An Embedded System's Design Verification Using Object-Oriented Simulation

Steve J. Cunning, Stephan Schulz and Jerzy W. Rozenblit

SIMULATION 1999 72: 238
DOI: 10.1177/003754979907200403

TECHNICAL ARTICLE

SIMULATION 72:4, 238-249

ISSN 0037-5497/99
Printed in the United States of America

An Embedded System's Design Verification Using Object-Oriented Simulation

Steve J. Cunning, Stephan Schulz and Jerzy W. Rozenblit
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, Arizona, USA
E-mail: {scunning,sschulz,jr}@ece.arizona.edu

...ility for the embedd... ...signer to ...del and simulate proposed designs prior to implementation is increasingly valuable in today's competitive market place. Simulation-based design is a methodology that uses the virtual prototype as a means of producing consistent and reduced design time. This paper presents this approach in the context of an automotive embedded system application. Structural models are used to capture design knowledge and define the space of possible design alternatives. The Discrete Event Specification (DEVS) formalism is used to develop behavioral models which can be simulated. Results are obtained through experimental frames which are used to define the scope of simulation. These results allow the designers to confirm that the proposed design solution meets the system requirements and constraints.

Keywords: Modeling, discrete event simulation, embedded systems design, hardware/software codesign

1. Introduction

Simulation modeling is being increasingly recognized as a useful tool in assessing the quality of design choices and arriving at acceptable trade-offs. This approach is often called "simulation-based design." However, computer simulation and other advanced computational tools are of limited effectiveness without a methodology to induce a systematic handling of the multitude of goals and constraints impinging on a design process. Our work focuses on the development of techniques in which design models can be synthesized and tested through simulation within a number of objectives, taken individually or in trade-off combinations.

Given a set of specifications, we use a structured representation of the project's domain to derive alternative object models for the system being designed. Object models are specifications of the components from which a system is to be built. Object definitions include attributes that express various properties of system's components as well as relationships to other objects. Behavioral descriptions are associated with object models in order to execute simulations.

We view simulation in the following, two-fold perspective: (a) as a means of verifying the functionality of the proposed solutions by executing the model's dynamics so that we can ensure that the model's behaviors are consistent with those perceived for the system being designed, and (b) as a way of assessing

how well performance requirements are met by the proposed design solution.

Simulation studies are carried out using an experimental frame [1], i.e., a structure that represents design objectives in the form of standard attributes. Such attributes express measures of input/output performance, utilization of resources, reliability assessments, etc. Alternative design models are evaluated through computer simulation in experimental frames that reflect design performance questions. Results are compared and trade-off decisions are made in the presence of conflicting criteria.

We believe that simulation-based design will become increasingly important in the development of heterogeneous systems that comprise hardware, software, and interface components. In [2], we have presented the fundamentals of a codesign methodology that leverage from model-based techniques. In this article, we demonstrate how simulation can be used to verify design properties of such systems.

2. Design Context: An Embedded System Application

In the last decade, we have witnessed a sizable growth in the application of electronics in the automotive industry. Early applications ranged from electronic fuel injection devices to motor control units. With the advent of more powerful microprocessors, a new generation of devices aimed at improving the safety of automobiles is emerging. Such devices are often highly complex, embedded systems.

In what follows, we show how a hierarchical, modular, object-oriented simulation framework is used to verify design specifications of an embedded system. Such a system inherently contains hardware, software, and interface components. An autonomous, intelligent cruise controller (AICC) has been selected to illustrate our approach. The AICC is an extension of the common cruise control that not only keeps a fixed speed, but also adapts to the speed of the vehicle ahead. It has no lateral control. The system is autonomous, i.e., it does not rely on communication between vehicles. The driver remains in full control since he or she can override the device by braking. The cruise controller should only be activated for speeds higher than 56 kilometers per hour (km/h). We require the unit to keep the speed of the vehicle within ± 2 km/h of the *safe speed* or *set speed*—whichever is lower. The *safe speed* is the maximum speed that keeps the car within a safe distance to the vehicle ahead. The *set speed* is the speed requested by the driver.

In the design of the AICC, it is necessary to guarantee that the safety distance is kept within a small margin of error under normal traffic conditions. The device should differentiate between obstacles and moving objects, warn the driver, and suggest or take appropriate actions.

The AICC is part of a complex, vehicle communication and control system as shown in Figure 1. The modeling work presented in this paper focuses on simulation-based verification of the AICC control unit. The experimental frame concept is used to describe the environment, e.g., sensor and actuator units, road conditions or terrain, that this control unit is embedded in. Our approach to simulation makes a clear distinction between the model and the experiment to allow testing objectives to be separated from the system being modeled.

