Introducing Embedded Software and Systems Education and Advanced Learning Technology in an Engineering Curriculum

JANOS SZTIPANOVITS, GAUTAM BISWAS, KEN FRAMPTON, ANIRUDDHA GOKHALE, LARRY HOWARD, GABOR KARSAI, T. JOHN KOO, XENOFON KOUTSOUKOS, and DOUGLAS C. SCHMIDT

Institute for Software Integrated Systems (ISIS), Vanderbilt University

Embedded software and systems are at the intersection of electrical engineering, computer engineering, and computer science, with, increasing importance, in mechanical engineering. Despite the clear need for knowledge of systems modeling and analysis (covered in electrical and other engineering disciplines) and analysis of computational processes (covered in computer science), few academic programs have integrated the two disciplines into a cohesive program of study. This paper describes the efforts conducted at Vanderbilt University to establish a curriculum that addresses the needs of embedded software and systems. Given the compartmentalized nature of traditional engineering schools, where each discipline has an independent program of study, we have had to devise innovative ways to bring together the two disciplines. The paper also describes our current efforts in using learning technology to construct, manage, and deliver sophisticated computer-aided learning modules that can supplement the traditional course structure in the individual disciplines through out-of-class and in-class use.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: Embedded Systems

General Terms: Embedded Systems Education

Additional Key Words and Phrases: Computer-aided learning

1. INTRODUCTION

This paper describes our efforts in building an embedded software and systems specialization in the framework of the current Engineering School curriculum at Vanderbilt University. Vanderbilt's School of Engineering is typical for mid-sized engineering programs (i.e., about 100 faculty, 1200 undergraduate, and 400 graduate students), where new areas of specialization cannot be created by simply adding new courses due to limited faculty resources. We have, therefore,

Authors' address: Institute for Software Integrated Systems (ISIS), Vanderbilt University, Nashville, TN 37221.

adopted a more pragmatic approach, where we adjust the content of existing courses and add a minimal number of new courses as technical electives. This structure enabled us to form a new embedded software and systems concentration within existing majors in electrical engineering, computer engineering, computer science, and mechanical engineering.

The approach we followed was strongly influenced by the following four factors:

Insertion of the embedded software and systems concentration in the engineering curriculum at Vanderbilt has benefited by the research programs in Model-Integrated Computing (MIC), Distributed and Real-Time Embedded (DRE) Systems, and Systems Science and Engineering at the Institute for Software Integrated Systems (ISIS) [http://www.isis.vanderbilt.edu]. MIC is a technology that supports model-based development of embedded systems [Karsai et al. 2003; Sztipanovits and Karsai 1997]. The MIC approach advocates the use of models in the design, analysis, and synthesis of embedded systems, thereby bringing design automation to software systems that operate in a physical environment. DRE Systems cover a wide range of research on software platforms for DRE systems, including middleware for distributed and networked embedded systems, component-based architectures and quality-of-service (QoS) architectures [Schmidt et al. 1998; Wang et al. 2003; Gokhale et al. 2005]. Our research on Systems Science focuses on Hybrid Systems [Antsaklis 2000; Koutsoukos and Antsaklis 2003] as mathematical models for embedded systems. These three research thrusts are integrated in different applications including diagnostic systems [Mosterman and Biswas 1999; Narasimhan and Biswas 2002], networked embedded systems [Maroti et al. 2004] and integrated tool chains for avionics, automotive, and signal-processing applications [Karsai et al. 2002, 2004].

Many of our projects in embedded software and systems are cooperative efforts. Among them, our collaboration with the CHESS Center at Berkeley in the NSF Information Technology Research (ITR) project on the Foundations for Hybrid and Embedded Systems has a strong educational thrust. In this project we investigate approaches to teaching new systems science foundations, which is both physical and computational and new tool-based design technologies, such as platform-based design [Sangiovanni-Vincentelli and Martin 2001]. We also take advantage of our participation in the US-EU cooperative program between with the ARTIST-2 Network of Excellence and the DECOS project in EU.

At Vanderbilt, we are actively developing a learning technology that educators can use to create interactive web-based learning experiences that can adapt to individual differences between learners by altering content or flow. Our interests in adaptive learning technologies arose from participation in the NSF-sponsored Vanderbilt-Northwestern-Texas-Harvard/MIT (VaNTH) ERC for Bioengineering Educational Technologies [Cordray et al. 2003]. The outcome of this research is the Courseware Authoring and Packaging Environment (CAPE) and the Learning Management System (eLMS). CAPE is used to design *how* learning materials are used to create an adaptive learning experience and eLMS is a web-based delivery platform.

ISIS has a strong industrial partnership program, which enables us to obtain practical experience in the construction of complex embedded system applications. In cooperation with the CHESS Center, ISIS has incubated and spawned off the ESCHER Research Institute [www.escherinstitute.org], which is an independent nonprofit organization that operates a quality-controlled open-source repository of embedded systems tools and platforms. ESCHER's industry-funded maturation program for open-source research tools helps create reference tool chains and applications that can be used in industry and at academic institutions as an infrastructure for teaching and research.

As discussed earlier, we have built our embedded software and systems curriculum by combining traditional courses in the Electrical Engineering (EE), Mechanical Engineering (ME), Computer Engineering (ME), and Computer Science (CS) disciplines, such as Signals and Systems, Circuit Analysis, Dynamic Systems, Control Theory, Computer Organization, Microprocessors and Microcontrollers, Data Structures and Algorithms, Automata and Theory of Computation, and Software Design with several new courses that have been introduced at the undergraduate and graduate levels.

