Scanning the Issue

Special Issue on Modeling and Design of Embedded Software

I. Introduction

Perhaps the biggest impact of the "IT explosion" in the last decade has been the emerging role of computing and software as the "universal system integrator." Systems are formed by interacting components. The new trend is that an increasing number of components and interactions in real-life systems are *computational*. From electric shavers to airplanes and from cars to factory robots, computers monitor and control our physical environment. Distributed control and process automation systems integrate manufacturing production lines. Flight control and avionics systems keep airplanes flying. This trend is based on a fundamental technical reason: computing is uniquely suitable for implementing and controlling complex interactions among physical system components.

Information processing that is tightly integrated with physical processes is called *embedded computing*. The pervasiveness of this technology is well illustrated by the following facts: 1) the total shipment of microprocessor units (MPUs) and microcontrol units (MCUs) in 2000 was more than 8.2 billion units; of this, about 98% related to embedded applications [1]; 2) between 1994 and 2004, the need for embedded software developers is expected to increase tenfold [2]. The profound technical and economic implications of embedded computing and the well-documented difficulties of embedded software development present a significant challenge for the research community: we need to obtain a much deeper understanding of the nature of embedded software design and need to use this understanding for developing much improved design technology.

Is there any *essential* difference between embedded software development and software development in general? Although no one argues that embedded software is much harder to design, the source and effects of differences have not been investigated well in the past. The most frequently mentioned differentiators—such as hardware closeness, domain specificity, or real-time response—capture only some attributes (although interesting ones) of embedded software. Recently, there is an increasing recognition in the research community that existing software design techniques are not suitable for building large embedded software systems. The differences are fundamental requiring a full rethinking of basic principles. The U.S. Department of Defense, whose development programs are most exposed to flaws in embedded software concluded that "there may be some scientific problems which are intrinsic to all military systems that systems developers are not grasping" [3].

Broad-based discussion in the research community at a number of workshops led by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) (e.g., [4]) has led to the recognition of the following challenges embedded software and system presents for developers.

1) *Physicality of Embedded Software*: Embedded computers are surrounded by physical processes: they receive their inputs from sensors and send their outputs to actuators. Accordingly, embedded computing devices, viewed from their sensor and actuator interfaces, act like physical processes, with dynamics, noise, fault, size, power, and other physical characteristics. The role of the embedded software is to "configure" the computing device so as to meet physical requirements. This deep integration of computing with physical systems implies that essential physical characteristics of systems (such as latency, noise, power consumption) are strongly influenced—or simply determined by—software. Consequently, software requirements become multifaceted, i.e., computational platforms and software must satisfy logical (computational) and physical requirements *simultaneously*. Although the state of practice has clearly shown that many of the abstractions of mainstream software technology are either indifferent to or at odds with the requirements of embedded software, embedded software programming has never transitioned into an independent software development paradigm. In nonembedded software, where physical properties are secondary, functional composition is the focus of software technology. The best concepts of modern software component technologies such as objects, application program interfaces, connector mechanisms, all support functional composition. It is not surprising that using current software technology, *physical properties are not composable*; they appear as crosscutting constraints in the development process. The effects of these crosscutting constraints can be devastating for the design. Meeting specifications in one part of the system may destroy

Digital Object Identifier 10.1109/JPROC.2002.805816

performance in others; additionally, many of the problems will surface at system integration time. Consequently, we need to change our approach to the design of embedded software: productivity increases must come from modeling methods and tools that directly address the design of the whole system with its many different physical, functional, and logical aspects.

2) *Assurance of Embedded Software*: Embedded computing implements and controls physical interactions in systems. Physical interactions mean energy and material flows that have direct and immediate impact on the physical environment and the people involved. As a result, almost all embedded software is subject to high or extremely high assurance requirements. As it is frequently phrased: "In embedded software systems, *crash* is not just a metaphor." There are tremendous roadblocks toward achieving high-confidence design technology for embedded software. Modeling and verification techniques for physical and computational systems have developed along very different paths. Advancement in design technology for embedded software and systems requires full integration of these technologies, and in some cases, such as hybrid systems [5], even the development of new theoretical foundations. Fault management in embedded systems requires the propagation of effects of physical and logical faults across physical and information system boundaries. Our current software technology, which builds systems in layers of abstraction, completely loses traceability of the effects of physical faults on system behavior.

