysis results are presented in consistent displays
3. Concept evaluation should be based on specific system configurations	3. HiVal incorporates selectable PSA representative system configurations (RSC) in its model network construction logic
4. Performance profiles will be generated	4. HiVal produces the low-level results and MOEs that are combined together to produce aggregate performance profiles for different needs
5. Be objective and balanced and ensure validity of data generated	5. HiVal facilitates comparable, quantitative analyses, uses already-validated simulations and models, and allows easy cross-comparison of MOEs computed using different simulations
6. Easily calibrated	6. HiVal uses already-validated simulations and models, and allows easy cross-comparison of results at multiple levels of detail
7. Clear presentation of data -- easily understood and comparable	7. HiVal design stresses unified, consistent displays of MOEs and simulation outputs, including graphic, tabular, and 2-D/3-D animation

TABLE 4-2: Specific Simulation and Decision Support System Functional Requirements (Continued)

SPECIFIC CHARACTERISTICS AND FUNCTIONS	HiVal DESIGN AND IMPLEMENTATION
8. Accommodates all viable concepts -- has flexibility to blend/reconstitute concepts to form and evaluate new ones	8. HiVal's structure is designed to accommodate multiple models, supporting different concepts, that the analyst can "plug and play" for evaluation. HiVal's ability to interchange models makes it easy to flexibly accommodate new concepts
9. Use requirements-based evaluation criteria and accommodate increasingly refined levels of definition	9. HiVal's modular, hierarchical network structure allows individual high-level modules to be replacesd by sub-networks of increasingly refined models as they are defined
10. Accommodate evaluation criteria metrics -- be flexible and accommodate sensitivity and tradeoff analyses	10. HiVal and its central results database can produce any evaluation metric the user specifies with minimal development. HiVal provides full access to all model input parameters and presents unified graphic output formats for consistent sensitivity and tradeoff analyses
11. Provides profiles for each user view	11. Incorporated using HiVal's post-processor and MOE display functions
12. Develops data from varying sources, including use of demonstration test data where possible	12. HiVal's modular, distributed client/server architecture makes it transparent to the user whether the data is from a dynamic simulation or a static database of test results
13. System must be implementable under the constrained NAHSC schedule. Desirable to modify and use existing models and simulations as much as possible	13. HiVal has been implemented. Its basic design and implementation focuses on using modified existing models, while allowing them to be linked with newly-developed modules as well
14. Ability to select most appropriate modeling/simulation, analysis, or test data for evaluating an alternative concept	14. HiVal has implemented decision logic, based on user-selected MOEs and system configurations, which automatically constructs the correct linked network of simulations, data, etc. for evaluating an alternative
15. Incorporate a wide range of modeling categories	15. HiVal currently includes traffic, vehicle control, environmental, safety, and cost model classes

TABLE 4-3: Design Requirements for an AHS Simulation and Decision Support System

DESIGN GOALS	HiVal DESIGN
1. It is realized that implementation should involve modification/extension of existing models (such as SMARTPATH) , and their integration	1. A fundamental premise of HiVal's implementation has always been the incorporation of existing models. A general, flexible computing structure to integrate them has been demonstrated
2. Incorporation of evolving AHS operational effectiveness models as they are developed	2. It will be easy to incorporate such new models into the HiVal computing hierarchy as alternative "traffic" models
3. Information used by a Driving Simulator would be stored in the system database file	3. HiVal DIS (Distributed Interactive Computing) connections allow linkages between HiVal traffic modules interactive driving simulators. The HiVal prototype has a uni-directional DIS connection that allows interactive 3-D visualization of traffic scenes. With "DIS-enabling" of traffic simulations and full-scale taffic simulators (e.g. Iowa Driving Simulator), direct interfaces between HiVal and driving simulators can be achieved
4. Provide data files for concepts, scenario and system configuration databases	4. HiVal incorporates similar files in its concept evaluation decision logic, its storage of pre-defined roadway geometries and default parameter input files, and its storage of files of representative system configurations
5. Provide data files -- weighted evaluation criteria, results to evaluators	5. HiVal stores MOE and evaluation output data in its central database, and provides unified graphical modules to display the results extracted from these files to evaluators

4.2 *System Architecture Specification*

This activity defined both the functional and computational architectures for the HiVal system. The functional architecture includes the topology of the overall HiVal hierarchy, how modules are linked, how modules are implemented, what the interface standards between modules are, and what level of fidelity a module operates at. These details have been developed for those elements of HiVal that comprise the prototype demonstration (e.g. SmartPath, FRESIM/AHS, Mobile5a), and are illustrated in Section 6. The requirements for developing "transfer functions" or aggregate measures which combine detailed outputs of a low level module into a "lower fidelity" input for a higher-level module have been defined for each module pair in HiVal. The computational architecture includes a description of the hardware and software used to implement HiVal. This includes the choice of computing paradigm as well, e.g. a client-server structure.

4.2.1 *Major Functional Elements*

The overall functional architecture of HiVal is given in Figure 4-1.

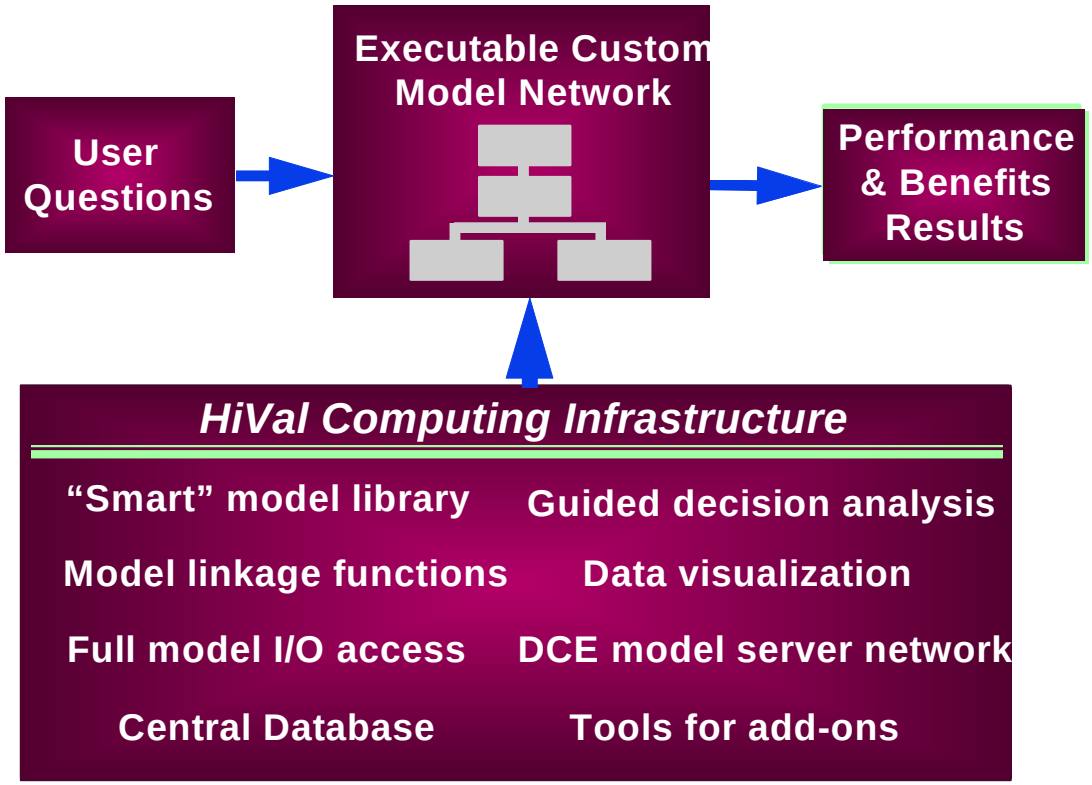

Figure 4-1 HiVal Computing Architecture for Flexible Guided Analysi s

The top part of the figure shows HiVal's high-level operating principles. A series of analysis or decision "questions" are posed by the user. These are a combination of scenario definitions and

output measures of effectiveness the user wishes to evaluate. Based on these inputs, a custom executable model network is built by HiVal. HiVal uses a small system of rules to define the right network. Along with the analysis network, HiVal provides appropriate sets of default input parameters need to run the network. Figure 4-2 presents one of HiVal's interface screens showing a simple analysis network.

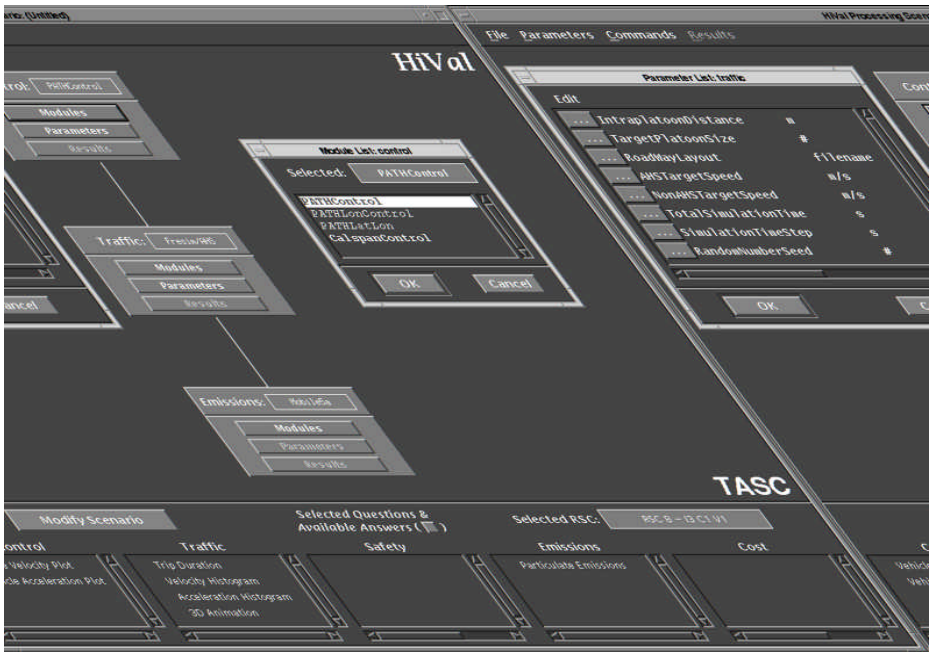

Figure 4-2 HiVal Interface and Analysis Network

Once the network had been run, analysis results are provided in consistent graphical, tabular, and animation formats. Results are stored in a central database which can be accessed later for additional analysis and comparisons. The lower portion of the figure summarizes the key elements of HiVal's computing infrastructure that provide simulation and decision support capabilities. The next sections describe each of the computing infrastructure elements in more detail.

"Smart" Model Library -- HiVal's model library is currently divided into five categories of module types. With each module category, there are one or more models of that type. For example, for the AHS "Traffic" module type, there are currently two specific models, FRESIM/AHS and SmartPath. Table 4-4 lists the current HiVal module categories and models in the library. Descriptions of these models can be found in the PSA final reports and other technical produced by each of the appropriate organizations. The number and definition of model categories is flexible and can be easily modified -- these were chosen as representative for the prototype. Additional AHS models were evaluated for inclusion into HiVal, but were not included at this time due to the constrained scope of the prototype development. The Simulink ™ application, not

originally planned for HiVal, is needed to run two late-arriving control models. These models are included in HiVal and are ready-to-run once Simulink ™ is acquired.

TABLE 4-4 Module Categories and Specific Models in the HiVal Prototype

MODULE CATEGORY	MODEL DESCRIPTION AND CONTRIBUTOR
Vehicle Control	• Lat/Lon Control -- California PATH • Platoon Lon Control* -- Calspan • Platoon Lon Control*-- Martin Marietta
AHS Traffic	• FRESIM / AHS -- FHWA • SmartPath -- California PATH
Safety	• Platoon Collision -- Calspan
Emissions	• Mobile5a -- EPA (Environmental Protection Agency) • Power Demand Emissions -- UC Riverside
Cost	• Cost Estimation -- Parsons Brinckerhoff

* Requires Simulink Application, not included in delivered prototype system

These models, along with additional RSC configuration files and stored highway geometries, constitute the HiVal model library. When a HiVal user is defining an analysis scenario, he selects a set of measures of effectiveness and a RSC. HiVal uses a small system of rules to select appropriate models based on these inputs. Appropriateness is defined along several dimensions:

- Which models are appropriate for use with the selected RSC (appropriateness means consistency of RSC and underlying modeling assumptions)

- Which models are appropriate for the selected measures of effectiveness (which models can compute these MOEs)

- Which models can be legitimately linked together to form an analysis network that correctly accounts for dependenci es between models (which models can be linked in a way that is compatible with the theoretical assumptions and algebraic outputs of the models).

HiVal's "smart" model library analyses the requests against these dimensions and automatically selects an appropriate network. The system allows the user to override the automatic choices when more than one option exists, but prevents the user from making incorrect or incompatible network definitions.

DCE Wrappers and Network -- Client server technology is the primary element of HiVal's distributed computing architecture. The role of software "wrappers" in HiVal is to encapsulate existing or legacy software, without the need to rewrite or extensively modify existing software, and allows two previously incompatible elements to operate together in a common computing

environment. This is extremely important for an AHS simulation and decision support environment that can be adaptively expanded as new AHS elements, and the software to analyze them, are continually developed as AHS concepts themselves evolve. For HiVal, that environment is provided by DCE, Distributed Computing Environment, a commercial client server development and run-time environment. DCE also provides full network management services for both local and distributed networks, including communications management, management of dynamic network resources, file service, time synchronization, and security services. Additional discussion of client server technology is presented in section 4.2.3.

Model Linkage Functions -- While code wrappers permit models to be connected in a software sense, linkage functions allow models to be connected in an analytic sense. They specify how data from one model is exchanged between other models. The exchange can be a direct mapping to account for differences in data formats, but is usually more elaborate, requiring transformation or aggregation of data. An example of this is the conversion of time-domain vehicle time history data from the SmartPath AHS traffic simulation into aggregate histograms for use as input to the Mobile5a particulate emissions model. Linkage functions are also constructed to ensure that the analytic connections made are compatible with the underlying theoretical assumptions of individual models. Examples of linkage functions are provided in Section 6.

Guided Decision Analysis -- At the highest level, HiVal can operate as an automated decision support tool, providing fully-guided decision analysis. Once the user has specified the analysis questions and scenarios of interest, all the remaining steps of the analysis can take place automatically. These steps include: formation of the analysis network; selection of default input parameters for each model; execution of all models in the network; standardized graphical display of all output results and measures of effectiveness. This process can proceed under the complete guidance of HiVal.

Full Model I/O access -- HiVal is designed to allow the user to operate on multiple levels of sophistication in order to serve a wide range of analysts and their needs.
While HiVal can operate as an automated, system-level decision support environment, the need to be able to support “expert” users requiring detailed access to elements of the system is also provided in the system architecture. Users can override the default analysis network by making alternative selections of models within a module category (where HiVal will only allow the user to select options that are consistent). For any model, the default input parameters can be accessed and edited. To facilitate consistent analyses, HiVal provides the user with pre-defined graphic and numeric outputs corresponding to selected output results. Recognizing that some users may want to see more customized or traditional outputs from a single model, HiVal permits full on-screen access to the

"raw" output files from any given simulation. These output files are also stored in HiVal's database. Thus any model within HiVal can be run on its own with all of the functionality that it would have if run independent of HiVal. In addition, users can modify HiVal's lists of pre-stored output types to include any other results they wish to standardize.

Central Database -- In order to facilitate scalabilty of the system and simplify the explicit connections required between models, HiVal uses a central database to store all system data. Both dynamic (generated by a simulation) and static (pre-stored in the database) parameters are managed by the database. Linkage functions connect models via the database, working as "pre-processing" functions for model inputs, and as "post-processors" for model outputs. The central database in the HiVal prototype is currently a flat file database maintained by the database server. However, HiVal's modular design and client server architecture make it easy to include a more powerful database structure (such as a commercial SQL relational database) into HiVal as required.

Standardized Data Visualization -- Individual models developed independently over time all present their outputs in different ways, usually textual rather than graphic. The analyst is required to design compatible and consistent ways to compare outputs, and to manually process those outputs. HiVal, however provides the analyst with standardized data visualizations that are automatically selected, prepared, and displayed. Visualizations include histograms, x-y multiple line plots, tabular displays, and 2-D animations and 3-D animations output data. These visualization routines are written as reusable tools which can be applied to any suitable data within HiVal.

Tools for add-ons -- A key element of HiVal is its extensibility and flexibility -- the architecture has been designed under the assumption that the models and databases within the system will change. The client server paradigm, in conjunction with software wrappers, linkage functions, and re-usable output prototypes have been implemented to facilitate adding new module categories, models, and results displays. The procedures for adding new elements to HiVal are discussed further in Section 6.