The remainder of this paper is organized as follows. Section 2 describes our new undergraduate embedded software and systems course offerings that supplement the existing relevant courses that we have listed above; Section 3 describes our graduate course offerings on embedded software and systems; Section 4 describes our adaptive learning technologies and examines potential applications; and Section 5 presents the concluding remarks and discusses future work toward further enhancements to our curriculum.

2. TEACHING EMBEDDED SOFTWARE AND SYSTEMS TO UNDERGRADUATES

The current curriculum at Vanderbilt is characteristic of many engineering schools. Relevant courses and capable students are spread over various departments. The Departments of EECS and ME are the two largest departments both in terms of students, and faculty who share common interests in embedded systems. The traditional courses in the EE, CompE, CS, and ME programs have the typical disciplinary focus, but this did not align well with the needs of a strong embedded systems curriculum. When designing an acceptable solution, therefore, we had to consider the following constraints:

- Launching a completely new line of courses focused on embedded software and systems was not feasible due to resource constraints in terms of faculty time and the availability of laboratory facilities.
- Embedded software and systems require an expanded foundation that combines the fundamentals of physical and computational systems theories. Extending the set of existing disciplinary foundational courses to accommodate embedded systems fundamentals is not possible given the need to satisfy ABET accreditation requirements and limits on the number of courses that can be crammed into a 4-year curriculum.

- The interested student body is enrolled in several majors, including EE, CS, CompE, and ME, so the curriculum must cross traditional disciplinary boundaries.

It was clear that we had to adopt a nonconventional solution since embedded software and systems represent fundamentally different challenges that cannot be integrated effectively into existing, well-established (and entrenched) disciplinary frameworks. This section introduces the conceptual issues we had to address in teaching embedded systems and then describes the undergraduate course offerings, laboratory program, and summer research internships that we created to give our undergraduates adequate exposure to the discipline.

2.1 New Conceptual Elements in the Undergraduate Curriculum

Engineering disciplines require the understanding of modeling formalisms, analysis methods, and design/synthesis techniques. Teaching embedded software and systems creates substantial new challenges in all three areas. Moreover, it requires two types of thinking:

- Abstractions are a key concept in the study of computer science and all engineering disciplines. In computer science, students learn to create and formalize abstractions of computational processes and to relate layers of abstraction to each other. These abstractions involve logical and conceptual formalisms, but they do not cover the theory and analysis of physical processes and systems. In fact, "physicality" brings problems (such as "cross cutting") to clean abstraction hierarchies, which makes conceptualization much harder.
- In most engineering disciplines besides computer science, a typical pedagogical approach is to teach a particular abstraction (such as discrete time linear systems) and then build analysis and design methods that exploit the abstraction formalism. In these engineering disciplines, however, there is a tacit understanding that physical reality is more complex than the "clean abstractions" used for teaching, which are always approximations of the real world.

A successful embedded software and systems curriculum must encourage both types of thinking, i.e., students need to be taught to use heterogeneous abstractions and even to design new systems of abstractions by combining and integrating existing ones.

Analysis of engineering models requires precisely defined semantics for the abstractions used in systems modeling. The level of precision and formality differs not only between computer science and other engineering disciplines, but among engineering disciplines. as well. A unique challenge of embedded software and system curricula is to include the appropriate level of formal semantic foundations that satisfy the needs of systems integrating physical and computational components. Teaching hybrid systems modeling and analysis on the level suitable for undergraduates is a particularly hard problem because students must conceptually integrate continuous-time models with discrete behaviors, such that one cannot be understood without understanding the other. Moreover, analysis methods for real-valued, differential-equation based

models differ significantly from automata-based and algebraic models that form the core of analysis methods for computational processes. Analysis of embedded systems therefore requires a new way of thinking that in some way must systematically combine disparate continuous and discrete mathematical domains.

Design methods are highly discipline dependent. Logic circuit design in computer engineering and linear circuit design in electrical engineering are well understood and highly developed areas, with standard formalisms, semantics, and methodology. It is very common in embedded software and systems design that design flows need to be adjusted according to the needs and requirements of specific domains. Students need to examine specific design flows as examples and understand how to change those if requirements are changing.

2.2 Undergraduate Courses

Based on the considerations described in Section 2.1, we have established an embedded software and systems concentration for EE, CS, CompE, and ME undergraduate students. This concentration includes courses that reflect the unique challenges of embedded software and system engineering and brings together students from different majors. Below we present the selection of our courses that form the backbone of the embedded systems curriculum for undergraduates. The courses are 3 credit hour courses following the scheduling at Vanderbilt (i.e., 3 hours of lectures per week, 14 weeks per semester).

Microcontrollers. This course is oriented toward the engineering principles and techniques of applying microcontrollers in various embedded systems. The course covers topics such as principles and techniques for programming real-time embedded systems, as well as various systems engineering topics such as fault tolerance and system integration practices. The course introduces design techniques, covering both procedural and object-oriented design, as well as the analysis techniques such as Rate Monotonic Analysis. The course places strong emphasis on hands-on experience with the topic, including eight structured labs (3 hours each) and has a significant design project. A typical design project involves designing and developing a controller for a small, mobile robot, and implementing it using a real-time operating system. Model-based techniques are introduced on the elementary level: students learn how to model behaviors using Stateflow and use these techniques in their projects. This course also includes a significant lab component (of 3 hours per week lab-time).

