An Experiment in Applying Knowledge-Based Software Engineering Technology

Paul D. Bailor, Frank C. D. Young, and Kim Kanzaki
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433-7765

Abstract

This paper presents the results of an experiment at applying knowledge-based software engineering technology to hardware/software co-design. The Reacto Verification System developed by Kestrel Institute was used to create a high level, formal-based interface to VHDL which can effectively model both hardware and software design components. In addition to the theorem proving and simulation capabilities already provided by Reacto, extensions were made to incorporate time constraints, and compiler-based language mappings for generating VHDL from Reacto specifications were defined. Our experimental results clearly indicated the complimentary nature and benefits of developing high level, formally defined interfaces between languages like Reacto and VHDL.

1 Introduction

Experiments at applying knowledge-based software engineering (KBSE) technology to substantial engineering problems is an issue that has not yet been seriously addressed. This paper presents the results of one such application experiment to systems engineering problems from the perspective of hardware/software co-design. For such tasks, it is desirable to have high-level, abstract design languages and tools that provide automated theorem proving, a sophisticated treatment of time, and automated system generation support at multiple levels of abstraction via behavior preserving design refinement techniques.

Ideally, such design languages naturally model engineering techniques for analyzing and designing systems. Consider one of the basic analysis and design languages used by both hardware and software engineers — finite state machines (FSMs). FSMs can model sequential circuits, object behavior, and reactive systems. Also, FSMs are well studied and are easily formalized for both analysis and synthesis tasks. Therefore, an appealing approach is to have a system in which the engineer models a system using FSMs (in particular hierarchical FSMs) and has a variety of tools available to support the remaining phases of the analysis and design effort. For example, a verifier to prove properties about FSMs, a simulator to simulate the specified behavior, and a behavior preserving synthesis capability to generate lower level, more efficient descriptions in languages like Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) and Ada.

Two goals of such an approach are the formal determination of system properties by means other than exhaustive testing and the potential to greatly compress design to implementation time. KBSE technology has been shown to provide support for achieving the goals listed above, and for this reason, it was decided to conduct an experiment investigating the feasibility of applying KBSE technology to the hardware/software co-design problem. Our hypothesis was that there exists sufficient KBSE technology to provide high level, abstract system specification and design languages with the desired underlying formal foundations to support verification, simulation, and highly efficient design transformation tasks.

The remaining sections of this paper describe the steps used in conducting this experiment. First, a description of one possible hardware/software co-design process is presented, and the Reacto and VHDL tools are introduced. This is followed by an in-depth discussion of the experiment itself. This includes the engineering approach taken to integrate the Reacto system with VHDL, extensions that were necessary to augment Reacto with time information, and an investigation of how to automate the generation of VHDL code from Reacto specifications. Next, the results of the experiment are provided based on applying the technique to two well-known benchmark problems, a cruise control and an elevator system.

2 Description of Hardware/Software Co-Design

From a systems engineering perspective, the intent of hardware/software co-design is to initially specify and design a system without regard to whether individual system components are implemented in hardware, software, or a combination of both. The expectation is that through detailed design analysis, the proper decisions regarding the implementation form can be made. For this experiment, our perspective is one of using higher level, abstract languages to initially specify and design the system. Through behavior preserving design refinements, a language like VHDL is used to model, simulate, and analyze design components at a lower level of detail. Once this analysis is completed, the final decision concerning implementation form is made, and the VHDL models are directly used via synthesis techniques to achieve the final implementation. Concrete results which support this approach have already been achieved in the synthesis of Very Large Scale Integrated (VLSI) circuits from VHDL descriptions.

