Verification Tools for Embedded Systems

Edmund Clarke, David Garlan,
Bruce Krogh, Reid Simmons, and Jeannette Wing

Department of Computer Science Department of Electrical and Computer Engineering
Institute for Software Research, International The Robotics Institute

Carnegie Mellon University
Pittsburgh, PA 15213-3890

November 21, 2000

Abstract

We propose a five-year research initiative at Carnegie Mellon University to create a new generation of formal verification tools that can be integrated into design environments for the complex, high-assurance embedded systems that will be required by future military systems. These systems are increasingly distributed, complex dynamic systems that must operate with a high degree of autonomy and survivability in diverse and unpredictable environments. We will focus on the development of new verification methods and tools to provide a rigorous means for checking the integrity and correctness of designs for these systems before they are deployed on target platforms. Our initiative has three broad research thrusts:

- *Verifying System Integrity.* System integrity refers to the correctness of the embedded system operation with respect to the interactions among the distributed software and hardware components. The embedded system must satisfy synchronization constraints, resource constraints, and real-time constraints imposed by the implementation architecture and application requirements. Methods for performing formal verification of these types of constraints that have been successful in hardware and protocol applications will be extended to embedded system applications.

- *Modeling the Environment.* Embedded software systems for military applications interact in complex ways with physical systems and adverse environments. It is thus essential to capture correctly and effectively the continuous dynamics, feedback loops, and unpredictable features of the environment in the models used for formal verification. Our research will draw on recent developments in (i) hybrid system verification to integrate continuous state dynamics with discrete-state models used in formal verification; and (ii) probabilistic verification to model and evaluate critical system attributes such as survivability and fault tolerance.

- *Usability.* In addition to performing research on fundamental methods in formal verification, we will address the formidable barriers that have kept formal verification techniques beyond the reach of practicing design engineers. We propose to develop tools that will extract automatically models used for verification from models already being constructed by design engineers for simulation studies and software development. We will also develop new methods for presenting and interpreting the results from formal verification tools so that they can be used effectively to evaluate the system and to correct the sources of faulty behaviors.

To assure this research initiative addresses the needs of real-time embedded systems in military applications, we have established a partnership with the Honeywell Technology Center (HTC), which will participate with in-kind matching funds. We will meet regularly with HTC researchers who will provide requirements specifications and review research plans and results. HTC will offer challenge problems from real avionics and combat systems and will use tools developed in this project on internal HTC projects to help with the evaluation and guidance of the CMU research.

This initiative will also be an integral technical component of the newly formed High Dependability Computing Consortium, a partnership among Carnegie Mellon University, NASA, the Silicon Valley corporate sector, and other private and public sector organizations.

Project Description

1 Introduction

In virtually all military systems, from land vehicles to aircraft, there is an increasing reliance on embedded systems to improve performance, increase flexibility, and enhance autonomy. Consequently, system testing and certification has become one of the most time-consuming and costly phases of deploying military systems. There is a crucial need for better tools that make it possible to guarantee the correctness of embedded systems when they are designed, before they are deployed on target platforms. The availability of such tools could contribute to significant savings in the time and cost developing and certifying military systems. For safety-critical systems, these tools could save lives. To address these needs, we propose a new research initiative at Carnegie Mellon University to develop innovative techniques and tools for the formal verification of complex embedded systems.

Assuring embedded systems will operate correctly and safely is becoming increasingly difficult for military systems because:

- embedded systems have become highly distributed, multi-task, real-time systems, wrought with all the problems that can arise due to concurrency in complex real-time systems;
- embedded systems are controlling increasingly sophisticated and complex physical systems, which makes it necessary to take into account the continuous physical dynamics and feedback interactions in determining their correctness;
- embedded systems must operate with more autonomy under increasingly adverse and unpredictable conditions, making fault tolerance and survivability key requirements that must be assessed using models of uncertainty in the environment; and
- embedded systems are being developed with a wider, more sophisticated array of models and tools, making it more difficult to abstract essential features for verification and provide useful insight to the engineers.

These factors contribute to the growing concern that current methods for designing and evaluating complex embedded systems are inadequate.

The degree of assurance we can provide today is based on extensive simulation and testing. In some cases, simulation is the only option; testing may be impossible because the system's intended effect is irreversible, or because the environment is inaccessible. Both simulation and testing suffer from being incomplete: they evaluate the system performance for only selected subsets of operating conditions and inputs. In many applications, it is impossible to cover even a small fraction of the total operating space with simulation, and testing is already too expensive. Building a test harness to simulate a component's environment is often more expensive than building the component itself.

The goal of our proposed research is to complement the traditional methods of embedded system validation (simulation and testing) with formal verification, that is, with methods that analyze the behavior of a system over a large (possibly infinite) set of operating conditions without resorting to exhaustive simulation.

Our long-term vision is to see a "row of buttons" on the desktop of the design engineer's workstation that provide immediate access to formal verification procedures that are practical and efficient. Each button would be specific to a property relevant to the engineer's application; for example, in the design of the Enhanced Ground Proximity Warning System, ensuring that we can always go to a "restart" state might appear as a "CheckRestart" button, hiding from the engineer the details of the verification procedure performed underneath. We might have similar buttons that have the effect of checking for deadlock freedom, race conditions, maximum resource usage, minimum delay time, and so on. We believe such property-specific and domain-specific tools based on formal methods will make it possible to design and deploy complex embedded systems with a level of confidence that far exceeds what can be achieved today using simulation and testing. Toward this vision, there are many technical challenges, pragmatic and theoretical. Our proposal addresses the challenges on both fronts.

To address the problems arising in the verification of embedded systems for future military systems, our research initiative has three main thrusts:

- *Verifying System Integrity.* By system integrity we mean correct behavior involving interactions between different components of an embedded system implementation. There are special classes of bugs that frequently arise in embedded system architectures, stemming from *synchronization violations*, *resource constraints*, and *real-time constraints*. These problems are particularly difficult to find using traditional testing because of the large number of situations that must be tried and because the problems often arise only after particular *sequences* of events occur, which are often not reproducible. Real-time performance affects the integrity of an embedded system, particularly when tasks are executed on a distributed hardware platform. Recent efforts in the design of embedded systems have resulted in defining architectural frameworks to support system development. We intend to exploit the constraints inherent in these classes of architectures to make the formal verification of real-time and decisional problems more tractable. This research will build directly on techniques—primarily model checking—for formal verification of hardware systems and protocols, where similar problems of concurrency are prevalent.

- *Modeling the Environment.* In military applications, crucial aspects of correct behavior are defined in terms of the interactions between the embedded system and its environment. These aspects include both the dynamics of physical systems being controlled and the inherent uncertainty of a system's environment. To deal with physical systems, it is necessary for the formal model to include features of the environment, which typically involves continuous-valued variables and continuous dynamics. To deal with uncertainty, it is necessary to assess a system's survivability and fault tolerance with respect to the likelihood of failure events and other contingencies. Thus we plan to explore probabilistic models and probabilistic verification techniques for analyzing a system's robustness.