Modeling and Simulation. The primary theme for this course is to introduce students to different modeling formalisms that are relevant to embedded systems, and use simulation tools for analyzing the behavior of these systems. This course has evolved from an older CS course on system simulation that focused on the study of numerical techniques to ensure precision and accuracy in complex real-valued computations using finite arithmetic on computer systems. The current version of the course has been revamped significantly to understand the basics of modeling and simulation of embedded systems. This vision is achieved by introducing students to modeling processes and simulation tools that span computational systems (discrete and discrete-event) and physical

processes (discrete-time and continuous). Rather than focus on the mathematical theories (e.g., theory of automata, discrete-event systems, difference, and differential equations) associated with analyzing such systems, this course focuses on realistic systems, defines a task for analyzing aspects of system behavior, and then develops computational models at the appropriate level of abstraction so that behavior analysis is performed by simulation and interpretation of the observed behaviors. Example systems, such as computing processes, manufacturing systems, physical devices, and biological systems, provide students insight into the theories of different computational paradigms.

Rather than take a traditional computer science approach to developing and coding simulation algorithms manually using third-generation programming languages, students use the Simulink and Stateflow components of Matlab to build and execute models. The differences in the models of computation are explained through the abstraction mechanisms that are employed in model building and the computational form of the models that students have to develop for the Simulink and Stateflow environments. Toward the end of the course, students are introduced to modeling of embedded systems using hybrid automata models. Again, Simulink models introduced into Stateflow blocks provide the modeling and simulation environment for students to build and analyze embedded systems models.

The specific topics covered in our undergraduate modeling and simulation course include introduction to systems modeling concepts, modeling formalisms and their simulators, cellular automata, discrete-event systems modeling, continuous system modeling, a bond graph approach to modeling physical systems, stochastic systems, and hybrid (continuous + discrete) systems. Over the period of the semester, students work on a number of Matlab-based assignments. Students also work on a class project in pairs, and apply the modeling techniques learned to their particular domain of interest. In the last several years students have worked on a variety of domains, such as modeling advanced life support systems for NASA manned missions, supply chain systems, complex electromechanical devices, UAV path planning systems, stock market analysis, traffic flow systems, missile defense systems, and shipboard facility layout systems.

System Dynamics and System Dynamics Laboratory. This two-part course has been in the ME curriculum for many years. The focus of the course is the integration of the physical sciences of dynamics, mechanics, fluids, heat transfer, and circuits, with the mathematical tools used in conceptualizing, modeling, and analyzing system response. Moreover, the associated laboratory (described below) reinforces these concepts. The overall goals of the course focus on analysis of the physical processes in embedded systems, which required almost no modification to fit within the embedded software and systems program at Vanderbilt. Some introductory material has been added to benefit EE, Comp E, and CS students and some elements have been added to better mesh with the Foundations of Hybrid and Embedded Systems course described in Section 3.

The topical progression of the System Dynamics course begins with a review of basic physical systems modeling techniques. This review is followed by the introduction of Laplace transforms and their use in system response (both

transient and steady-state) analysis. Next, concepts from hybrid and embedded systems are drawn into the course, including signal flow and block diagram system representations. Also included at this point is modeling of combined electromechanical systems, such as the acoustic loudspeaker that includes electrical circuit, electromechanical coupling, and mechanical response. Finally, the course covers introductory topics in feedback control for system response modification.

The laboratory associated with the System Dynamics course seeks to reinforce the in-class concepts, as well as expose students to the experimental techniques related to system analysis. Labs performed in this course include introduction/review of measurement instruments and circuit building, measurement of system transient response, measurement of system steady-state response, control system implementation, and programming and behavior of simple robotic systems. In each lab, students are first asked to develop computer simulation of the systems to be studied and then to compare the predicted results with the measured results.

Real-Time Systems. This is an introductory course on real-time systems design and implementation. Students learn basic design and implementation issues in real-time systems and theoretical formulations where appropriate. The course places heavy emphasis on design and implementation of real-time systems. Topics covered can be categorized into (1) concepts, which include hard and soft real-time systems, distributed real-time and embedded systems, global time, priorities, priority inversions, priority inheritance, event-triggered and time-triggered models of real-time, safety critical systems, resource management, scheduling, and schedulability analysis and (2) practical issues, which includes real-time programming primitives, real-time system software patterns, real-time operating systems, quality-of-service (QoS) issues in communication networks, and real-time middleware, such as real-time CORBA and real-time Java.

2.3 Laboratory Experience for Undergraduates

One of the most successful and appreciated courses for all majors in Vanderbilt's School of Engineering is the senior design course. This course focuses on a "capstone" design experience that integrates the material learned in prior years in the context of an industry-level design project. Students form teams, work in conjunction with an industrial advisor, and take a product through its full design and development cycle. This two-semester activity results in an industrial quality prototype, evaluated and endorsed by an industry mentor (typically from a local small business). Capstone design projects often involve traditional engineering disciplines (e.g., circuit design and mechanism construction), as well as computing tasks (e.g., designing a small microcontroller that implements a sensor and/or control function). These projects provide an excellent opportunity for students to gain hands-on experience with embedded computing in a realistic, physical context.