Some existing tools which support our approach are Statemate^{TM} and Synopsys^{TM}. Statemate uses a visual formalism known as statecharts to efficiently and rigorously describe the behavior of a real-time or reactive system [1, 2], and Statemate can generate VHDL code as a step towards "silicon compilation" of hierarchical state machines for hardware designers [2]. Synopsys is a tool used to design and synthesize large and complex circuits using VLSI circuit technology [3]. Synopsys has three high level, non-graphical interface mechanisms for the engineer: logic equations, FSMs in the form of decision tables, and a non-executable subset of VHDL. These forms can be input to the Synopsys synthesis system which produces both behavioral and structural VHDL [3]. Both types of VHDL are compilable and can be executed and tested using standard VHDL simulators. Structural VHDL is a subset of the VHDL language that provides a one-to-one mapping between its syntax and silicon components. Behavioral VHDL is algorithmic in nature and can be converted into structural VHDL or languages like Ada or C if a software implementation is desired.

While Synopsys provides an excellent structural VHDL to silicon component synthesis capability, it lacks a formal-based, abstract language one can use to model systems, and it lacks a transformation-based facility for behavioral VHDL. Statemate has the statechart formalism and a VHDL generation capability. However, Statemate does not contain a proof-based verification component nor does it use transformation-based code synthesis techniques. Additionally, Statemate has limitations in working with multiple instances of identical state machines [4]. Statemate generates VHDL as a flat structure of processes consisting of procedures, and it is currently difficult to use the behavioral VHDL generated by Statemate for chip manufacture without a major restructuring [5]. Another drawback of Statemate and Synopsys are their inability to specify and verify temporal constraints. For these reasons, KBSE technology was investigated to provide the desired features of an abstract and formal-based systems engineering language and tools.

The KBSE technology selected for our experiment was the Reacto Verification System developed by Kestrel Institute. However, this should not be interpreted that Reacto was the best or only KBSE technology available. We selected Reacto because it was readily available and its development was sponsored by the Department of Defense. Additionally, we selected it because of its use of hierarchical FSMs in the form of statecharts. FSMs are a well-known and proven technique for specifying real-time systems, and the incorporation of time constraints was very important to our research sponsors.

2.1 Reacto Verification System

Reacto is a tool for modeling reactive systems via hierarchical FSMs in the form of statecharts, and it contains facilities for verification of system properties, simulation, and synthesis of lower level representations such as VHDL and Ada [6]. The Reacto specification language is built on top of the Software Refinery^{TM} environment which is used to provide [6]: a graphical interface for creating statechart-based specifications, a verification subsystem that includes a verification condition generator and theorem prover, a simulator for the rapid prototyping of specification behavior, and a transformation system. Reacto has three subsystems, the *Reacto Editor*, *Reacto Compiler*, and the *Reacto Simulator*.

- The Reacto editor is used to create and edit the graphical structure of the FSM by specifying and naming hierarchical states and transitions between states.
- The Reacto Compiler transforms a Reacto state machine Specification (R-Spec) into Refine executable target code. The Reacto Compiler's consistency checking function checks to see if the R-Spec satisfies the syntactic and semantic constraints imposed by the system [6]. The Reacto

Compiler's verifier function proves consistency of R-Spec behavior with regard to any well-formed state assertions users make [6].

- The Reacto Simulator highlights active states and transitions in a graphical display window depicting FSM behavior to users. This capability allows users to execute and examine the behavior of an R-Spec.

2.2 Description of VHDL

For the lower level co-design representations, VHDL was selected since it supports the design of components that can be implemented in either hardware or software [7, 8], and it is already an IEEE standard [9], a DoD standard (MIL-STD-454), and proven VHDL compilers and simulators are readily available. Additionally, VHDL allows for the description of concurrent units with temporal constraints, i.e., real-time designs.

The basic VHDL unit of description is a ***design entity***. A design entity represents individual components or functions which make-up a system, and multiple copies of an entity can be instantiated in a system. A design entity consists of an *Entity Declaration* and an *Architecture Body*. Entity Declarations define the inputs and outputs so that other components can interface with it, and the Architecture Body describes the design entity as either a ***structural description*** (a composition of existing design entities) or a ***behavioral description*** (an algorithmic description of the entity's transformation of inputs to outputs).