3) *Systems with Dynamic Structure: Networked Embedded Systems*: Embedded systems are increasingly becoming distributed, providing interaction among multiple, spatially distributed information and physical processes. Monitoring, control, and diagnostic functions penetrate deeper and with smaller granularity in physical component structures. The transition to networked embedded computing is being accelerated by inexpensive microelectromechanical system (MEMS)–based sensors and actuators, and by continued progress in microprocessor and communication technology. Given this trend, the strong separation between physical and information processing architectures no longer makes sense and is not sustainable. Building highly dependable, robust, distributed applications with hundreds or thousands of nodes is a significant software and systems challenge. As an example, we mention two extremely hard problems facing application developers in this area. First, applications must be highly reliable and robust, must operate in real time, and must be formed as the coordination of many activities. The coordinated operation of distributed embedded systems makes *coordination*, *distribution*, and *embedding* the fundamental technical challenge for software. Second, many of these distributed applications are dynamic, i.e., they continuously change their shape and interaction patterns as the environment is changing. Design and execution frameworks, which constrain design decisions to bound the behavior of these systems and the related development paradigms, constitute a major challenge that must be addressed in the future.

These new challenges and their tremendous practical significance inspired a profound shift in the interest of the scientific community, the funding agencies, and the industry toward embedded software and system development. Embedded software development is one of the grand challenges in computer science today.

The objective of this Special Issue of PROCEEDINGS OF THE IEEE is to provide a source of reference for the ongoing and future research by collecting the new intellectual directions and by exposing the present and expected challenges embedded software poses. The issue includes 13 contributions from outstanding researchers working on fundamental aspects of embedded software and systems. Unfortunately, space limitations prevented us from including many more contributions representing excellent, important research directions in this emerging field. Still, we believe that the selection of papers will give the readers perspective on the surprising richness and the breadth and depth of the research, which need to be addressed to achieve progress.

II. Paper Descriptions

The issue is divided into four sections. The first three papers deal with the *mathematical foundations of embedded software*: the mathematical models for describing and analyzing tightly coupled physical and information systems. The second section provides an overview of the *synchronous design framework*, which has already resulted in a wide range of practical applications. This section includes four papers covering synchronous languages, analysis methods for synchronous design, and the time-triggered architecture (TTA) providing hardware architecture and communication protocols for synchronous design. The central theme of the third section of the issue is *approaches to manage heterogeneity* in embedded software and system design. Because embedded systems are both physical and computational, heterogeneity is an intrinsic characteristic of embedded software design. The four papers in this section provide examples for managing design heterogeneity in models of computations, modeling languages, code optimization, and scheduling. Two papers on *embedded software design methodologies* complete the issue by showing practical examples for end-to-end design processes.

A. Mathematical Foundations for Embedded Software and Systems

Embedded systems have two distinct types of components: computational and physical. In the early 1990s, a realization began to set in that, on the one hand, systems modeling techniques from classical engineering (e.g., electrical systems) are inadequate for capturing the computational aspects of systems implemented increasingly in software, and, on the other hand, the computational models from classical computer science are inadequate for capturing the physical aspects of software that interacts with physical processes. This led to the foundation of the field of *hybrid systems* with the goal that both bodies of knowledge need to be combined for the design and analysis of embedded software. There are two ways in which such a combination may be achieved [9]. In a *shallow* combination, hybrid systems are described in a language that results from connecting expressions describing physical processes (such as difference and differential equations) with expressions describing computational processes (such as state machines or pseudocode). Although a shallow combination enables the description (requirements specification, architectural and behavioral description) of mixed physical-computational systems, it does not, *per se*, support the design and analysis of such systems. For this, a *deep* combination of the two worlds is being developed. Deep composition requires that the properties of a composite system can be derived solely from the properties of the component systems and the type of the connection. In the case of hybrid systems, this must apply to both computational properties—functionality, efficiency, accuracy—and physical properties—stability, timing, resource usage.[1]

