Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

A GENERAL FRAMEWORK FOR LARGE SCALE SYSTEMS DEVELOPMENT

Aleks O. Göllü
Farokh H. Eskafi

Partners for Advance Transit and Highways (PATH) and
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720, U.S.A.

ABSTRACT

This paper describes a general framework for the modeling, design, simulation, and prototyping of large scale systems. The framework uses a coherent set of tools that model the system at hand, take a control design and analyze, verify, and simulate it; and then can generate code that can be run in a target real-time software platform in the physical system. The paper emphasizes the specification language SHIFT and the simulation tools used by the framework. We present the development of the Automated Highway System as an example.

1 INTRODUCTION

Large engineering systems, such as automated highway systems (AHS), autonomous vehicle systems (AVS), material handling systems, air traffic management systems (ATMS) face the challenge of providing reliable services using scarce resources. Clients of such systems demand performance, safety, comfort, and efficiency.

The problem is often compounded by physical resources that are saturated, inefficiently utilized, or technologically outdated. In many industries, failure to improve the performance of such systems results in significant financial or social costs.

Due to the heterogeneity of the system elements and the large system size, the planning and control of such systems cannot always be done in a mathematical framework. Experimentation with the actual physical system is often not feasible; in many cases the physical system is not yet built. Furthermore, most real systems have an abundance of unstructured information, too many superfluous details, no well-defined observation and control mechanisms, and no single access location due to their distributed nature.

Complex software applications are needed to specify, simulate, evaluate, and manage the behavior of such large scale systems. Currently, there are no coherent software tools that can facilitate large scale system development from concept inception to actual deployment. There is a gap between the specification and implementation constructs required to build such systems on the one hand, and the interfaces provided by software design tools and programming languages on the other hand.

This paper describes a general framework for the modeling, design, simulation, and prototyping of large scale systems. The framework uses a coherent set of tools that model the system at hand. It can take a control design and analyze, verify and simulate it, and then can generate code that can be run in a target real-time software platform in the physical system. This paper emphasizes the specification language SHIFT and the simulation tools used in the framework.

The general framework as well as the tools that implement it have evolved at PATH (Partners for Advance Transit and Highways) over the last seven years as large number of researchers have worked on Intelligent Transportation Systems, AHS, and AVS.

Our simulation tools have been evolving through the years. Since the systems we were working on were inherently complex the need for simulation was obvious and we have developed C and C++ based simulation frameworks to evaluate PATH's and other organizations' proposals for highway automation (Eskafi 1995 and Göllü 1995). In parallel to our AHS work, we were involved with several other projects, such as ATMS, 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 systems. No language, product, or tool in the market nor in academia came close to satisfying all requirements. Many simulation frameworks are available 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. 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. Consider vehicles moving on a highway, where the behavior of a vehicle depends on the behavior of its front vehicle. The block diagram paradigm may be able to define a single vehicle, but fails to represent the "front vehicle" which could be any vehicle in the simulation as time passes.

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 (Deshpande, Göllü, and Semenzato 1997a and 1997b) which we briefly discuss in Section 4 is the first programming language with well defined simulation semantics that addressed our requirements. It combines system-theoretic concepts into one consistent and uniform programming language with object-oriented features. SHIFT 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. SHIFT's strength lies in the ability of instantiating agents and evolving the interaction network among them at run-time as part of the simulation.

Our work on the general framework has been more recent. As the control designs for automated vehicles matured we started in-vehicle prototype experiments. At this stage it became essential to streamline the translation of the control law specifications that had been simulated and verified into in-vehicle real-time control instructions.

This paper is organized as follows. In Section 2 we discuss the framework and the general methodology for large scale system development. In Section 3 we summarize requirements for simulation frameworks. In Section 4 we discusses the SHIFT language that is used as the specification and simulation language of the overall framework. In Section 5 we use the example of the AHS to describe how the methodology is used. Section 6 has the conclusions.

2. GENERAL FRAMEWORK FOR LARGE SCALE SYSTEM DESIGN

The overall methodology we use for large scale system design, prototype, and deployment is depicted in Figure 1.

Large scale system development goes through several stages. The first stage typically consists of feasibility studies that lead to models and designs on paper. The second stage broadens the scope with prototype experiments.

2.1. Stage One

Any large scale system design task requires the specification of an overall architecture within which controllers can be designed to coordinate the system. The architecture design decomposes the control design problem into the control of sub-systems.

The highway automation architecture of PATH is briefly discussed in this paper. An architecture for Air Traffic Control is proposed in Koo et al. (1997), and an architecture for Autonomous Underwater Vehicles is proposed in Sousa and Göllü (1997).