In addition to the capstone design project for seniors, we also provide selective and intense research internship opportunities to our and other institutions'

rising junior and senior students during the summer through the NSF ITR project "Foundations for Hybrid and Embedded Software Systems" summer program called: Summer Internship Program in Hybrid and Embedded Software Research (SIPHER) [http://fountain.isis.vanderbilt.edu/fountain/Teaching/]. One of our major challenges in the CS and EE disciplines is to attract underrepresented (minorities and women) talent and channel them into successful research careers. Whereas the original focus of SIPHER was on recruiting and training underrepresented students in hybrid and embedded systems, faculty at Vanderbilt have expanded this program and offer internships to a number of additional students. All of the students undergo a 10-week intense training program in the science and technology of embedded systems developed by ISIS and UC Berkeley researchers, and work on specific design research problems. They use software design tools developed as part of the NSF ITR research project, including the Generic Modeling Environment (GME) [Ledeczi et al. 2001; http://www.isis.vanderbilt.edu/Projects/gme/], Ptolemy [http://ptolemy.eecs.berkeley.edu/ptolemyII/] and Metropolis [http://www-cad.eecs.berkeley.edu/Respep/Research/asves/index/html], to gain an early exposure to model-based design and implementation techniques for embedded software systems. Their work in these projects is mentored by graduate research assistants and supervising Vanderbilt faculty.

The SIPHER program emphasizes the model-based approach and students have used it to implement nontrivial embedded systems. In most of the projects the student were using a Small Modeling Language for Embedded Systems (SMOLES) [Szemethy and Karsai 2004] that illustrates how model-based composition, behavioral specification, and integration works. The SMOLES language is supported by a GME-based visual modeling tool, it has a code generator, and a small run-time system that runs on the embedded platform. The run-time system implements a data flow model of computation and it is closely coupled to the concepts of the modeling language. The system is available in C++ and Java, the latter can be used on a Lego RCX platform. SMOLES have been developed and updated by ISIS graduate students as part of their training in designing Domain-Specific Modeling Languages (DSMLs) and MIC environments.

3. TEACHING EMBEDDED SOFTWARE AND SYSTEMS TO GRADUATES

Our graduate-level courses on embedded software and systems at Vanderbilt emphasize the formal foundations and analytical tools needed to develop embedded systems, as well as cover advanced application topics, such as engineering large-scale distributed real-time, and embedded systems, model verification, and quality-of-service (QoS) issues. This section describes the key courses that we include in our embedded software and systems graduate curriculum.

3.1 Model-Integrated Computing (MIC)

This course is the core, graduate-level course on the fundamentals of MIC. The course teaches the principles and techniques of MIC, with significant, emphasis on the design, formal specification, and implementation of DSMLs, model

transformations, model-based code generators, and building end-to-end applications. A core concept in the course is that different embedded systems categories require a unique set of abstractions that are determined by what the properties designers want to achieve. DSMLs are thus not only used in the design process, they are also the objectives of design. Students learn about modeling in various disciplines, as well as practice meta-modeling that provides the formal specification for DSMLs. Another core principle discussed in the course is platform-based design [Sangiovanni-Vincentelli and Martin 2001]. Using examples, students study design flows, understand the nature of mapping between models in a design flow, and learn the techniques for formally specifying model transformations [Karsai et al. 2003]. A third concept in the MIC course is model-based system integration. Students learn about modeling key design aspects of embedded systems, such as functional behavior, hardware architecture, fault behavior and others, and understand how to use these interdependent modeling aspects in generating various artifacts, such configuration tables, schedule tables, and code.

The MIC course also covers the theoretical foundations of DSML construction and the technology that enables building domain-specific modeling environments. This course heavily relies on the use of the open-source GME [Ledeczi et al. 2001] that was developed at ISIS over the last decade, and has evolved significantly through its use in a number of research projects [http://www.isis.vanderbilt.edu/projects.asp]. Students work on class projects that include the full definition of a DSML, and build software generator tools for creating artifacts (e.g., executable code or simulations) from the domain-specific models.

While MIC is applicable to domains beyond embedded systems, this domain is a key area in which the model-based paradigm has made a significant impact on software research and practice. Hence, many student projects in the MIC course are related to embedded systems. A typical student research project results in a visual modeling environment that provides a tool for building embedded system applications for a specific category (e.g., reactive controllers specified using StateChart-like notations), as well as in additional software tools for verification (using a model checker, such as SMV), or code generators.

3.2 Foundations of Hybrid and Embedded Systems

The design of embedded systems requires the use of heterogeneous models of computation to address specification, analysis, synthesis, and validation problems. The objective of this course is to introduce various formal models of computation and study their use in embedded system applications. At the graduate and advanced undergraduate levels, this class provides students with the foundations required for studying embedded systems, control systems, and robotics.

This class starts with an overview of hybrid and embedded systems, emphasizing the necessities for and the rewards of precise mathematical representations. The first part of the course presents several models of computation, including finite state machines and statecharts. Subsequent topics include timed discrete-event systems, discrete-event simulation, continuous

time-driven systems, and numerical integration methods. Finally, data flow process networks are presented, focusing on scheduling of synchronous data flow. Students are exposed to numerous applications that demonstrate the value of these models.

The second part of this class focuses on hybrid systems as heterogeneous models of computation. Hybrid systems are viewed as models that combine continuous and discrete processes, time- and event-driven dynamics, data flow and control flow, or differential equations and finite-state machines for describing the interaction between embedded software systems and the physical world. After introducing the hybrid automata model, the course studies hybrid system simulation using many practical examples, such as robot control systems, and flight control systems. Formal verification of hybrid systems is also investigated and solutions based on discrete abstractions are presented. Assignments include modeling, simulation, and analysis problems and class projects involve modeling a reasonably complex embedded system application using model-based tools, such as Matlab/Simulink/Stateflow and Ptolemy II.