The emerging theory of hybrid systems provides the new semantic foundation for embedded software design. Although it is not the purpose of this special issue to provide a review of progress in hybrid system theory (interested readers are referred to the PROCEEDINGS OF THE IEEE Special Issue on Hybrid Systems [5]), we do intend to show the significance of sound semantics in modeling languages and analysis and verification tools for embedded software and systems. The first paper in this section, "Hierarchical Modeling and Analysis of Embedded Systems," by Alur *et al.*, describes the modeling language CHARON for modular design of interacting hybrid systems. The language allows specification of architectural as well as behavioral hierarchy, and discrete as well as continuous activities. The modular structure of the language is not merely syntactic, but is exploited by analysis tools, and is supported by a formal semantics with an accompanying compositional theory of refinement. The authors illustrate the benefits of CHARON in development of embedded software using a case study involving programming of coordinated strategies for soccer using Sony's legged AIBO robots.

The verification of embedded software is a particularly challenging task because the verification of interesting properties (such as safety properties) necessarily involves the integrated behavior of the hybrid system. The paper by Tiwari *et al.*, "Invisible Formal Methods for Embedded Control Systems," presents tools and techniques for performing formal analysis on hybrid models using a symbolic approach. The authors present a wide range of formal technologies for manipulating symbolic representations of state of hybrid control systems, including theorem provers and decision procedures. An interesting aspect of the proposed methods is the high degree of automation, which enables hiding formal analysis behind widely used formal notations.

Recent advances in MEMS technology, wired and wireless networking, and ultralow-power processor technology have created a new wave of embedded systems characterized by the fine-grain integration of physical and information processes. The resulting massively distributed networked embedded systems lead to revolutionary new technologies, such as smart structures [6] and active surfaces [7]. These systems require fundamentally new approaches in modeling, model representation, and analysis. In the final paper in this section, "Physics-Based Encapsulation in Embedded Software for Distributed Sensing and Control Applications," Zhao *et al.* present a spatial aggregation language (SAL), which abstracts sensing and control properties of a distributed MEMS system as a set of interacting objects. Using a mechanism of influence graph, SAL enables software for distributed systems to capture and exploit locality and continuity constraints from distributed applications directly interfacing with physical processes. The paper describes the methodology and a design environment for prototyping software for distributed embedded MEMS systems, with an application to decentralized optimization.

B. The Synchronous Design Framework

The interaction of computation with physical processes makes the model of time one of the fundamental issues in embedded computing. The synchronous design frameworks adopt a model of time, which is similar to that of used in synchronous digital circuits: there is a global clock, and at each clock tick all components of the system reads inputs from the environment and computes the new state and the outputs before the next clock tick arrives. The synchronous programming model has been used in a family of synchronous languages, which were discussed in detail in a special section of PROCEEDINGS OF THE IEEE in 1991 [8]. By today, the synchronous design framework has become one of the most trusted, well-understood, and safe technologies for constructing safety-critical real-time systems. Building on this success, many exciting new developments have emerged during the past decade. This section presents groundbreaking work, which shows current and future directions in synchronous design.

The paper by Benveniste *et al.* "The Synchronous Languages 12 Years Later," provides an overview of synchronous languages. The paper starts with the fundamentals of the synchronous approach: synchrony and concurrency. They then show how the fundamentals have been incorporated in the synchronous languages Lustre, Esterel, and Signal. The paper includes examples for new research in modeling, code generation, verification, and test generation in the synchronous languages approaches. Important conclusion of past experience is that simplicity of the basic model enabled the use of the technology in challenging application. The authors also discuss the frontiers of current research on synchrony: architecture modeling, deployment on asynchronous architectures, and building systems from components.

