The SHIFT language is currently being used to create applications for detailed specification and design of vehicle/highway control systems, simulation of formation of mini-submarines for marine survey, design and simulation of air traffic control system, material handling systems modeling.

7. References

[1] H. Al-Deek, M. Martello, A. May, and W. Sanders. "Potential Benefits of In-Vehicle Information Systems in a Real Freeway Corridor under Recurring and Incident Induced Congestion", UCB-ITS-PRR-88-2, Institute of Transportation Studies, University of California, Berkeley, CA 94720, 1988.

[2] H. Al-Deek and A. Kanafani. "Some Theoretical Aspects of the Benefits of En-Route Vehicle Guidance", UCB-ITS-PRR-89-2, Institute of Transportation Studies, University of California, Berkeley, CA 94720, 1989.

[3] J.G. Bender. "An overview of systems studies of automated highway systems", IEEE Transactions on Vehicular Technology, Vol. 40, No 1, 1991.

[4] C. Thorpe, M.H. Hebert, T. Kanade, and S.A. Shafer. "Vision and navigation for the Carnegie-Mellon Navlab", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No 3, 1988.

[5] D. Koller, T. Thorhallson, and H. H. Nagel. "Model-based Object Tracking in Traffic Scenes", Euorpean Conference on Computer Vision, S. Margherita, Italy, 1992.

[6] P. Varaiya. "Smart Cars on Smart Roads: Problems of Control", IEEE Trans. Automatic Control, Vol. 38, No 2. Feb. 1993.

[7] IVHS America. Strategic Plan for Intelligent Vehicle-Highway Systems in the United States. Report No IVHS-AMER-92-3. 20 May 1992.

[8] A. Deshpande, D. Godbole, A. Göllü, P. Varaiya. "Design and Evaluation Tools for Automated Highway Systems", In DIMACS 1995 and in Hybrid Systems III, LNCS, Springer-Verlag, 1996.

[9] C.F. Daganzo. "The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory", In Transportation Research, August 1994, V28B-4, P69-287.

[10] M. Broucke, P. Varaiya. "A theory of traffic flow in automated highway systems". In Transpn Res.-C 1996, V4 P181-210.

[11] F. Eskafi, D. Khorramabadi, and P. Varaiya. "An Automated Highway System Simulator", Transportation Research Journal, part C, Vol. 3, No 1, 1995.

[12] A. Göllü, P. Varaiya. "SmartAHS: An Object Oriented Simulation Framework for Highway Systems", To Appear in Iternational Journal of Mathematical and Computer Modeling.

[13] M. Andersson. "Discrete event modelling and simulation in Omola", in: 1992 IEEE Symposium on Computer-Aided Control System Design (CACSD), New York, NY, USA: IEEE, 1992. p. 262-8.

[14] H. Schwetman. "CSIM Reference Manual (Revision13)", Microelectronics and Computer Technology Corporation, 3500 West Balcones Center Drive, Austin, TX 78759, 1989.

[15] A. Deshpande, A. Göllü, L. Semenzato. "The SHIFT Programming Language and Run-time System for Dynamic Networks of Hybrid Automata". California PATH Technical Report UCB-ITS-PRR-97-7.

[16] A. Deshpande, A. Göllü, L. Semenzato. "SHIFT Reference Manual", California PATH Technical Report UCB-ITS-PRR-97-8.

[17] A. Deshpande. "AHS Components in SHIFT", PATH technical report, 1996.

next set of users.

The `AutomatedVehicle` creates an architecture within which controllers, sensors, and communication devices can be designed and modeled, e.g. if a vehicle is autonomous, it does not contain a `Transmitter` or `Receiver`. The `AutomatedVehicle` also aggregates the behavior of the types it contains.

The overall architecture created at this level localizes and simplifies the design task for the control and communication engineers that are the next set of users of the framework.