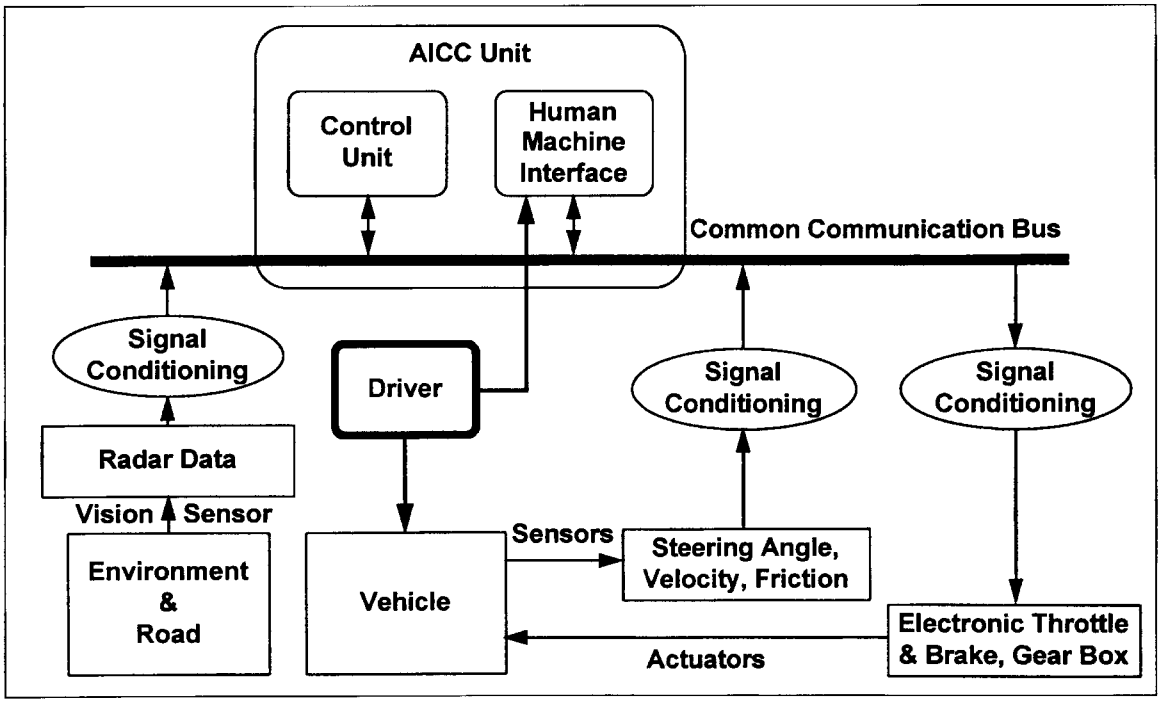


Figure 1. High-level diagram of the AICC and its environment

3. Design Modeling

Design modeling refers to developing a simulatable representation of the system being designed (SBD). This process is guided by a set of specifications and requirements. Its goal is to generate a model of the SBD. This model is the basis for assessing how well the design solution (given in the form of this model) meets the set of specifications and requirements. We offer a means of constructing such models. However, we do not attempt to carry out model validation and credibility assessments as defined by Balci [3]. Since we are developing models of a design concept (in essence, a system that does not exist yet), we have no repository of data against which the design model can be validated. Nonetheless, we assess the credibility of the design models by checking to see if the design requirements and expected functionality are met through the simulation results.

We develop models in a hierarchical, modular fashion, adhering to the well accepted principles of decomposing the model space into object (structural) and behavioral (functional and dynamic) models [4]. The object model explains the structural decomposition of the design. The functional model describes the overall functionality of the model and its integration into the surrounding environment. The dynamic model assigns timing constraints to the internal functionality and shows the details of state changes within the model. These three different model descriptions are sufficient to generate a simulatable specification for our embedded system example, i.e., the autonomous intelligent cruise controller.

3.1 The Object Model

To develop the AICC object model, we analyzed the domain of automotive safety and selected an instance of AICC [5]. Recent literature [6, 7] guided our decomposition of the unit into various components. We use the Object Modeling Technique (OMT) notation [4] for structured knowledge representation. Figure 2 shows the corresponding diagram for the AICC control unit to be modeled.

3.2 The Behavioral and Functional Model

The functional and behavioral models of the AICC Control Unit were defined based on the conversion of assumptions and functional requirements into informal specifications. The first is that data inputs are generated by the driver and from internal sensors. The driver inputs originate from common switches which are part of the Human Machine Interface (HMI) and are typically found on a regular cruise controller: ON/OFF, COAST, and RESUME/ACC. The internal sensors may include radar units, steering angle, momentum, speed, throttle angle measurement devices, etc. Data arrives with a specific protocol at a rate depending on the type of source. From these data, information about the distance and relative speeds of

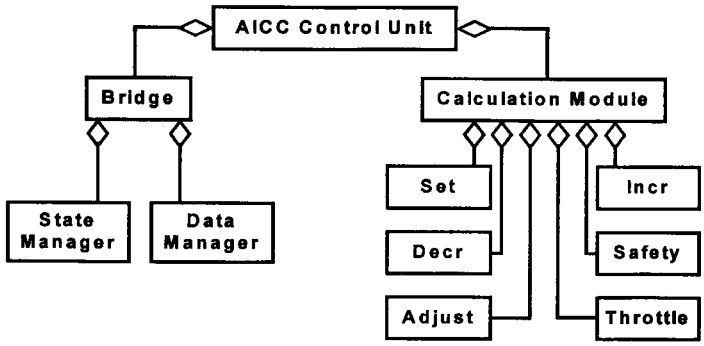


Figure 2. OMT diagram for AICC control unit

surrounding vehicles can be obtained. To capture the behavioral model, we have used the Statechart formalism [8]. The complete Statechart can be found in [5].

3.3 Discrete Event Specification (DEVS) Models

In our system, models are developed using the Discrete Event System Specification (DEVS) formalism. The structural, behavioral and functional aspects are combined in each DEVS model. This formalism underlies the construction of models in the verification environment DEVS-Java [9]. A discrete modeling environment was chosen for this task because the application was limited to an embedded system which only required the use and communication with digital components. Continuous modeling would have to be included if the system were to support analog functionality or needed to interface with analog devices.

The DEVS hierarchical, modular set theoretic formalism for sequential implementations was developed by Zeigler [1]. The implementation of DEVS-Java reflects the parallel DEVS formulation [9]. In such a formalism, one must specify basic models from which larger ones are built, and how these models are connected together in a hierarchical fashion. A basic model, called an atomic DEVS model, is defined by the following structure:

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

where:

- X is a set, the external input event types
- S is a set, the sequential states
- Y is a set, the external output event types
- δ_{int} is a function, the internal transition specification
- δ_{ext} is a function, the external transition specification
- λ is a function, the output function
- ta is a function, the time advance function

with the following constraints:

a. The total state set of the system specified by M is

$$Q = \{(s, e) \mid s \in S, 0 \le e < ta(s)\}$$

where e is the elapsed time since the last transition;

b. δ_{int} is a mapping from S to S:

$$\delta_{int}: S \rightarrow S:$$

c. δ_{ext} is a function:

$$\delta_{ext}: Q \times X^b \rightarrow S$$

where X^b represents a bag of external events which are held until internal transition and output generation is complete;

d. ta is a mapping from S to the non-negative real numbers with infinity:

$$ta: S \rightarrow R_{0^+ \rightarrow \infty}$$

and

e. λ is a mapping from Q to Y:

$$\lambda: Q \rightarrow Y$$

The second form of a model, called a coupled model, tells how to couple several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to the hierarchical construction of models. A coupled DEVS model is defined as a structure:

$$DN = <X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}>$$

where:

X_{self}	is a set of input events
Y_{self}	is a set of output events
D	is a set of components
for each i in D,	M_i is a model component
for each i in $D \cup \{self\}$,	I_i is a set of influences of i
for each j in I_i,	$Z_{i,j}$ is a function, the i-to-j output translation

The structure is subject to the constraints that for each i in D,

M_i is a parallel DEVS atomic model as described above

I_i is a subset of $D \cup \{self\}$, i is not in I_i

$Z_{self,j} : X_{self} \rightarrow X_j$

$Z_{i,self} : Y_i \rightarrow Y_{self}$

$Z_{i,j} : Y_i \rightarrow X_j$

where *self* refers to the coupled model itself and is a device for allowing specification of external input and external output couplings.