3.3 QoS-enabled Component Middleware

This course focuses on QoS issues in large-scale DRE systems built using component technology. Much of the complexity and cost of developing and validating DRE systems can be alleviated by the application of the advanced modeling and QoS-enabled component middleware concepts, platforms, tools, and techniques covered in this course, which reside between the applications and the underlying hardware and network infrastructure and provide reusable services that can be generated, composed, configured, and deployed to create mission- and safety-critical DRE systems rapidly and robustly.

The theoretical foundations of this course cover the following advanced concepts for component-based DRE systems that possess deterministic and statistical QoS requirements: (1) global time synchronization services in distributed systems, (2) priority preservation, e.g., priority inheritance/ceiling protocols, (3) computational models for DRE system concurrency and distribution, e.g., event and time-triggered systems, (4) validation techniques for mission- and safety-critical DRE systems, (5) distributed and dynamic resource management strategies, and (6) scheduling strategies and schedulability analysis, e.g., maximum urgency first (MUF) and least laxity first (LLF).

The experimentation aspects of the QoS-enabled Component Middleware course cover the following advanced topics: (1) programming language, operating system, communication network, and QoS-enabled component middleware capabilities, (2) architectural and design patterns of DRE software, and (3) model-driven specification, analysis, and synthesis of DRE systems. Students use the open-source TAO [Schmidt et al. 1998] and CIAO [Wang et al. 2003] QoS-enabled component middleware developed at ISIS as the basis for class projects. TAO is a real-time CORBA object request broker that allows clients to invoke operations on distributed objects without concern for object location, programming language, OS platform, communication protocols and interconnects, and hardware. CIAO is a real-time CORBA Component Model (CCM) implementation built on top of TAO.

3.4 Model-Driven Middleware

Rapid advances in hardware, networking and software technology are fostering widespread deployment of complex distributed real-time and embedded (DRE) applications in several different domains, such as avionics, telecommunications, telemedicine, industrial process control, healthcare, military combat systems, and enterprise. A challenge requirement for these new and planned applications involves supporting a diverse set of quality of service (QoS) properties, such as predictable latency/jitter, throughput guarantees, scalability, 24×7 availability, dependability, and security that must be satisfied simultaneously in real-time. Although these applications are increasingly based on commercial off-the-shelf (COTS) hardware and software elements, the complexity of life-cycle management is a growing challenge for the application developers. For example, substantial time and effort are spent integrating these elements into DRE applications. Integration challenges stem largely from a lack of higher-level abstractions for composing complex DRE applications.

The Model-Driven Middleware (MDM) course deals with the challenges faced by DRE application developers and delves into the solutions used to address these challenges. In particular, it focuses on MDM, which is an emerging paradigm to address these challenges by integrating the strengths of model-based engineering of systems, such as Model Integrated Computing (MIC) and Model-Driven Architecture (MDA), and component-based middleware technologies, such as the real-time CORBA Component Model, J2EE, Microsoft .Net, and Web Services. Student projects in this course use the open-source CoSMIC [Gokhale et al. 2005] MDM toolsuite developed at ISIS. CoSMIC provides a collection of GME-based domain-specific modeling languages and their associated analysis/synthesis tools that support various phases of component-based DRE system development, assembly, configuration, and deployment.

3.5 Advanced Real-Time Systems

The course serves as a graduate-level course on real-time systems that must react to the dynamic environments under timing constraints. The course focuses on the analysis and design of real-time systems. The course covers topics on system modeling, using the tagged signal models and timed models of computation [Lee and Sangiovanni-Vincentelli 1998], specifications and scheduling techniques for real-time tasks [Butazzo 1997], simulation and verification of real-time systems, software architecture and language for constructing real-time systems [Kopetz 2003; Henzinger et al. 2003; Horowitz et al. 2003]. Special attention is paid to computational and simulation tools for real-time systems. Applications ranging from robotics, embedded control systems, drive-by-wire systems, space missions, telecommunication systems, industrial automation, logistics systems, and middleware software systems are covered.

3.6 Topics in Embedded Software and Systems

The objective of this course is to review and discuss new advancements in model-based design of embedded systems and software. The selection of topics reflects a MIC perspective, i.e., there is a strong emphasis on DSMLs and

embedded software and systems platforms, which can offer effective support for model-based system and software generation. The course uses a seminar format where students review and discuss papers assigned for each class by the instructor. The class preparation includes the following steps: (1) reading the papers assigned, (2) preparing answers to control questions and submitting them electronically before the class, and (3) preparing material for introducing the discussion (twice during the semester). There are three main topics discussed in the class based on the critical review of current literature: (1) formal specification and composition of DSMLs, (2) design approaches for embedded systems, and (3) new platforms for embedded systems.

4. TOOLS FOR TEACHING AND LEARNING IN A TOOL-RICH ENVIRONMENT

As embedded software and systems engineering becomes more heavily dependent on tools, the challenges grow for bridging theory and practice in curricula. Innovations in curricular designs, such as the senior design course described earlier, are a vital part of responding to these challenges. Another area of opportunity is extending the traditional learning environment through the use of new learning technologies to complement learning in the classroom and the lab. Learning resources made available through the web and directly embedded into tools offer *asynchronicity* (that is, they are available on demand) and the potential for *adaptivity* (that is, they can be designed to change in response to an individual learner). Technology can facilitate more active forms of learning, where its role is to provide "scaffolding" for learners in reflecting on their individual learning goals and outcomes and selecting appropriate learning resources and activities. Such active learning experiences contribute to developing meta-cognitive skills that are important to life-long learning in a tool-rich (and information-rich) environment, which will be predominantly demand-driven, self-directed, and informal. The creation of technology-based learning resources and scaffolding by educators is nontrivial, especially when they are adaptive, and it is essential that these tasks also be supported by technology.