5.4 Control Design in SmartAHS

The control designers may further decompose the controller into a hierarchy, such as feedback controllers responsible for safe execution off follow, lane change, entry and exit activities, and coordination controllers for tactical decisions and for executing inter-vehicle protocol actions which coordinate the maneuvers possibly involving multiple vehicles and the roadway infrastructure. Further hierarchical decomposition of the controller design may be necessary to implement a multi-level controller with multiple maneuvers and a supervisor.

The behavior specification syntax of the SHIFT language merges the design and implementation of controllers. State machines, discrete events, differential equations, and existential queries are used in the specification of controllers.

Controllers specified by different users can easily be integrated into future SmartAHS releases.

Upon completion of this stage an overall concept and control design is ready for evaluation.

5.5 Evaluation in SmartAHS

In order to obtain the working simulator it is necessary to create a scenario by instantiating an initial set of components. The scenario creation steps are: highway geometry description, placement of sinks and sources to represent origin-destination data and flow volumes, specification of weather and road conditions, and placement of the detailed monitors to generate the data to be used for concept evaluation.

A detailed highway layout is obtained by instantiating base highway components in a sequence and with parameter values of a highway being modeled. A graphical highway designer is used to quickly create the SHIFT code for the highway part of the simulator. Instances of `Source` and `Sink` types are connected to the designated locations on the highway layout. Finally custom `Monitor` types may be used to collect specific statistics.

6. Conclusion

In this paper we have presented major features of the SHIFT programming language, requirements for the AHS modeling framework, and the methodology in its implementation.

At this time, the SHIFT simulation environment is available at

http://www.path.berkeley.edu/shift

This includes a compiler that takes a SHIFT specification and translates it into a C file that should be compiled and linked with the run-time library to produce a simulator; a command line (tty) debugger that allows you to browse through program's entities at run time; a Tcl/Tk-based GUI environment that allows to visualize running simulations and debug SHIFT programs; a C Application Program Interface; and full set of the SHIFT language documentation, including description of the mathematical model, language reference, user manual, tutorials and examples. The SmartAHS release can be obtained from:

http://www.path.berkeley.edu/smart-ahs

desired Origin-Destination patterns and flow volumes.

Basic set of `Monitor` types collect necessary statistics used for concept evaluation.

A basic `AutomatedVehicle`[1] consists of several types. It contains a `Vehicle`, which models vehicle dynamics, a `Controller` which is to provide the throttle, steering, and brake inputs, and a `VREP` (Vehicle-Roadway Environment Processor). The AutomatedVehicle may also contain Sensor and Communication devices and their environment processors.

The `VREP` is a logical object that maintains a Vehicle's position on the highway. It performs the coordinate translations between the vehicle coordinate frame, and the roadway and global coordinate frames.

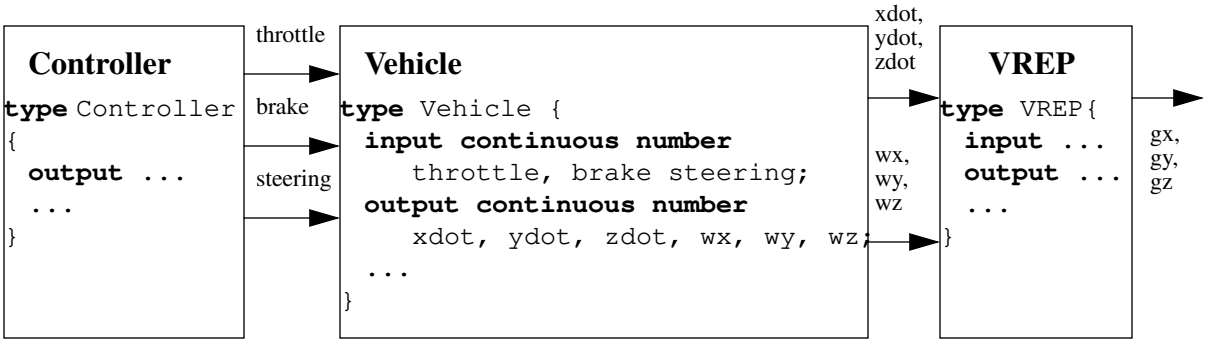

Figure 1: Basic AutomatedVehicle

5.2 Additional SmartAHS Libraries

The next step of the methodology involves specific library development. Using the foundation types, library developers provide a vehicle dynamics model library, detailed sensor, actuator, communication device model libraries, as well as environment processor implementations for these devices.