An implementation of DEVS using Java has been developed at the University of Arizona. The interested reader may refer to [10, 11]. DEVS-Java follows the design of previous implementations using Scheme and C++ [12]. As with the other implementations, DEVS-Java provides the object-oriented constructs for inheritance and message passing. In fact, DEVS-Java provides the model builder with the base classes from which all models are inherited.

Each model in DEVS-Java has a set of input and output ports for receiving and sending external events, state variables for maintaining the model's state, and the internal transition, external transition, output, and time advance functions for specifying the behavior of the model. The internal transition function is called by the DEVS simulation engine when the current simulation time is equal to the model's next scheduled internal event. Internal events are scheduled by the time advance function. The external transition function is called by the DEVS simulation engine when an external event has arrived at one of the model's input ports. The output function is called just prior to the call to the internal transition function and is used to map the current state of the model to any necessary output events.

The implementation of DEVS in Java also provides a few features that were not previously available. The first is the multi-threading capability of Java. Each model is given its own thread of control which simplifies the scheduling in the simulator. Although execution is still sequential on single-processor machines, threads conceptually are able to execute concurrently and there is the opportunity for true concurrency on multi-processor systems. Another feature is the platform independent nature of Java which should allow easier porting of simulations. Coupled with this is advanced graphical features of Java which also endeavor to be platform-independent. This allows for the relatively easy addition of graphical user interfaces to the DEVS models.

3.4 *DEVS Model for the Autonomous Intelligent Cruise Controller*

After initially building and testing the atomic components of the AICC control unit, the model was extended to the final coupled model. The STATE MANAGER keeps track of the state the cruise controller is in. This is mostly influenced by actions taken by the driver. The DATA MANAGER obtains, manages and distributes all data received from the corresponding sensors, and sent or requested by functions of the calculation module. The calculation module contains components that perform specific computations on arriving data, e.g., calculating the safe distance or the new throttle angle. For a more detailed description of these components, the reader is referred to [13].

A complete diagram of the DEVS model of the AICC control unit is shown in Figure 3. The lines labeled "req" comprise the request network. Each component connected to this line is implied to have a connection to the "request_in" port of the DATA MANAGER and a connection from the "request_out" port of the DATA MANAGER back to the component. This model of the

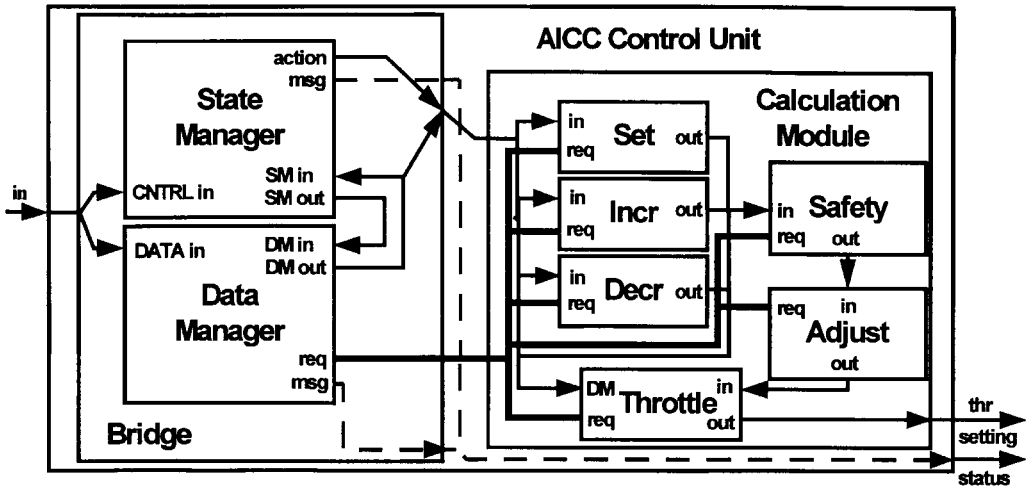


Figure 3. Final AICC DEVS model

control unit was then connected to an experimental frame during the simulation phase.

Having described the model for the AICC Control Unit and DEVS modeling in general, we will now proceed to describe the experimental frame that allows us to test and evaluate the AICC Control Unit design.

4. Experimental Frame Specification

The experimental frame (EF) concept is presented in detail in [14]. It is a construct which helps the model builder to keep the primary model (the object of interest) separate from the stimuli and responses used for evaluation during simulation. This separation is analogous to the idea of a test bench and unit under test used in hardware simulations.

The separation of the primary model and the EF has many benefits. An EF helps to define the aspects of interest for the primary model. Since it is not practical for a model to accurately describe every possible behavior that the real-world object might exhibit, a subset of behaviors is usually selected. The EF will generate input parameters and observe output parameters that are related to these behaviors of interest. More than one EF may be used with the same model in order to test different behaviors or to test at different levels of model granularity. As long as the interface for the primary model is well defined, alternative primary models may be tested with the same EF.

The EF provides three primary functions. The first is that of a generator, which provides the stimulus needed to drive the primary model through all tests of interest. The second primary function is that of a transducer, which observes the outputs from the primary model and converts them to a form desired by the user. This output may also be fed into the generator to influence future stimuli. The last is that of an acceptor, which provides control over the simulation run. Termination of a simulation run, if not provided by the underlying simulation engine, should be easily controlled by the user. This may be determined prior to the start of a simulation for a batch type simulation, or at execution time if the simulation is interactive. The coupling of the EF components and the coupling of the EF to the primary model are illustrated in Figure 4.

4.1 AICC Experimental Frame

The objective of the AICC EF is to provide a suitable environment for the evaluation of AICC Control Unit designs. The initial purpose of the simulation is to verify the proper operations of the AICC, such as controlling the speed of a vehicle as a standard cruise control does, the calculations of the safe speed and safe distance for the primary vehicle, and the proper control of the speed of the vehicle when presented with an obstacle. After an initial AICC Control Unit design has been developed that meets the objectives

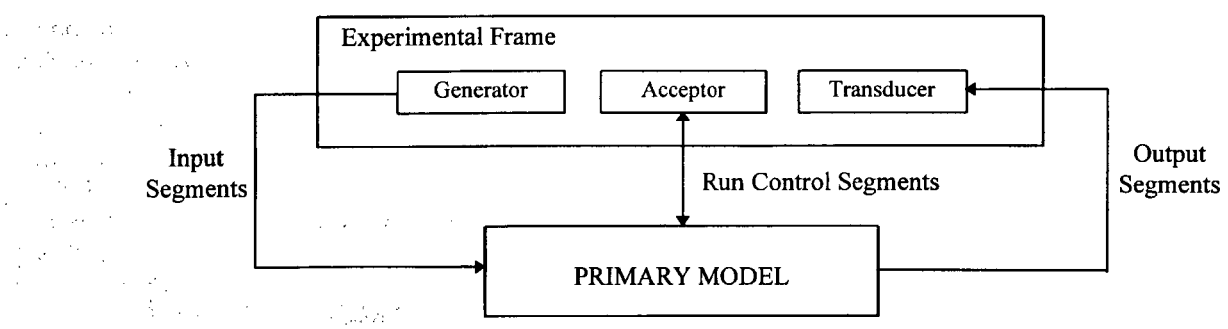


Figure 4. Structure of an experimental frame

listed above, alternative designs intended to optimize selected parameters may be evaluated.