In this section, we describe roles for adaptive learning technologies in facilitating broader and deeper roles for design tools in embedded software and systems education. We use our experiences developing and applying a technology infrastructure for adaptive on-line learning as a lens for examining the opportunities. This infrastructure includes a design technology called CAPE that is based on the MIC paradigm and implemented using the GME tools mentioned in Section 3. This means that educators can use a design tool based on technology used *for* embedded software and system design to create learning experiences *about* embedded software and system design. We begin by introducing the concept of adaptive learning technologies, continue by describing our particular learning technology infrastructure, and conclude by presenting and discussing a set of relevant application examples.

4.1 Adaptive Learning Technologies

A learning experience is *adaptive* if what is known about a particular learner, *a priori* or through embedded formative assessment, can alter the experience

itself. Technologies that support adaptive learning experiences have their roots in over 30 years of research on intelligent tutoring systems (ITS) [Lave and Wenger 1990]. These technologies, whose evolution continues to the present [see Graesser et al. 2004], seek to emulate human tutors instructing a single pupil on a particular knowledge domain. A canonical ITS employs explicit representations of learner knowledge and the knowledge of an expert in the domain. Questioning or interlocution is used to build and revise a model of a particular learner's knowledge and the system is concerned with incrementally aligning this knowledge with that of the expert through the engagement of particular learning content and activities. Extensions of ITS techniques to web-based learning technologies can be seen in work on adaptive hypermedia [Brusilovsky 1998]. Technologies for so-called "e-learning," widely employed on university campuses in the form of course management systems, such as Blackboard and WebCT, are also beginning to offer limited capabilities for adaptive learning experiences. This support currently offers relatively weak knowledge representation and reasoning capabilities, however, with a primary focus on instructor-directed conditional sequences.

Our interests in adaptive learning technologies arose from participation in the NSF-sponsored Vanderbilt-Northwestern-Texas-Harvard/MIT (VaNTH) ERC for Bioengineering Educational Technologies [Cordray et al. 2003]; [Howard 2003]. This multidisciplinary center involves learning scientists, experts in assessment and evaluation, learning technologists, and bioengineering educators (among others) who are collaboratively pursuing improvements to pedagogy and applications of learning technology in training future generations of bioengineers. In creating the Courseware Authoring and Packaging Environment (CAPE), we were motivated to exploit the vast middle-ground between ITS capabilities and those found with "e-learning" technologies, while at the same time making it easier for educators to create, evolve, and share designs for powerful web-based adaptive learning experiences. We created the experimental Learning Management System (eLMS) to support communities of educators in providing such experiences to learners and learning about how they are used. Figure 1 illustrates the connection between CAPE and eLMS. As the figure shows, CAPE is used by educators to construct learning products that are stored in a repository. In this construction process, the educators can rely on the work of instructional designers, learning scientists, media designers, and others who have produced instructional design patterns, courseware design elements, and other repository-based assets. eLMS provides (web-based) delivery and other services to make courseware available to learners, educators, and administrators. The details of CAPE and eLMS are discussed below.

4.2 CAPE: A Visual Authoring Language

CAPE learning designs focus on a number of interrelated issues, with the central concerns focusing on how the learning process unfolds, e.g., what learning materials are involved, how formative assessments are employed, and what roles adaptation will play. Associated with these kinds of concerns are statements of learning objectives and how these are associated with elements of

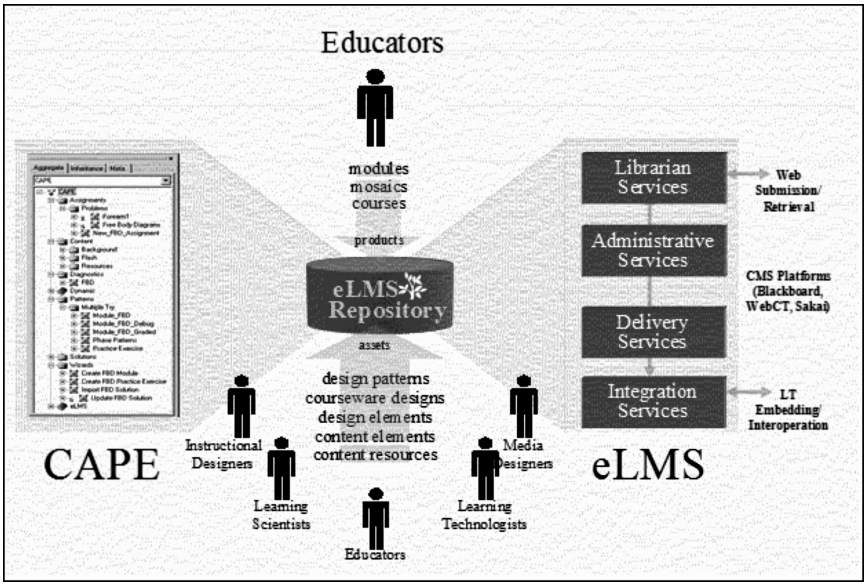

Fig. 1. CAPE and eLMS functional connections.

domain knowledge. Metadata, or descriptive data, are employed to describe the learning process to many potential stakeholders and to different agents along the path to delivery to learners.