4.2.2 *Distributed Computing Architecture*

HiVal uses a general distributed computing environment to provide flexibility and functionality. Distributed computing involves the cooperation of more than one computational engine communicating over a network. The machines in the network can range from supercomputers to PC's to "virtual" machines resident in a single physical machine. The networks can be local, or can span large geographic distances. Two types of distributed computing paradigms are implemented in HiVal: client server-based simulation, and, in a secondary way, DIS (Distributed Interactive Simulation)

protocol links for interactive 3-D visualization. The primary paradigm is client-server-based simulation.

4.2.2.1 *Client Server Computing*

One architecture for implementing distributed simulations is the client server architecture (the material in this section is adapted from Ref. 3). In this architecture, the distributed application is divided into two parts, one on each of two computing elements (which are often, though not necessarily, different computers). The two parts communicate over a network, either local or wide area. Figure 4-3 illustrates the basic client server architecture.

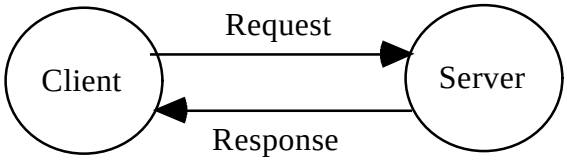

Figure 4-3 The Basic Client Server Architecture

The *client* portion of the application resides on the node that initiates the distributed request and receives a "service" from another node. The *server* process receives and executes the distributed request. The terms "client and server" are relative roles. They depend on which process is invoking a request for service, and which is providing it. For example, client process may also act as a server process if *it* receives a request for service from another client. Often, major clients in an application are implemented as continuous processes, whereas the server is implemented as part of a library, i.e. as a process which runs when called, returns a result, then awaits the next call to execute.

In HiVal, the client process is initiated from HiVal 's graphical user interface, and includes the selection of models, construction of the hierarchy, and management of input and output requirements. Individual models, such as FRESIM/AHS or Mobile5a, are implemented as servers. Figure 4-4 illustrates the HiVal client server architecture, software wrappers, and linkage functions.

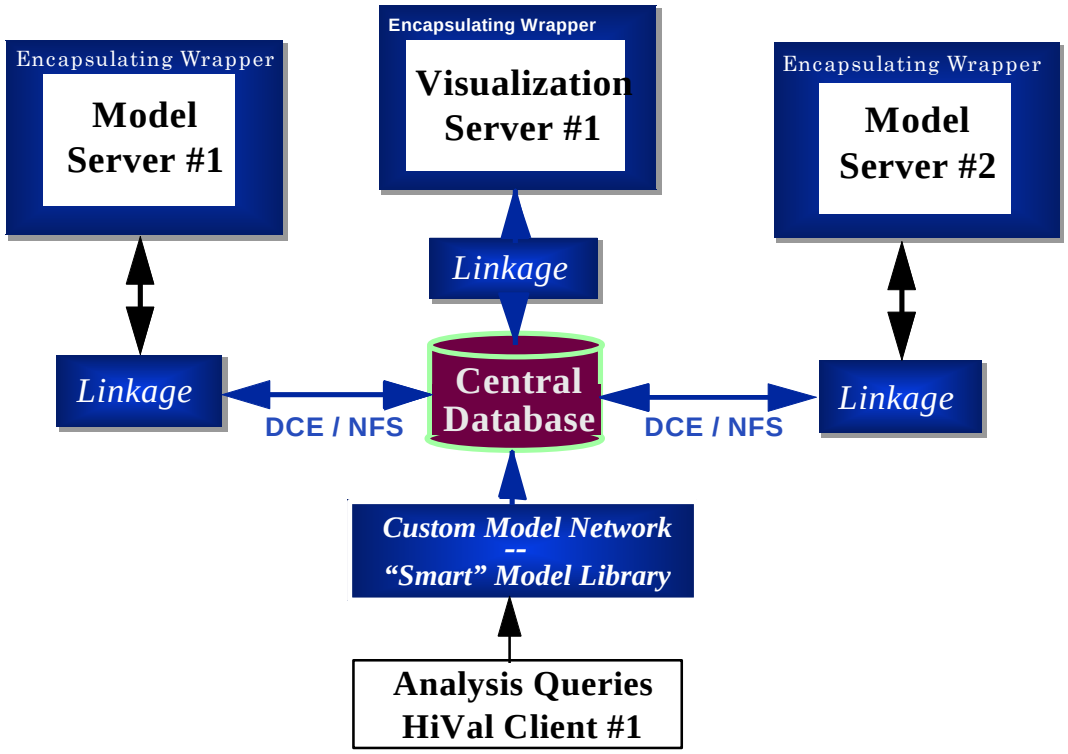

Figure 4-4 HiVal's Client Server Architecture

HiVal implements the client server architecture using the OSF 's (Open Software Foundation) DCE (Distributed Computing Environment). DCE is a layer between the operating system and network on one side, and the distributed applications on the other. DCE provides the services that allow a distributed application to interact with a collection of heterogeneous operating systems, computers, and networks as if they were a single system. DCE is an open standard, meaning that maximum compatibility with present and future commercial products is ensured. DCE supports various specialized elements of the client server architecture, including the data sharing model and the RPC (Remote Procedure Call) model. In HiVal, RPC's are used to allow high-rate feedback loops between connected models, in contrast to models that can run sequentially without feedback. Both kinds of client server connections are supported by HiVal's design. For example, a linkage between a traffic model like SmartPath and an emissions model like Mobile5a requires no feedback. A linkage between FRESIM / AHS and a Lat/Lon vehicle control model, on the other hand, will require high-rate feedback, and the RPC model would be used in this case. RPC have been implemented in the HiVal software architecture, but have not yet been implemented for any specific pair of models in HiVal. This is due to the constraints on the scope of the HiVal prototype development weighed against the extra analysis and software effort required to modify an existing traffic model to connect with a sophisticated vehicle control model. The software infrastructure for this is in place within HiVal, and there are no major theoretical obstacles to performing such a linkage. For additional information on DCE, see Ref. 3.

4.2.2.2 *DIS Connections to HiVal*

HiVal's computing infrastructure has been designed to explicitly provide a flexible architecture that can be expanded to integrate new analysis resources as they are developed. One example of such a resource is HiVal's connection to DIS (Distributed Interactive Simulation) tools. This technology was originally developed for military applications under DoD sponsorship, and has been adopted as an International simulation standard, IEEE-Std-1278-1993 (Ref. 4). The Principal Investigator for HiVal serves on a DIS Standards Special Interest Group for ITS applications. As part of a TASC internal R&D project to apply DIS technology for ITS, HiVal was used as a testbed for linkage to a DIS -based tool called IVHSim. As a result of this synergistic work, HiVal has a linkage to DIS visualization tools. This linkage allows outputs of conventional traffic simulations, such as FRESIM / AHS, to be rendered and displayed in an interactive 3-D animated scene. 3-D vehicles move down the highway, and the analyst can interactively move through the traffic scene, affix himself to particular vehicle, and experience multi-directional views. Figure 4-5 is an example of this DIS-based traffic visualization, showing different views available simultaneously in different display windows.

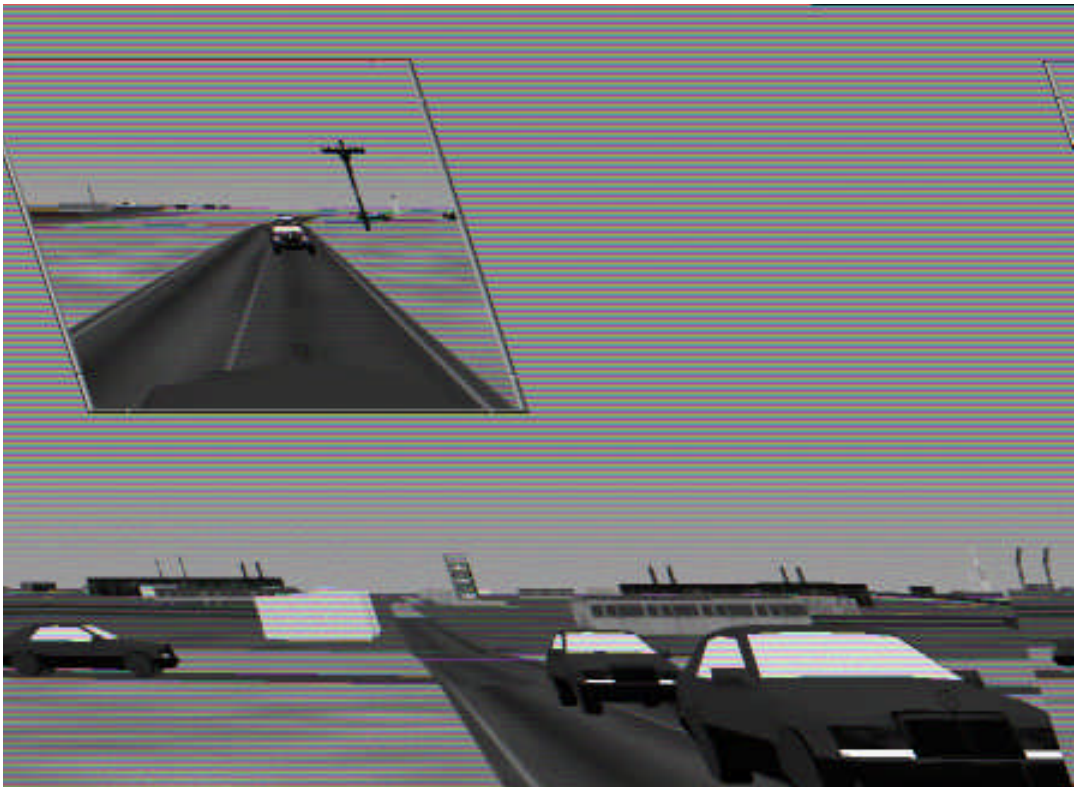

Figure 4-5 DIS-based Traffic and Infrastructure Visualization

Compatibility with DIS technology offers several advantages for an AHS simulation and decision support environment. These include:

- Multi-perspective 3-D views of traffic simulation outputs for qualitative analysis and verification of simulation outputs

- Presentation of AHS concepts to stakeholders in a natural way that has high visual impact and conveys concepts effectively and efficiently

- High-fidelity representations of traffic and infrastructure for selected AHS human factors studies

- Linkage to DIS-enabled driving simulators that take advantage of DIS two-way communications to allow drivers to interact directly with DIS-enabled AHS traffic simulations.

The HiVal architecture and its IVHSim linkage provide a substantial foundation for continued transfer of military DIS technology for AHS analysis needs.

4.3 *Standards and Interface Requirements*

This activity defined the hardware and software interface standards, as well as the data interchange protocols for elements of the HiVal system. Standards appropriate for different distributed computing environments were evaluated for their use in HiVal. Client server standards based on the Open Software Foundation's Distributed Computing Environment (DCE) was selected. Technologies and concepts used in Distributed Interactive Simulations (DIS), and object-oriented distributed simulation, such as the Common Object Broker Request Architecture (CORBA), are also used as appropriate. Data interchange protocols are defined for specific modules that are derived from PSA and related AHS studies. Some general standards and interface guidelines for those developing software specifically for HiVal are included in Section 6.

4.4 *Prototype System Development*

The major output of this project is a working AHS simulation and decision support system: a prototype of the HiVal system. Rapid prototyping and interactive software development strategies were used to ensure that flexible, reusable software components resulted. In addition to validating the HiVal concept, the prototype serves as a valuable, concrete "design tool" for development of overall system requirements and architecture specification. A variety of models from both the PSA and broader AHS research communities are included in HiVal. Modules operating at different levels of fidelity are employed, and demonstrate the integration or "transfer" of outputs to inputs for disparate module structures. Guidelines for procedures and techniques for developing software, interfaces and protocols for new elements of the HiVal testbed that will be added in the future have also been developed, and are presented in Section 6.

4.4.1 *Hardware and Software in the Prototype*

The HiVal product has two dimensions: the extensible computing infrastructure it provides, and the specific software and hardware within the system at any one time. The HiVal prototype collects together the best existing AHS simulation and decision support software resources, and modern computing hardware, to provide a useful tool for today's AHS analysis. Table 4-5 lists the software in the prototype, the language it is written in, and the organization that provided the software for HiVal. The Table points out the wide range of different software languages that have been seamlessly integrated together within the distributed client server architecture.

TABLE 4-5 Software Elements in the HiVal Prototype

FRESIM / AHS Traffic Model	Fortan	FHWA
SmartPath Traffic Model	C	PATH
Mobile5a Emissions	Fortran	EPA
Power-demand Emissions	Matlab	U.C. Riverside
Lat / Lon Control	C	PATH
Platoon Lon Control *	Simulink Calspan	
Platoon Lon Control *	Simulink Martin Marietta	
AHS Platoon Collision	Matlab	Calspan
Cost Estimation	Matlab(Lotus 1-2-3)	Parsons-Brinckerhoff
2D Traffic Animator	C	TASC
3D Visualizer link	C++, DIS	TASC
System Configuration Files	C	Calspan
Stored Highway Geometries	ASCII	TASC

* Requires Simulink, not included in delivered prototype system

The HiVal prototype consists of three primary pieces of hardware: a Sun workstation running Unix/X Windows, a 486-PC running MS Windows, and an ethernet connector hub for the HiVal LAN. Although not part of the HiVal system itself, a Silicon Graphics workstation interfaces with HiVal for DIS-based visualization. This hardware configuration is illustrated in Figure 4-6.

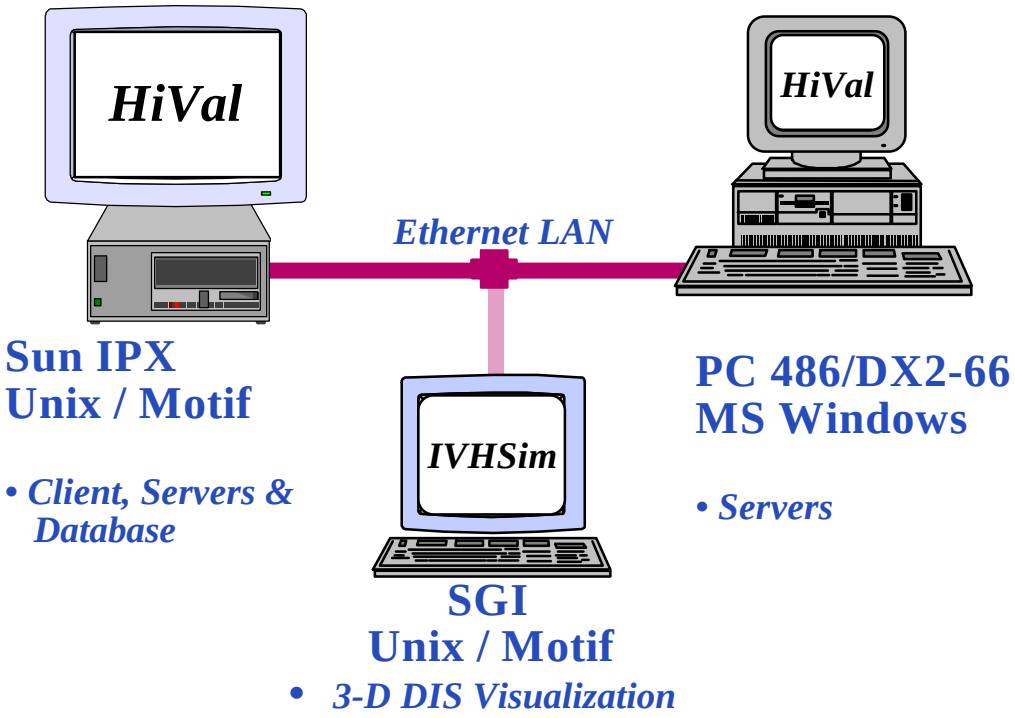

Figure 4-6 Hardware Configuration for the HiVal Prototype and Adjunct IVHSim Visualizer

4.5 *HiVal's Ability to Grow*

The HiVal system architecture has been specifically designed to be extensible. This is in recognition of the fact that AHS modeling and simulation is not static -- it will continue to evolve and improve as research continues. An AHS simulation and decision support tool must be able to adapt and expand as well, without the need to "break" and rebuild the tool. HiVal's software architecture achieves this adaptibility by using distrubuted client server computing, software wrappers, analytic linkage functions, a central database, and modular design principles.

Software Architecture -- HiVal's software design *puts no limit* on the number of module categories or specific simulation models that can be part of the system. Any number of models can be added to any of the existing module categories. Any number of pre-selected MOE can be added to HiVal. HiVal's software architetcure facilitates such additions. Additional module categories can also be easily added to the system. This would require minor modification of the HiVal client graphical interface, which currently has five module category "slots" displayed. The underlying code for the GUI is object-oriented and can easily be extended to add additional module categories. If a large number of additonal module categories were added (more than ten total, for example), the current GUI design might require some modification to ensure that it remains user-friendly.