With these objectives in mind, the AICC EF has been designed to operate in one of two modes, both of which are selected and controlled through a Graphical User Interface (GUI). The first is the interactive mode where the operator acts as the driver of the primary vehicle. This mode is useful for model development and debugging and for developing test scenarios. After an initial AICC control unit model has been developed and test scenarios have been determined, our focus shifts toward multiple repeated simulation of alternative AICC designs. For this situation, a programmed primary vehicle driver that would stimulate the AICC in a repeatable manner is needed. This capability is provided by the script mode of EF operation. This mode allows test scenarios, which are saved as a sequence of events in a system file to be selected prior to the start of a simulation run.

In addition to the control functionality described above, the EF must model the environment in which the AICC is expected to function, i.e., it must complete a closed-loop system with the AICC Control Unit. This environment (as illustrated in Figure 1) has the following components and functions:

1. A vehicle to be controlled by the AICC
2. A sensor to provide distance and speed information for an object ahead of the primary vehicle
3. A Graphical User Interface (GUI) to display information and accept user input
4. A transducer to monitor selected parameters during simulation runs

The generator functions of the EF are provided by the driver (acting through the GUI model), the sensor model, and the vehicle model. The driver provides various inputs such as throttle control, brake control, clutch control, and AICC controls (ON/OFF, COAST, RESUME/ACC). The sensor model contains an acceleration profile for the lead vehicle. When the operator starts this profile through the GUI, the sensor model provides the AICC Control Unit with the distance to and speed of the lead vehicle. The vehicle model takes throttle and brake control input from both the driver and AICC Control Unit and produces the speed of the vehicle, which is fed back through a bus controller component to the AICC.

The transducer functions of the EF are provided by the GUI and the transducer model. The items that are easily observable are displayed on the GUI for the operator. An example would be observing that the distance to the lead vehicle was allowed to reach zero. For items that require computation or summation, the transducer is used. The parameters to be monitored by the transducer are the time that the primary vehicle speed exceeds the computed safe speed and the time that the distance to the lead vehicle is less than the computed safe distance.

4.2 EF Model Base

The model base for the AICC EF system contains the following atomic models which correspond to the EF functions listed in the previous section:

- Primary Vehicle
- Sensor
- GUI
- Bus Controller
- Transducer

In order to illustrate the design of a DEVS model, the primary vehicle model will be described in detail. Only brief descriptions of the other EF models will be provided.

Primary Vehicle Model

The primary vehicle model is, in a sense, the master for all of the other models in the experimental frame. It periodically outputs its speed to the Sensor and AICC Control Unit models and also generates running clock information which is passed to the GUI and Transducer models. The Sensor, GUI, and Transducer are essentially passive models since they rely on an external input to drive their actions. Each of these models does nothing until an external message is received. After the message is processed, a transitory output state is immediately scheduled so that outputs may be generated. This was done to provide synchronization between the EF models.

Figure 5 illustrates the input and output ports associated with the primary vehicle model. The port names are used in the pseudo-code description of the external transition, internal transition, and output functions for this model (see Table 1). In this description the variable sigma represents the time advance function in that it determines the scheduled time for the next internal event for the model. The variable *phase* represents the model's current state.

This model continuously schedules internal events every *delta_t* time units. Since the output and internal transition functions are called as a result of each internal event, the speed of the vehicle is updated at a rate of $1/delta\_t$. External events are handled as they arrive, and the appropriate parameters (which influence the vehicle's speed equations) are updated.

To determine the Primary Vehicle's speed, accelerations are calculated using $a = F/m$. The forces modeled are power train, resistive, and braking system forces. The force exerted by the power train is determined by

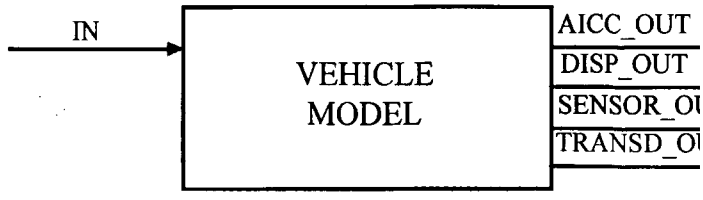


Figure 5. Port diagram for primary vehicle model

Table 1. Pseudo-code description for primary vehicle model shown in Figure 5

```
ATOMIC-MODEL: AICC_VEHICLE

PARAMETERS:        Speed = 0                      Driver_throttle_command = 0
                   CC_throttle_command = 0        Throttle_position = 0
                   delta_t = 100

STATE VARIABLES:   sigma = delta_t                phase = "running"
                   Elapsed_time = 0

EXTERNAL TRANSITION FUNCTION:

        sigma = sigma – elapsed time in current state
        if message on port IN
            if message.name is "driver throttle command"
                Driver_throttle_command = message.value
                determine maximum throttle position using Driver_throttle_command
            if message.name is "cruise control throttle command"
                CC_throttle_command = message.value
                determine maximum throttle position using CC_throttle_command
            if message.name is "driver brake command"
                Driver_brake_command = message.value
                determine maximum brake position using Driver_brake_command
            if message.name is "cruise control brake command"
                CC_brake_command = message.value
                determine maximum brake position using CC_brake_command
            if message.name is "initial speed"
                Speed = message.value
            if no valid message on port IN found
                output warning for invalid message on IN port
        else
            output warning for message on undefined port

INTERNAL TRANSITION FUNCTION:

        phase = 'running'
        sigma = delta_t

OUTPUT FUNCTION:

        calculate speed of vehicle
        Elapsed_time = Elapsed_time + delta_t
        send Speed on ports aicc_out, disp_out, sensor_out, and transd_out
        send Throttle position on port disp_out
        send Brake position on port disp_out
        send Elapsed_time on port disp_out and transd_out
```

the throttle setting and the present speed of the vehicle. The resistive forces are assumed to be of two types: (a) constant, e.g., tire and bearing resistance, and (b) varying forces, e.g., wind resistance. The braking force is determined by the present brake pedal position, which is also controlled by both the driver and AICC.