SmartAHS currently provides a comprehensive set of vehicle dynamical libraries that support different levels of granularity, as well as, ideal sensor and communication device models such as, `SphericalSensor`, `Message`, `Transmitter`, `Receiver` and Receiver Environment Processor (`REP`). Completely different sets of the communication and sensor libraries may be provided by other development groups, and this interchange does not affect other blocks of the SmartAHS framework.

The types provided by these libraries can be used interchangeably and can be combined within the same simulation since they all support the same input and output interfaces (inherited from the base type). The libraries described in this section can be used in **bottom-up** design when going through the architecture design phase (explained in the next section).

5.3 Architecture Specification in SmartAHS

Concept design stage uses these libraries to decide how to best represent a concept by a set of available types. This step combines **bottom-up** and **top-down** design, since for the Control types they only specify the input/output interfaces and delegate their design and implementation to the

1. The SimpleVehicle example in the previous section is not part of SmartAHS. SmartAHS uses AutomatedVehicle as the root type for automated vehicles.

Consider the Buick type above. It may be implemented by a single type, or it can be decomposed into many different types in a complex containment hierarchy. Its implementation has no implications for its use by the SimpleVehicle. Conversely given a very complex implementation of a Buick that consists of many types, we can say that the container type Buick aggregates the behavior of its containees by its input/output interface.

4.4 Specialization of a Top-Down Design

During top-down design it is not always possible to predict all containees of a type and all input, output connections among them. Furthermore specialized containees may have new inputs and outputs that need to be connected.

Consider the SimpleVehicle example above and assume that in the next step of the top-down design we want to create a DetailedVehicle that has a Vehicle Roadway Environment Processor (VREP) that maintains the position of the vehicle in the global coordinate frames. The x, y, z, wx, wy, and wz outputs of the SimpleVehicle must be inputs of the VREP. In SHIFT subtypes do not inherit any behavior from their parents, if DetailedVehicle were to inherit from the SimpleVehicle it would have to re-specify the input/output connections. Instead the DetailedVehicle uses SimpleVehicle and augments it functionality.

```
type DetailedVehicle{
  state VREP myVREP := create(VREP); SimpleVehicle myAV;
  state Vehicle myVehicle; Controller myController;
  ...
  setup do {
     myAV := create(SimpleVehicle, myVehicle := myVehicle,
                                      myControoller := myController); }
        connect {
           x(myVREP) <- x(myVehicle);
           y(myVREP) <- y(myVehicle);
           ...   }
}
```

In the example above, it is assumed that the create statement for the DetailedVehicle initializes the myVehicle, and myController variables.

5. SmartAHS Methodology

In this section we illustrate the application of the Object-Oriented methodology to the creation of an AHS simulation framework which facilitates quick prototyping and efficient implementation of simulators for different AHS concepts, such as independent (autonomous) vehicles, cooperative vehicles, infrastructure-supported designs etc. Particularly, we show the convenience of using the SHIFT language for specifying modules for AHS models and linking them together yielding a simulator for desired level of granularity. The approach we take results in a simulation framework compliant to the requirements discussed in Section 2.

5.1 SmartAHS Building Blocks

Basic SmartAHS building blocks provided by SmartAHS developers are the discussed in [17]. The Highway library provides building blocks to create arbitrary highways. The roadway is represented in terms of components of types Section, Segment, Lane. Sections consist of segments and lanes. Segments represent the geometry of a highway. Highway types reside in the global coordination frame meaning that each point of the highway is referenced by its global (x, y, z) coordinates. Each section has its own coordinate frame.