Architecture specification can go through several steps of refinement. For AHS, the first step of decomposition separates the roadside controllers from in-vehicle controllers. In-vehicle controllers are further decomposed into feedback controllers responsible for safe execution of follow, lane change, entry and exit maneuvers, and coordination controllers for tactical decisions as to what maneuver to perform.

The actual controllers may be implemented by classical feedback systems, discrete event systems, or hybrid systems that combine the two. Other approaches such as rule-based systems and neural nets can also be used.

Analysis, simulation, and verification tools are needed within which these tasks can be carried out. Verification tools provide guarantee conditions for specific designs. In general, verification is possible only with very high level abstractions or very isolated subsystems. Analysis tools are useful for high level models for which closed form solutions can be

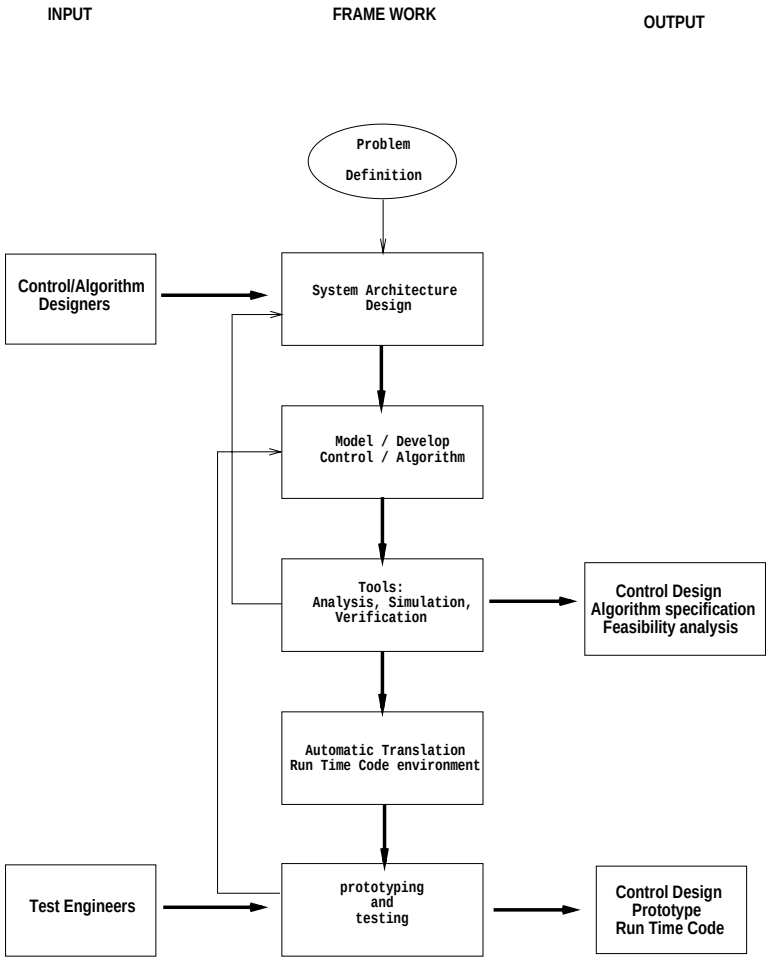

Figure 1: A General Framework for System Development

obtained. Simulation is needed to see the detailed performance of a design.

These tools are used in successive iterations until the first stage starts delivering satisfactory architecture and control designs.

2.2- Stage Two

As the confidence in the system architecture and control designs grows the second stage starts which involves physical prototype experiments.

At this stage it is essential to leverage the control specifications of the earlier stage into actual code that can be used in physical experiments. This requires the implementation of a real-time control environment in the actual devices that can execute control instructions. Such a run-time environment must provide the necessary abstractions to interface with sensory and communication inputs and outputs to actuators.

Once a real-time control environment is implemented, it becomes possible to automatically translate specifications from the design stage into the target physical prototype system. Such automatic translation increases efficiency of experiments and prevents loss of information between the two stages.

Physical experiments may require further refinement of the control designs or the overall architecture.

3- SIMULATION FRAMEWORKS

In this section we briefly discuss simulation frameworks that facilitate the design, specification, simulation, and evaluation of large scale systems.

Frameworks shield their users from software implementation details and allow them to concentrate on their particular specification or evaluation task. The requirements for simulation frameworks were discussed in detail in Eskafi and Göllü (1997).

These frameworks must address the needs of sev-

eral categories of users who use it in successive stages. System engineers develop automation architectures, control and communication engineers design, implement, and test individual control algorithms; system analysts test and evaluate the overall system; and system planners select the automation strategy for deployment based on evaluation results.