- *Usability.* To develop "push-button" verification tools that can be used routinely by engineers (non-specialists in formal verification), innovative work is needed to make effective connections to the design environments and languages that they currently use for embedded system development. Our research in this area will emphasize: (i) translators that can produce formal models automatically for embedded system software; and (ii) techniques for

presenting verification results to the user so that they can be interpreted easily and to provide effective information for correcting and modifying designs.

The following section elaborates on the key challenges to developing verification methods for embedded systems and describes target applications for the proposed research in avionics and future combat systems. Section 3 presents the technical approach to be pursued in this initiative. Section 4 summarizes our research plan.

2 Verification of Embedded Systems

Verification is the process of determining whether a system satisfies a given property of interest. In *formal* verification, we use an abstract mathematical model of the system to be checked, and describe the desired behavioral properties with precise, mathematical rigor. Today the best known verification methods are model checking and theorem proving, both of which have sophisticated tool support and have been applied to non-trivial systems, including the design and debugging of microprocessors, cache coherence protocols, and internetworking protocols. Model checking in particular has enjoyed huge success in industry for verifying hardware designs. Within the past four years, companies such as Fujitsu, IBM, Intel, and Motorola have started in-house model checking groups. Companies such as Cadence, IBM, Lucent, and Siemens market formal verification tools to hardware designers.

Despite these successes, formal verification has not been used to a great extent in the development of embedded systems. One reason for this lack of use is that the formalisms and techniques have not been distilled into tools that are accessible to the practicing design engineer. Too much expertise is required to construct the abstract models and formal specifications, making formal verification practical only for applications that merit the attention of personnel with specialized advanced training. Making formal verification part of the embedded system tool kit is more than simply "technology transition," however. The techniques themselves require further research to alleviate existing limitations: at present, they do not scale well to problems with large state spaces and they do not handle important features of embedded systems applications.

2.1 Motivating Applications

Embedded systems are essential to current and future plans for virtually all military and civilian technologies. We now describe two examples that illustrate the challenges of bringing formal verification to bear on the development of these systems.

Future Combat Systems. The Army and DARPA currently conceive future combat systems (FCSs) as highly versatile combat vehicles capable of operating and surviving in diverse battle situations. These vehicles will integrate a wide array of sensory and actuation technologies to achieve a high degree of autonomy at all levels, from navigation to weapons control. Unmanned robotic vehicles are envisioned to be deployed with limited human supervision in "network-centric" combat teams.

The FCS vision will be realized by a large number of complex, highly interdependent embedded systems. Formal verification methods could play a significant role in demonstrating that these embedded systems can indeed deliver the high-assurance performance that they demand. FCS

systems will be very complex dynamic systems expected to operate with high precision and accuracy. They must be self-sustaining, intelligent, autonomous, and robust in adverse and hostile conditions. These characteristics mean that the verification tools must be able to evaluate complex interactions of the system components with each other as well as system interactions with the physical environment.

Avionics. Increasing aviation safety and reducing flight delay are two of the greatest challenges facing the aviation industry. There are several technologies, either recently deployed or in development, aimed at addressing these challenges, including the Enhanced Ground Proximity Warning System (EGPWS), Controller/Pilot Data Link Communication (CPDLC), Cockpit Display of Traffic Information (CDTI), and Airborne Weather Information (AWIN). In addition, over the past decade Integrated Modular Avionics (IMA) has gained popularity as a more cost-effective method (reducing size, weight, power and recurring cost) of fielding advanced avionics systems. IMA systems use a shared resource environment to simultaneously host functions of differing criticality. Such platforms are thus a natural location to place new functionality. This system architecture, however, places a special burden on the IMA operating system to keep functions of different criticality levels from interfering with each other.

These developments require the addition of new systems into the on-board avionics of both existing and newly developed aircraft. New radio standards, critical databases, display systems, and information management and decision systems will be added to support the required on-board functionality. Shared resources, including both processing and I/O, involve a large amount of concurrency, and the software threads can interact in complex patterns. It is critical to ensure that these system changes do not have unintended consequences.

2.2 Research Issues

The FCS and avionics domains described above illustrate the general challenges to bringing formal verification into the set of standard tools used for embedded system development. These challenges include:

- interleavings of multiple task execution;
- inter-task dependencies and synchronization requirements;
- system resource constraints;
- hard real-time constraints;
- interactions with complex physical dynamic systems;
- adverse, unpredictable environments; and
- stringent requirements for autonomy, fault tolerance, and survivability.

In this research initiative we propose to meet these challenges by focusing on both the fundamental science of formal verification and the practicality of doing verification in a design engineer's work environment. Thus, we will focus on the aforementioned critical features of embedded systems for distributed real-time control of dynamic systems, i.e., the type of embedded software designed

for weapons control, avionics, and control loops in unmanned vehicles for water, land and air. Furthermore, to make the results of our research accessible to, and usable by, the design engineers that create the embedded systems. we will develop prototype tools that embody the verification techniques we devise.

3 Technical Approach

Our technical approach has three major thrusts. First, we propose to develop formal methods to address the principal difficulties arising in assuring the internal system integrity of military embedded systems applications. This research will emphasize methods for dealing with synchronization constraints, resource constraints, and real-time constraints. Second, we will develop new methods for integrating models of the continuous and stochastic features of the environment into the verification process. This research will draw on recent developments in the areas of hybrid system verification and probabilistic verification. Third, we will address the major barriers to making formal verification part of the design engineer's toolbox. In particular, we will focus on making verification tools usable by developing new methods for extracting abstract models needed for verification from models and languages used by practitioners, and by creating new methods for presenting verification results to engineers so that they are meaningful and useful for evaluating and re-designing the system to meet user specifications.

The following sections describe the research plans in each of these three areas. Figure 1 depicts how the different pieces of our research initiative fit together. At the top is an embedded system that interacts with an environment. Provided in the design engineer's toolbox is a set of "push-button" checkers; each tool is used to check a property about a system's integrity or about an interaction between the system and its environment, given a possibly stochastic model of the system's environment. Underlying these tools are foundational building blocks such as algorithms for model checking, languages for specifying tasks, and analyses over continuous domains. The Carnegie Mellon team's basic research efforts will focus on advancing the state of the art in the bottom two layers of the figure. Our applied research and technology transition efforts, in close collaboration with Honeywell and HDCC consortium members, will focus on the second layer from the top.

Interwoven throughout the following subsections are some cross-cutting themes of our research approach:

- Exploit *model checking techniques and tools*: extending their capability, scalability, applicability, and usability. Our starting points are off-the-shelf model checkers such as SMV, Spin, and FDR; research prototypes for checking discrete real-time systems (e.g., Verus) and hybrid systems (e.g., HyTech and CheckMate); and new model checking techniques for continuous real-time systems and probabilistic verification.

- Focus on the *task level*: First, rather than verify code, we focus on verifying designs; these code abstractions come in the form of task-level system architectures for embedded systems (Section 3.1.1) and synchronization skeletons (Section 3.3.1). Second, we plan to build tools for existing and new task executive languages that will improve the usability by design engineers to use our verification methods.