Since the throttle and brake commands are generated by both the driver and AICC Control Unit, the vehicle model contains maximum functions for both of these parameters. The vehicle determines the actual positions as the maximum of either the driver or AICC command.

Sensor Model

Sensors should provide the AICC Control Unit with the distance to and speed of the object in front of the primary vehicle. This implies that the object in front of the primary vehicle (the Lead Vehicle or LV), needs to be modeled. In order to simplify the system, the lead vehicle will be modeled within the model of the sensor which will contain an acceleration profile that

may be started by the operator through a button on the GUI window.

GUI Model

This DEVS model acts as an interface between the DEVS simulator and the Java object that controls the GUI window. During initialization it opens a window frame and populates it with all of the necessary GUI controls. Figure 6 is an illustration of the GUI window frame.

The status of interesting simulation parameters is provided by labels and scroll bars (the Lead Vehicle scroll bars are for display only). The operator can provide input to the simulation through the text edits, buttons, and the brake and throttle scroll bars. The simulation may be operated in interactive or script mode to allow easily repeatable experiments.

The messages that are sent to the GUI model from other EF models are intended to update a GUI control (e.g., a label or scrollbar object). The GUI controls are updated by calling the methods provided by the Java window frame developed for the GUI interface. After all external messages are processed, the model enters a transitory state that is used to check if the operator has acted on any of the user modifiable controls. If this is the case, appropriate messages are generated to all models concerned with the control(s) containing the actions.

Bus Controller Model

This component models a bus controller chip that is used to interface the processor of the AICC control unit to the communication bus. Its purpose is to buffer up messages or data sent by all of the components listed above, and to transfer this information in the form of data packages to the control unit.

Transducer Model

The items monitored by the transducer are: the speed of the primary vehicle, the safe speed and safe distance computed by the AICC Control Unit, and the distance to the lead vehicle reported by the sensor. These items are used to accumulate the amount of time that the primary vehicle speed exceeds the safe speed and the amount of time the distance to the lead vehicle is less than the safe distance. The transducer outputs statistics to the Java system standard output window whenever any statistic has changed.

5. Coupled Model

Figure 7 illustrates the internal couplings for the AICC evaluation system model. In addition, the possible messages are listed near the source of each coupling. Referring back to Figure 4, the AICC model represents the primary model with the dashed line enclosing the EF. Since this is the top level model,