Hardware Architecture -- HiVal's general hardware architecture also *puts no limit* on the number of models or databases that can be added to the system. Under the prototype hardware configuration, simulation servers can be hosted on either the Unix workstation or the MS Windows™ PC. The ethernet hub on the HiVal prototype LAN has six unused ports, to which up to

six additional computers that host servers can be added. This permits an easy, major expansion of HiVal's computing capability. While there is no limit to the number of simulation or database servers that can be hosted on a single computer, performance of the system will be degarded if any single computer is "overloaded" with servers. A computer is overloaded with servers if the amount of RAM available for each server is low enough to require frequent paging / memory swaps, or if any single server is computationally intensive enough to "block" the execution of other servers. In the prototype HiVal configuration, up to five servers are available to run simultaneously on the PC hardware with 16 MB RAM. While several of these (using Matlab and Simulink) are computationally intensive, system performance remains good. The HiVal prototype has approximately 1.5 GB total of hard disk storage space available. Obviously, there are limitations on the size that the HiVal database can be under this

configuration (after disk space has been allocated to operating system and application software). However, there is no limit on how much additional hard disk storage can be added to HiVal.

5. HiVal USER GUIDE

This section presents an informal user guide to the HiVal system. Since HiVal was developed as an engineering prototype under the PSA activities, the software documentation presented in the section and in Section 6 are informal.

5.1 *Requirements for Running HiVal*

HiVal uses a variety of specialized system and applications software to implement its AHS simulation and decision support environment. It uses advanced, but relatively standard, workstation and PC technology. All of the system software is commercially available. The applications software was supplied for use in the HiVal project by the organizations identified in Table 4-3 above. Table 5-1 lists the hardware and software that make up the HiVal system and are required in order to run all of the applications listed in Table 4-3. Licenses to several commercial products are required for HiVal.

TABLE 5-1 Minimum Requirements for Running HiVal Prototype

Hardware:	
1	Sun workstation with 1 Gb disk space, minimum of 16 Mb memory, running Solaris 2.3 (or Sun OS 5.3)
1	RGB monitor, preferably 19"
1	PC486-compatible with monitor and at least 500 Mb disk space, minimum 16 Mb physical memory, one 3.5" floppy drive, running DOS 5.1 or later
1	PC-compatible ethernet card with an ethernet 10BaseT adapter
1	Ethernet 10BaseT concentrator
1	MicroMau Thicknet-to-10BaseT converter for the Sun workstation
2	Unshielded, twisted pair ethernet cables with 10BaseT connectors

TABLE 5-1 Minimum Requirements for Running HiVal Prototype (Continued)

System Software:

1 TransArc™ DCE 1.0.3 base services run-time license for the Sun workstation (Solaris 2.3)

1 OSF Motif libraries for the Sun

1 Perl scripting environment

1 Gradient™ DCE 1.0.2 license for the PC

1 MS Windows™ 3.1 or higher

1 FTP Software run-time license for the PC

Applications Support Software:

1 Lahey™ Fortran Compiler

1 ANSI-C Compiler for Solaris 2.3

1 ANSI-C Compiler for DOS/MS Windows

1 Matlab™ with Control toolbox for PC

1 Simulink™ for PC

Applications Software:

1 Each of the application models and databases listed in Table 4-5

5.2 Accessing the HiVal System

Inquiries about gaining access to the HiVal system should be directed to J. Richard Bishop, FHWA Turner Fairbank Highway Research Center.

5.3 Using HiVal for Analysis

HiVal employs an interactive GUI (Graphical User Interface) developed using human factors and man-machine interface design principles. The X / Motif - based GUI provides the use with access to the full range of HiVal analysis products, from animations of traffic flow to MOE histograms to ASCII files of output results. This section presents a sequence of screen images from a run of the HiVal system that illustrate how the system is used.

5.3.1 *Starting the HiVal System*

The first step in starting up the HiVal system is to initailize the server processes on their respective hosts. The client server paradigm allows servers in the system to be registered either statically or dynamically with respect to clients. The prototype HiVal system is static in this regard and therefore requires all servers to be started and registered before launching the main client process, which is the HiVal graphical interface. During the runtime initialization of a server process, the server is automatically registered in the HiVal system database.

Windows-based Servers -- For the modules hosted on the PC platform the server is started by opening an individual server window from its iconic state in the Gradient DCE group, and then pressing the `Start` button. Two registration shells are automatically be spawned and reside on the screen in iconic state. The server window posts a message that it has registered with the HiVal system and is awaiting remote client requests. Once the captions of the registration shell icons indicate that they are inactive, those shells may be closed.

Unix-based Servers -- Modules hosted on the Unix platform also have server processes that must be started before the main client process. To start a Unix based server, open a new window, change directories to the appropriate module directory and start the server process by entering the executable name at the command line. The Unix server processes also automatically register the server with the HiVal system.

HiVal Client -- Once all servers have registered and are “listening” , the HiVal client interface process can be started in the same fashion as any other Xwindow process. Since the system is currently an engineering prototype, its directory structure does require that the interface be launched from the directory `~hival/hival`. Initialization messages describing which servers are currently registered are posted by the client interface. At this point the user begins to build a HiVal scenario, as described in Sections 5.3.2 ff.

Stopping the Servers -- Servers can be left active for an arbitrary length of time over the course of an arbitrary number of client lifetimes. However, when the server is halted it must remove its registration from the HiVal system. PC-hosted servers should be terminated by pressing the `Cancel` button on the server window, and Unix-based servers should receive the appropriate key sequence, nominally '`q`' , as directed on the screen. The server process must terminate NORMALLY to have its registration removed automatically. In the case that a server terminates abnormally (i.e. `GPF` on a PC or a fatal error such as `SIGSEG` on the Unix platform), it is recommended that the server be started again and terminated correctly to accomplish the registration removal.

5.3.2 *Menu Options*

HiVal has a conventional set of pull-down menus as part of its interface, illustrated in Figure 5-1. These menus and the items they contain are illustrated in Figure 5-2. Underlined items are

currently implemented in the HiVal prototype. Remaining items are part of the system design, but are not yet implemented.

FILE	PARAMETERS	COMMANDS	RESULTS
New	Control	Execute	Control
Open	Traffic	Cancel	Traffic
Save Scenario	Safety		Safety
Save Scenario As	Emissions		Emissions
Save Output	Cost		Cost
Save Output As	All		All
Print			
Exit			

Figure 5-2 HiVal Menu Options

The functions of these commands are:

- `New` -- create a new analysis scenario
- `Open` -- open an existing scenario
- `Save Scenario` -- save to a file a newly create scenario not previously saved
- `Save Scenario As` -- save to a file an already-saved scenario under a new name
- `Save Output` -- save to a file the output created by a run
- `Save Output As` -- save to a file the an already-saved output under a new name
- `Print` -- print the contents of a window
- `Exit` -- exit HiVal
- `Execute` -- execute a scenario
- `Cancel` -- cancel an executing scenario

5.3.3 *Scenario Definition*

A HiVal session is started by selecting *New* from the file menu on HiVal's opening screen. A dialog box entitled *HiVal: Create New Scenario* appears, and the user types a name for the scenario that is being started. *Scenario* is HiVal's term for an analysis session. Figure 5-3 presents an example of the dialog box for naming scenarios. Once a name has been entered, click *OK* to proceed or *Cancel* to remove the dialog box.

After a scenario name is submitted, the *Scenario Editor: Name* screen appears, where *Name* is the name given to HiVal in the *Create New Scenario* dialog box. This is illustrated in Figure 5-4. The editor screen is divided into an upper and a lower half. In the lower half, the user can select one of the 12 Calspan RSC. The Current RSC section of the lower panel is a mapping of the Calspan RSC onto another set of descriptive parameters to provide another view of the RSC characteristics and facilitate comparisons with other PSA RSC. In the upper half of the screen, the user can make queries in any of the five module categories in the HiVal prototype (Control, Traffic, Safety, Emissions, Cost). Selections in these categories can be made in either the *Question* or *Module* mode. In *Question* mode (selected by means of diamond-shaped push buttons at the top of the screen), MOE

(measures of effectiveness) choices appear under each of the five categories. For example, velocity histogram is a traffic module MOE, while particulate emissions is an emissions module MOE. In Module mode, names of the modules, rather than MOEchoices, appear in each category as selectable items. This provides functionality for a range of users -- from those who wish to select models directly, to those who prefer to work at the MOE / decision support level.

The selection of a RSC and MOE (or module) combinations triggers HiVal 's "smart" model library manager. Only those models in the library which are theoretically and functionally compatible with *both* the choice of RSC and MOE can be accessed. HiVal prevents the user from making incompatible choices by disabling (and showing in italic font) selection options which are not valid. For example, Figure 5.3 shows that RSC#2 has been selected. Under the traffic module MOE listing, this has caused the *Aborted Lane Change* MOE to be disabled, because there is no model in the library which can simultaneously model RSC#2 and produce outputs to compute this MOE. Once the user has defined the RSC and MOE desired, clicking *OK* begins HiVal's analysis hierarchy construction.

5.3.4 *Editing and Running A Scenario*

HiVal constructs an analysis hierarchy consistent with the RSC and MOEs selected, choosing a default set of parameters and models (if more than one set is applicable). This construction is represented by the network display of the analysis hierarchy, as shown in Figure 5-5. The upper portion of the screen shows the selected module categories and specific modules chosen by HiVal, for example FRESIM/AHS for Traffic, and Mobile5a for Emissions. Lines connecting modules show data dependencies between analysis modules. The bottom segment of the screen summaries the RSC and echoes the Questions (MOE) selected. There is a *Modify Scenario* button which allows the user to return to the *Scenario Editor* screen.

Each of the boxes representing module categories has three push buttons on it. Two of these boxes are enables at this stage of the HiVal analysis, *Modules* and *Parameters*. Pushing these buttons brings up a dialog box, as illustrated in Figure 5-6. Clicking on Modules brings up a *Modules List* box which indicates what module has been selected, what modules are in the library, and which of them are compatible with the scenario the user has defined (incompatible items appear in italics and are not selectable). The user can change the selected module in a category at this point. When the *Parameters* button is pushed, a dialog box appears showing input parameters for the selected module that can be edited as desired. Pre-stored values of the parameters derived from HiVal's database appear by default, including their units and ranges of acceptable values where applicable. The parameters can be edited by pushing the associated buttons with three dots ([. . .]) in the *Parameter List* window. For each of these buttons, another dialog box appears into which new parameter values can be entered. Clicking *OK* closes the dialog box and applies the changes the user has made.

To now run the network that has been created and initialized, the user selects *Execute* from the *Commands* menu. This initiates the analysis and the calls to the various servers that host the different analysis modules. To help in monitoring the status of the run, the selected module name that appears at the top of each of the module category boxes is highlighted in red when the module is running. The entire top portion of the module category box is highlighted in red when execution of that module has completed. When a module has completed its run and produced outputs required to compute a selected MOE (or *Answer* to a *Question* in HiVal's terminology), a push button appears next to that MOE, as indicated in Figure 5-7. Pushing one of these buttons brings up an appropriate display of the individual analysis result.

5.3.5 ***Displaying Results***

HiVal provides a variety of pre-specified results displays for each MOE in the database. When the *Answers* button next to each *Question* at the bottom of the screen is pressed, a new window with an appropriate pre-selected output format appears. *Answers* can be viewed repeatedly, and in any order. Figure 5-8 presents an example of velocity histograms for a scenario in which AHS lanes and conventional lanes exist on the same highway. Figure 5-9 illustrates tabular output of particulate emission concentrations for two vehicle classes. In addition to pre-selected MOE, the user has direct access to complete raw outputs of a module run by clicking on the *Results* button on a module box in the network. As appropriate, a dialog box appears when there are multiple output files to choose from. For example, Figure 5-10 shows a selection made from the Results button in the Traffic Module. It provides a scrollable window containing the standard ASCII text outputs of the FRESIM / AHS run. In this way, the user has maximum flexibility to analyze results.

In addition to numerical outputs, the HiVal project developed two forms of animators than can be used with HiVal. The first is a simple 2-D traffic vehicle animator which is contained within HiVal as one of the traffic *Question*/MOE options. Figure 5-11 shows an example of the outputs, where six manual and one automated highway lanes are simulated. The 2-D animator displays the roadway in terms of links, rather than physical geometry, to simplify the presentation. It is designed for quick "sanity checks" of simulations. Other more sophisticated 2-D animators, such as those being developed under FHWA sponsorship for the CORSIM family of models, can be easily integrated into HiVal. HiVal, when used in conjunction with the DIS-based animator IVHSim in a post-processing mode, can produce high-fidelity 3-D interactive visualizations of dynamic traffic scenes.