Sink and Source types provide the flows of vehicles facilitating the representation of

- The discrete state of each component;
- The values of the continuous number variables in each component; and
- The values of number, symbol, and link variables in the components.

Note that we have distinguished continuous number variables from other variables. The values of continuous variables can change as time passes, as well as on transitions. The values of other variables can change only on discrete transitions.

4. Object-Oriented SHIFT Constructs used by SmartAHS

SHIFT supports inheritance in the data model. A subtype inherits all input and output variables and exported events of its parent type and can add new ones. In SHIFT, it is possible to create input-output connections between two types. These two constructs are combined in a number of ways to address traditional software engineering requirements such as top-down and bottom-up design, hierarchical decomposition and aggregation, abstraction, and modularity.

4.1 Bottom-Up Design

Given a set of types, these can easily be combined to create larger more complex types by simply connecting their inputs and outputs. Given a `Controller` with outputs throttle, brake, and steering and a `Vehicle` with the same inputs a `SimpleVehicle` can be created that interconnects the two:

```
type SimpleVehicle {
  state Vehicle myVehicle; Controller myController;
  ...
  setup connect {
     throttle(myVehicle) <- throttle(myController);
     brake(myVehicle) <- brake(myController);
     steering(myVehicle) <- steering(myController);
}
```

4.2 Top-Down Design

In the above example consider that we deliver only vanilla implementations of `Controller` and `Vehicle` types and expect other users to implement the detailed Controller and Vehicle types. In this case the other users will inherit from the framework provided base classes and create their Controllers, e.g. `PATHController`, that conform to the same Input-Output interfaces.

A SimpleVehicle can be created with specialized types:

```
type Buick : Vehicle { ... }
type GMController : Controller { ... }
type Honda : Vehicle { ... }
type PATHController : Controller { ... }
type MiscVehicleSource {
     ...
        SimpleVehicle newVehicle := create(SimpleVehicle,
                                    myVehicle := create(Buick);
                                    myController := create(GMController);
     ...
        SimpleVehicle newVehicle := create(SimpleVehicle,
                                    myVehicle := create(Honda);
                                    myController := create(PATHController);
        ...
}
```

4.3 Hierarchical Decomposition and Aggregation

When a transition is executed numerical and link variable values may be changed and new components can be created as part of the reset actions.

A type can establish input-output connections among variables of types accessible through its link variables. Alternatively, a type can provide algebraic or constant definitions for other types' inputs.

SHIFT supports global variables.

The data model supports inheritance. A subtype inherits the input and output variables and event labels of its parents. It may add new variables and event labels. At this time the behavior model does not support inheritance, primarily because of the difficulty of defining a consistent and useful inheritance operator for differential equations and state machines.

Components evolve in time according to their continuous behavior rules until a discrete transition becomes possible. At that point the discrete transition is executed in zero time. Several transitions can be executed before time passage resumes.

Under the current implementation SHIFT programs are translated into C code and linked with SHIFT run-time libraries to create an executable. SHIFT programs can link in C functions. The run-time executable supports programmatic API, command-line, and graphical interfaces for user interaction.

3.1 Variable Kinds

SHIFT supports the following built-in variable kinds.

CONTINUOUS NUMBER. These are comparable to the *double* data type in C. Continuous numbers evolve, i.e. change their value as time passes, or on discrete transitions. The domain of the continuous numbers are the reals.

NUMBER. These are comparable to the *double* data type in C. Unlike, "continuous" numbers they change their value on discrete transitions only. The domain of the numbers are the reals.

SYMBOL. These are comparable to the *enumerated* data type in C, although in SHIFT, the user does not declare possible values. Symbols are prefixed with the '$' sign, e.g $leftlane, $front. Symbol variables can change their value on discrete transitions only.