Algorithms used to define behavioral descriptions are implemented as processes. A process is a communicating sequential program, defined in an architecture body, which runs concurrently with all other processes during a VHDL simulation. Entities communicate via *Signals*. Signals are like variables, except they are managed by ***signal drivers*** which hold the current value and all currently scheduled future values for the signal it is associated with. The event driven simulator updates all signals according to values scheduled for the current simulation time in the signal driver, then it executes all processes that are sensitive to the events. When there are no more signal events scheduled for the current time, the simulator increments the clock to the next event time. The simulation cycle ends when no processes are sensitive to any updated signals.

3 Engineering Design Approach with Reacto and VHDL

Figure 1 provides a "big picture" perspective of the focus of this experiment. From the System Specification store, behavior and constraints information are extracted to generate a Reacto state machine Specification (R-Spec). Three activities can be performed next. The Reacto Verification activity consists of using Reacto's automated verification system to formally verify properties of the R-Spec. The Reacto Simulation activity exercises the R-Spec interactively by updating inputs to the state machine manually or by inputting a simulation file. In the Reacto-to-VHDL Transformation activity R-Spec objects are used to generate a formal VHDL state machine Specification (V-Spec).

While Figure 1 represents the focus of this effort, there are significant supporting activities necessary to implement it. Getting to the point where we can apply Reacto and VHDL to hardware/software co-design problems requires additions to the Reacto system to manage time, concurrency of state machines, and a definition of the transformation process from Reacto-to-VHDL. The time and transformation issues were addressed in this experiment; however, the concurrency issue was left for future research.

3.1 Reacto Augmentations to Specify Time Constraints

The Reacto system doesn't model time, and we defined a means to track and quantify time by declaring a global variable, *clock*, of type integer. One time unit is referred to as the *simulation granularity*, and for each domain, a small enough time value is chosen to allow meaningful examination of system behavior. Time constraints are modeled as *stimulus-response* (the time between an input event and the responding system's output event) and *response-response* (the time between consecutive system output events) constraints [10].

Constants of type integer are declared to represent time constraints. Maximum stimulus-response constraints are called *time limits* and minimum stimulus-response constraints are called *min times*. We call response-response constraints *durations*. Additionally, some means to express the amount of time work takes is needed. We use *transition delay* to express the amount of time the work assigned to a transition takes. Like the clock and time constraints, transition delays are declared as constants of type integer. To model the

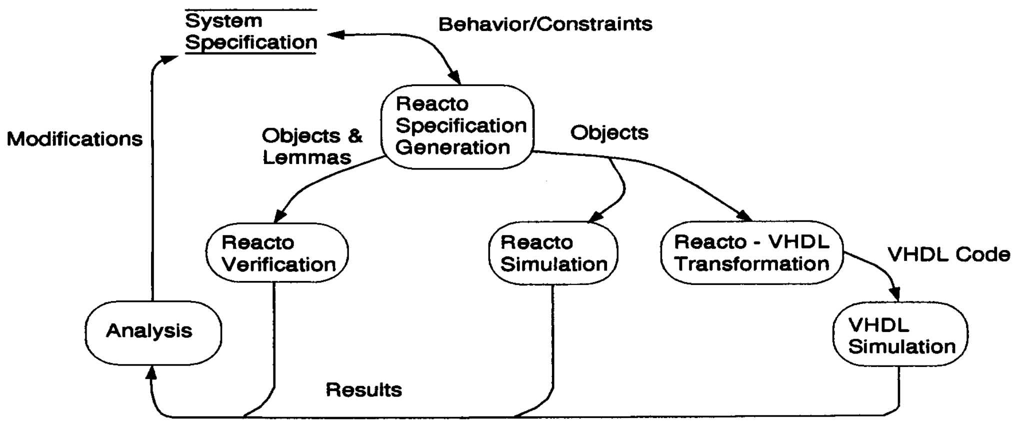

Figure 1: Reacto and VHDL Validation Process

passing of time, the clock is incremented by the value of the transition delay when the transition executes.