CAPE is not a content-authoring environment: it is used to design *how* learning materials are used in the learning process. Web-deliverable learning materials (HTML pages, images, movies, etc.) are authored using conventional tools for these content types. Some of the resources employed in a learning design may be involved in adaptation schemes: *dynamic* resources. CAPE makes adaptation services available that enable the replacement of isolated placeholders in content such as HTML pages at delivery time or the dynamic generation of entire content elements. CAPE provides an integral assessment authoring capability, reflecting the heavy involvement of formative assessments in adaptation schemes and the needs for assessments themselves to sometimes be adaptive. CAPE supports the embedding of interactive content into a learning design and provides the ability to organize the interoperation of such content.

CAPE provides support for *instructional design patterns*. Such patterns are represented by CAPE as abstract models that can be reused by refinement. At a high level, these patterns can capture structural invariants of recurring (and preferred) pedagogical strategies. At lower levels, they can be used to represent idioms or techniques that a single author or group of authors find useful. CAPE allows authors to fashion *wizards*—stepwise automation agents—that can assist in instancing these patterns. CAPE wizards are built inside CAPE using the same language features that support adaptive delivery, including programming extensions in Python [van Rossum and Drake 2001], a dynamic object-oriented language. These capabilities are being extended to later design processes, including assignment-time and delivery-time adaptation.

Other forms of assistance provided to authors by CAPE include observation-based automation, context-specific help resources, and "just-in-time" learning facilities. The actions of CAPE authors are continually monitored by an event-based agent programmed to offer particular task-completing steps. Context-specific on-line help associates web-based reference materials with particular models, elements, and aspects. CAPE provides access to adaptive tutorial resources authored with CAPE and delivered via eLMS. These "just-in-time" learning capabilities will be discussed in Section 4.4.

4.3 eLMS: A Delivery Platform and Repository

eLMS is a repository-based web services platform that supports the delivery of learning experiences authored with CAPE to learners and the sharing of CAPE learning designs and their constituent materials among communities of authors. As a repository platform, eLMS features are strongly specialized to CAPE and enable authors to share design patterns, wizards, design elements, content elements, and content resources. As a delivery platform, eLMS provides administrative services to instructors to form classes, to establish rosters, and to make assignments to individuals, groups, or an entire class. Learners manage their courseware assignments, which can be suspended, resumed, and reviewed as often as they like without abandoning any ongoing activities.

The heart of the eLMS delivery platform is a model-based delivery engine that enacts CAPE learning designs with individual learners. These deliveries are extensively instrumented. Every interaction between the learner and the delivery engine is time-stamped and entered into a *delivery record*. The result is a detailed account of what each learner experienced. The collection of these records can be mined for particular information using data mining services of the eLMS Instructor interface. Delivery records serve not just to evaluate learner performance, they also serve to inform improvements to the learning designs themselves.

CAPE and eLMS are intended to be used in the context of "blended learning" where learning experiences performed outside class are used to augment classroom-based instruction. eLMS capabilities that are specific to blended learning include support for *interventions*. When a learner encounters difficulties that are beyond the diagnostic reach of a learning design, the learner can be directed to a human diagnostician, be it the instructor or a teaching assistant. Based on the results of this face-to-face diagnosis and remediation, state can be set in the learner's delivery record that will determine how they continue with the learning experience. Such a facility can be used to support a kind of triage for learners who need assistance most and can also focus attention on opportunities to further improve learning experiences for future learners.

To make its easier to use CAPE-authored learning experience, we are actively pursuing the integration of eLMS with popular course management systems. These platforms are the primary "faces" of on-line learning resources at universities, and it makes sense to us for instructors and students to access eLMS through these platforms. Our first offering—a "building block" integration with the Blackboard Learning System (Figure 2)—is expected to be available by the

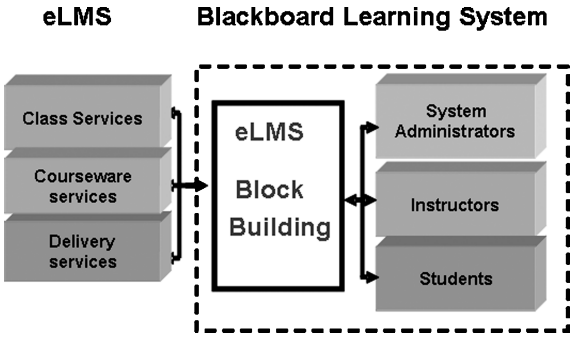

Fig. 2. Integration with a campus-wide course-management system.

time this article is in press. Figure 2 shows how eLMS services are connected to and made available through Blackboard. In the future, we see the possibility that eLMS can devolve into strictly a delivery service.

4.4 Embedding Instructional Capabilities into Existing Tools

"Just-in-Time" (JIT) learning, mentioned in the description of CAPE in Section 4.2, can ease the cognitive burden of using complex technologies, such as tool-based design of embedded software. For initial learning, JIT learning activities can present tasks in a situated context using learning strategies like anchored inquiry [Cognition and Technology Group at Vanderbilt 1993]. Especially when a tool is not used routinely, it is natural for users to forget precisely how things are done, requiring relearning. Sometimes integrated help resources are sufficient to scaffold this learning, although the reference style of such resources often fails to sufficiently contextualize the information provided. Task or problem-based tutorials provide the opportunity to put information into context. Adaptive tutorials allow this information to be specialized to particular classes of learners.