The next paper, "Giotto: A Time-Triggered Language for Embedded Programming," by Henzinger *et al.*, discusses a synchronous language, Giotto, which provides an abstract programmer's model for the implementation of embedded control systems with hard real-time constraints. A typical hybrid control application consists of periodic software tasks together with a mode-switching logic for enabling

[1]ITR proposal

and disabling tasks. Pure Giotto specifies time-triggered sensor readings, task invocations, and mode switches independent of any implementation platform. Pure Giotto can be annotated with platform constraints such as task-to-host mappings, and task and communication schedules. The annotations are directives for the Giotto compiler, but they do not alter the functionality and timing of a Giotto program. By separating the platform-independent from the platform-dependent concerns, Giotto enables a great deal of flexibility in choosing control platforms as well as a great deal of automation in the validation and synthesis of control software.

The paper by Sifakis and Yovine, "Building Models of Real-Time Systems from Application Software," develops a method for building timed models of real-time systems by incrementally restricting their application software. The applied restrictions take into account execution times of atomic statements, the behavior of the system's external environment, and scheduling policies. The timed models of the application obtained in this manner can be analyzed by using timed analysis techniques to check relevant real-time properties. An advantage of this approach is the use of a unified modeling framework relating functional to nonfunctional aspects of the system's behavior. Furthermore, the framework encompasses general scheduling and schedulability analysis problems usually tackled separately from behavior modeling by using domain-specific theory. The authors show an instance of the method to modeling real-time systems programmed in the Esterel language. This language has been extended to describe time constraints imposed by the execution and the external environments by using pragmas. An analyzable timed model of the application is produced by composing instrumented C-code generated by the compiler.

The paper "The Time-Triggered Architecture," by Kopetz and Bauer, presents the TTA, which is a framework for the design and implementation of well-structured dependable distributed embedded systems. The TTA provides the following services to the application designer: predictable communication with small latency and minimal jitter, fault-tolerant clock synchronization, consistent membership service, and transparent handling of redundancy. Central to the TTA is a two-phased design methodology for the development of TTA applications: the architecture design phase and the component implementation phase. During the architecture design phase, the interactions among the distributed components and the interfaces among the components are fully specified in the value domain and in the temporal domain. During the component implementation phase, the components are designed, taking these interface specifications as constraints. This two-phased design methodology is a prerequisite for the composability of the TTA and the reuse of components. The paper presents the design principles of the TTA, and explains the two communication protocols that are at the core of the TTA, the time-triggered fault-tolerant protocol TTP/C, and the time-triggered sensor bus protocol TTP/A. The paper introduces the design process for developing TTA applications, supported by an appropriate tool set.

C. Approaches to Manage Heterogeneity

While hybrid systems theory provides a semantic, mathematical foundation for the integrated modeling of physical and information systems, model-based design focuses on the formal representation, composition, and manipulation of models during the design process. It addresses system specification, model transformation, synthesis of implementations, model analysis and validation, execution, and design evolution. The primary challenge (as well as the ultimate justification) for model-based design is the intrinsic heterogeneity of embedded systems. The physical and computational aspects of embedded systems need to be modeled simultaneously, and the models need to be deep enough to expose their interdependences. The semantic frameworks in which these models are applied may be domain specific, offering embedded system designers methods and syntaxes that are closer to their application domain. For example, domain-specific semantic frameworks for embedded systems might represent physical processes using ordinary differential equations, signal processing using dataflow models, decision logic using finite-state machines, and resource management using synchronous models. Heterogeneity of applications brings about the need for domain-specific languages (DSLs) and modeling tools for system specification, makes model transformation a central component of design environments, and extends the traditional embedded software tool chains with model synthesis, model analysis and validation, and model-based code generation components. Since physical characteristics of computations such as memory size, speed, and power are primary factors in embedded systems; software needs to be optimized to improve these characteristics. The contributions in this section present important research directions in addressing heterogeneity in composition, modeling languages and generators, code optimization, and scheduling.