Next, we define a means to validate that the FSM behavior does meet the stimulus-response time constraints. We could use error states and error transitions to the error states to verify FSM behavior as described in Dasarathy [11], but by using Reacto assertions, we can discover inappropriate temporal behavior without error states. To do so, we mark the time that events occur using start time logs to measure stimulus-response time constraints. A transition uses start time logs to log the time when it starts responding to an event. As the transition executes, it updates the clock by its delay. After the transition moves the FSM to the next state, Reacto evaluates the state assertion and notifies us if the transition failed to meet the time constraint. The general form of a Reacto assertion to check a stimulus-response time constraint on the object X with both a minimum and maximum constraint is shown below.

$$\begin{aligned} \text{assertion Clock - Min\_X\_Start\_Time} &\geq \text{Min\_X\_Time \&} \\ \text{Clock - Max\_X\_Start\_Time} &\leq \text{X\_Time\_Limit} \end{aligned}$$

As transitions perform behavior subject to a response-response constraint, they set timers. Reacto examines transition predicates sensitive to response-response constraints and executes the response-response sensitive transition when the predicate is true. Response-response constraints always generate a transition whose predicate involves a time value of the form below.

$$\text{predicate Clock - X\_Timer\_Duration} \geq \text{X\_Timer}$$

The Reacto FSM as we have defined it represents a "filing cabinet" allowing us to organize the theoretical process model as described in Levi and Agrawala [12]. The FSMs follow the Mealy machine model, and each transition action represents a process. Computation time is transition delay. Begin constraint is the time the transition predicate becomes true. Process period is the highest frequency of events the transition is expected to respond to. Process deadline is defined by the start time plus the time constraint the transition is meeting.

3.2 Generation of VHDL

For this experiment, a more traditional compiler-based language mapping approach was taken as an initial step. The disadvantages of such an approach are the loss of flexibility provided by a knowledge-based synthesis approach as well as the need for extensive testing of the defined language mappings. While a synthesis approach was out of the scope of this initial effort (but is planned as future work), the compiler-based approach provided a sufficient specification of a Reacto-to-VHDL language mapping that provides a substantial code generation capability.

The first step in defining the mapping of an R-Spec into a V-Spec is to verify that the Reacto state machine and corresponding VHDL state machine display equivalent behavior. Reacto primitive states and transitions map directly to VHDL states and transitions. Although the two state machines are basically the same, Reacto and VHDL themselves are significantly different. We attempted to provide a general behavior preserving mapping from Reacto FSMs to VHDL FSMs. However, a more rigorous and formal mapping between the two is necessary to insure that all Reacto FSMs can be transformed into VHDL in a behavior-preserving manner. A general mapping of Reacto elements to VHDL elements is shown in Table 1.

One fundamental difference between the R-Spec and V-Spec models is clock handling. In Reacto, the clock is controlled with the FSM transitions. In VHDL, the simulator controls the clock. Therefore, events in the VHDL model may occur independently of transition execution, i.e., we can model asynchronous events in VHDL. Events are no longer constrained to happen at discrete times dictated by transition updates of the clock. This allows us to model what we call a *Preemptive Execution Model* (PEM) in VHDL. In the PEM, a transition executing at time $\mathcal{T}$ changes the internal and external state of the FSM at time $\mathcal{T}+$ *Transition-Delay.* If subsequent input events occur before time $\mathcal{T}+$ *Transition-Delay* such that a higher priority transition predicate becomes true, the higher priority transition executes preempting the scheduled state changes of the lower priority transition. Investigating the effects of asynchronous events in the VHDL PEM can shed a great deal of light on transition dependencies when analyzing the behavioral specification.