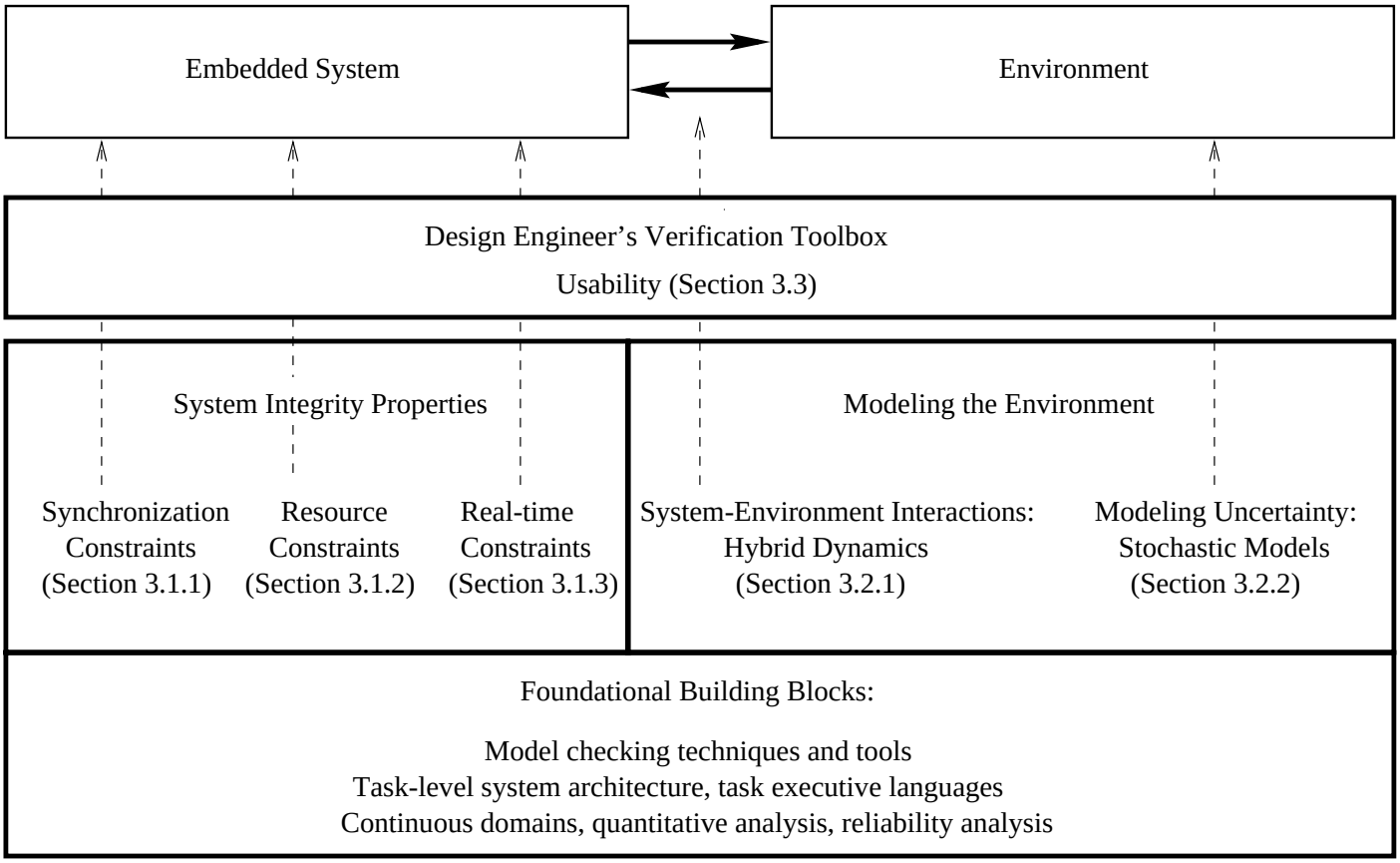

Figure 1: An Integrated View of Our Verification Initiative

- Incorporate *continuous domains* in our models: specifying and verifying new classes of system behavior by using: "time/clock" variables (for handling resource constraints), quantitative models (for capturing numeric domains such as for air pressure, electromechanics, hydraulics, temperature, and wind speed); and stochastic models (for doing cost-benefit analysis such as whether to add physical redundancy for increased reliability despite the additional weight it incurs).

3.1 Verifying System Integrity

By system integrity, we mean the correctness of the system with respect to the interactions that occur between the system components and the constraints imposed by the system platform and the application requirements. The following subsections describe the proposed research on problems inherent to the integrity of embedded systems for military applications.

3.1.1 Synchronization Constraints

Most embedded systems are constructed as loosely coupled sets of tasks. Synchronization between tasks is needed to ensure proper functioning of the system. Such synchronization constraints may be relative (e.g., "To switch from autonomous mode to manual mode requires going to STANDBY first.") or more quantitative in nature ("Weapons systems must be on for at least 10 minutes before use.").

Software errors that cause a system to violate these constraints often arise from subtle interactions among concurrent tasks (e.g., race conditions, missed events, deadlocks, resource contention, etc.). Since these bugs may manifest themselves only for particular interleavings of actions, the programmer may not be able to reproduce the problem at will. Thus, it is often very difficult to track down the sources of synchronization bugs in using traditional testing and code inspection techniques.

Fortunately, such problems are particularly well-suited to automated formal verification using model checking. Model checking starts with a finite state model of the software system under consideration. The model checking tool then examines all possible interleavings of computations in the model, to expose any violations of task synchronization constraints. To the extent that the model faithfully reflects the actual system, errors found in the model represent problems with the system and vice versa.

Our research on synchronization constraints will focus on specific synchronization mechanisms used in embedded systems, and extraction of the appropriate models to analyze these mechanisms. To support coordination among behaviors, while maintaining loose coupling between tasks, embedded systems typically use one of two forms of component integration:

1. Using *publish-subscribe interaction* [39] tasks are responsible for announcing (or "publishing") significant events. Other tasks may register to be informed of (or "subscribe to") a set of events. When one task publishes an event, it is sent to all subscribers. In this way tasks remain independent since a publisher does not know the identity, or even existence, of other subscriber tasks.

2. Using *cyclic tasks with shared variables,* a similar effect is typically achieved by assigning each task to a task slot that is repeatedly run at some fixed period—say every 50 milliseconds. At each activation a task examines a set of input variables in some shared variable space, and based on those values, it writes derived values to a set of output variables. Readers of shared variables do not know which tasks set those values, or which tasks will "consume" the values of the variables to which they write. Thus the shared variables serve the role of the events in the publish-subscribe architectures of embedded systems, with the analogous goal of decoupling tasks while permitting communication.

While both architectures are good from an engineering point of view, reasoning about their aggregate behavior is problematic. One would like, for example, to be able to guarantee that if a sensor detects that when a temperature value goes over a maximum limit then some other part of the system will take appropriate corrective action to preserve overall stability. This detection is hard to do with such systems, however, because interactions between the parts of the system are indirect and asynchronous. In particular, the nondeterminism inherent in the invocation of tasks and the communication delays inherent in a distributed publish-subscribe system (many events may be in transit) make it difficult to reason about properties such as global invariants or timely response to certain trigger events. (For example: Are sufficient events published to allow interested parties to respond in a timely fashion? Is there potential interference between components that subscribe to the same event?) Analogously, for shared-variable cyclic systems it is difficult to determine whether a given assignment of tasks to periodic buckets and the ordering of tasks within a bucket are sufficient to guarantee some property under all possible scenarios. (For example: Are there implicit ordering assumptions in the reading and writing of shared variables that may be violated?)