The first paper of this section, by Eker *et al.*, "Taming Heterogeneity—The Ptolemy Approach," addresses the composition of embedded systems from subsystems with very different characteristics, that communicate and interact in a variety of ways—synchronous or asynchronous, buffered or unbuffered, etc. When designing such systems, a modeling language needs to reflect this heterogeneity. Today's modeling environments usually offer a variant of what we call amorphous heterogeneity to address this problem. This paper argues that modeling systems in this manner leads to unexpected and hard-to-analyze interactions between the communication mechanisms and proposes a more structured approach to heterogeneity, called hierarchical heterogeneity, to solve this problem. It proposes a model structure and semantic framework that support this form of heterogeneity, and discusses the issues arising from heterogeneous component interaction and the desire for component reuse. It introduces the notion of domain polymorphism as a way to address these issues.

DSLs have significant impact on the design process. In embedded systems, where computation and communication interact with the physical world, DSLs offer an effective way to structure information about the system to be designed along the "natural dimensions" of the application. The paper "Model-Integrated Development of Embedded Software," by Karsai *et al.*, describes a metaprogrammable tool infrastructure that enables rapid composition of domain-specific modeling languages on different levels of abstraction, representing heterogeneous models of computations, semantically solid pattern languages, and others. The paper describes a generative approach for embedded software development that is based on domain-specific, multiple-view models. Models explicitly represent the embedded software and the environment it operates in, and capture the requirements and the design of the application, simultaneously. Models are *descriptive* in the sense that they allow the formal analysis, verification, and validation of the embedded system at design time. Models are also *generative* in the sense that they carry enough information for automatically generating embedded systems from them using the techniques of program generators. To decrease the cost of defining and extending domain-specific modeling languages and corresponding analysis and synthesis tools, the model-integrated approach is applied in a "metacircular" architecture in the form of metamodeling, meta-analysis, and metasynthesis.

In a broad category of embedded computing applications, the memory system is a key factor in performance, power, and manufacturing cost. The paper "Memory System Optimization of Embedded Software," by Wolf and Kandemir, provides an in-depth survey of memory system analysis and optimization techniques for embedded software. The prevailing view in embedded software design is that the behavior of caches is too unpredictable for embedded systems. The authors show that advancement in trace-based analysis techniques and worst case analysis enables the safe use of cache-based architectures. They also argue that new directions in optimization techniques such as integration of internest loop optimization with layout transformation, software-managed on-chip memories, and hardware-based locality optimization techniques will have strong impact on future generations of embedded software.

Large-scale, dynamic, distributed applications represent very different challenges. Although the complexity of the behavior of the individual nodes is limited, the global state and behavior of the overall system composed of a large number of interacting nodes can be extremely complex. The problem in these systems is not the design of a specific global behavior, which may not even be monitored or known with perfect accuracy, but bounding the behavior in "safe regions" of the overall behavior space. This goal can be achieved by *execution frameworks* that introduce constraints in the behavior of and interaction among the distributed components. Since the development of robust execution frameworks is extremely hard today and requires a long maturation process, a new application-independent software layer, the middleware, has emerged as a promising solution. The last paper of this section, "Multiparadigm Scheduling for Distributed Real-Time Embedded Computing," by Gill *et al.*, describes technologies created and demonstrated at Boeing, BBN Technologies, Washington University, and DARPA related to adaptive middleware that supports the quality-of-service requirements for next-generation avionics systems. This adaptive middleware, which is based on commercial standards and products, has been successfully applied to support: 1) the interaction of diverse temporal, physical, and architectural scales of systems; 2) end-to-end integration of measurement and control paths within and especially between multiple distributed resource management layers; and 3) physical constraints on choice of transport media and protocols (i.e., Link-16 and land-line connections) and the corresponding implications for the middleware and end-system, and application design. In addition to describing how the adaptive middleware was modeled, designed, and optimized to meet demanding avionics real-time mission needs, this paper also presents the key lessons learned and future research opportunities stemming from this work.