Developing the Reacto-to-VHDL mapping required a mapping of their data and control models. The data model is relatively straightforward with a few exceptions in transforming higher level data types like sets to equivalent data types in VHDL. However, the VHDL control model is very rich and presents many possible choices for implementing the Reacto control model. For this experiment, a canonical form based on implementing VHDL FSMs as processes was used. A summary of our VHDL code generation technique is provided below:

- Generating Entity Declarations — VHDL entity declarations are used to define the architecture of FSM entities and their interface to the outside world. Inputs and outputs declared in the R-Spec are used to generate input and output port declarations. The first part of the architecture consists of variable declarations and VHDL signal and constant declarations, and these are generated from the Reacto constant and variable declarations. The architecture body and its associated declarations are generated as a concurrent process to model the behavior of the FSM entity.

- Generating Auxiliary VHDL Functions — We must generate functions to perform operations which are defined in Reacto, Refine, or Lisp but not in VHDL. Simple examples include the refine operator implies, the Lisp operator Min, set types, set operations, and logical quantifiers.

- Generating Assertion Procedures — R-Spec assertions are the key to verifying R-Spec behavior and consistency. We use them to verify our V-Spec by transforming them into V-Spec assertion procedures.

- Generating Transition Procedures — We generate V-Spec transition procedures from each R-Spec transition action.

- Generating the FSM Process Body — The FSM process body consists of a simple if statement and a controlling case statement. The if statement resets the transition priority after the VHDL simulator completes a scheduled transition execution. This allows any transition whose predicate is true to execute subject to the case statement. The case statement controls the V-Spec FSM. It maintains the current state and calls the assertion and transition procedures. There is a case statement option for each primitive state in the FSM. It evaluates transition predicates in priority order, executing the first transition with a true predicate if that transition's priority is greater than any currently scheduled priority.

- Testbench Generation — We do not generate the Testbench from the R-Spec. A Testbench has two parts, an entity declaration, and an architecture containing the behavioral description of the Testbench. The final step before VHDL simulation is to generate the test configuration. The test configuration simply identifies which library components we wish to connect into the Testbench architecture.

3.3 Automating The Transformation

Most of the transformation process can be automated. The automated transformation can use the R-Spec source files as input or simply use the R-Spec represented as an abstract syntax tree in the Refine knowledge base producing the V-Spec source code directly from it. Since Reacto is more abstract than VHDL, we typically generate a lot of VHDL code to support Reacto operators, sets, and set operations. Additionally, several other complications make it difficult to completely automate the transformation process.

1. VHDL allows for asynchronous and multiple synchronous events. Resolving potential consistency

Table 1: Reacto to VHDL Mapping

Reacto		VHDL
States		
Name	⟶	Current-State declaration
Own-Vars	⟶	Signals and Variables
Assertion	⟶	Assertion Procedure
Runtime-Check	⟶	Assertion Procedure
Transition		
Predicate	⟶	if-then-else predicate
Label	⟶	Procedure Name
Action	⟶	Transition Procedure Body
Type Declarations	⟶	Type Declarations
Sequences	⟶	Arrays
Sets	⟶	Integer Sets
Tuples	⟶	Records
Input Variables	⟶	Entity declaration, **in** Port
Output Variables	⟶	Entity declaration, **inout** Port and Variable Declaration
Global Variables	⟶	Signal and Variable Declarations
Constants	⟶	Constants
Functions	⟶	Functions
Quantification	⟶	Functions

problems associated with these capabilities requires the use of event history data. VHDL provides this capability via attributes on signals. Therefore, the VHDL assertion procedures generated for Reacto assertions must be augmented with this consistency preserving code. A general procedure for generating this code still needs to be developed.

2. Generating the V-Spec entity declaration requires knowing which Reacto variables are inputs and which are outputs. The R-Spec inputs and outputs are identified by comments and such comments are not currently present in Refine's knowledge base. Adding a Reacto graphical interface specification capability for inputs and outputs would make it easier to specify them and understand the state machine in the context of its environment.

3. VHDL is more strongly typed than Reacto. This causes some difficulty mapping from Reacto symbol types to VHDL enumerated types. Strong typing also causes problems with expressions of type time since VHDL's strong typing forces us to add a conversion factor and explicit type conversion.