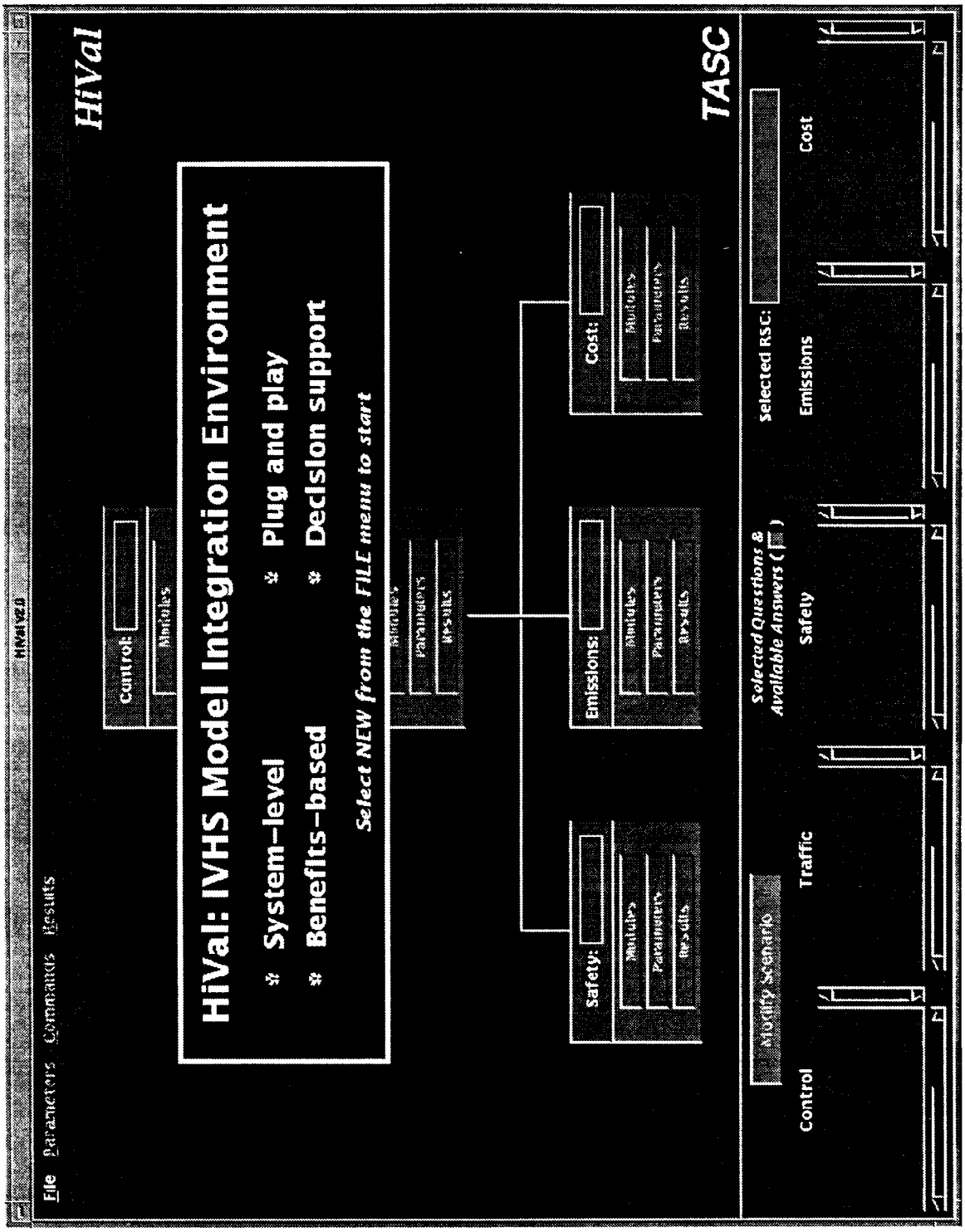

Figure 5-1 Introductory Screen

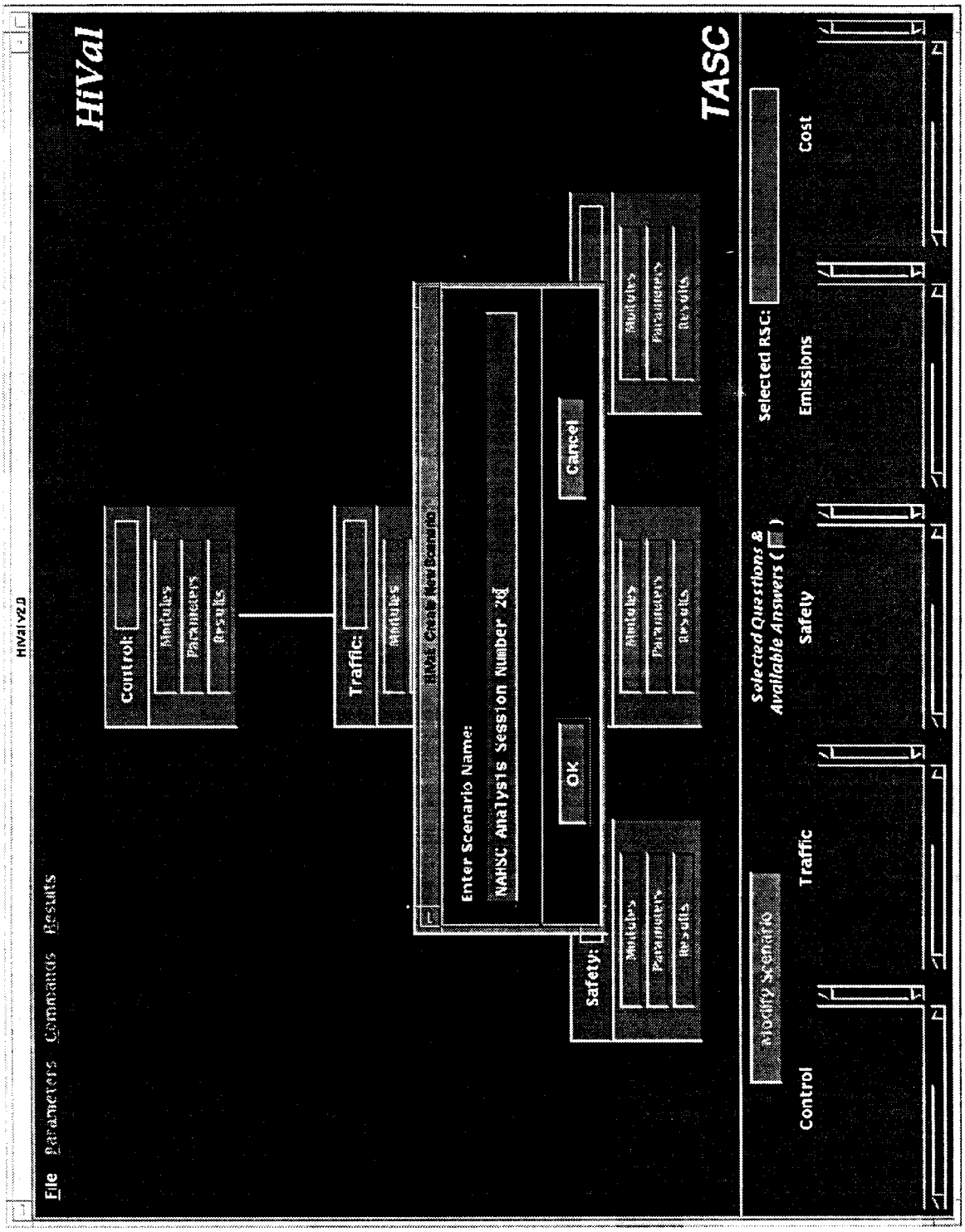

Figure 5-3 Scenario Creation

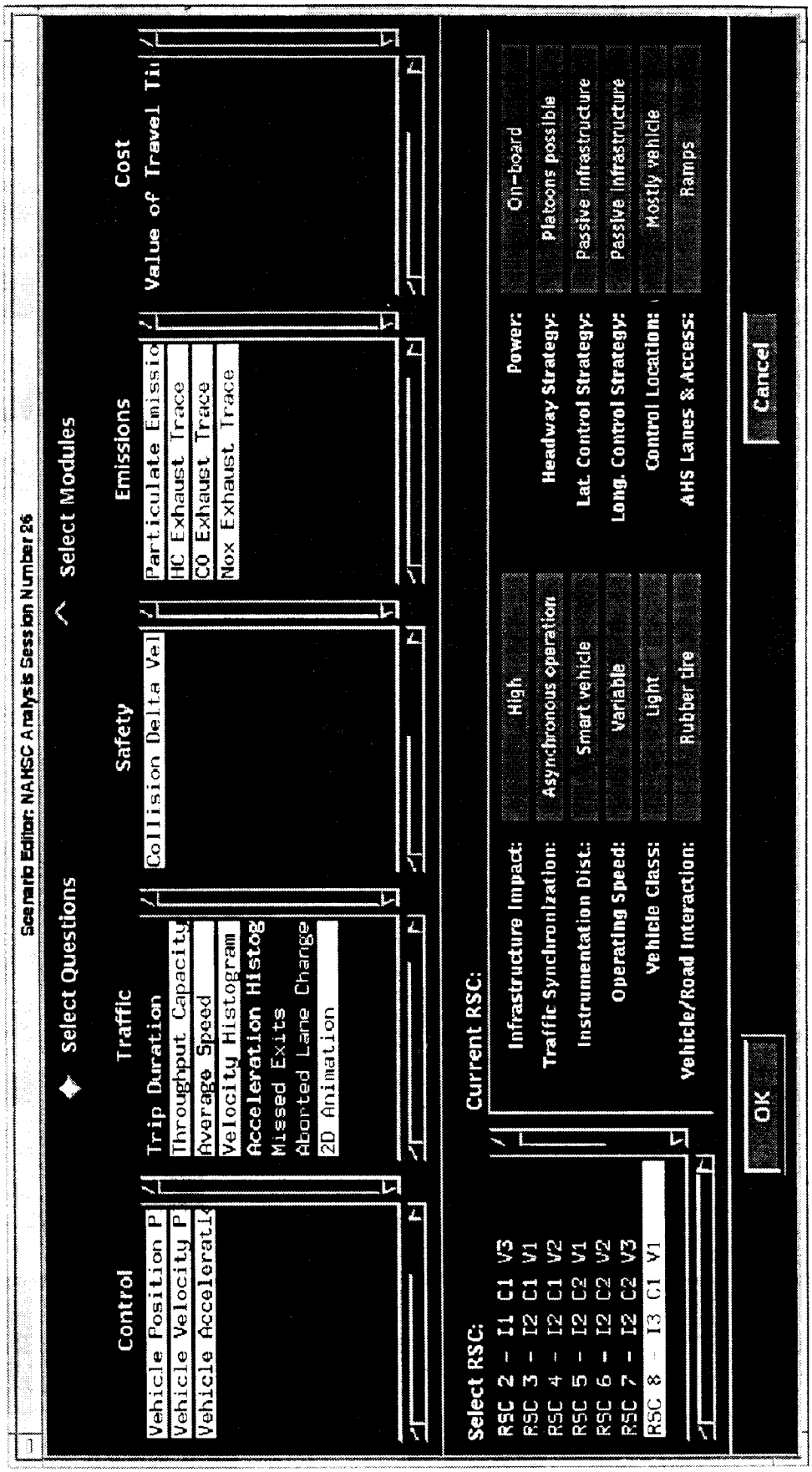

Figure 5-4 Scenario Editor

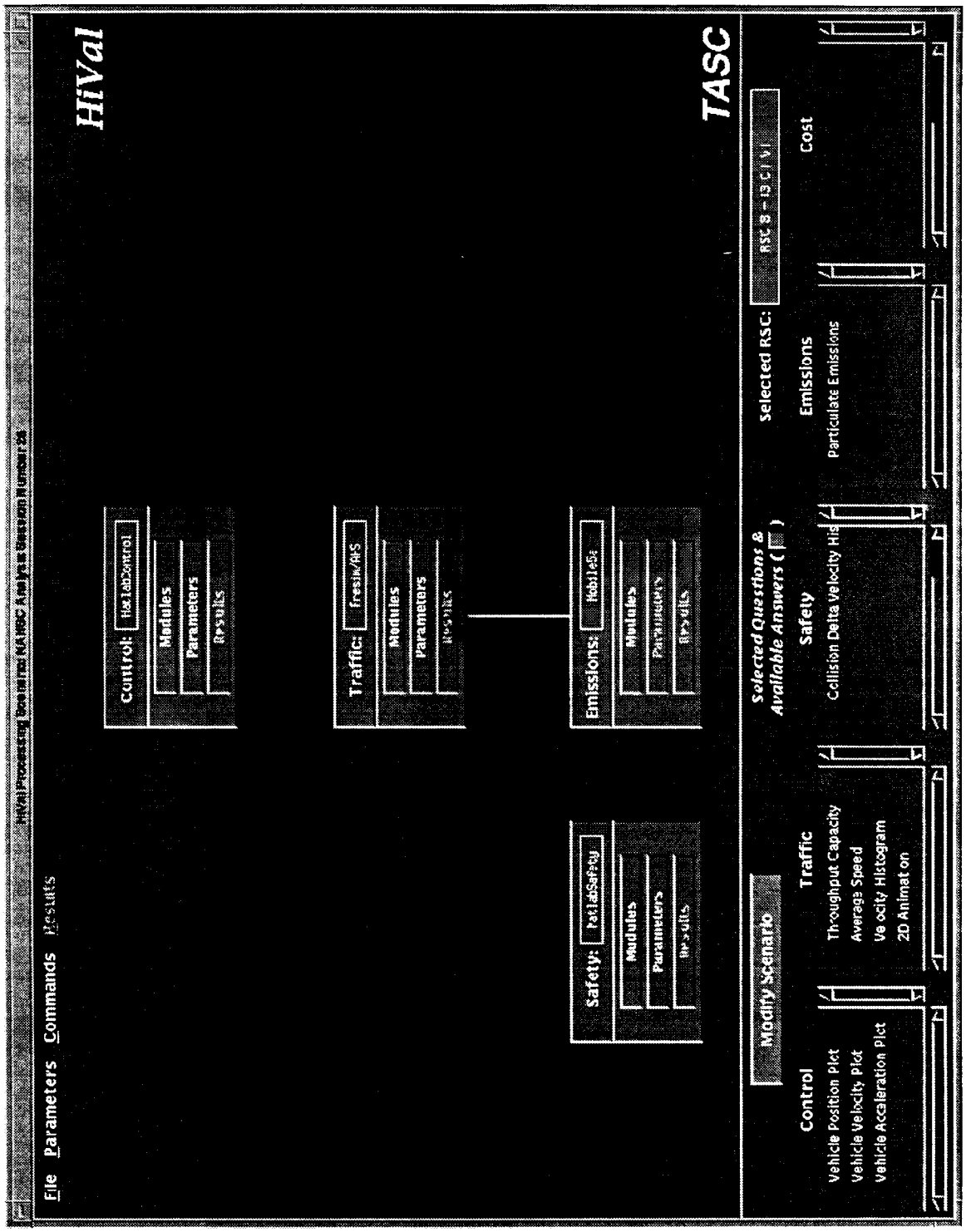

Figure 5-5 Analysis Network

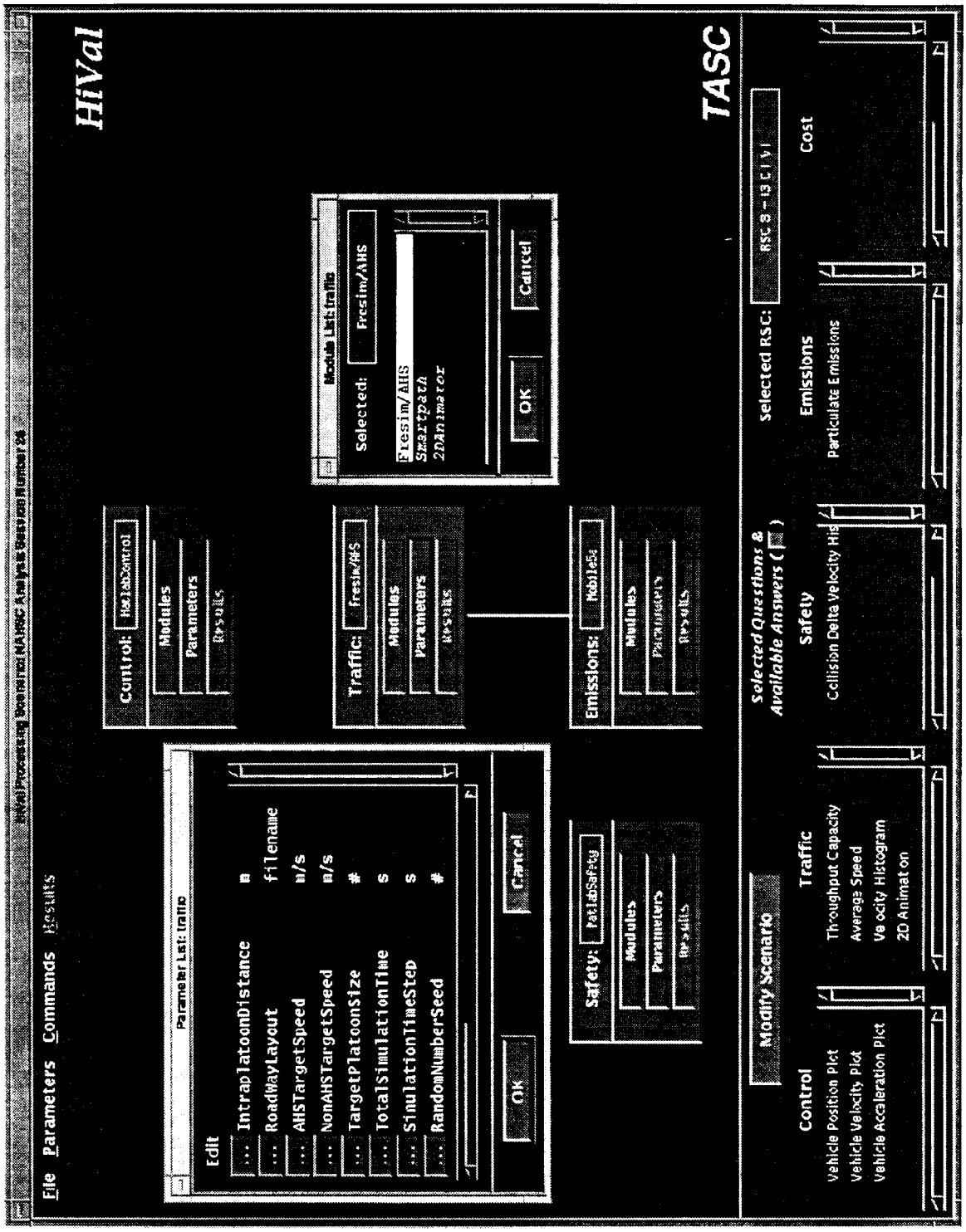

Figure 5-6 Parameter and Module Selection

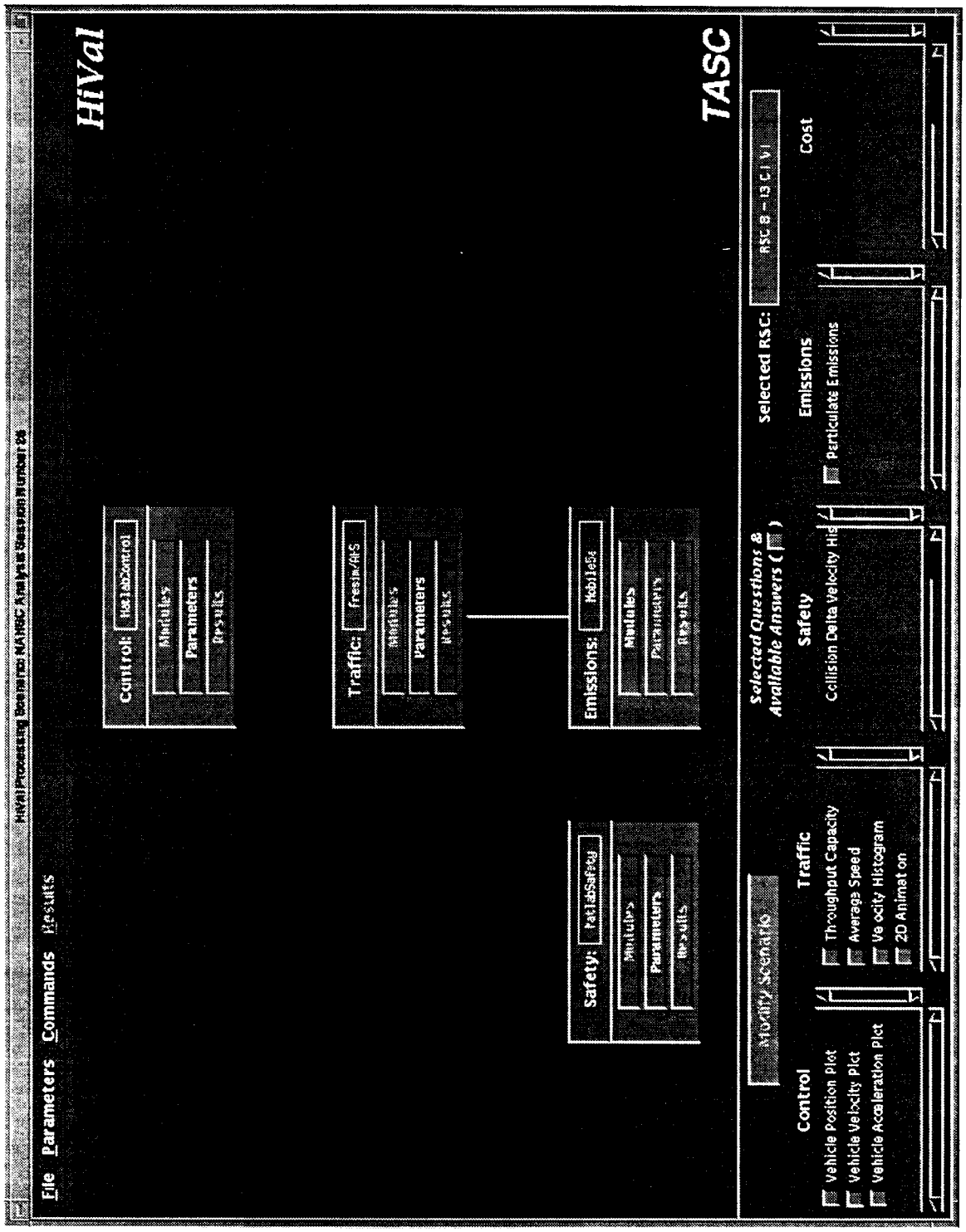

Figure 5-7 Running a Network

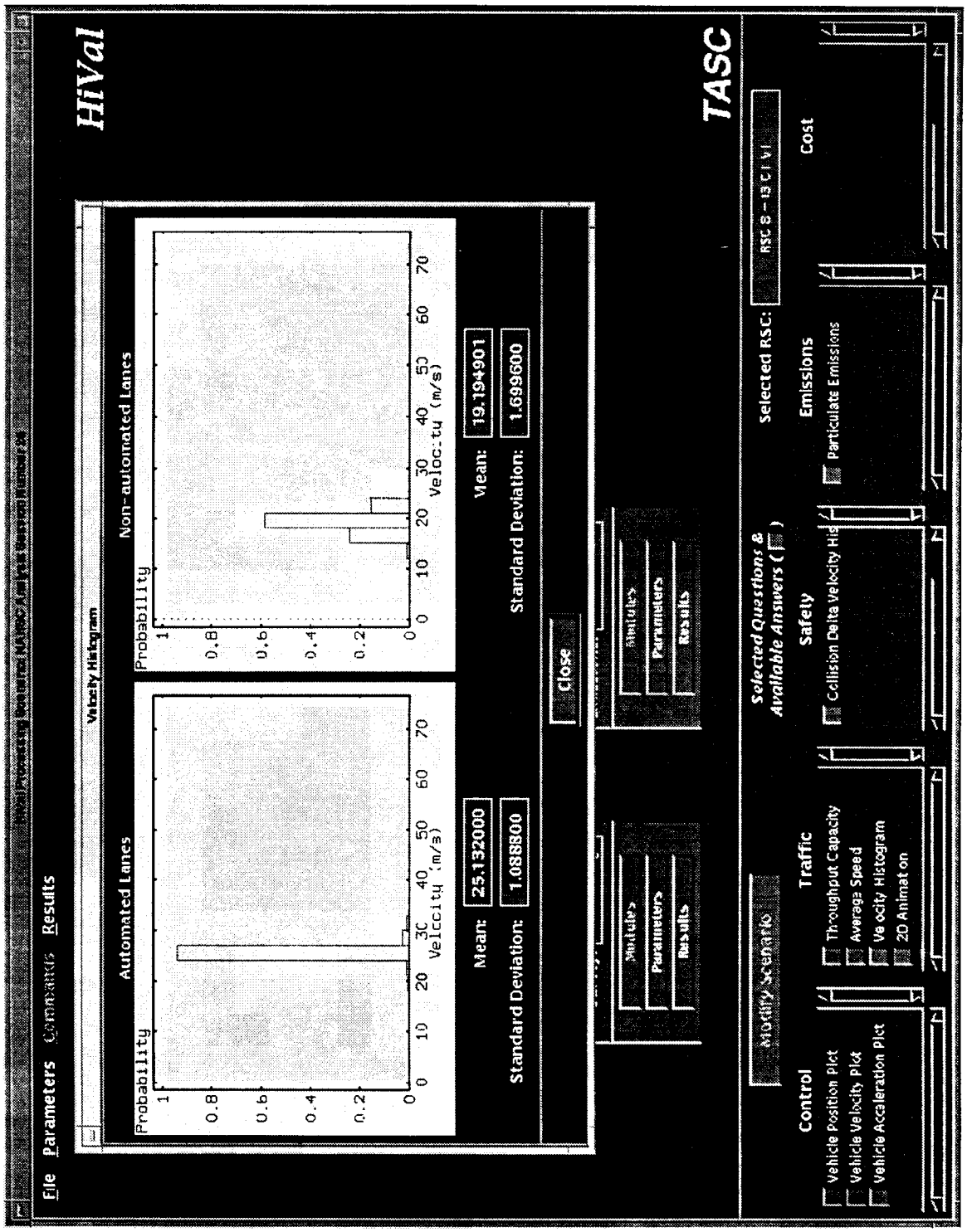

Figure 5-8 Results Histogram

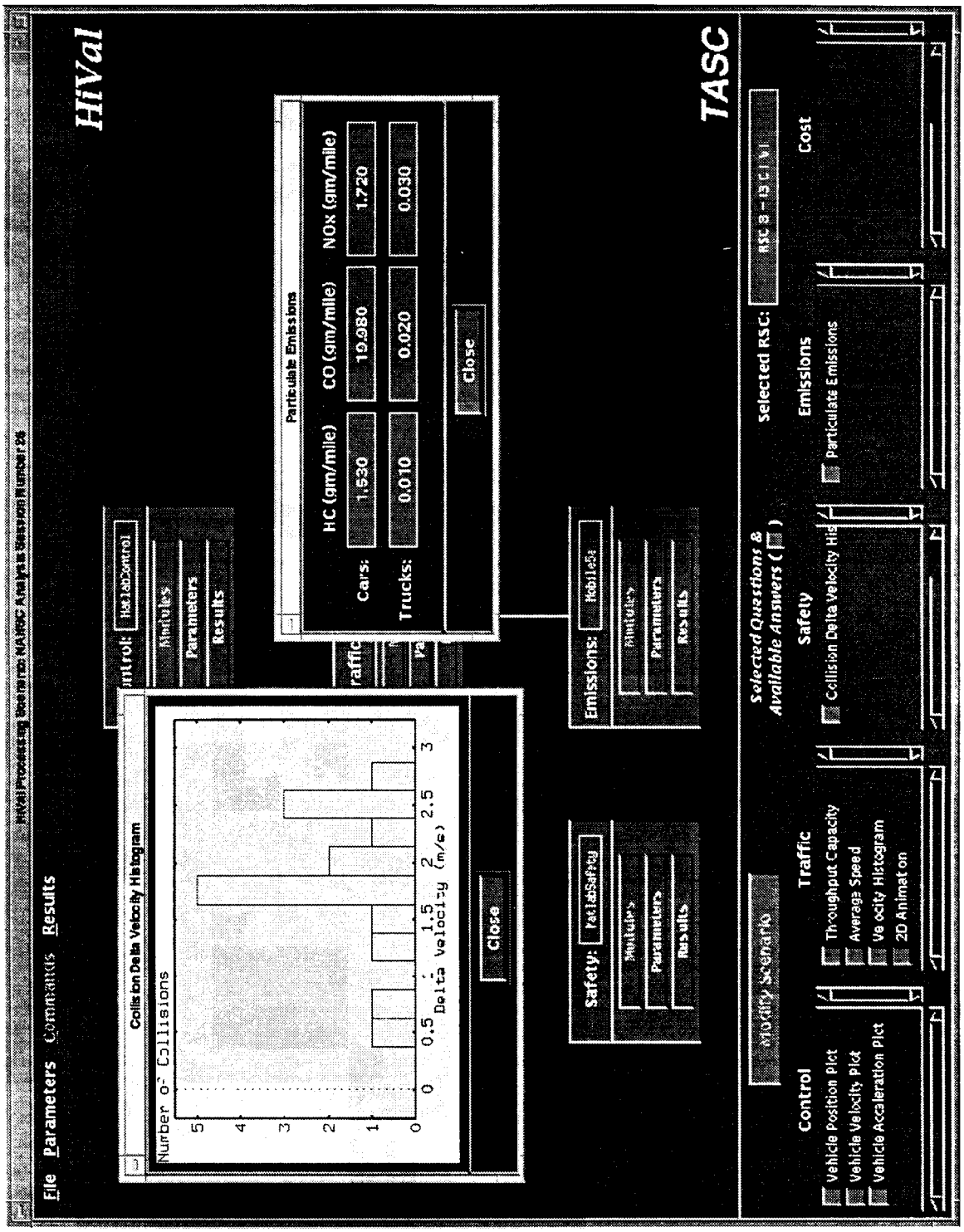

Figure 5-9 Tabular Results