D. Embedded Software and System Design Methodologies

Advances in model-based design have significant impact on both the design process and the architecture of the designed systems. In the design process, the shift toward high-level design languages and modeling tools naturally creates an opportunity for increased automation in verifying, producing, and integrating code. The increased use of generative programming bridges the gap between design-time models and implementation by the use of generator tools that can synthesize platform-specific code customized for specific middleware and application properties. The existence of design-time models offers significant opportunities in building *model-based systems*. New generations of intelligent, robust embedded systems are being developed, which include design-time models in their operation. This section presents two papers representing breakthroughs in the design process and in system architecture.

Since embedded systems are simultaneously computational and physical, it is not surprising that the need for layered abstractions in system design has emerged very strongly in this area. Although layered design approaches are known in many engineering fields, a clean conceptualization and systematic description of the method in embedded systems is a recent development. The basic tenets of platform-based design by Horowitz *et al.* (from the point of view of our discussion) are the following: 1) the design proceeds in precisely defined layers of abstractions; 2) each layer of abstraction is defined by a platform representing a family of designs; and 3) a design is obtained by defining platform instances by composing platform components and by mapping one platform to the successive one. Their paper, "Platform-Based Embedded Software Design and System Integration for Autonomous Vehicles," presents the platform-based design process in the design of the flight control system for the automatic control of an unmanned helicopter.

The paper by Williams *et al.*, "Model-based Programming of Intelligent Embedded Systems and Robotic Space Explorers," describes a new generation of sensor-rich, massively distributed model-based systems developed for deep space explorers, ubiquitous computing environments, and sensor webs for monitoring the earth ecosystem. Programming these systems involves reasoning through complex system interactions along lengthy paths between the sensors, control processors, and control actuators. The resulting code lacks modularity, and is fraught with error. Model-based programming supports modularity by enabling engineers to program reactive systems by specifying high-level control strategies and by simply articulating and plugging together commonsense models of the hardware and software modules being controlled. To execute a control strategy, model-based interpreters reason about the models on the fly to coordinate between modules. The paper describes a model-based interpreter, recently demonstrated on the Deep Space One probe, that is able to respond to novel situations on the order of hundreds of milliseconds, by performing extensive deduction, diagnosis, and planning within its reactive control loop. The paper also describes how the model-based programming paradigm extends to the coordination of networks of robots that will perform a diverse set of tasks such as search and rescue, Mars exploration and the study of life around other stars.

E. The Future

Embedded computing is a transformational technology that rapidly changes the world around us. It revolutionizes product lines in established industries, creates industries, and deeply impacts the foundation and practice of engineering. Full understanding the profound difference in design technology between embedded systems and their pure physical and computational components remains the greatest scientific and technical challenge. Recognition of these differences and rapid adoption of new development paradigms is increasingly becoming a key differentiator in industrial competitiveness. We expect that the broad interdisciplinary research, which is sampled in this special issue, will ultimately lead to the emergence of a new scientific and engineering discipline.

ACKNOWLEDGMENT

The Guest Editors wish to thank the many reviewers who volunteered their time to provide feedback to the authors. They would also like to thank the authors for their important contributions to this special issue.

SHANKAR SASTRY
University of California, Berkeley
Department of Electrical Engineering
and Computer Science
Berkeley, CA 94720–1770

JANOS SZTIPANOVITS
Vanderbilt University
Institute for Software Integrated Systems
Nashville, TN 37235

RUZENA BAJCSY
National Science Foundation
Computer Information Science
and Engineering Directorate
Arlington, VA 22230

HELEN GILL
National Science Foundation
Hybrid and Embedded Systems
Arlington, VA 22230

REFERENCES

[1] D. Tennenhouse, "Proactive computing," *Commun. ACM*, vol. 43, no. 5, pp. 43–50, May 2000.

[2] R. H. Bourgonjon, "Embedded systems in consumer products," in *Lecture Notes in Computer Science, Lectures on Embedded Systems*. Heidelberg, Germany: Springer-Verlag, 1996, vol. 1494, pp. 395–403.

[3] H. Mark, Speech to the National Defense Industrial Association Science and Technology Conference, May 9–11, 2000.