While model checking would appear to be a good solution for verification of such properties, it is not immediately obvious how to map these kinds of systems into an appropriate model. In previous research we have developed techniques for reasoning about these kinds of systems [14]. The basic idea underlying the work is that one can provide a precise description of the cause and effect of each event to reason locally about how a component behaves in isolation. Then using local correctness of the components and properties of events, we infer the desired global property of the overall system. We believe it should be possible to map these descriptions automatically into a form that can be model checked. Moreover, many of the standard problems about event-ordering and interference (mentioned above) can be built-in as standard checks that are automatically verified. We have already taken initial steps towards developing such an automated translator for publish-subscribe systems [17].

3.1.2 Resource Constraints

Embedded systems are typically implemented on distributed hardware architectures with limited resources. Therefore, designers of embedded systems must be concerned with resource allocation, ensuring that they do not exceed resource bounds. Model checking is particularly well suited for detecting potential violations of resource constraints because it can examine all possible execution traces and interleavings of tasks.

Resources can be categorized into four types [13]:

1. *Single sharable*—a discrete, single unit, such as a camera. Sharable resources can be used by only a single task at any given time.

2. *Multiple sharable*—multiple discrete units, such as multiple rovers exploring enemy terrain. Multiple sharable resources may be interchangeable, but difficulties arise when the resources are interchangeable for some purposes but not others (e.g., wheeled rovers, but with different sensors).

3. *Consumable*—continuous resource, such as fuel. Consumable resources can be used for any task, at any time, but eventually can be used up.

4. *Renewable*—continuous resource that can be replenished, such as disk space or wireless communications bandwidth. Renewable resources are critical in field operations, and must be very carefully managed to avoid over-subscription.

Single sharable resources can be synchronized using a mutex boolean variable. It is straightforward to model this with model checkers. Multiple sharable resources can be represented using an integer variable with a given, finite range. While, in general, *allocating* multiple sharable resources is a computationally complex problem, *verifying* the absence of resource conflicts is no more difficult than other discrete, symbolic model-checking problems.

On the other hand, the major research issue in dealing with resource constraints is reasoning about continuous (consumable and renewable) resources. Here, we propose to adapt real-time model-checking algorithms [24, 6] (Section 3.1.3). In many cases, resources are consumed and renewed at a constant rate and their constraints are independent. This corresponds to multi-rate clocks, or rectangular automata, for which many verification problems of practical interest are decidable [2, 20]. Consumption and renewal rates that are not constant can be modeled in this framework using piecewise constant approximations.

Often, it is important to know not just whether a resource constraint has been violated, but how close one is to a violation in the best and worst cases. This gives the engineer an understanding of potentially critical points in the design. Again, multiple clock rates can be used to model the dynamics of the consumption/renewal processes. Reachable sets of the system trajectories can be computed as fixed points of operators in the space of continuous clock values. This leads to polyhedral regions that can then be introduced as constraints over optimization problems (linear programming problems) to evaluate the worst-case proximity to specified resource constraints. We propose to create tools that automate this process of computing these values for critical constraints selected by the designer.

Another significant research issue is how to specify resource constraints and utilization in a way that can be automatically processed for formal verification. We propose to develop a syntax for specifying such constraints and embed this language in the engineer's programming environment—either by extending an existing task description language, such as ESL[18] or TDL [37], or by defining them as stylized comments that can be added to general-purpose programming languages, such as C. For this effort, we will borrow heavily from work in planning and scheduling [31], where declarative representations of resources are commonly used.

We will then develop translators to create SMV models automatically from the resource constraint specifications, which would then be combined with the models generated for synchronization constraints (Section 3.1.1) to determine if the constraints hold for all possible task executions. We will also extend our work in automatic generation of explanations to handle counterexamples that demonstrate resource violations (Section 3.3.2. The development of these automatic transla-

tion and explanation capabilities, together with the development of novel model-checking algorithms to handle continuous resources, will provide engineers with important and valuable new tools for verifying resource-limited embedded systems.

3.1.3 Real-time Constraints

Many specifications for real-time embedded systems are quantitative ("X must occur at least T seconds before Y"). Although there has been considerable work already on verifying real-time systems, including research at Carnegie Mellon [6], more research is needed in order to handle models of the complexity needed for future combat and avionics systems.

In understanding why a quantitative synchronization constraint has failed, it is often useful to know how far the system actually falls outside of the desired bounds. Previous research at Carnegie Mellon on model checking of discrete real-time systems has introduced the concept of quantitative timing analysis [7]. Using this new method it is possible not only to check the correctness of a system but also to compute quantitative timing information about the behavior of the system, such as minimum and maximum times between the occurrences of events. The method can also determine the number of occurrences of an event in all paths between two other specified events. Moreover, it is also possible to identify how often a task has been blocked, or if lower priority tasks have been executing in the interval between the task's execution start and finish times. With this information it is possible not only to assert the correctness of a system but also to understand its behavior and in many cases to identify optimizations to the design. Even more detailed information can be obtained using selective quantitative analysis, which allows the analysis of specific execution sequences in the model, instead of checking all of its behaviors.

While quantitative timing analysis is a powerful technique, more research is needed before it can become a practical tool. One issue is that distributed systems are often better modeled using continuous time rather than discrete time. This is particularly true for quantitative timing analysis. Currently, it is possible to determine minimum and maximum time delays between events. However, other verification algorithms such as condition counting or selective quantitative analysis are not supported. We propose to extend quantitative timing analysis to continuous-time models and to develop model checking tools that can handle task execution systems of realistic complexity.

A second limitation of model checking technology is its inability to handle systems whose task structure varies at run time. Current techniques require one to create models that contain all possible tasks, even though some may not be active at any given moment. The creation or deletion of a new task is then modeled by activating and deactivating tasks. This approach has the disadvantage of requiring very complex models, and of not being a close match with the point of view taken by many task executives, in which tasks are freely created and deleted. We propose to investigate extensions to model checking that will permit more natural representation of such systems, and in particular, dynamic task creation and deletion.

3.2 Modeling the Environment

For many military applications, embedded systems control dynamic physical systems and operate in complex environments. Consequently, many aspects of correct behavior are defined in terms of how the embedded system components interact with the external world. Specifications defining

only the internal system integrity as discussed in the previous section are inadequate to completely characterize all of the important system requirements. Some requirements involve performance of feedback loops, such as guarantees that the embedded system delivers commands in a certain sequence when the physical system being controlled moves at a certain speed. Other requirements may be given in terms of probabilities with respect to environmental conditions, such as likelihood of survival given a particular stochastic characterization of the battle conditions.

Specifications of these types make it necessary for the verification procedure to include models of the embedded system's environment. In this initiative, we propose to address two of the most challenging problems posed by environmental models. First, we propose to develop new methods for incorporating models of continuous dynamics as they arise in controlled systems and the environment into the inherently discrete models used for model checking. Second, we propose to extend methods for probabilistic verification to address the problems arising in verification of systems embedded in environments modeled stochastically. Research in each of these directions is described below.