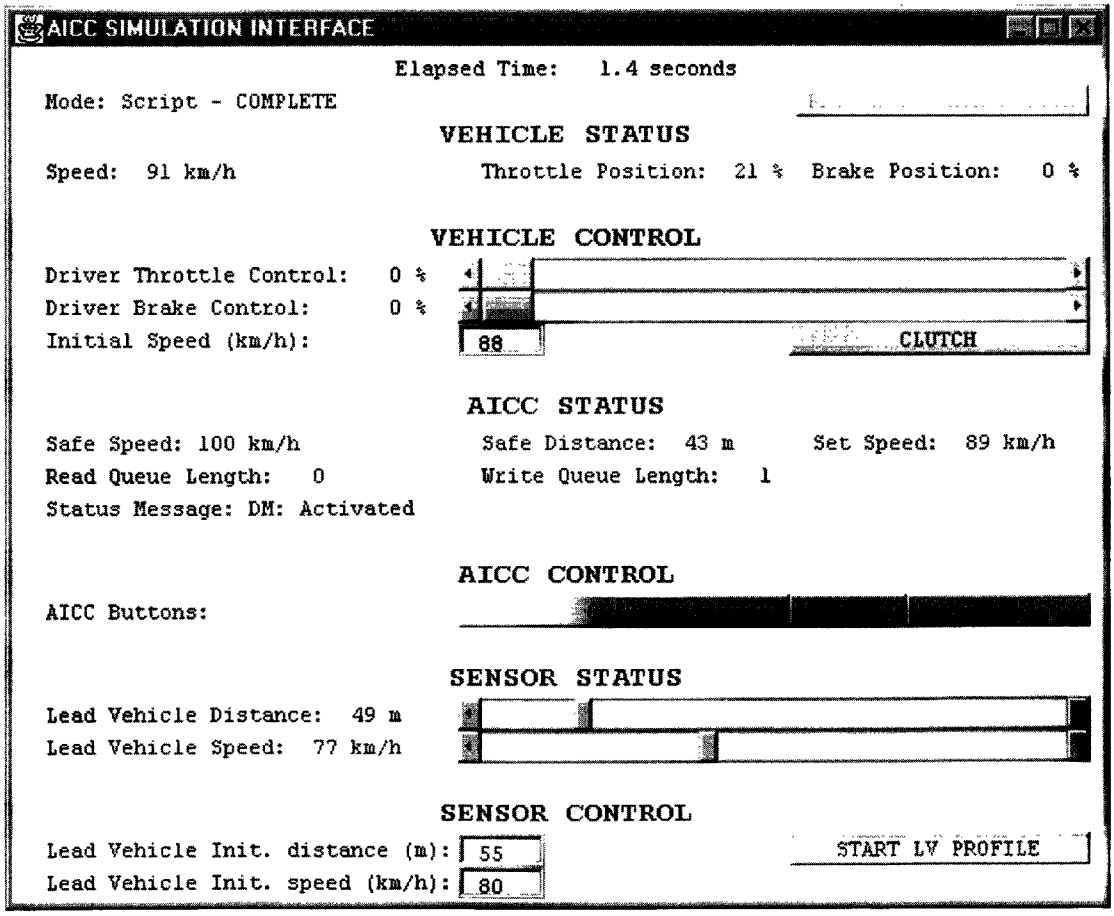


Figure 6. AICC EF GUI window

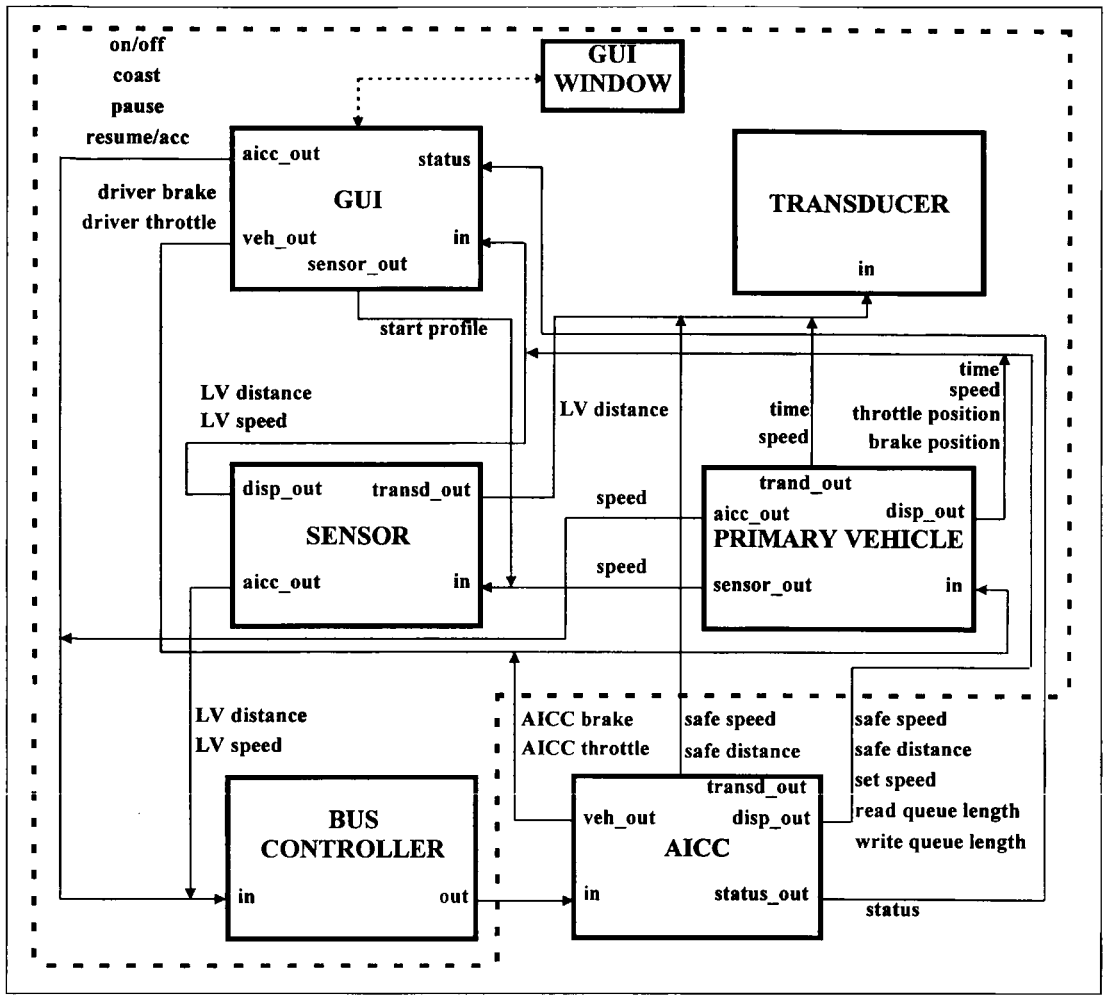


Figure 7. Coupled AICC evaluation system model

there are no external couplings. Execution control of this model is handled through the DEVS-Java simulator window.

6. Simulation Results

As discussed previously, our design approach calls for the use of simulation to verify the behavioral correctness of proposed design solutions. This allows for the incorporation of corrections at the virtual level rather than after physical prototypes are built, where it is more costly. The results presented in this section provide an illustrative example of this approach as it was applied to the design of the AICC Control Unit.

The first requirement for the AICC is that it be able to control the speed of the primary vehicle. The scenario selected to evaluate this function is as follows: (a) the cruise control is set by pressing *COAST*, (b) the brake is applied to slow down the vehicle, and (c) the cruise control is re-engaged by pressing *RESUME*. Figure 8 and Figure 9 show the simulation results obtained from the initial AICC design. It can be seen that the controller was not stable and produced an oscillating speed by alternating between application of the brake and full throttle. The simulation results indicated that the speed control function of the AICC needed to be corrected. Figure 10 and Figure 11 show the simulation results obtained by executing the same scenario after the AICC speed control calculations were revised.

The most important aspect of the AICC is its ability to maintain a safe distance from the lead vehicle. One of the scenarios used to evaluate this function of the AICC starts by setting the AICC to control the primary vehicle speed at about 90 km/h. The lead vehicle then appears (by changing lanes for example) 55 meters in front of the primary vehicle. The lead vehicle is initially traveling slightly slower than the primary vehicle and is decelerating. After about two seconds the lead vehicle begins an acceleration that lasts for about four seconds and leaves the lead vehicle speed traveling faster than the original set speed of the primary vehicle.

The simulation results from this scenario are provided in Figure 12 and Figure 13. The graphs show how the primary vehicle was slowed down and that its speed was kept within the calculated safe speed. It can also be seen that the distance between the vehicles was allowed to get slightly less than the calculated