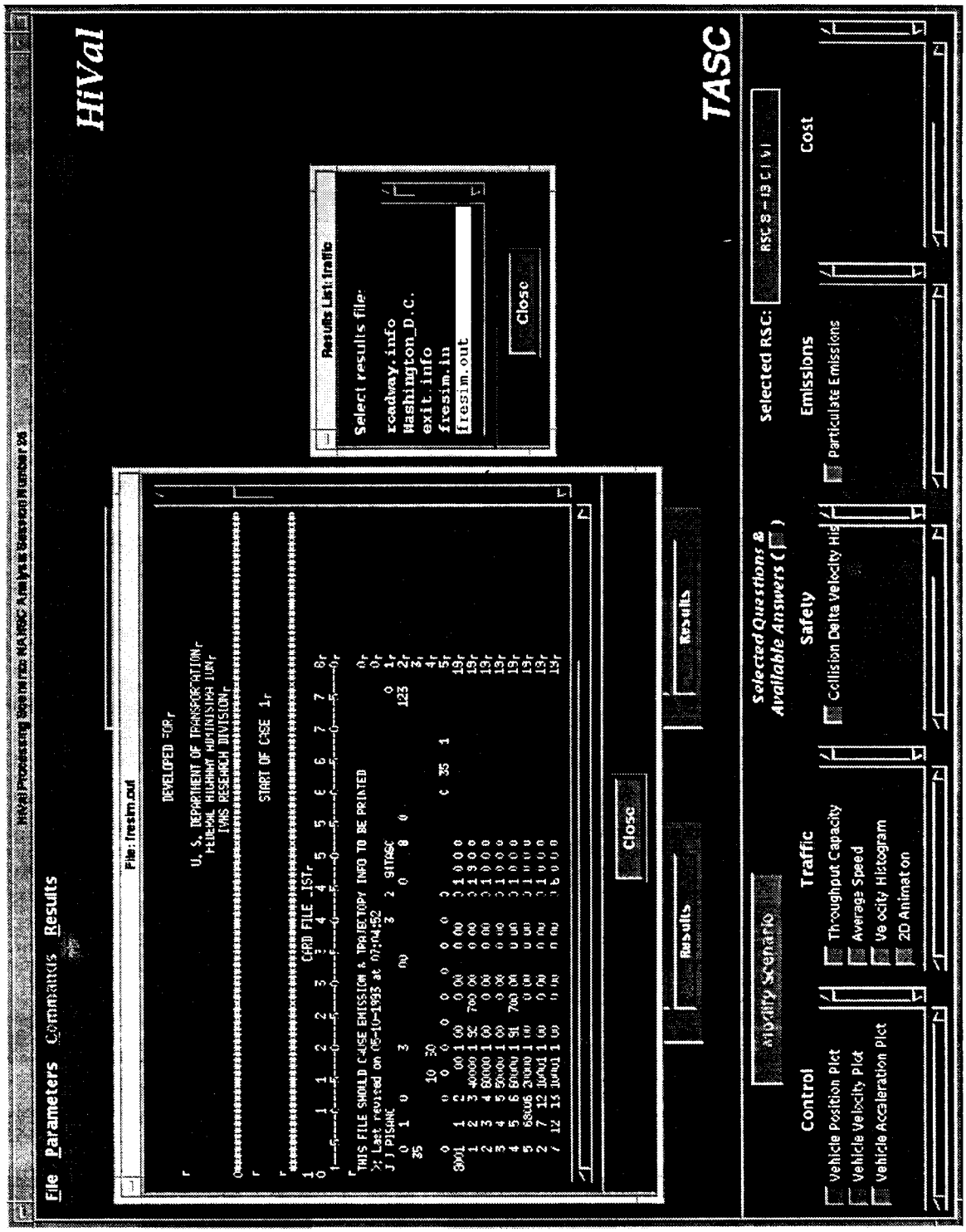

Figure 5-10 Results Data Files

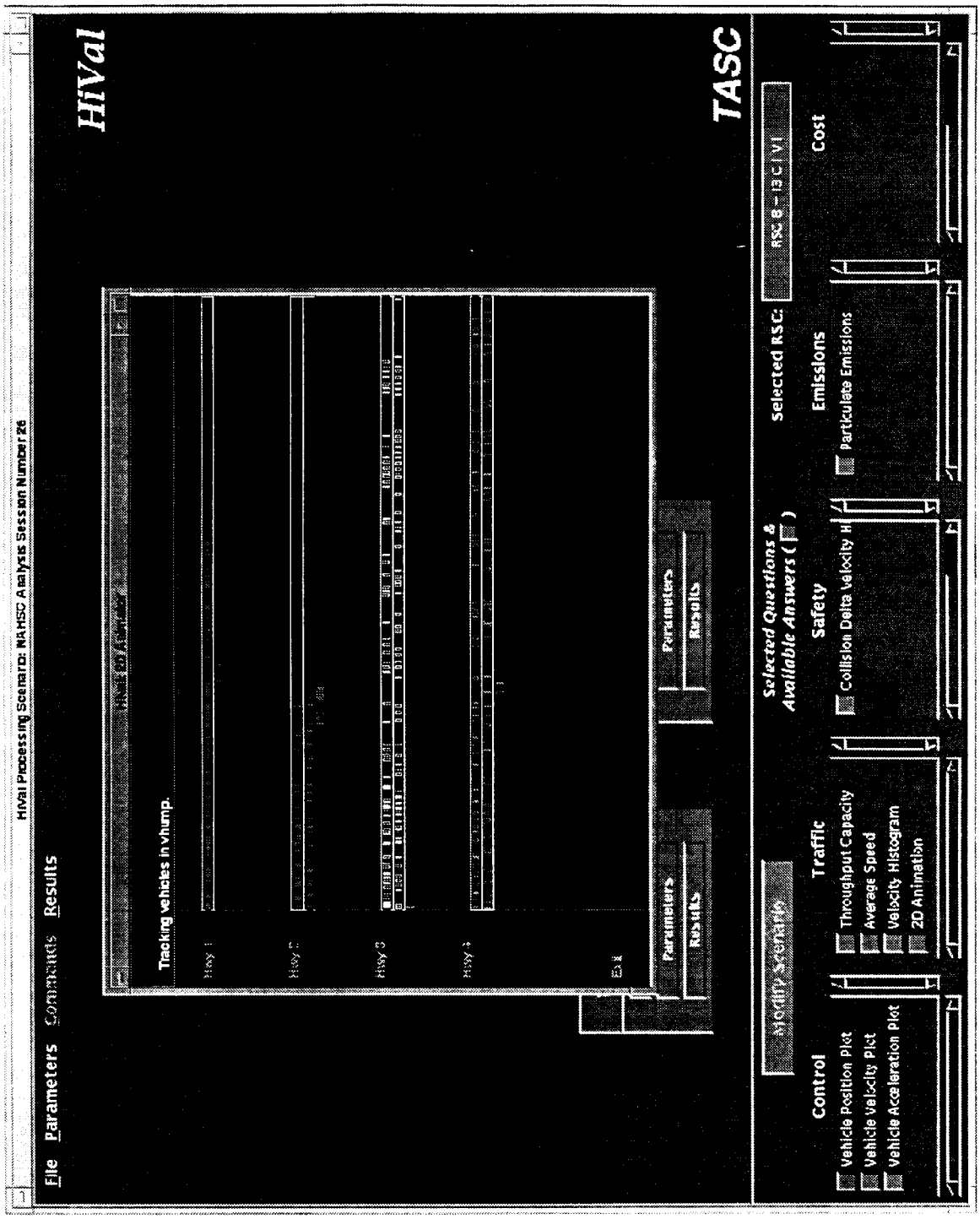

Figure 5-11 2-D Animation Output

6. HiVal PROGRAMMING GUIDE

This section provides a guide to the detailed software structures a nd elements underlying HiVal. It describes the Unix directory structure of the HiVal prototype, presents lists of MOEs and database elements, and provides information on how to add new elements to HiVal. This includes suggested programming guidelines for new models being developed, and examples of linkage functions and C-code “wrappers” for use as templates.