3.2.1 Hybrid Dynamics

Constraints related to the dynamic behavior of an embedded system and its interactions with the environment are among the most difficult to incorporate into a formal verification procedure because traditional formal methods are based on discrete-state models that cannot easily capture the continuous-parameter variations that characterize the system dynamics and environmental interactions. It is essential that these constraints be represented and evaluated as faithfully as possible so that the system can be operated to its full limits while avoiding catastrophes. Assuring dynamic constraints are not violated by over designing the system is not an attractive option when there are stringent constraints on the available size, weight, and energy.

The operation of embedded systems and their interactions with the physical world are governed by the laws of continuous dynamics, usually modeled by differential and algebraic equations. In contrast, tools for formal verification are based on models of computation that are inherently discrete state/event models. Timing is typically the only mechanism for incorporating features of physical environments into these models. Our goal in this part of our project is to create tools for analyzing models of physical dynamic systems to abstract tractable behavioral models to verify embedded system performance with respect to environmental constraints.

Embedded systems interacting with continuous dynamic environments are *hybrid dynamic systems*, that is, systems with both continuous and discrete state variables. Hybrid dynamics can also appear in the environment itself when the continuous dynamics change depending on discrete conditions, such as changes that occur in robotic mechanisms depending on whether or not certain surfaces are in contact. Central issues in recent research on hybrid dynamic systems include numerical methods for simulation, identifying and characterizing qualitative behaviors such as invariants and limit cycles, synthesizing controllers for various objectives such as stability and reachability, and formal verification [30, 35].

This latter research on formal verification is most germane to the present proposal. In contrast to model checking for finite-state systems, algorithmic verification is possible for hybrid dynamic systems with only very simple classes of continuous dynamics [22, 21, 29]. Consequently, the best one can hope for is the verification of properties of conservative models that represent outer or

inner approximations to the families of exact system behaviors. Based on our experience and the experience of others in building tools to verify properties of hybrid dynamic systems, we plan to generate discrete system models from continuous ones, and thus generally handle continuous state variables, where time is just a special (but important) case.

We propose to use the differential equation models to generate discrete models of the environment that can be composed as needed with the existing discrete models of the computing system. This approach has been taken in the past, but the discrete models of the continuous dynamics have been either created manually and in an ad hoc manner [33], or generated using numerical techniques that do not guarantee the model is correct [27]. On the other hand, tools for verifying hybrid dynamic systems directly (combining the discrete and continuous dynamics in a single model) can handle only very small examples [3, 23, 36].

Our work on discrete-state approximate quotient transition systems will form the basis for these models [8]. The fundamental problem in obtaining discrete models of hybrid systems is the computation and representation of reachable sets for the continuous dynamics. Current methods for computing these reachable sets are effective only when there are no more than three or four continuous state variables [9, 34]. To deal with this problem, we will investigate methods for decomposing the continuous dynamic models into low-dimensional interacting subsystems, each of which can be handled effectively with our computational tools. Formally, this approach corresponds to projecting the full state vector into lower-dimensional subspaces as proposed in [19]. Alternative representations of continuous dynamics will also be investigated, including ellipsoidal approximations that are possibly less complex than the polyhedral approximations we are currently using [28].

The key to this approach will be the ability to select appropriate operating regimes for the continuous dynamics that need to be approximated. Toward this end, we will create tools that make it possible for the domain expert who is familiar with the dynamics and the requirements to be verified to define the range of operation that needs to be captured.

Our approximation approach here is analogous to our approach for extracting synchronization skeletons of Section 3.3.1, which are yet a different abstraction of the system. It is also consistent with our focus on task-level execution, allowing us to analyze tractable models of inherently large state spaces due to continuous dynamics.

3.2.2 Stochastic Models

Unpredictable behavior of the environment frequently makes the verification problem more complex than current techniques can handle. We can model the environmental events as nondeterministic transitions. In some cases, the probabilities of events can be estimated using information like past mission data. This makes it possible to associate probabilities with the state transitions that correspond to these events. As a result, the environment of an embedded system can often be modeled as a stochastic process.

Numerous theoretical studies have been written on the probabilistic verification problem. Algorithms for solving the problem have been given in several papers; for example, there is a model checking algorithm for linear temporal logic which is exponential in the size of the specification and polynomial in the size of the Markov chain. However, there are currently no probabilistic model checkers that are able to verify realistic systems. The bottleneck is the construction of the

state space and the necessity to solve huge systems of linear equations. We are developing a more efficient alternative, which performs the probability calculations using Multi-Terminal Binary Decision Diagrams (MTBDDs) [10]. MTBDDs differ from Binary Decision Diagrams (BDDs) in that the leaves may have values other than 0 and 1; in this case the leaves contain transition probabilities. Preliminary results indicate that MTBDDs can be used to represent extremely large transition probability matrices.

Transition probability matrices represented as MTBDDs can be integrated with a symbolic model checker and have the potential to outperform other matrix representations because they are so compact. For example, Hachtel and his colleagues [4] at the University of Colorado, Boulder have developed symbolic algorithms to perform steady-state probabilistic analysis for systems with finite state models of more than 10^{27} states. While it is difficult to provide precise time complexity estimates for probabilistic model checking using MTBDDs, the success of BDDs in practice indicates that this is likely to be a worthwhile approach.

For writing specifications we use Probabilistic real-time Computation Tree Logic (PCTL) introduced by Hansson and Jonsson in 1989 [5]. PCTL augments the logic CTL (developed by Clarke and Emerson) with time and probability. Formulas are interpreted over finite state discrete-time Markov chains. The logic is very expressive and has a simple model checking algorithm that can be implemented using MTBDD-based techniques. A model checker for this logic can be a useful tool in the dependability analysis of fault-tolerant real-time control systems, performance analysis of commercial computer systems and networks, and operation of automated manufacturing systems.

We are implementing a model checker for PCTL within Verus, a verification tool tailored for analysis of real time systems [6]. In order to model and verify stochastic systems in Verus, we have extended the specification language and implemented a subset of the PCTL operators. We have already verified a few examples, including a simple communication protocol, an automated manufacturing system comprising two machines, and a fault-tolerant computing system. Currently, we are verifying a railway interlocking controller with a state space in the order of 10^{22} states. For this part of the research initiative, we propose to implement the full PCTL logic, continue the development of the required MTBDD operators, and evaluate how well PCTL model checking performs on avionics applications.

Our work on implementing an MTBDD-based PCTL model checker for a version of Verus lets us incorporate the stochastic nature of an embedded system's environment into our models. However, it fundamentally relies on the assumption that events occur independently. In practice, events, especially failure events, do not necessarily occur independently. For example, in an aircraft with two redundant nodes used for load balancing, if one node fails, then the other node is more likely to get overloaded and subsequently fail. Also, an exceptional or unanticipated external environmental event usually triggers a fail-safe shutdown procedure (e.g., if the sensor's current reading exceeds the maximum threshold for safe operation). The shutdown procedure will either (1) cause further failure recovery subtasks to execute, effecting a cascade of repair events, or (2) simply abort the mission (e.g., stop the unmanned tank).