In addition to traditional software engineering requirements, these frameworks must allow the designers to use a specification language that fits their domain, in this case differential equations and finite state machines; they must provide a structured specification, simulation, and evaluation environment with formal semantics; they must represent dynamic interaction dependencies among components in the system; and last but not least they must provide models of entities that are particular to the application domain.

Our experience with generations of simulation frameworks that we have implemented (Eskafi 1996, Göllü 1995) lead us to the conclusion that simulation frameworks are best implemented in a programming language with explicit simulation syntax and semantics. Hence, we have developed the SHIFT language.

4- SHIFT PROGRAMMING LANGUAGE

4.1- Introduction

SHIFT is a special-purpose object-oriented programming language designed to simulate large 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) with continuous and discrete behavior. A simulation starts with an initial set of components that are instantiations of these types. The world-evolution is derived from the behavior of these components.

The world evolves in a sequence of phases. During each phase, time flows while the configuration of the world remains fixed. In the transition between phases, time stops and the set of components in the world and their configurations are allowed to change.

The data model of a type consists of numerical variables, link variables, a set of discrete states, and a set of event labels.- The variables are grouped into input, internal (state), and output variables. A type has read only access to its input variables and read/write access to its internal and output variables. Types can access other types 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.

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.

The behavior model is hybrid, i.e., the model has both continuous and discrete behaviors. Each discrete state has a set of differential equations and algebraic definitions that govern the continuous evolution of numeric state and output variables. These equations are based on all numeric 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. 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.

At this time the behavior model does not support inheritance.

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 runtime executable supports programmatic, command-line, and graphical interfaces for user interaction.

4.2- SHIFT Use

Currently, we are using SHIFT to implement the controllers to populate SmartAHS (the AHS simulator) structure. We describe the structure in Section 5. Other frameworks under implementation in

the SHIFT language include air traffic management planning, underwater submarine operation, and cellular mobile frequency allocation schemes.

We used SHIFT extensively to generate dynamical models of entities in various fields, effectively creating class libraries for other users. Control engineers have used the language to design and specify control algorithms to manage and improve the behavior of large system. And test engineers have used the language to evaluate systems designed by domain experts and control engineers. They have implemented simple types to monitor the system and collect statistics as the system evolves during successive simulation runs.

To see examples of the SHIFT specification language, user and programmers manuals, and up-to-date information about SHIFT and the developed frameworks, we refer the interested parties to our home-page,
`http://www.path.berkeley.edu/shift`.

5- DESIGN OF THE AUTOMATED HIGHWAY SYSTEM

5.1- Introduction to AHS Controllers

The Automated Highway System consists of two main elements: the automated highway and the automated vehicle. The technical challenge facing any AHS proposal is the design, development, and testing of a set of controllers both on the highway and in the vehicle that result in superior performance compared to today. In general, the controllers on the highway should help the driver choose the route with the shortest travel time, and the controllers on the vehicle should be able to drive the vehicle safely and efficiently. We can, therefore, group the controllers in loosely coupled control layers according to their function and domain of operation. This allows for independent design and development of controllers by different groups of control engineers, so long as we can define a robust interface among the layers.

A hierarchical control design proposed originally in a 1991 PATH report (Varaiya and Shladover 1991) and later extended in Varaiya (1993) has four layers: network, link, coordination, and regulation layers. The first two layers are on the highway and the last two on the vehicle. This proposal, though originally designed for the PATH-AHS proposal, is the most elaborate proposal in terms of generality and completeness. It only specifies the control layers and not any specific controller, and it encompasses both the highway and the vehicle.

5.2- SmartAHS Building Blocks

Basic SmartAHS building blocks provided by SmartAHS developers are discussed in Deshpande (1996). The highway library provides building blocks to create arbitrary highways. A *highway topology* is divided into sections. A *section* has a specific length defined as the length of the inner most lane within that section (lane 1 or the high speed lane). The only requirement for a section is that it should have the same number of lanes throughout its length, and all the lanes within a section have the same geometry. However, the number of lanes from one section to another can change. Every section comprises a number of segments. A *segment* can be either a line or an arc; in the former case its only attribute is its length, but in the latter case it has length, the arc radius, and the direction of turn which can be left or right.

Sink and *source* types provide the flows of vehicles facilitating the representation of desired origin-destination patterns and flow volumes.

A basic *automated vehicle* consists of several types. It contains a *vehicle* which models vehicle dynamics, a *controller* which is to provide the throttle, steering, and braking inputs, and a *vehicle roadway environment processor*, which provides information regarding the environment (e.g., highway, lane, curvature, etc.) to the vehicle. The *automated vehicle* also contains sensors and communication devices.