[4] A. Porter and J. Sztipanovits. (2001) New visions for software design and productivity: research and applications. Interagency Working Group for Inform. Technol. Res. and Develop., Software Design and Productiv. Coordinating Group, Vanderbilt Univ., Nashville, TN. [Online]. Available: http//:www.isis.vanderbilt.edu/sdp

[5] P. J. Antsaklis, "Special issue on hybrid systems: Theory and applications," *Proc. IEEE*, vol. 88, pp. 879–887, July 2000.

[6] R. L. Clark, W. R. Saunders, and G. P. Gibbs, *Adaptive Structures*. New York: Wiley, 1998.

[7] W. B. Jackson, M. P. J. Fromherz, D. K. Biegelsen, J. Reich, and D. Goldberg, "Constrained optimization based control of real time large-scale systems: Airjet object movement system," in *Proc. 40th IEEE Conf. Decision and Control*, vol. 5, Orlando, FL, Dec. 2001, pp. 4717–4720.

[8] A. Benveniste and G. Berry, "Prolog to the special section on another look at real-time programming," *Proc. IEEE*, vol. 79, pp. 1268–1269, Sept. 1991.

[9] L. de Alfaro, T. A. Henzinger, and R. Jhala, "Compositional methods for probabilistic systems," *Lecture Notes in Computer Science, CONCUR 2001—Concurrency Theory*, vol. 2154, pp. 351–365, 2001.

S. Shankar Sastry (Fellow, IEEE) received the M.S. degree (*honoris causa*) from Harvard University, Cambridge, MA, in 1994, and the Ph.D. degree from the University of California, Berkeley, in 1981.

From 1980 to 1982, he was an Assistant Professor at Massachusetts Institute of Technology, Cambridge. In 2000, he was Director of the Information Technology Office at the Defense Advanced Research Projects Agency, Arlington, VA. He is currently the NEC Distinguished Professor of Electrical Engineering and Computer Sciences and Bioengineering and the Chairman of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. He has coauthored more than 250 technical papers and has authored, coauthored, or coedited several books, including his latest, *Nonlinear Systems: Analysis, Stability and Control* (New York: Springer-Verlag, 1999). Books on embedded software and structure from motion in computer vision are in progress. He served as Associate Editor for numerous publications, including *Journal of Mathematical Systems, Estimation and Control, IMA Journal of Control and Information, International Journal of Adaptive Control and Signal Processing*, and *Journal of Biomimetic Systems and Materials.* His research interests are embedded and autonomous software, computer vision, computation in novel substrates such as DNA, nonlinear and adaptive control, robotic telesurgery, control of hybrid systems, embedded systems, sensor networks, and biological motor control.

Dr. Sastry was elected into the National Academy of Engineering in 2001 “for pioneering contributions to the design of hybrid and embedded systems.” He also received the President of India Gold Medal in 1977, the IBM Faculty Development award for 1983–1985, the National Science Foundation Presidential Young Investigator Award in 1985, the Eckman Award of the American Automatic Control Council in 1990, the distinguished Alumnus Award of the Indian Institute of Technology in 1999, and the David Marr prize for the best paper at the International Conference in Computer Vision in 1999. He was a chaired Gordon McKay professor at Harvard University, Cambridge, MA, in 1994. He has served as Associate Editor for IEEE TRANSACTIONS ON AUTOMATIC CONTROL, IEEE CONTROL SYSTEMS MAGAZINE, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS.

Dr. Janos Sztipanovits (Fellow, IEEE) graduated from the Technical University of Budapest, Budapest, Hungary, in 1970. He received the Candidate of Technical Sciences degree from the Hungarian Academy of Sciences, Budapest, in 1980, and the Distinguished Doctor degree (Golden Ring of the Republic) in 1982.

From 1999 to 2001, he was Program Manager and Acting Deputy Director of the Information Technology Office, Defense Advanced Research Projects Agency, Arlington, VA, where he worked on the Autonomous Negotiating Teams, Model-Based Integration of Embedded Software, and Networked Embedded Software Technology programs. He is currently E. Bronson Ingram Distinguished Professor of Engineering in the Electrical Engineering and Computer Science Department, Vanderbilt University, Nashville, TN. He is founding director of the Institute for Software Integrated Systems. His research interests include model-integrated computing, structurally adaptive systems, and embedded software and systems. He has published more than 150 papers and is the coauthor of two books.