In our work on modeling and analyzing survivable systems [26], we need to address the same problem. We use a stochastic model called *Constrained Markov Decision Processes (CMDPs)* [1]. A CMDP can have a mix of nondeterministic and probabilisitic state transitions. Consequently, a CMDP is more general than a nondeterministic state machine. CMDPs are also a generalization of Markov Chains, since the transition probabilities can depend on past history. Finally, CMDPs

allow one to attach cost constraints to state transitions. Thus, CMDPs can be used to model a wider range of systems than the methods discussed previously. For example, CMDP models have been successfully used in applications [1] from the telecommunications industry, e.g., to perform maximization of the throughput of certain traffic, subject to constraints on its delays; and in decision sciences, e.g., to develop a pavement management system for Arizona to produce optimal maintenance policies for a 7400-mile network of highways.

Existing CMDP algorithms can be used to do both *reliability analysis*, e.g., to compute the worst-case probability that a task started will eventually finish, and *latency analysis*, e.g., to compute the worst-case expected finishing time of a task. However, in addition to modeling history-dependent events and enabling reliability and latency analysis, another important advantage of using a general model like CMDPs is its ability to model cost constraints. Using costs associated with state transitions, we can perform *cost-benefit analysis*. For example, suppose we have a fixed budget and need to decide which communication links should be upgraded to achieve maximum increase in system reliability. We may not be able to upgrade all links and must choose those that would be most cost-effective. In a CMDP, cost information is represented very generally in terms of functions that assign real values to $\langle \mathrm{state}, \mathrm{action} \rangle$ pairs in the state transition model. One promising application of this CMDP capability is to use these functions to model the kinds of resources described in Section 3.1.2. This idea is still unexplored territory; we propose to understand how the CMDP model can help us do cost-benefit analysis of resource usage.

In our own preliminary work we have built a model checker that can verify properties of CMDPs [25] and have successfully applied it to two non-trivial examples in the financial services application domain. However, we have not yet tried to use the model checker to verify properties of embedded systems. We have also hit against the limitations of the NuSMV-based model checker in our current tool because we need to model explicitly in NuSMV a history variable in our states, thereby exploding the state space. Thus, for this part of the research initiative, we plan to build a newer version of our tool based on an "explicit-state" model checker (e.g., Spin) to avoid modeling history explicitly, and to apply our model and newer tool on embedded systems applications.

More generally, we propose to identify the practical and realistic environmental conditions under which the Markov assumption applies. We can use tools based on MTBDDs and PCTL to analyze such examples. More ambitiously, we propose to identify and express the dependencies (especially due to failure events) resulting from dynamic interactions between the system and environment. Here, we need to develop our CMDPs methodology further, for example, in pursuing how cost constraints in a CMDP model relate to resource usage.

3.3 Usability

There are two significant barriers to the use of model checking-based verification of complex embedded systems. We refer to these as problems of *usability.* To use a model checker, a designer must first describe the system and requirements in the formalism used by the model checking technology. For a complex system this can be a non-trivial task, since it involves manually mapping a (usually) infinite-state software system to a finite-state model. Moreover, the model must be one that is small enough that it can be checked in a reasonable amount of time. Finally, the designer must be able to interpret the results—typically a set of counterexamples showing execution paths where a property is violated. In combination these represent significant barriers for system

designers and developers to use these verification tools on a routine basis.

3.3.1 Extracting Models

We propose to develop new static analysis methods to extract models, called *synchronization skeletons* [11], automatically. A synchronization skeleton is a projection of the code that includes only details relevant to analysis of properties of concurrency. Our research will be based on the use of *slicing*. In earlier work we applied slicing to hardware description languages [12]. In this project, we will extend these techniques to software by developing ways to use slicing techniques for extracting synchronization skeletons from analysis of concurrent programs. The second approach will be to use formal specifications to extract automatically code fragments that are relevant for model checking. This work is similar to that of Engler [15], who has investigated extraction of finite state descriptions from C programs by eliminating fragments of the program that are unnecessary for verifying a property of interest. However, in his work properties are specified operationally using C programs and extracted manually from the source code. In this project, we intend to develop more automated techniques in which properties of interest are specified declaratively.

Our second attack on the problem of extracting models will focus on automated creation of models for *task execution languages.* In many cases, specialized languages are used to implement task executives for autonomous systems. Typical languages, such as RAPs [16], ESL [18], and TDL [37], contain constructs for hierarchical goal decomposition, task synchronization, resource management, execution monitoring, and exception handling. In such languages, the synchronization constraints are explicit, so extraction and translation of constraints are facilitated. In previous research at Carnegie Mellon, in collaboration with NASA/Ames [32, 38], we developed a tool to produce SMV models automatically from models written in MPL, the language used in the Livingstone fault diagnosis system [40].

We propose to create similar translators for task execution languages. The difficulty here is that such languages are typically quite expressive, and care must be taken to produce formal models that can be verified efficiently. In addition, we propose to develop specialized languages that enable engineers to encode properties of interest easily.

We will focus on being able to encode requirements specified by our Honeywell partners in order to test the approach. In some cases, we will extend the task execution languages so as to be able to support directly the inclusion of such requirements, much as the “assert” property in C.

3.3.2 Interpreting Verification Results

The second way we will improve usability is by developing new ways to help designers understand the meaning of the counterexamples produced by a model-checker. Current work is developing algorithms that can produce causal explanations of counterexamples for MPL models [38]. This work instantiates a counterexample and its associated SMV model into a TMS (Truth Maintenance System). By propagating dependencies, a justification tree is created, which can then be manipulated to form a linear explanation. For this project, we will apply and extend this work to explain counterexamples for task executive programs. In particular, we will use techniques adopted from AI planning to find concise explanations that focus on the most germane portions of the justification tree. We also propose to develop techniques that enable developers to view and browse

the temporal evolution of counterexamples. The idea is to create virtual execution traces from the counterexamples that can be utilized by existing visualization tools for task executives.

4 Project Plan

This project is envisioned as a major new research initiative at Carnegie Mellon, bringing together researchers with expertise in formal verification techniques, real-time control, and embedded and autonomous system applications. We believe that the whole will truly be more effective than the individual parts in this endeavor because of the need to bring many perspectives and skills together to create the kinds of tools that are needed for verifying embedded systems. We will also be working with researchers at the Honeywell Technology Center (HTC) and High Dependability Computing Consortium (HDCC) partners throughout the project who will help us identify the key barriers that currently exist to putting these tools in the hands of practicing engineers. The HTC researchers will also provide insight into key military applications, including avionics systems and future combat systems. The following paragraphs describe the project plan in more detail.

4.1 Organization

The research initiative will be housed in the School of Computer Science at Carnegie Mellon University. The five principal researchers are from the Department of Computer Science (Clarke, Wing), the Robotics Institute (Simmons), the Department of Electrical and Computer Engineering (Krogh), and the Institute for Software Research, International (Garlan). Additional personnel includes eight Ph.D. candidates, a staff research programmer, and an administrative assistant. The roles of the principal personnel are as follows.

Jeannette Wing (PI) will have overall management responsibility for the project. She will also be responsible for the stochastic models that enable reliability and cost-benefit analyses. She will provide expertise in formal specification language design.