LINK. Link or reference variables are comparable to pointers in C. As the name suggests links implement access by reference. Link variables can change their value on discrete transitions only. Link declarations in a type specify the user defined type of the link. The domain of a link variable is a component id of the specified type or one of its subtypes.

SET(). The user can create sets of variables of any kind. Sets are homogeneous and their declaration specifies their kind. There is no order to the elements of a set. The user does not need to specify the size of a set. The user can add/remove elements to/from a set or recreate it.

ARRAY[]. The user can create arrays of variables of any kind. Arrays are homogeneous and their declaration specifies their kind. Array elements are ordered. The array size is specified during its initialization. Once initialized, the user can access or change elements of an array at a given index that is within its current size. At run-time. the user can re-initialize the array, with a different size.

3.2 The World State

During the execution of a SHIFT program the world state is given by the following:

- The type definitions; The type definitions are an implicit part of the world state and are static during execution.
- The set of components; At a given type the world consists of a set of components. This set may change as the world evolves.

model developers who provide detailed vehicle, sensor, and communication device libraries, system engineers who will develop automation architectures, control and communication engineers who will design, implement, and test individual control and communication components; system analysts who will test and evaluate automation strategies; and system planners who will select the automation strategy for deployment based on evaluation results.

Along with traditional software engineering requirements, the framework must satisfy the following requirements: it must provide time and event driven evolution model; it must allow the designers to use a specification language that fits their domain, in this case differential equations and finite state machines; it must provide a structured specification, simulation, and evaluation environment with formal semantics; it must provide constructs that are object-oriented; and it must represent dynamic interaction dependencies.

Additionally the framework must model a number of entities that are particular to simulating vehicles on the highway. These entities must be able to represent arbitrary highways; incoming and outgoing traffic patterns; vehicles consisting of many components; roadside controllers consisting of many components; different types of vehicles on the highway; inter-vehicle and vehicle-to-roadside communication; accidents; and collection of arbitrary statistics.

SHIFT and SmartAHS satisfy these requirements.

3. Overview of the SHIFT language.

SHIFT is a special-purpose object-oriented programming language designed to simulate large hybrid dynamical systems. It bridges the gap between system and control theory, formal methods, and programming languages for a focused yet large class of applications. SHIFT users define types (classes) that are prototypical representations of components. A simulation starts with an initial set of components that are instantiations of these types. At run-time SHIFT assigns a unique id to each component and derives their behavior from their type descriptions.

The data model of a type consists of numerical variables, link variables, a set of discrete states, and a set of event labels. In a type, the user specifies the variable names. In a component, these variables take values. In a type, the user specifies the possible discrete states, at a given time, a component is in one of these discrete states. The variables are grouped into input, state (internal), and output variables. A type has read-only access to its input variables and read/write access to its internal and output variables. Components can access other components through their link variables. Such access is limited to write-only access for inputs and to read-only access for outputs. In SHIFT, write access to a variable constitutes the ability to define the evolution of that variable. Read access constitutes the ability to use that variable in specifying the evolution of variables.

Users specify time evolution rules for the state and output continuous number variables of a type. These rules, called the flow of a type, are given by differential equations and algebraic definitions, alternatively they can assign constant values to variables. Each discrete state can have a different flow. These equations are based on all variables of this type and outputs of other types accessible through link variables. The algebraic definitions cannot have cyclic dependencies.

The discrete behavior is given by a set of transitions among the discrete states. A transition is given by a from state, to state, a set of events, a guard, and reset actions. Events consist of event labels of this type (local events), and event labels on types accessible through link variables (external events). External events create a connection (synchronization) between transitions in different components and require concurrent execution of such transitions. Transitions are executed when guard conditions on variables and synchronization requirements on events hold.

In time, several other highway automation architectures were proposed and it became evident that a generalized simulation framework was needed that could facilitate the specification, simulation, and evaluation of different highway automation architectures and the first version of SmartAHS was developed [12].