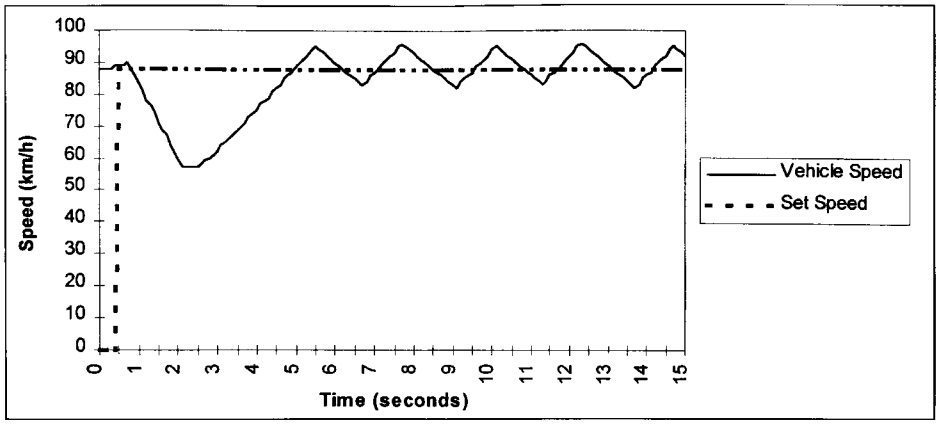


Figure 8. Primary vehicle speed control by initial AICC design

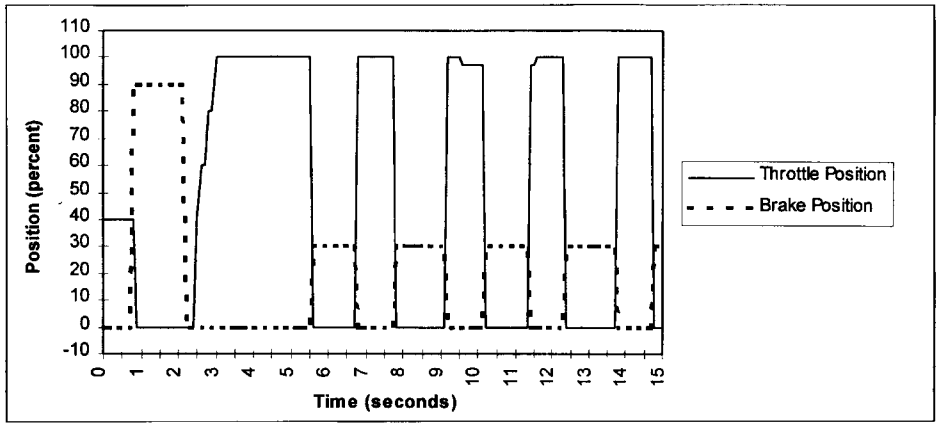


Figure 9. Primary vehicle throttle and brake control by initial AICC design

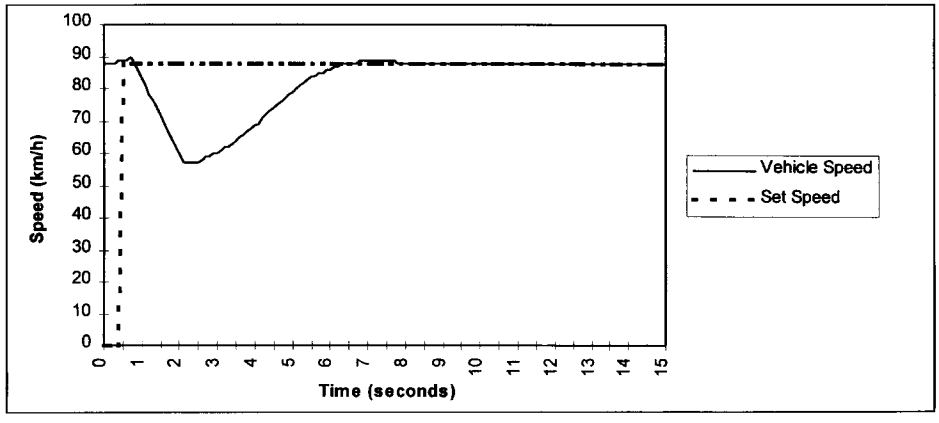


Figure 10. Primary vehicle speed control by revised AICC design

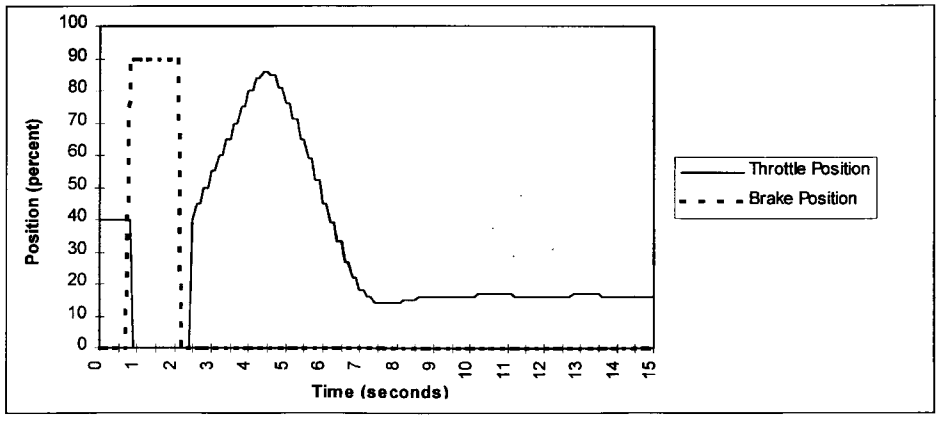


Figure 11. Primary vehicle throttle and brake control by revised AICC design

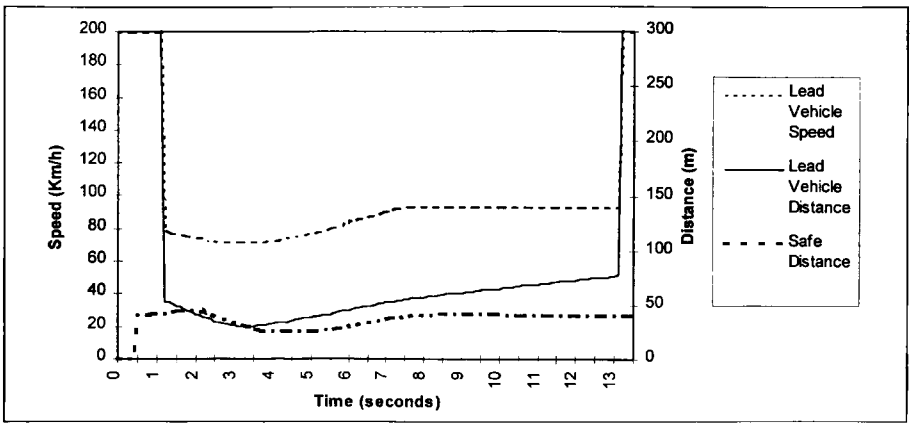


Figure 12. Lead vehicle speed, distance and calculated safe distance

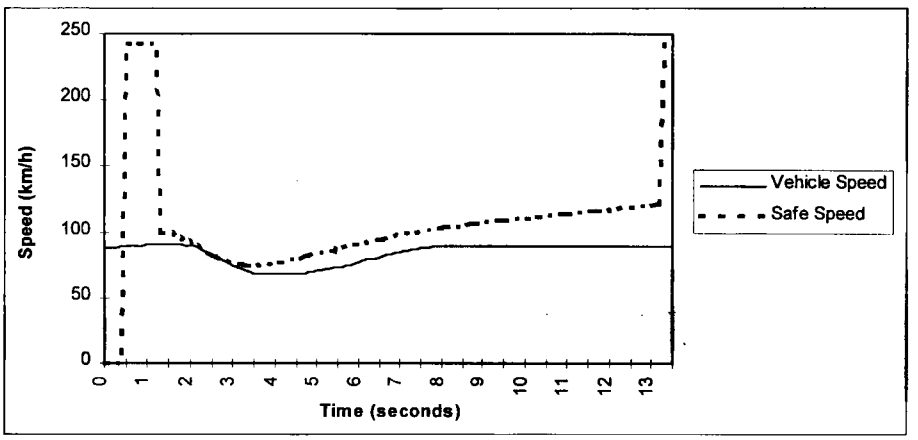


Figure 13. Primary vehicle speed and calculated safe speed

safe distance. This result indicates that an additional adjustment to the AICC is required.