Dr. Ruzena Bajcsy received the M.S. and Ph.D. degrees in electrical engineering from Slovak Technical University, Bratislava, Slovakia, in 1957 and 1967, respectively. She received the Ph.D. degree in computer science in 1972 from Stanford University, Stanford, CA.

During the 1950s and 1960s, she was an Instructor and Assistant Professor in the Department of Mathematics and Department of Computer Science at Slovak Technical University. In the 1970s, 1980s, and 1990s, she taught and did research in the Department of Computer and Information Science, University of Pennsylvania, Philadelphia; after 13 years, she became Chair of the department. At the University of Pennsylvania, she was a Professor in both the Computer and Information Science Department and in the Mechanical Engineering and Applied Mechanics Department, and was a member of the Neuroscience Institute in the School of Medicine. She was also Director of the university's General Robotics and Active Sensory Perception Laboratory, which she founded in 1978. She served as adviser to more than 20 Ph.D. recipients. From 1998 to 2001, she was the Assistant Director for the Computer Information Science and Engineering Directorate (CISE) at the National Science Foundation, Arlington, VA, where she managed a budget of approximately $300 million annually. She was the sixth person to be named to this position since the directorate was created in 1986. In 2001, she was appointed as Professor and Director of CITRIS, University of California, Berkeley. CITRIS is a multicampus (involving the University of California, Berkeley; the University of California, Santa Cruz; the University of California, Davis; and the University of California, Merced), interdisciplinary endeavor that brings together scholars and practitioners from engineering, sciences, and humanities and social sciences in order to explore the potential of information technology at the societal scale and applications such as energy conservation, safety and security, transportation, environment monitoring, education, and health care.She has done seminal research in the areas of human-centered computer control, cognitive science, robotics, computerized radiological/medical image processing, and artificial vision. She is highly regarded not only for her significant research contributions but also for her leadership in the creation of a world-class robotics lab, recognized worldwide as a premiere research center. She is especially known for her wide-ranging, broad outlook on the field and cross disciplinary talent and leadership, successfully bridging such diverse areas as robotics and artificial intelligence, engineering, and cognitive science.

Dr. Bajcsy is a member of the National Academy of Engineering as well as the Institute of Medicine.

Helen Gill received the B.A. degree in mathematics from the University of Missouri, Columbia (General Honors) in 1964, the M.S. degree in computer science from the University of Colorado, Boulder, in 1981, and the Ph.D. degree in computer science from Auburn University, Auburn, AL, in 1997.

From 1985 to 2000, she was a Principal Scientist with the MITRE Corporation, McLean, VA, and from 1993 to 1996, directed programs in software engineering and programming languages at the National Science Foundation (NSF), Arlington, VA. From 1997 to 2000, she was a Program Manager in the Information Technology Office of the Defense Advanced Research Projects Agency, Arlington, VA, where she developed programs in software-enabled control and hybrid systems and in programming technology for embedded systems, and she managed research in modeling and formal methods for software development and evolution. She is currently Program Director for the Computer-Communication Research Division of the NSF, where she directs a new area of research in hybrid and embedded systems. She cochairs the coordinating group for high-confidence software and systems under the auspices of the Interagency Working Group on Information Technology Research and Development of the National Science and Technology Council. Her research publications are in graph decomposition for concurrency analysis, partitioning and scheduling software for parallel execution, distributed programming environments, and discrete event simulation. Her current research interests are in embedded systems, middleware, hybrid discrete and continuous systems, software analysis, applied formal methods, and technology for high-confidence software and systems.

Dr. Gill's academic honors include Pi Mu Epsilon; Phi Kappa Phi; MU Curators, Powell B. McHaney, Burroughs, and CU Regents Fellowships. She is a member of Phi Beta Kappa.