6.1 *Adding a New Module to HiVal*

HiVal has implemented a variety of formal software structures to modularize and faciltate the addition of new modules. The steps that are required to add a new module or database element to HiVal are :

- Specify new configuration file
- Define any new questions (MOE) and scripts to process them
- Define linkage functions (input and output)
- Specify software wrappers and/or RPC interfaces
- Modify module selection logic for the scenario editor.

The easist way to understadn this procedure for the HiVal prototype system is to review these elements for an existing HiVal software module, such as FRESIM / AHS. The following sections present examples of the elements needed to add a new software module to HiVal. Programmers can review the HiVal source code for additional examples.

6.2 *Directories and Configuration files*

This section presents the Unix directory structure of the HiVal prototype, as well as a directory of the configuration files specifying models, questions (MOEs), and RSC.
The directory structure specified below should be replicated by server platforms to the extent of the server and application directories. The modules pre- and post-processors will read from and write to the data base through database system calls. Servers should
mount to the directory:

`~hival/hival`

and put data in the directory `$(CATEGORY)/$(MODULE)/io` for local use and transfer.

Each module entered into the HiVal infrastructure must have a module configuration file. The configuration files have a structured naming convention, `$(CATEGORY).##.module`, where ## is the next number in sequence starting from 00 for the given module category. Configuration files must reside in the directory:

`~hival/hival/database/config/modules`

Examples of module configuration files are provided below.

The RPC servers for each module must register their string binding at each start up and unregister when they are shut down. Scripts for performing the registration steps are provided. All that need be changed in the script is the module name which must match the module name in the configuration file.

Adding a new module to the HiVal infrastructure currently requires a re-compilation of the interface. At this time the server header file, `$(MODULE).h`, must be placed in the `~hival/hival/include` directory and the client code body `$(MODULE)_cstub.c` must be added to the hival makefile target `$(CLIENTS)` and copied into the directory `~hival/hival/interface/src`. In version 1.0, the HiVal client file `server_execution.c` must be edited. Add a new case for the module in the function CallServer, where the literal "`$(MODULE_NAME)`" is identical to the title argument (the first word) in the module's configuration file.

HiVal V1.0 Directory Structure

```
hival/include:  \   database and gnuplot
hival/lib:      /   libraries

hival/interface:        | HiVal client and user interface
     /interface/audio:
     /interface/uid:
     /interface/src:
               /src/uil:

hival/database:
     /database/bin:     | db system calls and question pre-processor binaries
     /database/src:
     /database/config:
              /config/modules:    \   dynamic configuration
              /config/questions:   >  files scanned by HiVal
              /config/rscs:       /   at launch

hival/matlab:

hival/control:
hival/control/Questions:         #
     /control/cs_cont:           | Calspan control models (MATLAB)
             /cs_cont/io:        %
             /cs_cont/src:
     /control/ucbcontrol:        | PATH control models (C)
             /ucbcontrol/io:     %
             /ucbcontrol/src:
                       /src/application:
                       /src/client:
                       /src/server:
     /control/mmccontrol:         / pre- and post-processors,         %
             /mmccontrol/io:     <  static, temporary, and local      %
             /mmccontrol/src:     \ data files                        %

hival/cost:
hival/cost/Questions:     | Perl scripts for processing answers  #

hival/emission:
```

```
hival/emission/Questions:           #
     /emission/mobile5a:            | EPA emissions model (FORTRAN)
               /mobile5a/io:        %
               /mobile5a/src:
                        /src/client:
     /emission/pathemissions:       | PATH emissions model (MATLAB)

hival/safety:
hival/safety/Questions:

hival/traffic:
hival/traffic/Questions:      #
hival/traffic/animator:       | 2D Animator (Xwindow,C)
              /animator/io:   %
              /animator/src:
                       /src/application:
                       /src/client:
                       /src/server:
     /traffic/fresim:         | FHWA freeway simulator (FORTRAN)
              /fresim/io:     %
              /fresim/src:
                      /src/application:
                      /src/client:
     /traffic/shahsam:        | Stealth Animator (DIS,C++,C)
              /shahsam/io:    %
              /shahsam/src:
                       /src/application:
                       /src/client:
                       /src/server:
     /traffic/smartpath:      | PATH traffic simulator (C)
              /smartpath/io:  %
              /smartpath/src:
                        /src/application:
                        /src/client:
                        /src/server:
```

HiVal V1.0 Configuration Files

```
hival/database/config/modules:

control.00.module              Lat/Lon Control       PATH (C)
control.01.module              Hi-Rate Lon Control   PATH (C)
control.02.module              Hi-Rate Lat/Lon Control PATH (C)
control.03.module              Platoon Lon Control   Calspan (Simulink)
cost.00.module                 Time/Cost Calculation P-B (Matlab/Lotus
                                                         1-2-3)
emission.00.module             Mobile5A              EPA (Fortran)
emission.01.module             Power Demand Emissions UCDavis (Matlab)
safety.00.module               Platoon Collision Det. Calspan (Matlab)
traffic.00.module              Fresim/AHS            FHWA (Fortran)
traffic.01.module              Smartpath             PATH (C)
traffic.02.module              2D-Animator           TASC (X,C)
traffic.03.module              3D-Animation Streamer TASC (DIS,C++)

hival/database/config/questions:

control.00.question          Position Trace
control.01.question          Velocity Trace
control.02.question          Acceleration Trace
cost.00.question             Dollars Saved
emission.00.question         Particulate Table
```

```
emission.01.question          HC Exhaust Trace
emission.02.question          CO Exhaust Trace
emission.03.question          NOx Exhaust Trace
safety.00.question            Delta-v Histo
traffic.00.question           Trip Duration Histo
traffic.01.question           Throughput Capacity Histo
traffic.02.question           Average Speed per Lane
traffic.03.question           Velocity Histogram
traffic.04.question           Acceleration Histogram
traffic.05.question           Exit Success Rate
traffic.06.question           Lane Change Success Rate
traffic.07.question           2D-Animation
traffic.08.question           3D-Animation Stream

hival/database/config/rscs:

rsc01.rsc
rsc02.rsc
rsc03.rsc
rsc04.rsc
rsc05.rsc
rsc06.rsc
rsc07.rsc
rsc08.rsc
rsc09.rsc
rsc10.rsc
rsc11.rsc
rsc12.rsc
rsc13.rsc
```

6.3 *Constructing Representative System Configurations (RSC)*

HiVal currently implements the twelve RSC developed by Calspan under the AHS PSA program. These RSC are defined in terms of three parameters: infrastructure impact, vehicle intelligence, and communications. They have also been mapped to an additional set of descriptive parameters, consistent with some of the global RSC consolidation work performed by MITRE during the PSA project.

The RSC parameter file contains the current set of twelve representative system configuration parameters and their possible values. If new parameters are added to this file, each of the `rscXX.rsc` files must be structurally augmented with the new parameter and an appropriate value. If new values are added to an existing parameter , modification of the `rscXX.rsc` files is not necessary unless the new value supersedes a previous value.

To create a new representative system configuration, construct the file `` `rscXX.rsc' `` where "XX" is the next integer in the series of existing file names. The file must have a title on the first non-comment line (a comment line is any line beginning with #). The file must then have a single line entry consisting of a colon separated parameter name/value pair for every RSC parameter. The parameter entries may be in any order and separated by an arbitrary number of comment lines.

RSC Parameter File:

```
infrastructure_impact ::
                          High :
                          Low :
traffic_synchronization ::
                          Asynchronous operation :
                          Mixed synch/asynch :
                            Highly synchronized :
instrumentation_dist ::
                        Smart vehicle :
                        Mixed vehicle/roadway :
                        Smart roadway :
operating_speed ::
                         Low :
                         Variable :
                         High :
vehicle_class ::
                         Light :
                         Heavy :
vehicle_road_interaction ::
                         Rubber tire :
                         Pallet :
power ::
                         On-board :
                         Roadway-provided electric :
headway_strategy ::
                         Single vehicles only :
                         Platoons possible :
lat_control_strategy ::
                         Passive infrastructure :
                         Active infrastructure :
long_control_strategy ::
                         Passive infrastructure :
                         Active infrastructure :
control_location ::
                         Mostly vehicle :
                         Vehicle & infrastructure :
                         Mostly infrastructure :
ahs_lanes_and_access ::
                         Ramps :
                         Transition lanes :
                         Mixed traffic :
```

6.4 *Measures of Effectiveness*

HiVal uses a variety of pre-specified MOEs that are used to support consistent, automated analyses. It is easy to add additional specific MOEs to accommodate the needs of individual users. A directory listing of the MOE function implementations is presented below, along with examples of two types of implemented MOEs that can be used as templates for developing additional customized MOEs.

HiVal V1.0 Measures of Effectiveness

```
hival/control/Questions:
```

```
PlotAccelerationTrace*
PlotPositionTrace*
PlotVelocityTrace*

hival/cost/Questions:

PlotValueSaved*

hival/emission/Questions:

PlotCOExhaustTrace*
PlotHCExhaustTrace*
PlotNoxExhaustTrace*
ShowParticulateEmissions*

hival/safety/Questions:

PlotDeltaVelocityHistogram*

hival/traffic/Questions:

PlotAccelerationHistogram*
PlotExitPerformanceBarGraph*
PlotLaneChangePerformanceBarGraph*
PlotMeanVelocityBarGraphs*
PlotThroughputCapacityBarGraphs*
PlotTripDurationHistogram*
PlotVelocityHistogram*
```

Graphical Display -- An example of a MOE (i.e. Question) which displays its results as a plot function. This example is for a CO (Carbon Monoxide) exhaust trace from an emissions model.

```
#!/usr/local/bin/perl
#-----------------------------------------------------------------------#
# Title:            PlotCOExhaustTrace
# Date:             11/11/94
# Revision History: None.

# This perl script plots the Carbon Monoxide traces in response
# to the HiVal question "CO Exhaust Trace"

# COMMAND LINE PARAMETERS:
#  AnswerFileName - the name of the file to which output is written.
#   Format for AnswerFileName is dictated by the
#   single_plot_form.uil file which specifies the X/Motif
#   display for this question's answer.  Format is as follows:
#    PlotFile - name of the file to which the GNUPLOT is written.
#    Title - title for the file in PlotFile.

# ERROR HANDLING: Errors such as incorrect number of command line arguments
# are trapped and diagnostic text is written to STDERR via the perl "die"
# command.

# HiVal DATABASE DEPENDENCIES: 1 record(s):
# CoExhaustTrace
# The database utilization defined above should be reflected in the
# question configuration file emission.01.question.
```

```
# See "Programming Perl" by Larry Wall and Randall Schwartz
# (O'Reilly copyright 1991) for complete perl syntax descriptions.
#------------------------------------------------------------------------#

# Check the number of input arguments: $#ARGV = 0 means 1 argument was
# passed not counting the name of this script.
if ($#ARGV != 1)
{
  die "Incorrect number of arguments.\n" .
  "Stopped";
}

# Initialize some default variables.
$DB = $ENV{'HOME'} . "/hival/database/hival.db";
$SEM = $ARGV[0];
$answerfile = $ARGV[1];

# Determine unique name(s) for the plot file(s) using the Solaris tempnam
# C library function - place file(s) in /tmp with a prefix of "hival."
$plotfile = `tempnam /tmp hival`;

# Check the exit value of the `tempnam...` call.
# Exit if it is nonzero.
if ($? != 0)
{
  die "Could not create a temporary plot file name.\n" .
  "Stopped";
}

# Remove the newlines from the end of the plot file names:
chop $plotfile;
$datafile = $plotfile . ".dat";

# Determine the location of the exhaust trace data file.
@tracefilenamerow = `val_db $DB CoExhaustTrace`;
# Check the exit value of the `val_db...` call.
# Exit if it is nonzero.
if ($? != 0)
{
  die "Could not determine file name of the CO exhaust trace: ".
  "Stopped";
}
$tracefilename = $tracefilenamerow[2];
chop ($tracefilename);

if (!open(INPUTFILE, $tracefilename))
{
       die "Could not open temporary input file.\n" .
           "Stopped";
}
# Assuming we have not exited, INPUTFILE must be available.
# Determine the time range and co range.
while (<INPUTFILE>)
{
   next if /^#/;
   next if /^$/;
   s/^\s+//;
   s/\s+/ /g;
   chop($exhaustrow = $_);
   ($time, $co) = split (/ /,$exhaustrow);
   if ($. == 1)
   {
      $timemin = $time;
```

```
      $timemax = $time;
      $comin = $co;
      $comax = $co;
   }
   if ($time < $timemin)
   {
      $timemin = $time;
   }
   if ($time > $timemax)
   {
      $timemax = $time;
   }
   if ($co < $comin)
   {
      $comin = $co;
   }
   if ($co > $comax)
   {
      $comax = $co;
   }
}
close(INPUTFILE);
# Put link the data to the place where plot_trace can find it.
`ln -s $tracefilename $datafile`;
if ($? != 0)
{
  die "Could not link the co exhaust data to GNUPLOT input.\n" .
  "Stopped";
}

# Draw the graphs.
`plot_trace $timemin $timemax $comin $comax \"CO Exhaust\" \"Time (s)\" \"CO
(g/mile)\" $plotfile`;

# Check the exit value of the `plot_trace...` call.
# Exit if it is nonzero.
if ($? != 0)
{
  die "Could not GNUPLOT the co exhaust trace on emission output.\n" .
  "Stopped";
}

open ($answerhandle,">$answerfile") || die "Could not open the answer file.\n"
.
  "Stopped";
printf ($answerhandle "%s\n", $plotfile);
close ($answerhandle);

`touch $SEM`;
# This perl script has run successfully.
exit (0);
```

Tabular Display -- Below is an example of a MOE (i.e. Question) which displays its results in tabular form. This example is for an emissions model.

```
#!/usr/local/bin/perl
#-----------------------------------------------------------------#
# Title:            ShowParticulateEmissions
# Date:             9/30/94
```

```
# Revision History: None.

# This perl script posts the particulate emissions in response
# to the HiVal question "Particalate Emissions."

# COMMAND LINE PARAMETERS:
#  AnswerFileName - the name of the file to which output is written.
#   Format for AnswerFileName is dictated by the
#   emission_form.uil file which specifies the X/Motif
#   display for this question's answer.  Format is as follows:
#    NOxCarEmissions - total NOx emissions in grams from cars.
#    COCarEmissions - total CO emissions in grams from cars.
#    HCCarEmissions - total HC emissions in grams from cars.
#    NOxTruckEmissions - total NOx emissions in grams from trucks.
#    COTruckEmissions - total CO emissions in grams from trucks.
#    HCTruckEmissions - total HC emissions in grams from trucks.

# ERROR HANDLING: Errors such as incorrect number of command line arguments
# are trapped and diagnostic text is written to STDERR via the perl "die"
# command.

# HiVal DATABASE DEPENDENCIES: 2 record(s):
#  CarEmissions
#  TruckEmissions
# The database utilization defined above should be reflected in the
# question configuration file emission.00.question.

# See "Programming Perl" by Larry Wall and Randall Schwartz
# (O'Reilly copyright 1991) for complete perl syntax descriptions.
#-------------------------------------------------------------------#

# Check the number of input arguments: $#ARGV = 0 means 1 argument was
# passed not counting the name of this script.
if ($#ARGV != 1)
{
  die "Incorrect number of arguments.\n" .
  "Stopped";
}

# Initialize some default variables.
$DB = $ENV{'HOME'} . "/hival/database/hival.db";
$SEM = $ARGV[0];
$answerfile = $ARGV[1];

# Retrieve the emissions data from the database:
@caremissions = `val_db $DB CarEmissions`;

# Check the exit value of the `val_db...` call.
# Exit if it is nonzero.
if ($? != 0)
{
  die "Could not determine value of CarEmissions " .
  "database record.\n" .
  "Stopped";
}

@truckemissions = `val_db $DB TruckEmissions`;

# Check the exit value of the `val_db...` call.
# Exit if it is nonzero.
if ($? != 0)
{
  die "Could not determine value of TruckEmissions " .
```

```
  "database record.\n" .
  "Stopped";
}

# Remove the newline characters from the car and truck emission variables.
chop $caremissions;
chop $truckemissions;;

open ($answerhandle,"> $answerfile") || die
  "Could not open specified answer file.\n" .
  "Stopped";
for ($i = 2; $i <= $#caremissions; $i++)
{
  printf ($answerhandle "%.3f\n", $caremissions[$i]);
}
for ($i = 2; $i <= $#truckemissions; $i++)
{
  printf ($answerhandle "%.3f\n", $truckemissions[$i]);
}
close ($answerhandle);

`touch $SEM`;
# This perl script has run successfully.
exit (0);
```

6.5 *Software Wrappers and RPC*

Two important architectural elements of HiVal are RPC (Remote Procedure Call) and software wrappers. RPC enable function-like calls between different software models in HiVal, especially in the case of modules that work iteratively or with feedback. Software wrappers allow code written in different languages and/or running on different hardware to communicate. Figure 6-1 illustrates the relationship of RPC and wrappers in HiVal.