Edmund Clarke (co-I) will have primary responsibility for extracting synchronization constraints, metric model checking (for temporal and resource constraints), and probabilistic verification.

David Garlan (co-I) will have primary responsibility for formalizing publish-subscribe systems and developing related tools. He will share responsibility for developing languages for specifying properties.

Bruce Krogh (co-I) will be responsible for issues relating to verification of hybrid systems. In addition, he will provide expertise in real-time control and embedded systems.

Reid Simmons (co-I) will have primary responsibility for modeling task executive languages and explaining counterexamples. He will share responsibility for developing languages for specifying properties. He will also provide expertise in autonomous systems.

Project meetings will be held regularly with the following objectives: to present technical background and new results, to define case studies and requirements, to establish collaborations between individual researchers in the group. We will also meet quarterly with industrial partners

from HTC. These meetings will alternate between Carnegie Mellon and HTC, Minneapolis, to enhance the presentation and exchange of ideas between the two research groups. These meetings will focus initially on requirements and challenges in applying formal verification to embedded systems. Proposed solution techniques will be reviewed by the research groups. During years two and three, discussions will focus on test cases and demonstrations of prototype software. Years four and five will emphasize the transfer of technologies developed at Carnegie Mellon to HTC and HDCC consortium members for evaluation and testing.

4.2 Milestones

Here are our annual milestones:

- **Year 1**
  - develop languages for specifying synchronization and resource constraints
  - extend real-time verification to continuous-time constraints
  - develop model checking algorithms for hybrid models of environments
  - develop techniques for extracting task-level models
  - investigate automation of counterexample interpretation
  - identify test cases from target application scenarios
- **Year 2**
  - implement prototype tools for constraint verification (synchronization, resource, real-time)
  - formulate theory of correctness of hybrid system model checking algorithms
  - evaluate probabilistic verification algorithms for test case scenarios
  - develop stochastic models of uncertainty of environments for embedded systems
  - extend and design new task executive languages
- **Year 3**
  - hold workshop with industrial researchers
  - test prototype tools at HTC
  - begin development of tools for usability, connecting to engineering environments
- **Year 4**
  - establish test cases for technology evaluation
  - work with industrial partners to set up test cases and evaluation procedures
- **Year 5**
  - evaluate fully prototype tools developed in this initiative

- hold industry workshop to disseminate results
- develop plans for technology transfer
- identify directions for future research

Throughout Years 1 through 5, the Carnegie Mellon team will work on the underlying scientific and mathematical foundations that support the engineering needed to make verification tools accessible to design engineers.

4.3 Industrial Partnerships

As the world's leading supplier of advanced avionics systems to civilian, space, and military markets, Honeywell is uniquely qualified to help define requirements for formal verification of embedded systems and to evaluate the potential impact of the results of this research initiative on the development of both military and commercial avionics products. Honeywell engineers have already pursued numerous research projects aimed at verifying the correctness of a variety of software and hardware systems. The work to be done under this proposal will build on the results of existing research, and extend these methods to achieve comprehensive verification of highly complex embedded systems.

Avionics components under development by Honeywell constitute ideal test cases for formal verification of safety critical properties in embedded systems. Researchers at HTC will apply prototype tools from this initiative to in-house projects focusing on the development of in-flight integrated modular avionics systems and future combat systems. Further, HTC can help define paths for the transition of this technology into the aircraft avionics certification process, setting a precedent for the use of formal methods as part of the standard process for ensuring safety-critical software correctness. As a global controls company, Honeywell is positioned to utilize the results of this research to improve embedded control systems in areas beyond aviation. Honeywell's Industrial Automation and Control division, for example, produces numerous embedded industrial control systems which will benefit from better tools for formal verification.

In addition to our close collaboration with Honeywell, our initiative will be an integral technical component of the newly formed High Dependability Computing Consortium, a partnership that includes Carnegie Mellon University, NASA, the Silicon Valley industrial sector, and other public and private agencies. HDCC's mission is "to promote and conduct research that will enable us to create highly dependable software systems at an affordable cost." Thus, since Carnegie Mellon is a key partner in HDCC, our research team has ample opportunity for validating—by both industry and government users—the results from our work on the formal verification for embedded systems.

4.4 Educational Activities

This research initiative will support eight new Ph.D. candidates drawn from the Computer Science Department; the Robotics Institute; the Software Research Institute, International; and the Electrical and Computer Engineering Department at Carnegie Mellon University. Thus, we will have a research team that will give the participating students considerable experience with collaborative research in a multidisciplinary project. The interaction with the Honeywell researchers and insight into the real world applications at HTC will provide these students an unusual chance to

see the immediate importance and relevance of their research. Honeywell has also agreed to host Ph.D. candidates for summer internships. This opportunity will enhance further the educational experience from working on this initiative, as well as helping with the transfer of technology to industry.

A research initiative of this size will also have an impact on the education program beyond the students that are supported directly by this program. Graduate research seminars will be held regularly throughout the duration of the project, focusing on the recent literature in the areas of real-time embedded systems, high assurance systems, formal verification and applications. A number of undergraduates will also be involved in this research program through projects and summer jobs. Finally, we anticipate the presence of this project on campus at Carnegie Mellon will have a direct impact on improving the several undergraduate and graduate courses, which we already teach, in the areas of verification and real-time embedded systems.

4.5 Dissemination of Results

The results of this research initiative will be disseminated through the regular publication of technical reports, presentations at conferences, and publication of journal papers. In addition to these conventional mechanisms for disseminating academic research, we will host two workshops for industrial, academic and military researchers. The first will be during the third year of the initiative. The objective of this workshop will be to highlight the results of this research initiative and make available the prototype tools for formal verification to the wider user community. The experiences and feedback from these users will be important during years four and five of the project as we focus on the transfer of the technology to target end users. The second workshop will be in the fifth year. The objective of this workshop will be to take a status check of new research techniques and methods, to report on and summarize "lessons learned" from the military and industrial case studies, and to outline needed/promising directions for technology transfer and future research.

4.6 Facilities

Carnegie Mellon's School of Computer Science (SCS) is the largest academic organization devoted to the study of computers. Its six degree-granting departments – the Computer Science Department, Robotics Institute, Human-Computer Interaction Institute, Center for Automated Learning and Discovery, Language Technologies Institute, and Institute for Software Research, International – include over 200 faculty, 300 graduate students, and a 200-member professional technical staff. SCS also collaborates with other University Research Centers, including the DoD-funded Software Engineering Institute (SEI); the NSF-sponsored Pittsburgh Supercomputing Center (PSC), the Information Networking Institute, and the Institute for Complex Engineered Systems (ICES). Faculty members have private offices, while postdoctoral staff and graduate students share office space.

Heterogeneous Distributed Computing. The SCS research facility provides numerous and diverse computers for faculty and graduate-student use – more than 3000 machines. All have transparent access to the Andrew File System, a 625Gbyte, shared file space, and to one another through the Network File System protocol. SCS maintains several terabytes of secondary storage. Beyond

these resources, the University provides various independent facilities for general use. Computationally intensive applications can also use PSC computers, including Cray T3E, C90-16/512, and J90 supercomputers.