7. Summary and Conclusions

We now summarize the major tenets of our simulation-based design approach and emphasize their benefits in the embedded systems development process. We use models as a means of generating a virtual, simulatable prototype prior to a realization of the design solution in a specific technology. This has profound consequences in the area of heterogeneous systems development. Such systems typically comprise a mix of hardware, software, and interface components. Conventional design techniques pursue the software and the hardware threads separately. Modeling and simulation tools are used that typically address these threads in isolation. Hence, it is difficult for the designers to assess design solutions in a holistic, unifying manner that addresses the system under development fully and comprehensively.

The approach we take postpones partitioning into hardware or software until we gain a high degree of confidence in the viability of the proposed design solution. We are able to assess the solutions by modeling the system as a whole and simulating it in a suite of various experimental conditions. The modular, hierarchical simulation modeling techniques that we use allow us to:

a. Flexibly construct a variety of design models, each reflecting a particular design solution approach,
b. Easily reuse design models when new designs are being considered,
c. Assess a proposed solution within a variety of experimental setups. This not only allows us to verify functionality and conformance to design performance requirements, but it also facilitates trade-off analyses to select designs that best meet the project specifications.

In our simulation-based design framework, we are developing techniques that will allow us to map model specifications onto a specific technology realization [13]. This has a significant potential to speed up the design cycles and reduce their overall costs. In addition to the definition of technology mapping techniques, we are further refining the simulation methodology as its application to the realm of embedded systems imposes various constraints.

8. Acknowledgements

This work has been sponsored in part by the National Science Foundation under Grant 9554561 and by Siemens AG, Central Research and Development Laboratories, Munich, Germany.

9. References

[1] Zeigler, B.P. *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, London and Orlando, 1984.

[2] Schulz, S., Rozenblit, J.W., Mrva, M., Buchenrieder, K. "Model-Based Codesign: The Framework and Its Application." *IEEE Computer*, August 1998.

[3] Balci, O. "Verification, Validation and Testing." *The Handbook of Simulation*, J. Banks, editor, John Wiley & Sons, New York, 1998.

[4] Rumbaugh, J. *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[5] Schulz, S., Rozenblit, J.W., Buchenrieder, K. "Towards Model-Based Codesign: An Intelligent, Autonomous Cruise Controller Application." *Proceedings of the 1997 IEEE Conference and Workshop on Engineering of Computer Based Systems*, IEEE Cat. 97B100105, pp 73-80, Monterey, CA, March 1997.

[6] Ioannou, P.A., Chen, C.C. "Autonomous Intelligent Cruise Control." *IEEE Transactions on Vehicular Technology*, Vol. 42, No. 4, pp 657-672, 1993.

[7] Palmquist, U. "Intelligent Cruise Control and Roadside Information." *IEEE Micro*, Vol. 13, No. 1, pp 20-28, 1993.

[8] Harel, D. "STATEMATE: A Working Environment for the Development of Complex Reactive Systems." *IEEE Transactions on Software Engineering*, Vol. 16, No. 4, pp 403-414, 1990.

[9] Chow, A. "Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism and its Distributed Simulator." *TRANSACTIONS of the Society for Computer Simulation*, Vol. 13, No. 2, pp 55-102, June 1996.

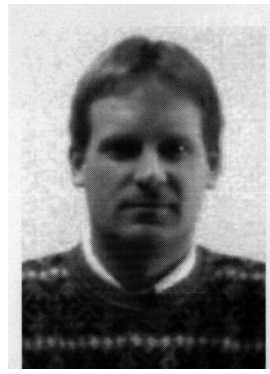
[10] Zeigler, B.P. *Objects and Systems: Principled Design with Implementations in C++ and Java*, Springer-Verlag, New York, 1996.

[11] Zeigler, B.P., Sarjoughian, H., Au, V. "Object-Oriented DEVS." *Enabling Technology for Simulation Science*, SPIE, AeoroSense, Orlando, FL, April 1997.

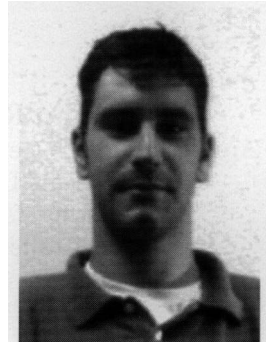
[12] Zeigler, B.P. *Object-Oriented Simulation with Hierarchical, Modular Models*, Academic Press, 1990; copyright B.P. Zeigler, 1995.

[13] Schulz, S. "A Model-Based Codesign Application: The Design of an Autonomous Intelligent Cruise Controller." Master's Thesis for the Department of Electrical and Computer Engineering, University of Arizona, Spring 1997.

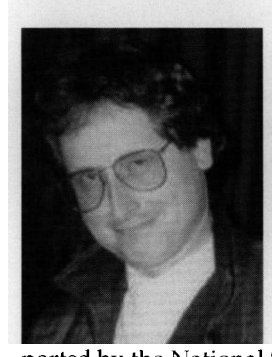
[14] Rozenblit, J.W. "Experimental Frame Specification Methodology for Hierarchical Simulation Modeling." *International Journal of General Systems*, Vol. 19, No. 3, pp 317-336, 1991.



Steve Cunning is a PhD candidate in Electrical and Computer Engineering at the University of Arizona. His research interests include model-based codesign for embedded systems, hardware/software interface synthesis, requirements engineering for real-time systems, and system-level test case generation. He is employed as a Systems Engineer at Raytheon, where he has taken part in the design and integration of a hardware-in-the-loop simulation facility and in the design and specification of real-time embedded systems. He received a BS in Electrical Engineering from the University of Iowa, and an MS in Electrical and Computer Engineering from the University of Arizona.



Stephan Schulz is a PhD candidate in Electrical and Computer Engineering at the University of Arizona. His research interests include embedded systems applications, model-based design, real-time operating systems, continuous and discrete-event simulation, and hardware/software codesign. He received a BS EE from the State University of New York–Binghamton, and an MS in Electrical and Computer Engineering from the University of Arizona.



Jerzy Rozenblit is Professor of Electrical and Computer Engineering at the University of Arizona. He holds PhD and MS degrees in Computer Science from Wayne State University, Michigan, and an MSc in Computer Engineering from the Technical University of Wroclaw, Poland. His research and teaching are in the areas of complex systems design and simulation modeling. His research in design has been supported by the National Science Foundation, Siemens AG, Semiconductor Research Corporation, McDonnell Douglas, and the US Army Research Laboratories, where he was a Research Fellow. Dr. Rozenblit serves as an associate editor of several international journals, and as a reviewer for a number of national and international funding agencies. In 1994 and 1995 he was Fulbright Senior Scholar and Visiting Professor at the Institute of Systems Science, Johannes Kepler University, Austria. He has also held visiting scientist appointments at the Central Research Laboratories of Siemens AG in Munich.