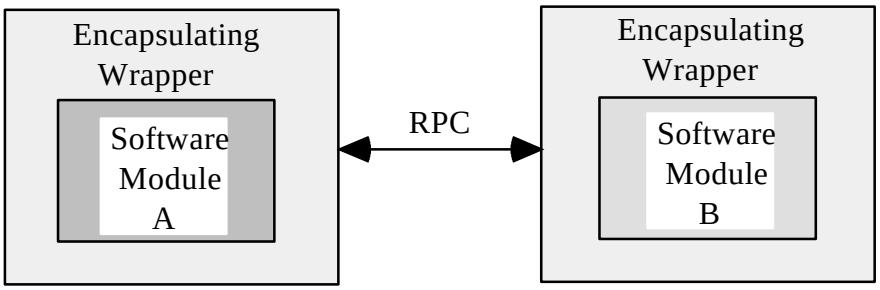

Figure 6-1 Wrappers and RPC Relationship

An example of key elements of HiVal's wrapper and RPC code for the FRESIM / AHS traffic model is given below.

HiVal FRESIM / AHS Wrapper Interface

```
/*
 * Copyright (c) 1991, 1992, 1993 by Gradient Technologies, Inc.
 * All rights reserved.
 *
 * fresim.c
 *
 * Implementation of Fresim/AHS interface in HiVal
 *
 */
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include "fresim.h"
#include "util.h"

void FreeSim (
    handle_t h,
    idl_char *client_greeting,
    idl_char *server_reply
)
{
    UINT  wReturn;
    char *semaphore;
    char *client;
    char *date;
    char  szMsg[80];

    client = strtok (client_greeting, ":");
    semaphore = strtok (NULL, ":");
    date = strtok (NULL ,"");
    printf ("Client: %s %s\n", client, date);
    wReturn = WinExec("/hival/fresim/fresim.pif", SW_SHOWMINIMIZED);
    if (wReturn < 32)
       sprintf (szMsg,"%d",wReturn);
    else
       sprintf (szMsg,"0");

    strcpy(server_reply,szMsg);
}
```

HiVal FRESIM / AHS RPC Interface

```
/* Generated by IDL compiler version DCE 1.0.0-1 */
#ifndef FresimIf_v1_0_included
#define FresimIf_v1_0_included

#if (__STDC__ != 1)
#define volatile
#endif

#include <dce/idlbase.h>
#include <dce/rpc.h>

#ifdef __cplusplus
    extern "C" {
#endif
```

```
#include <dce/nbase.h>
#define REPLYSIZE (100)
extern void FreeSim(
#ifdef IDL_PROTOTYPES
    /* [in] */ handle_t h,
    /* [in] */ idl_char client_greeting[],
    /* [out] */ idl_char server_reply[100]
#endif
);
typedef struct FresimIf_v1_0_epv_t {
void (*FreeSim)(
#ifdef IDL_PROTOTYPES
    /* [in] */ handle_t h,
    /* [in] */ idl_char client_greeting[],
    /* [out] */ idl_char server_reply[100]
#endif
);
} FresimIf_v1_0_epv_t;
extern rpc_if_handle_t FresimIf_v1_0_c_ifspec;
extern rpc_if_handle_t FresimIf_v1_0_s_ifspec;

#ifdef __cplusplus
    }
#endif

#endif
```

6.6 *Linkage Functions*

HiVal's linkage functions provide the analytic connections that allow independent simulation models to be correctly integrated together. For a new module to be installed in HiVal, both input (pre-processor) and output (post-processor) linkage functions are specified. Inputs and outputs are transferred via HiVal's central database, as illustrated in Figure 6-2.

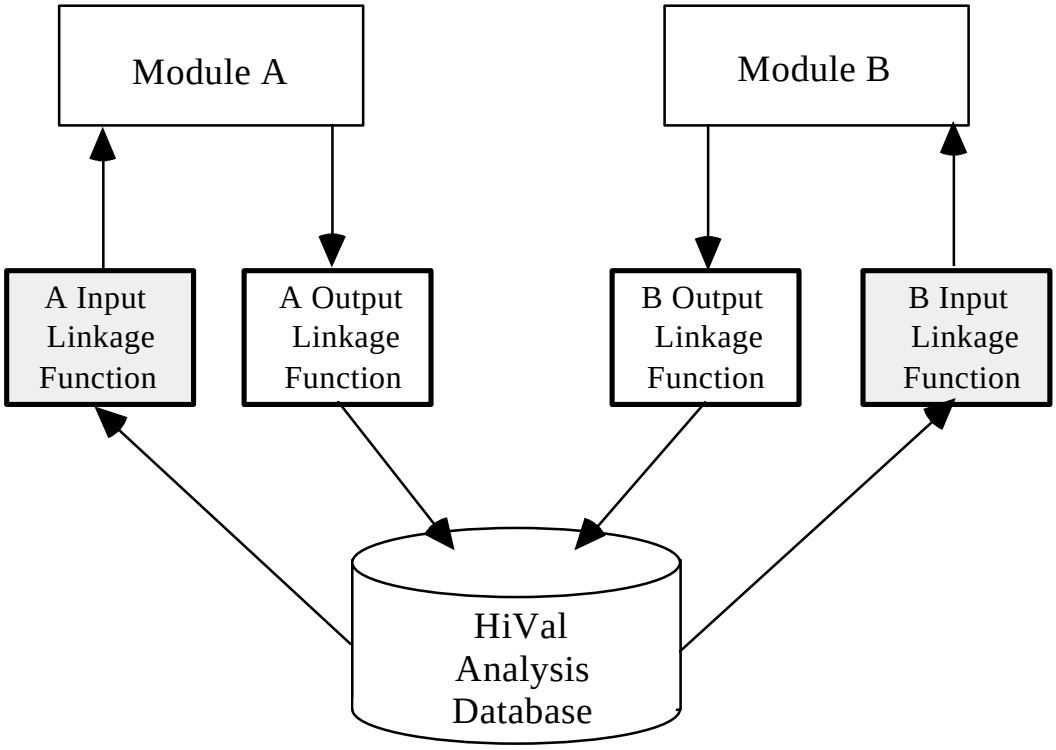

Figure 6-2 Pre- and post-processing linkage functions in HiVal

Examples of input and output linkage functions for both the FRESIM / AHS traffi c simulation and the Mobile5a emission model are presented below.

HiVal Input Linkage Function for FRESIM / AHS

```
#!/usr/local/bin/perl
# Title:            fresim.pre
# Date:             4 Oct 94
# Revision History: None.

# This perl script performs the pre-processing of static and
# database data as defined in the HiVal infrastructure.
# Accepts 2 command-line parameters:
#     pathname    # pointing to the local storage directory
#       semaphore   # key for process thread control
#
# Utilizes 8 record(s) from the HiVal database:
#     RoadWayLayout
#     RandomNumberSeed
#     TotalSimulationTime
#     SimulationTimeStep
#     AHSTargetSpeed
#     NonAHSTargetSpeed
#     IntraplatoonDistance
#     TargetPlatoonSize
#
```

```
# The database utilization defined above should be reflected in the
# module configuration file traffic.XX.module, where XX is the unique
# two digit integer corresponding to the Fresim/AHS server module.
# Errors such as incorrect number of command line arguments shall be
# trapped and diagnostic text is written to STDERR via the perl "die"
# command. See "Programming Perl" by Larry Wall and Randall Schwartz
# (O'Reilly copyright 1991) for complete perl syntax descriptions.

# Check the number of input arguments: $#ARGV = 0 means 1 argument was
# passed not counting the name of this script.
if ($#ARGV != 1)
{
  die "Incorrect number of arguments to fresim.pre.\n" .
  "Stopped";
}

chop ($HIVAL_DIR = `pwd`);
chdir ($ARGV[0]);
$SEM = $ARGV[1];

# Initialize some default variables.
$DBPATH = $ENV{'HOME'} . "/hival/database";
$DB = $DBPATH . "/hival.db";
$roadwayfile = "roadway.info";
$fresimfile = "fresim.in";
$exitfile = "exit.info";

@geomrow = `val_db $DB RoadWayLayout`;
if ($? != 0)
{
  die "Fresim pre-processor failed on fetch.\n" .
  "Stopped";
}
chop @geomrow;
$geomfile = $geomrow[2];

@randomnumberseedrow = `val_db $DB RandomNumberSeed`;
if ($? != 0)
{
  die "Fresim pre-processor failed on fetch.\n" .
  "Stopped";
}
$randomnumberseed = $randomnumberseedrow[2];

@totalsimulationtimerow = `val_db $DB TotalSimulationTime`;
if ($? != 0)
{
  die "Fresim pre-processor failed on fetch.\n" .
  "Stopped";
}
$totalsimulationtime = $totalsimulationtimerow[2];

@simulationtimesteprow = `val_db $DB SimulationTimeStep`;
if ($? != 0)
{
  die "Fresim pre-processor failed on fetch.\n" .
  "Stopped";
}
$simulationtimestep = $simulationtimesteprow[2];
$dumptimestep = $simulationtimesteprow[2];

@ahstargetspeedrow = `val_db $DB AHSTargetSpeed`;
if ($? != 0)
```

```
{
  die "Fresim pre-processor failed on fetch.\n" .
  "Stopped";
}
$ahstargetspeed = $ahstargetspeedrow[2];

@nonahstargetspeedrow = `val_db $DB NonAHSTargetSpeed`;
if ($? != 0)
{
  die "Fresim pre-processor failed on fetch.\n" .
  "Stopped";
}
$nonahstargetspeed = $nonahstargetspeedrow[2];

@intraplatoondistancerow = `val_db $DB IntraplatoonDistance`;
if ($? != 0)
{
  die "Fresim pre-processor failed on fetch.\n" .
  "Stopped";
}
$intraplatoondistance = $intraplatoondistancerow[2];

@targetplatoonsizerow = `val_db $DB TargetPlatoonSize`;
if ($? != 0)
{
  die "Fresim pre-processor failed on fetch.\n" .
  "Stopped";
}
$targetplatoonsize = $targetplatoonsizerow[2];

# filter roadway geometry information
`grep '^%' $geomfile | sed 's/%//' > $roadwayfile`;
if ($? != 0)
{
  die "Fresim pre-processor failed on geometry file.\n" .
  "Stopped";
}
`grep '^&' $exitfile | sed 's/&//' >> $roadwayfile`;
if ($? != 0)
{
  die "Fresim pre-processor failed on exit file.\n" .
  "Stopped";
}

$nats = sprintf("%2.0f", 2.237*$nonahstargetspeed);
$ats = sprintf("%2.0f", 2.237*$ahstargetspeed);
$ipd = sprintf("%4.0f", 100*$intraplatoondistance/$ahstargetspeed);
$tps = sprintf("%4.0f", $targetplatoonsize);
$tst = sprintf("%4d", $totalsimulationtime);
$sts = sprintf("%5.0f", 10*$simulationtimestep);
$rns = sprintf("%8d", $randomnumberseed);
if ($simulationtimestep < 1)
{
   $dts = sprintf("%4d", 1);
}
else
{
   $dts = sprintf("%4d", $dumptimestep);
}

`grep -v '^[#%&]' $geomfile | sed 's/:IntraplatoonDistance:/$ipd/g;
s/:AHSTargetSpeed:/$ats/g; s/:NonAHSTargetSpeed:/$nats/g;
s/:TargetPlatoonSize:/$tps/g; s/:TotalSimulationTime:/$tst/g;
```

```
s/:SimulationTimeStep:/$sts/g; s/:DumpTimeStep:/$dts/g;
s/:RandomNumberSeed:/$rns/g' | nawk '{print \$0 "\r"}' > $fresimfile `;

if ($? != 0)
{
  die "Fresim pre-processor failed on fresim.in.\n" .
  "Stopped";
}

# Return to the caller's directory.
chdir ($HIVAL_DIR);
`touch $SEM`;
# This perl script has run successfully.
exit (0);
```

HiVal Output Linkage Funtion for FRESIM / AHS

```
#!/usr/local/bin/perl
# Title:            fresim.post
# Date:             4 Oct 94
# Revision History: None.

# This perl script performs the post-processing of a Fresim/AHS output
# as defined in the HiVal infrastructure.
# Accepts 2 command-line parameter:
#     pathname    # pointing to the local storage directory
#       semaphore   # key for process thread control
#
# Utilizes 34 record(s) from the HiVal database:
#     RoadWayLayout
#     VehicleTrace
#     RoadWayInfo
#     RoadWayGeometry
#     XMilesLengthOfTrip
#     YMilesLengthOfTrip
#     XVMTMix
#     YVMTMix
#     XTripDuration
#     YTripDuration
#     XThroughputCapacityAutomated
#     YThroughputCapacityAutomated
#     XThroughputCapacityManual
#     YThroughputCapacityManual
#     XVelocityAverageAutomated
#     YVelocityAverageAutomated
#     XVelocityAverageManual
#     YVelocityAverageManual
#     XAutomatedVelocityHistogram
#     YAutomatedVelocityHistogram
#     AutomatedVelocityMean
#     AutomatedVelocitySigma
#     XManualVelocityHistogram
#     YManualVelocityHistogram
#     ManualVelocityMean
#     ManualVelocitySigma
#     XAutomatedAccelerationHistogram
#     YAutomatedAccelerationHistogram
#     AutomatedAccelerationMean
#     AutomatedAccelerationSigma
#     XManualAccelerationHistogram
#     YManualAccelerationHistogram
```

```
#     ManualAccelerationMean
#     ManualAccelerationSigma
#
# The database utilization defined above should be reflected in the
# module configuration file traffic.XX.module, where XX is the unique
# two digit integer corresponding to the Fresim/AHS server module.
# Errors such as incorrect number of command line arguments shall be
# trapped and diagnostic text is written to STDERR via the perl "die"
# command. See "Programming Perl" by Larry Wall and Randall Schwartz
# (O'Reilly copyright 1991) for complete perl syntax descriptions.

# Check the number of input arguments: $#ARGV = 0 means 1 argument was
# passed not counting the name of this script.
if ($#ARGV != 1)
{
  die "Incorrect number of arguments to fresim.post.\n" .
  "Stopped";
}

chop ($HIVAL_DIR = `pwd`);
chdir ($ARGV[0]);
$SEM = $ARGV[1];

# Initialize some default variables.
chop($CURRPATH = `pwd`);
$DBPATH = $ENV{'HOME'} . "/hival/database";
$DB = $DBPATH . "/hival.db";
$dumpfile = "vhdump";
$roadfile = "roadway.info";
$tempfile = `tempnam /tmp hival`;

`highway $dumpfile $roadfile > $tempfile`;

# Check the exit value of the `highway...` call.
# Exit if it is nonzero.
if ($? != 0)
{
  die "Highway parser failed.\n" .
  "Stopped";
}

# Assuming we have not exited, tempfile must be available.
# Put each line into the database.
if (!open(TEXTFILE, $tempfile))
{
       die "Could not open temporary data file.\n" .
           "Stopped";
}
while (<TEXTFILE>)
{
   next if /^#/;
   next if /^$/;
   `put_db $DB $_`;
   if ($? != 0)
   {
       print ("Database insertion failed on line " .
              $., " of ", $tempfile, ".\n");
       die "Stopped";
   }
}
close (TEXTFILE);
'rm $tempfile';
```

```
@geomrow = `val_db $DB RoadWayLayout`;
if ($? != 0)
{
  die "Fresim post-processor failed on fetch.\n" .
  "Stopped";
}
chop @geomrow;
$geomfile = $geomrow[2].".geom";

`put_db $DB VehicleTrace "tscalar : filename : $CURRPATH/$dumpfile"`;
if ($? != 0)
{
  die "Database insertion failed.\n" .
  "Stopped";
}

`put_db $DB RoadWayInfo "tscalar : filename : $CURRPATH/$roadfile"`;
if ($? != 0)
{
  die "Database insertion failed.\n" .
  "Stopped";
}

`put_db $DB RoadWayGeometry "tscalar : filename : $CURRPATH/$geomfile"`;
if ($? != 0)
{
  die "Database insertion failed.\n" .
  "Stopped";
}

chdir ($HIVAL_DIR);
`touch $SEM`;
# This perl script has run successfully.
exit (0);
```