In parallel to our AHS work, we were involved with several other projects, such as air traffic management, power transmission and distribution systems, and network management systems. In system engineering, we have observed a general shift towards hierarchical control of large systems that combined classical continuous feedback systems, with more recent discrete event based control algorithms and protocol specifications. This hybrid systems paradigm has proven ideal for the specification, control, and verification of such complex, large, dynamical systems.

Our experience with a multitude of such systems resulted in a set of requirements for frameworks for the design, specification, control, simulation, and evaluation of large dynamical systems. At the same time the National Automated Highway System Consortium (NAHSC) was formed and was charged with the development of the ultimate AHS concept, creating an imminent need for such a framework.

No language, product, or tool in the market nor in academia came close to satisfying all requirements. Many simulation frameworks are available [13] that consist of a set of class libraries developed in a programming language such as C++. These frameworks impose semantic notions such as inputs, outputs, events, differential equations etc. onto the C++ syntax and expect the user to follow framework rules for using the class libraries. This approach does not provide the user with any syntactic support for large scale system development. Several discrete event simulation tools exist. However, these tools do not provide enough support for continuous evolution [14]. Block-diagram based simulation tools are easy to use, but do not provide the necessary expressive power to represent dynamic interaction patterns among the simulated objects.

The design and implementation of a language that addressed all the requirements required expertise from several disciplines including computer science, electrical engineering, and mechanical engineering. Such a multi-disciplinary team was assembled at PATH/UC Berkeley and a new programming language, SHIFT, was born.

The SHIFT formalism [15,16] which we briefly discuss in Section 3 is the first programming language with well defined simulation semantics that addresses the requirements discussed in this paper. SHIFT combines system-theoretic concepts into one consistent and uniform programming language with object-oriented features. Our system is ideal for the design, specification, simulation, control, and evaluation of large dynamical systems that consist of multiple interacting agents whose behavior are described by state machines and ordinary differential equations. Our strength lies in the ability of instantiating agents and evolving the interaction network among them at runtime as part of the simulation. The SHIFT language is currently being used in automated highway planning, air traffic management planning, underwater submarine operation, and material handling system modeling.

The rest of this paper is organized as follows. In Section 2 we discuss the requirements imposed on the simulation framework. Section 3 describes basic concepts of the SHIFT language. In Section 4 the Object-Oriented features of SHIFT are exposed in more detail. Chapter 5 explains the methodology used to create the SmartAHS simulation framework.

2. Simulation Framework Requirements

The simulation framework must address the needs of several categories of users. These are

that with human drivers, there is a limit to the maximum achievable traffic flow (about 2000 vehicles/hour/lane).

• Increase the throughput by automating the decision-making for route selection and control of the vehicle which effectively removes the human driver from the driver seat. This approach claims dramatic improvements in capacity, safety, and energy efficiency and leads to the concept of the Automated Highway System (AHS).

Planning an Automated Highway System is a daunting task. It involves the design and integration of the intelligent vehicles and intelligent highways to increase throughput without compromising safety. There are several proposals about the structure of an AHS and its elements. At one extreme lie proposals in which a centralized controller determines the position of every vehicle similar to the way trains are controlled. These designs were studied by TRW, GM, Rohr Industries and some other groups, and are reviewed in [3]. At the other extreme are proposals that are inspired by robotics and AI-based approaches to the control of an autonomous vehicle navigating in an unstructured and even hostile environment [4,5]. These approaches emphasize recognition, learning, and trajectory planning in the face of diverse threats and obstacles. Proposals in between these extremes include the PATH proposal for AHS (PATH-AHS) [6].

The IVHS strategic plan [7] identifies modeling and simulation as important steps in realizing these transportation initiatives. A software framework in which alternative IVHS strategies can be specified, simulated, and uniformly evaluated is crucial for the cost-effective development and objective comparison of the proposed alternatives. Such a framework must be able to evaluate multiple Measure of Effectiveness (MOE) criteria such as safety, performance, throughput, and environmental impact. Clearly other criteria such as cost, deployability, social reaction will be important elements in the final comparison of various proposals. For a discussion of a complete set of tools that are being developed for the downselection process the reader is referred to [8].