4. The concept of a testbench used by VHDL to configure and perform simulation test cases is not used by Reacto. Generalized mechanisms for automating the generation of VHDL testbenches still need to be developed.

4 Results

The engineering design process described in Section 3 was validated using two benchmark specification problems — a cruise control and lift system. The cruise control problem was a valuable first problem because it has well-defined response-response constraints and there are many published solutions. The lift problem tests the ability of the proposed process to accommodate larger designs as well as the ability to model instantiations of an arbitrary number of identical objects. Additionally, the lift system includes response-response, average response time, and both minimum and maximum stimulus-response timing constraints. The lift system also has many published solutions.

Both problems were modeled and simulated using the Reacto system; however, the Reacto verifier was not used in this phase of the experiment. Additionally, VHDL code was manually generated using the procedures outlined in Section 3.2, and the code was extensively tested using the VHDL simulator. Both scope limitations were made only to ensure the masters student performing this experiment would be able to complete his work on time. Use of the verifier, automation of the VHDL generation, and some other research areas are currently being pursued and are explained later in Section 4.1.

Space limitations prevent a detailed example from being presented in this paper (Both example problems are developed in detail in Young's masters thesis [13] which is publicly available from the Defense Technical Information Center or the authors). However, a summary of the specification and design improvements directly attributable to using Reacto and VHDL are provided next.

The specification and design improvements can be categorized into two areas: behavioral and temporal. Behavioral improvements correct errors in the relationship between FSM inputs and FSM outputs (functional requirements). Temporal improvements correct errors in the relationship between input and output events, i.e., correct behavior that violates timing constraints (non-functional requirements). Unfortunately, we did not have a set of specifications based on informal techniques like Real-Time Structured Analysis (RTSA) that were produced in a controlled, experimental environment. Therefore, we used published solutions (many of them partial at best) based on RTSA. As expected, the use of formalized FSMs alone made significant improvements over the RTSA based behavioral specifications. The formality of the Reacto specifications tended to uncover and provide a means of clarify ambiguities in the problem statement. Additionally, even though our version of Reacto did not allow for the simulation of concurrent FSM's, it improved the definition of the interfaces between communicating and potentially concurrent FSMs. The single largest benefit of Reacto over the RTSA was the ability to simulate the specification and produce results *without assertion errors*!

In terms of improvements made from the Reacto to the VHDL design representations, we discovered the following. For the cruise control problem no behavioral improvements were made; however, three temporal improvements were made. For the lift system problem, three behavioral improvements were made and eight temporal improvements were made. The behavioral improvements were achieved because of VHDL's ability to combine FSM's together and simulate them concurrently. Also, the ability to easily generate more powerful test cases in VHDL helped to uncover problems. The temporal improvements were achieved because of VHDL's sophisticated treatment of time-based simulations. This provided the ability to examine the effects of asynchronous and multiple synchronous events. Without this ability, dependencies between transition actions may go unnoticed. Some example dependencies discovered were:

1. A dependency between the transition for accepting new lift destinations and the transition for turning on/off panel lights.
2. A dependency between the transition for accepting new lift destinations and the emergency buttons.
3. A dependency between the transitions for handling external events from the lift environment and the transitions for handling internal events used for lift scheduling purposes.

One other result worth mentioning is that this experiment was performed by a masters degree student. This student had an excellent background in computer engineering and software engineering; however, his knowledge and experience with automated theorem proving was limited. Without this background, use of the theorem prover was not practical. Fortunately, this didn't turn out to be a significant problem, and the student was able to perform extremely well on all other aspects of this experiment. Thus, while an advanced computer/software engineering education was needed to perform this experiment, this education is well within the realm of a masters program in computer or software engineering.