HiVal Input Linkage Funtion for Mobile5a

```
#!/usr/local/bin/perl
# Title:            mobile5a.pre
# Date:             5 Oct 94
# Revision History: None.

# This perl script performs the pre-processing of static and
# database data as defined in the HiVal infrastructure.
# Accepts 2 command-line parameter:
#       CarEmissions
#       TruckEmissions
#
# Utilizes 4 record(s) from the HiVal database:
#     ScenarioName
#     AutomatedVelocityMean
#     YMilesLengthOfTrip
#     YVMTMix
#
# The database utilization defined above should be reflected in the
# module configuration file emission.XX.module, where XX is the unique
# two digit integer corresponding to the Mobile5a server module.
# Errors such as incorrect number of command line arguments shall be
# trapped and diagnostic text is written to STDERR via the perl "die"
# command. See "Programming Perl" by Larry Wall and Randall Schwartz
# (O'Reilly copyright 1991) for complete perl syntax descriptions.
```

```
# Check the number of input arguments: $#ARGV = 0 means 1 argument was
# passed not counting the name of this script.
if ($#ARGV != 1)
{
  die "Incorrect number of arguments to mobile5a.pre.\n" .
  "Stopped";
}

chop ($HIVAL_DIR = `pwd`);
chdir ($ARGV[0]);
$SEM = $ARGV[1];

# Initialize some default variables.
$DBPATH = $ENV{'HOME'} . "/hival/database";
$DB = $DBPATH . "/hival.db";
$mobilefile = "mobile5a.in";
$staticfile = "static.data";

@scenarionamerow = `val_db $DB ScenarioName`;
if ($? == 0)
{
   $scenario = $scenarionamerow[2];
   chop $scenario;
   $scenarioname = sprintf("%14s", $scenario);
}
else
{
   print "Using default scenario name: HiVal\n";
   $scenarioname = "HiVal   client";
}

@velocityrow = `val_db $DB AutomatedVelocityMean`;
if ($? != 0)
{
  die "Mobile5a pre-processor failed on fetch.\n" .
  "Stopped";
}
$vel = $velocityrow[2];
chop $vel;
$velocity = sprintf("%4.1f", $vel);

@lengthoftriprow = `val_db $DB YMilesLengthOfTrip`;
if ($? != 0)
{
  die "Mobile5a pre-processor failed on fetch.\n" .
  "Stopped";
}
chop @lengthoftriprow;
# Any single element in the lengthoftrip array
# cannot be unity because of the format required
# by Mobile5a.
if ($lengthoftriprow[2] == 1.0)
{
   $lengthoftriprow[2] = .999;
   $lengthoftriprow[3] = .001;
}
if ($lengthoftriprow[3] == 1.0)
{
   $lengthoftriprow[3] = .999;
   $lengthoftriprow[2] = .001;
}
if ($lengthoftriprow[4] == 1.0)
```

```
{
   $lengthoftriprow[4] = .999;
   $lengthoftriprow[7] = .001;
}
if ($lengthoftriprow[5] == 1.0)
{
   $lengthoftriprow[5] = .999;
   $lengthoftriprow[7] = .001;
}
if ($lengthoftriprow[6] == 1.0)
{
   $lengthoftriprow[6] = .999;
   $lengthoftriprow[4] = .001;
}
if ($lengthoftriprow[7] == 1.0)
{
   $lengthoftriprow[7] = .999;
   $lengthoftriprow[5] = .001;
}
if ($lengthoftriprow[8] == 1.0)
{
   $lengthoftriprow[8] = .999;
   $lengthoftriprow[4] = .001;
}
$lengthoftrip = sprintf("%5.1f%5.1f%5.1f%5.1f%5.1f%5.1f",
                        $lengthoftriprow[2]*100,
                        $lengthoftriprow[3]*100,
                        $lengthoftriprow[4]*100,
                        $lengthoftriprow[5]*100,
                        $lengthoftriprow[6]*100,
                        ($lengthoftriprow[7] + $lengthoftriprow[8])*100);
@vmtmixrow = `val_db $DB YVMTMix`;
if ($? != 0)
{
  die "Mobile5a pre-processor failed on fetch.\n" .
  "Stopped";
}
chop @vmtmixrow;
# Any single element in the vmtmix array
# cannot be unity because of the format required
# by Mobile5a.
if ($vmtmixrow[2] == 1.0)
{
   $vmtmixrow[2] = .999;
   $vmtmixrow[3] = .001;
}
if ($vmtmixrow[3] == 1.0)
{
   $vmtmixrow[3] = .999;
   $vmtmixrow[2] = .001;
}
if ($vmtmixrow[4] == 1.0)
{
   $vmtmixrow[4] = .999;
   $vmtmixrow[7] = .001;
}
if ($vmtmixrow[5] == 1.0)
{
   $vmtmixrow[5] = .999;
   $vmtmixrow[7] = .001;
}
if ($vmtmixrow[6] == 1.0)
{
```

```
   $vmtmixrow[6] = .999;
   $vmtmixrow[4] = .001;
}
if ($vmtmixrow[7] == 1.0)
{
   $vmtmixrow[7] = .999;
   $vmtmixrow[5] = .001;
}
if ($vmtmixrow[8] == 1.0)
{
   $vmtmixrow[8] = .999;
   $vmtmixrow[4] = .001;
}
# The order of the indices in the vmtmix array of the
# following sprintf are the transformation from HiVal
# vehicle types to Mobile5a vehicle types.
#     LDGV  --- high performance car            [3]
#     LDGT1 --- low performance car       [2]
#     LDGT2 --- low performance car       [2]
#     HDGV  --- single unit truck         [6]
#     LDDV  --- bus                       [8]
#     LDDT  --- semi-trailer (small load) [4]
#     HDDV  --- semi-trailer (fuil load)  [7]
#     MC    --- no motorcycles in HiVal

$_ = sprintf ("%.3f %.3f 0.000 %.3f %.3f %.3f %.3f 0.000",
              $vmtmixrow[3],
              $vmtmixrow[2],
              $vmtmixrow[6],
              $vmtmixrow[8],
              $vmtmixrow[4],
              $vmtmixrow[7]);

$sum_v = 0.0;
@temp_out = split(/ /,$_);
foreach(@temp_out)
{
   $sum_v += $_;
}
$vmtmixrow[2] = $temp_out[1] + (1.0 - $sum_v);

$_ = sprintf ("%.3f%.3f0.000%.3f%.3f%.3f%.3f0.000",
              $vmtmixrow[3],
              $vmtmixrow[2],
              $vmtmixrow[6],
              $vmtmixrow[8],
              $vmtmixrow[4],
              $vmtmixrow[7]);
s/0\./\./g;
$vmtmix = $_;

`grep -v '^#' $staticfile | sed 's/:ScenarioName:/$scenarioname/g;
s/:AutomatedVelocityMean:/$velocity/g; s/:YVMTMix:/$vmtmix/g;
s/:YMilesLengthOfTrip:/$lengthoftrip/g' | nawk '{print \$0 "\r"}' >
$mobilefile`;
if ($? != 0)
{
  die "Mobile5a pre-processor failed to write mobile5a.in.\n" .
  "Stopped";
}

# Return to the caller's directory.
chdir ($HIVAL_DIR);
```

```
`touch $SEM`;
# This perl script has run successfully.
exit (0);
```

HiVal Output LinkageFuntion for Mobile5a

```
#!/usr/local/bin/perl
# Title:            mobile5a.post
# Date:             4 Oct 94
# Revision History: None.

# This perl script performs the post-processing of a Mobile5a output
# as defined in the HiVal infrastructure.
# Accepts 2 command-line parameter:
#     pathname     # path to the local data storage area
#     semaphore    # key for process control
#
# Utilizes 2 record(s) from the HiVal database:
#     CarEmissions
#     TruckEmissions
#
# The database utilization defined above should be reflected in the
# module configuration file emission.XX.module, where XX is the unique
# two digit integer corresponding to the Mobile5a server module.
# Errors such as incorrect number of command line arguments shall be
# trapped and diagnostic text is written to STDERR via the perl "die"
# command. See "Programming Perl" by Larry Wall and Randall Schwartz
# (O'Reilly copyright 1991) for complete perl syntax descriptions.

# Check the number of input arguments: $#ARGV = 0 means 1 argument was
# passed not counting the name of this script.
if ($#ARGV != 1)
{
  die "Incorrect number of arguments to mobile5a.post.\n" .
  "Stopped";
}

chop ($HIVAL_DIR = `pwd`);
chdir ($ARGV[0]);
$SEM = $ARGV[1];

# Initialize some default variables.
$DBPATH = $ENV{'HOME'} . "/hival/database";
$DB = $DBPATH . "/hival.db";
$mobilefile = "mobile5a.out";
$tempfile = `tempnam /tmp hival`;

$_ = `grep 'VMT' $mobilefile`;
s/^\s+//;
s/\s+/ /g;
@vmtmixrow = split(/\s/);

# The only emissions data applicable to AHS from
# Mobile5a is the exhaust breakdown of CO, HC, and NOX.
`grep Exhst $mobilefile > $tempfile`;
if ($? != 0)
{
       die "Could not open mobile5a output data file.\n" .
           "Stopped";
}
```

```
if (!open(TEMPFILE, $tempfile))
{
       die "Could not open temporary data file.\n" .
           "Stopped";
}
# Assuming we have not exited, mobile5a.out must be available.
while (<TEMPFILE>)
{
   s/^\s+//;
   s/\s+/ /g;
   @emissions = split(/ /);

   if ($emissions[1] eq "HC:")
   {
      $carhc = $emissions[2]*$vmtmixrow[2] +
               $emissions[3]*$vmtmixrow[3] +
               $emissions[4]*$vmtmixrow[4];
      $truckhc = $emissions[6]*$mvtmixrow[5] +
                 $emissions[7]*$vmtmixrow[6] +
                 $emissions[8]*$vmtmixrow[7] +
                 $emissions[9]*$vmtmixrow[8];
   }
   elsif ($emissions[1] eq "CO:")
   {
      $carco = $emissions[2]*$vmtmixrow[2] +
               $emissions[3]*$vmtmixrow[3] +
               $emissions[4]*$vmtmixrow[4];
      $truckco = $emissions[6]*$mvtmixrow[5] +
                 $emissions[7]*$vmtmixrow[6] +
                 $emissions[8]*$vmtmixrow[7] +
                 $emissions[9]*$vmtmixrow[8];
   }
   elsif ($emissions[1] eq "NOX:")
   {
      $carnox = $emissions[2]*$vmtmixrow[2] +
                $emissions[3]*$vmtmixrow[3] +
                $emissions[4]*$vmtmixrow[4];
      $trucknox = $emissions[6]*$mvtmixrow[5] +
                  $emissions[7]*$vmtmixrow[6] +
                  $emissions[8]*$vmtmixrow[7] +
                  $emissions[9]*$vmtmixrow[8];
   }
}
close(TEMPFILE);

$carems = sprintf ("CarEmissions \"fvector : g/mile : 3 : %6.2f %6.2f %6.2f
\"",
                   $carhc, $carco, $carnox);
$truckems = sprintf ("TruckEmissions \"fvector : g/mile : 3 : %6.2f %6.2f
%6.2f \"",
                   $truckhc, $truckco, $trucknox);

`put_db $DB $carems`;
if ($? != 0)
{
  die "Database insertion failed.\n" .
  "Stopped";
}

`put_db $DB $truckems`;
if ($? != 0)o
{
  die "Database insertion failed.\n" .
```

```
  "Stopped";
}
chdir ($HIVAL_DIR);
`touch $SEM`;
# This perl script has run successfully.
exit (0);
```

6.7 *Suggested Programming Guidelines for New Software to be Added to HiVal*

This is the set of suggested programming guidelines for HiVal that was distributed to the AHS PSA team during the course of the program. It contains some high-level guidance on programming techniques to follow when new code is being developed in order to facilitate its incorporation into HiVal.

Guidelines

Introduction -- As part of the PSA contractor team, TASC is developing a computing environment called HiVal for AHS simulation and analysis. The objective of this proof-of-concept project is to provide a computing framework within which simulations, analytic models, and databases can be integrated and made available to FHWA AHS researchers. Through the integration of individual elements, HiVal will support alternative system-level analyses working off of common models, databases and scenarios to facilitate comparative evaluations. It is anticipated that key elements of the HiVal system will be the models, simulations, and scenarios developed under the current PSA activities. Other simulation and modeling elements developed under non-PSA activities will also be included. These guidelines are designed to facilitate the process of integrating many disparate analysis elements written in different languages for different computing systems. However, the suggested guidelines are fairly benign -- they should not place any major constraints on the analytic creativity or software development processes. Many of the recommendations suggest practices you might already plan to implement as part of a modern structured programming project. In addition, these guidelines should not deter PSA teams from exploring areas that may be counter to any one of these recommendations.

Computing Architecture -- In order to facilitate the integration of software elements written in many languages that will interact in varying degrees, HiVal will be implemented using a client/server paradigm. Attachment 1 presents some of the basic concepts involved in open system architectures and client/server systems. While HiVal's integration task is made more complex by the variety of software and computer environments being used for development, it is our desire and purpose to preserve the flexibility and creativity this variety affords. To minimize the impacts of integration on the AHS software developer for this initial HiVal implementation, HiVal will be based on an open system architecture across a (distributed) network. The overall communication scheme is based on the client server / Distributed Computing Environment (DCE).

The prototype computing architecture is being developed in two phases, each defined by the level of interactive communication established between individual computing system elements. In the first phase, the communication will be a very simplified version of the DCE paradigm: inter-element communication will occur through ASCII files. During the second phase these links will be expanded to inter-element and inter-procedural communication using a full-scale DCE paradigm and protocol.

Encapsulation -- The HiVal environment will ultimately provide an "encapsulation toolkit" (or "wrapper") for incorporating individual models into the HiVal testbed with minimal impact on independent software development activities. The goal is to minimize the impact on individual models and preserve their operating environment. Although the encapsulation toolkits are still under development for HiVal, there are recommendations that will facilitate this encapsulation process -- they form the basis for these guidelines. The Level One recommendations apply to all elements, whereas the Level Two recommendations apply primarily to software that is being newly developed, or existing software that can be modified for more complete DCE-compliance. Depending on the nature of the PSA software elements you are developing, you may be able to follow either Level Two (most desirable) or Level One guidelines.

The Level One Guidelines (ASCII file communication) for PSA software development are:

- Provide a batch mode of operation that does not require user interaction, but rather uses ASCII file I/O. For example, a command line option for disabling the interactive mode and enabling the batch mode
- Implement routines which can write all program outputs to an ASCII file (e.g. vs. to a screen display)
- Incorporate command line switches for disabling graphical output

The Level Two Guidelines (Inter-procedural communication) are:

- Use ANSI standard C-language as the primary programming language and maintain ANSI standards
- Avoid the use of global variables, particularly for logical control
- Decompose complicated functions and procedures into multiple, simpler, procedures
- Decouple graphics and interface procedures from computational routines
- Isolate system calls to a single module

7. CONCLUSIONS

In order to meet the present and future needs of AHS concept analysis and tradeoff studies, a prototype integrated modeling, simulation, and decision support testbed, called HiVal, has been developed by FHWA under the AHS PSA Program. HiVal fulfills several major objectives:

- Provide integrated, system-level analysis and decision support for alternative AHS system concept evaluations and tradeoffs
- Preserve and make accessible the software and database products of PSA and other AHS researchers to support future elements of the National AHS research agenda and the NAHSC (National Automated Highway System Consortia)
- Develop a computing system which is flexible and easily extensible that will adapt to and evolve with the rapidly advancing state of AHS knowledge by using combinations of the best individual AHS models, simulations, and databases from a variety of sources

HiVal accommodates a range of user expertise and objectives, ranging from high-level, aggregate AHS performance metrics and tradeoffs, to low-level, detailed simulation of individual AHS subsystem elements. HiVal provides a computing environment which integrates a variety of simulations, models, and databases from both PSA activities and the broader AHS research community. More than a dozen simulations and models have been incorporated into the prototype system. HiVal uses a modular, distributed client-server computing architecture. Modern workstation technology (Unix & X-Motif, DOS/MS Windows, DCE, TCP/IP) is used throughout to support a wide variety of modeling and simulation needs, and allow for continued system growth. All of HiVal's basic interfaces, protocols, and control software uses COTS (Commercial Off The Shelf) technology. A functional prototype of HiVal has been implemented and provided to FHWA.

REFERENCES

1. Interim AHS Results Summary, Calspan Corporation, Buffalo, NY, April 1994

2. Stevens, W. B., Automated Highway System (AHS) Concepts Evaluation (Draft), MTR-94W000xxx, The Mitre Corporation, MacLean VA, June 1994

3. IEEE Standard for Information technology -- Protocols for Distributed Interactive Simulation Applications, IEEE-Std 1278-1993, IEEE Inc., Piscataway, NJ, May 1993

4. Introduction to OSF™ DCE, Revision 1.0, Open Software Foundation, Cambridge, MA, 1992