On the Roadside there may be communication devices for message transmissions and receptions, controllers to provide routing and traffic guidance to the vehicles, and sensors and monitors to gather information about the status of traffic.

5.3- SmartAHS Methodology

SmartAHS is extended and used by a series of users.

Domain experts use the SHIFT syntax and the SmartAHS framework to provide detailed vehicle dynamics, sensor, actuator, and communication device models. 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.

Concept designers use 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 next set of users. 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.

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. 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.

To simulate the system we 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 detailed monitors to generate the data to be used for concept evaluation.

For more information on SmartAHS release, see
`http://www.path.berkeley.edu/smart-ahs`

The next step is to implement the controllers on the actual vehicles.

5.4- Implementation and Prototyping

The automated translation of control specifications into the real time control environment is our current focus.

For early experiments, control specifications were translated manually. This translation was costly and prone to unwanted errors caused by the translators. Nonetheless the controllers have successfully been installed and tested on the vehicles.

6- CONCLUSION AND CURRENT STATUS

We have presented a general framework for modeling, simulation, verification, and prototyping of controllers in a large scale system. We have illustrated the framework for the automated highway system application. Other similar systems that are using this approach include ATMS, autonomous underwater vehicles, and wireless communication systems.

This paper emphasized simulation frameworks and the SHIFT simulation language that is used by the framework.

Currently, we are focusing our effort on providing the verification of SHIFT specifications and their direct translation into control instructions to run on real-time operating systems.

REFERENCES

Deshpande, A. 1996. AHS components in SHIFT, *PATH technical report.*

Deshpande, A., A. Göllü, and L. Semenzato. 1997a. The SHIFT programming language and run-time system for dynamic networks of hybrid automata. California PATH Technical Report UCB-ITS-PRR-97-7.

Deshpande, A., A. Göllü, and L. Semenzato. 1997b. SHIFT reference manual, California PATH Technical Report UCB-ITS-PRR-97-8.

Eskafi, F., D. Khorramabadi, and P. Varaiya. 1995. An automated highway system simulator, *Transportation Research Journal, part C*, 3(1): 1-17.

Eskafi, F. 1996.- Modeling and Simulation of the Automated Highway System, *Ph.D. Thesis*, UC Berkeley 1996. Also Path Report UCB-ITS-PRR-96-19.

Eskafi, F., and A. Göllü. 1997. Simulation requirements and methodologies in automated highway planning, to appear in *TRANSACTIONS of the Society for Computer Simulation.*

Göllü, A. 1995. Object Management Systems, 1995. *Ph.D. Thesis*, UC Berkeley 1995. Also Path Report UCB-ITS-PRR-95-19.

Koo, T. J., Y. Ma, G. J. Pappas, and C. Tomlin. 1997. SmartATMS: A Simulator for Air Traffic Management Systems, to appear in *Proceedings of the 1997 Winter Simulation Conference*, ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, IEEE, Piscataway, New Jersey.

Sousa, J., and A. Göllü. 1997. A simulation environment of the coordinated operation of multiple autonomous underwater vehicles, to appear in *Proceedings of the 1997 Winter Simulation Conference*, ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, IEEE, Piscataway, New Jersey.

Varaiya, P., and S. Shladover. 1991. Sketch of an IVHS systems architecture, *Proceedings of the Vehicle Navigation and Information Systems Conference*, 909-922.

Varaiya, P. 1993. Smart cars on smart roads: problems of control, *IEEE Transactions on Automatic Control*, 38(2): 195-207.

AUTHOR BIOGRAPHIES

ALEKS GÖLLÜ received his B.S. in Electrical Engineering from Massachusetts Institute of Technology in '87 and his M.S. and Ph.D. in Electrical Engineering and Computer Science from UC Berkeley in '89 and '95 respectively. He was a Systems Engineer and Project Manager at Teknekron Communications ('90-'92) Systems and a Software Engineer at Oracle ('89-'90). Currently he is a Research Engineer at PATH/UC Berkeley where his research interrests include simulation, modeling, real-time control of hybrid systems. His domain expertise includes telecommunications, power distribution, highway automation, large-scale software development, database management systems, and control technologies.

FAROKH ESKAFI received his B.S., M.S., and Ph.D. in 1991, 1992, and 1996, respectively, all in Electrical Engineering and Computer Science from University of California at Berkeley. Currently he is a research engineer at U.C. Berkeley and the project leader for integration and implementation of the vehicle controllers and communication protocols for the Automated Highway System. His current research interests include simulation as applied to large scale hybrid systems, distribution and parallelization of strongly coupled systems, and integration of simulation and verification processes of control algorithms.