Traditionally simulation granularity of highway traffic has been limited to macro-simulation or meso-simulation. Macro-simulators usually rely on fluid flow models to generate throughput and density information for the highway [9]. Meso-simulators use the fluid flow model to simulate the behavior of the individual vehicles [10]. The modeling challenges in human driver models have made it difficult to have accurate micro-simulation results for highway traffic. However, the elimination of the human driver (or reduction of their role) in automated vehicles obviates (reduces) this problem. Automated vehicles behave deterministically as designed. Hence, most accurate system information is obtained if simulation is at the individual vehicle or even at vehicle component level.

SmartAHS is such a simulation framework. This paper discusses the SHIFT language that was used to develop SmartAHS, the object-oriented constructs of the SHIFT language that have been used in the implementation of SmartAHS, and the development and use methodology of the SmartAHS framework.

It is important to distinguish the actual control and communication design of an automation strategy from its simulation and evaluation. The mandate of the automation strategy is to provide cheap, safe, speedy, comfortable, and clean transportation. The mandate of the framework is the objective comparison of alternative approaches.

Our work on simulation started in the early 90s when PATH (Partners for Advanced Transit and Highways) proposed a specific control hierarchy for the highway automation project. Since we were dealing with a complex and very large system, there was no hope for closed form mathematical analysis. The need for a simulation environment was obvious and SmartPath was developed [11].

OBJECT-ORIENTED DESIGN OF AUTOMATED HIGHWAY SIMULATIONS USING SHIFT PROGRAMMING LANGUAGE.

Aleks Göllü and Mikhail Kourjanski[1]
{gollu, michaelk}@path.berkeley.edu
California PATH
University of California at Berkeley

March 30, 1997

Abstract

SmartAHS is a specification, simulation, and evaluation framework for modeling, control and evaluation of Automated Highway Systems (AHS). SmartAHS is developed using SHIFT, a new programming language with simulation semantics.This paper discusses the requirements that have led to the development of SmartAHS and SHIFT, summarizes the main characteristics of the SHIFT language, and illustrates how the object-oriented features of SHIFT are used in the development of the SmartAHS framework.

1. Introduction.

Traffic congestion is an everyday problem for many people commuting to and from work in metropolitan areas of the United States. In 1995 the average speed of vehicles during peak hours was 35 mph and is expected to drop further to 11 mph by the year 2005. It is estimated that lost productivity due to traffic congestion costs $100 billion each year in the United States. Alongside congestion, safety continues to be a prime concern.

The traditional solution of constructing more highways to meet growing demand is no longer possible in many urban areas. Telecommuting opens a new approach to the congestion problem by keeping people out of the urban centers. However, for the telecommuting to be implementable across the work-force, affordable high speed communication links should be available, and more importantly the business control structure which relies on direct supervision and face-to-face interaction has to change. The expansion of public transportation cannot provide a cost-effective solution in areas facing dispersion of the work-force, and the vast majority of commuters continue to use private automobiles.

Intelligent Vehicles and Highway systems (IVHS) is a comprehensive program initiated by the U.S. Government under the Intermodal Surface Transportation Efficiency Act of 1991 to improve safety, reduce congestion, enhance mobility, minimize environmental impact, save energy, and promote economic productivity in the transportation system. The IVHS program combines several modern technologies, including information processing, communications, control, and electronics. Two main threads of IVHS research are:

- Increase the throughput by providing reliable traffic information and help drivers make better decisions regarding their route selections [1,2]. This approach is limited to the “behavioral law”

1. Supported by the California PATH program of the University of California, in cooperation with the National Automated Highway Consortium, California Business, Transportation, and Housing Agency, and the Department of Transportation