Networking. Carnegie Mellon operates a fully-interconnected, multimedia, multiprotocol campus network. The system incorporates state-of-the-art commercial technology and spans over 100 segments in a "collapsed backbone" infrastructure that enables mutual access among all campus systems, including the PSC supercomputers. To extend the network, the Information Networking Institute, with NSF funding, is developing a wireless infrastructure. Now supporting mobile computing throughout campus, the system will eventually deliver to users' handheld or laptop computers all the functionality of their desktop machines, on campus or off. Externally, SCS connects directly to the Internet, through T3 (45Mbit/s) and 10Mbit/s links, the NSF-sponsored vBNS (OC12) network, and the DARPA-sponsored CAIRN and ATM-based AAI (OC3) wide-area experimental networks. Carnegie Mellon is also actively engaged in the Internet-2 and NGI initiatives.

Experimental Systems. SCS has a reputation for developing innovative computers, devices, networks, and systems that benefit diverse applications. Current large-scale, experimental efforts include the Darwin "application-aware" networking project and the NASD project on storage interfaces with direct device/client communication. SCS's Robotics Institute is a renowned leader in developing embedded mobile robot systems that operate autonomously in natural environments. RI robots have logged thousands of miles of autonomous travel in office buildings, museums, the Antarctic, the Chilean desert, and all across America. Issues of reliability and safety are paramount in these endeavors, and receive much attention by the researchers and engineers of the Robotics Institute.

References

[1] E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall, 1998.

[2] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. 1993.

[3] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. on Software Engineering*, 22(3), Mar 1996.

[4] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *ICCAD '93*, Nov. 1993.

[5] H. Bansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, (6):512–535, 1994.

[6] S. V. Campos, E. M. Clarke, W. Marrero, and M. Minea. Verus: a tool for quantitative analysis of finite-state real-time systems. In *ACM Workshop on Languages Compilers and Tools for Real-Time Systems*, 1995.

[7] S. V. Campos, E. M. Clarke, W. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *Real-Time Systems Symposium*, 1994.

[8] A. Chutinan and B. H. Krogh. Approximate quotient transition systems for hybrid systems. In *2000 American Control Conference*, June 2000.

[9] A. Chutinan and B.H. Krogh. Computing polyhedral approximations to dynamic flow pipes. *The 37^{th} IEEE Conference on Decision and Control*, 1998.

[10] E. Clarke, M. Fujita, P. McGeer, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In *IWLS 93: Internl. Workshop on Logic Synthesis*, May 1993.

[11] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop*, LNCS, 1981.

[12] E.M. Clarke, M. Fujita, P.S. Rajan, T. Reps, S. Shankar, and T. Teitelbaum. Program slicing of hardware description languages. In *Proc. of Charme '99*. Springer-Verlag, 1999.

[13] Ken Currie and Austin Tate. O-Plan: The open planning architecture,. *Artificial Intelligence*, 52, 1991.

[14] J. Dingel, D. Garlan, S. Jha, and D. Notkin. Towards a formal treatment of implicit invocation. *Formal Aspects of Computing*, 1998.

[15] D. Engler, B. Chelf, A. Chou, and S. Hallern. Checking system rules using system-specific, programmer-written compiler extensions. In *Proceedings of Operating Systems Design and Implementation*, 2000.

[16] R. James Firby. An investigation into reactive planning in complex domains. In *Proc. National Conference on Artificial Intelligence*, Seattle, WA, July 1987.

[17] David Garlan and Serge Khersonsky. Model checking implicit invocation systems. In *Proc. of the 10th Internl. Workshop on Software Specification and Design*, San Diego, CA, Nov. 2000.

[18] Erann Gat. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Procs. of the AAAI Fall Symposium on Plan Execution*. AAAI Press, 1996.

[19] M. R. Greenstreet and I. Mitchell. Reachability analysis using polygonal projections. In *Hybrid Systems: Computation and Control, 2nd International Workshop, HSCC'99*, volume 1569 of *LNCS*. Springer, 1999.

[20] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *J. Comput. Sci.*, 57(1), 1998.

[21] T.A. Henzinger. Hybrid automata with finite bimulations. In *ICALP 95: Automata, Languages, and Programming*. Springer-Verlag, 1995.

[22] T.A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996.

[23] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Trans. on Automatic Control*, 43(4), April 1998.

[24] T.A. Henzinger, X. Nicolin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2), 1994.

[25] S. Jha and J. Wing. Survivability analysis of networked systems. Submitted to *International Conference on Software Engineering*, August 2000.

[26] S. Jha, J.M. Wing, R. Linger, and T. Longstaff. Analyzing survivability properties of specifications of networks. In *Proc. of the Internl. Conference on Dependable Systems and Networks, Workshop on Dependability Despite Malicious Fault*, June 2000.

[27] S. Kowalewski, S. Engell, and O. Stursberg. Verification of logic controllers for continuous plants. In *Advances in Control*. Springer, 1999.

[28] A. B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *Hybrid Systems: Computation and Control, 3rd International Workshop, HSCC'00*, volume 1790 of *LNCS*. Springer, 2000.

[29] J. S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In N. Lynch and B. H. Krogh, editors, *Hybrid Systems: Computation and Control, 3rd International Workshop, HSCC'00*, volume 1790 of *LNCS*. Springer, 2000.

[30] A. S. Morse, C. C. Pantelides, S. Sastry, and J. M. Schumacher (editors). A special issue on hybrid systems. *Automatica*, 35(3), March 1999.

[31] Nicola Muscettola. *HSTS: Integrating Planning and Scheduling*. Morgan Kaufmann, 1994.

[32] Charles Pecheur and Reid Simmons. From Livingstone to SMV: Formal verification for autonomous systems. In *Goddard Workshop on Formal Methods*, April 2000.

[33] H. A. Preisig. A mathematical approach to discrete-event dynamic modelling of hybrid systems. *Conmputers and Chemical Engineering*, 20(pt. B), 1996.

[34] J. Preußig, O. Stursberg, and S. Kowalewski. Reachability analysis of a class of switched continuous systems by integrating rectangular approximation and rectangular analysis. In *Hybrid Systems: Computation and Control, 2nd International Workshop, HSCC'99*, Berg en Dal, The Netherlands, 1999. Springer-Verlag.

[35] Elsevier Science. Special issue on hybrid systems. *Theoretical Computer Science*, 138(1), Feb 1995.

[36] I. Silva and B. H. Krogh. Formal verification of hybrid systems using CheckMate: A case study. In *2000 American Control Conference*, June 2000.

[37] Reid Simmons and David Apfelbaum. A task description language for robot control. In *Proc. International Conference on Intelligent Robots and Systems*, October 1998.

[38] Reid Simmons and Charles Pecheur. Automating model checking for autonomous systems. In *AAAI Spring Symposium on Real-Time Autonomous Systems*, Stanford, CA, March 2000.

[39] Kevin J. Sullivan and David Notkin. Reconciling environment integration and software evolution. In *Proceedings of SIGSOFT '90: Fourth Symposium on Software Development Environments*, Irvine, December 1990.

[40] B. Williams and P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings National Conference on Artificial Intelligence*. AAAI, August 1996.