4.1 Future Work

Results from the two example problems clearly indicated three limitations of the Reacto system. First, it needs to be extended to allow concurrency. Second, it needs to have a more sophisticated treatment of time. Last, the ability to better define FSM inputs and outputs in the context of its environment needs to improved. Since this experiment was conducted, concurrency features have been added to Reacto along with axiomatically defined abstract data types. The formally defined abstract data types help with the FSM input/output problems leaving an improved treatment of time as the only major limitation of Reacto. Under a Small Business Innovation Research contract with Rome Laboratory [14], a better treatment of time is being incorporated into Reacto. Also, the VHDL generation capabilities based on compiler like language mappings will be replaced by knowledge-based software synthesis techniques. The goal is to have a system that performs correctness preserving design refinements on well-founded engineering models to produce both structural and behavioral VHDL design components.

As an aside, an interesting observation made during the course of this experiment was the significant impact KBSE technology could have on analyzing and manipulating existing VHDL code as a part of reengineering efforts; however, this was not directly pursued as a part of this experiment.

5 Summary and Conclusions

The Reacto Verification System developed by Kestrel Institute was used to develop a high level, formal-based interface with VHDL that provides increased analysis and design level support for hardware/software co-design problems. This experiment demonstrated that existing KBSE technology can be quickly applied to substantial engineering problems, and this application can be performed by master's level graduate students. In addition to the theorem proving and simulation capabilities already provided by Reacto, extensions were made to incorporate time constraints and compiler-based language mappings were defined to automate the process of generating VHDL design components from Reacto specifications. Experiments with sample problems clearly indicated the complimentary nature and benefits of developing such interfaces between high level, formally defined analysis and design languages like hierarchical finite state machines and lower-level design and implementation langauges like VHDL or Ada.

6 Acknowledgments

We would like to thank Cordell Green, Allen Goldberg, Li-Mei Gilham, and David Zimmerman of Kestrel Institute for their responsive and professional help during the course of this experiment. Funding for this experiment was provided in part by the Air Force Office of Scientific Research and the Avionics Directorate of the Air Force Wright Laboratories.

References

[1] D. Harel, "On visual formalisms," *Communications of the ACM*, vol. 31, pp. 514–530, May 1988.

[2] D. Harel *et al.*, "Statemate: A working environment for the development of complex reactive systems," *IEEE Transactions on Software Engineering*, vol. 16, pp. 403–414, April 1990.

[3] Synopsys, Inc., *Synopsys Design Compiler Reference Manual*, 1991. Version 2.2.

[4] S. L. Smith and S. L. Gerhart, "Statemate and cruise control: A case study," in *Proceedings of the Twelfth Annual International Computer Software and Applications Conference (COMPSAC 88)*, (New York), pp. 49–56, IEEE, Oct. 1988.

[5] M. S. Cohen, "Graphical Behavior Capture to VHDL," in *Proceedings of the Conference on Using VHDL in System Design, Test, and Manufacturing*, (Scottsdale, AZ), pp. 1–8, VHDL International User's Forum, May 1992.

[6] Kestrel Institute, *Reacto Users Manual*, 1992. Version 2.0.

[7] R. Lipsett *et al.*, *VHDL: Hardware Description and Design.* Boston: Kluwer Academic Publishers, 1989.

[8] D. L. Perry, *VHDL.* New York: McGraw-Hill Book Company, 1991.

[9] IEEE Press, New York, *IEEE Standard VHDL Language Reference Manual - IEEE Std 1076-1987*, 1988.

[10] A. M. Davis, *Software Requirements : Analysis and Specification.* Englewood Cliffs NJ: Prentice Hall, 1990.

[11] B. Dasarathy, "Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them," *IEEE Transactions on Software Engineering*, vol. 11, pp. 80–86, January 1985.

[12] S.-T. Levi and A. K. Agrawala, *Real-Time System Design.* New York: McGraw-Hill Book Company, 1990.

[13] F. C. D. Young, "Formalizing, Validating, and Verifying Real-Time System Requirements with Reacto and VHDL," Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, Dec. 1992 (AD-A259224).

[14] Kestrel Institute, *Design Verification and Transformation of Hardware Specifications to VHDL*, 1993. Rome Laboratories SBIR: F30602-93-C-0150.