The JIT learning capability created for CAPE is integrated directly into the CAPE modeling environment and provides tutorials authored with CAPE and delivered via eLMS. We are currently investigating techniques for making the selection of JIT learning opportunities more sensitive to the context of the modeling task. For example, GME meta-models organize a domain-specific modeling language (DSML) into kinds of models constituted by kinds of elements and associations. Models can be further organized into *aspects* that subset these elements, focusing the modeler's attention on particular concerns. When the modeler expresses interest in JIT learning activities, the current model and aspect can provide cues for recommending particular resources.

4.5 Embedding Existing Tools into Learning Experiences

Adaptive learning technologies can be used to teach learners how to use another technology, or they can be used to teach a concept where another technology plays a role in exemplification. This is particularly important issue in teaching embedded software and systems, where the use of complex tools is integral and inseparable part of the learning experience. There are also situations where

an adaptive learning technology can play a scaffolding role. Here we examine a couple of examples of these kinds of "embedding" and how CAPE and eLMS can be used to support them.

In creating tutorial resources for CAPE, it was often necessary to construct web pages showing CAPE screenshots and describing procedures involved in completing some task. There are commercial tools for making narrated or annotated animations from screenshot sequences, such as ViewletBuilder from Qarbon (http://www.qarbon.com) and Camtasia Studio from TechSmith (http://www.techsmith.com). The result is basically a movie showing someone else performing some sequence of actions with a tool. As an alternative, we used automation services of the GME to open and manipulate tutorial projects. Our initial aim was to use GME itself as a visualization tool.

The automation capabilities of the GME are based on Microsoft's Component Object Model (COM) (http://www.microsoft.com/com/default.mspx) component technology. Programmatic interfaces are provided for manipulating the modeling representation. Other COM interfaces allow manipulating the model editor; for example, a particular model can be opened in the editor and objects selected. These interfaces can be accessed and scripted using any Microsoft-supported scripting language, such as the Javascript language natively supported by the Internet Explorer web browser.

A Javascript application-programming interface (API) was built to provide higher-level support for employing these manipulations from a CAPE-based learning experience. This approach is very useful for showing learners around in a tutorial GME project, yet teaching a learner how to use a technology typically involves asking them to perform actions and then confirming that these actions were correctly performed. Our interests expanded to using GME as an interactive embedded learning component by detecting user actions as differences from a known initial state of a tutorial project. Communication with the user is achieved using GME's capabilities to dynamically create and remove model annotations. With these visualization, interaction, and annotation capabilities in place, we can lead GME users through sequences of steps comprising some task, ask them to complete actions, evaluate the actions that they actually perform, and remediate as necessary. These capabilities greatly reduce the need to construct tutorial content pages filled with screenshots and assessment-based interactions.

Our work with bioengineering educators in the VaNTH ERC has presented many opportunities to embed learning technologies and other technologies within CAPE designs to teach learners concepts [Roselli et al. 2003]. We have been motivated to support technologies that these educators already know in order to reduce their "learning curve" in applying CAPE. One such technology is MATLAB from MathWorks. This technology offers impressive visualization, computation, and interaction capabilities. It also provides an automation interface, like the GME, that enables it to be manipulated from web-based learning experiences. An added advantage of MATLAB is that it is based on a scripting language.

Similar to the example above, we have implemented support for manipulating MATLAB from a CAPE-based learning design using another Javascript

API, but here the instructions to MATLAB can be communicated in its scripting language M. In addition to statically defined M-code, CAPE's facilities for dynamic content can be used to synthesize M-code at delivery time, which provides a very wide range of possibilities. CAPE-authored assessments can be used for interacting with the learner (say, to gather input parameters or terms), or MATLAB GUIDE user interfaces can be constructed that journal user actions and report these back to eLMS using web services to become part of the learner's delivery record. Also, as in the case of the GME embedding example above, CAPE's native features for adaptive instruction can be used for diagnosis and remediation in further scaffolding MATLAB based exercises.

4.6 Current Status

We believe strongly that the technology described above can be applied to teaching embedded system design. We have recently started building modules for course materials for teaching the theory and practice of embedded systems and we are in the process of investigating how they can be integrated into CAPE-based learning experiences. The CAPE and eLMS tools are currently in use at a number of institutions and are available as non-commercial implementations by agreement with the VaNTH ERC [http://www.vanth.org]. They are based exclusively on open-source technologies.

5. CONCLUDING REMARKS

The teaching of embedded software and systems in an engineering curriculum is a challenging problem because it requires the integration of two distinct disciplines and traditions: systems science and computer science. At Vanderbilt University we are introducing and revising courses in our curriculum that emphasize integration across these disciplines and we are offering a range of undergraduate and graduate level courses that focus on embedded software and systems. The undergraduate classroom experience is augmented by a senior capstone design lab course that enables the practical application of theory, tools, and techniques learned in previous courses. We have also developed a suite of embedded model-based software tools and QoS-enabled component middleware platforms that are used throughout our undergraduate and graduate courses to provide students with hands-on access to cutting-edge technologies. Most recently, we are developing tools that support model-based construction and delivery of sophisticated education content that assists students and practitioners master the complexity of developing embedded software and systems.

The introduction of these courses, tools, and platforms has been underway for several years, but we can already observe significant positive changes in our students. For instance, student feedback on the senior design project has been extremely positive, i.e., students appreciated the integrative nature of projects where computing was interlinked with traditional engineering. Therefore, we believe that the reengineering of our curriculum will provide better education for engineers who will design and build tomorrow's complex embedded software and systems. Our early observations show that this direction is not only viable, but it is essential for the future.

REFERENCES

ANTSAKIS, P. J. ED. 2000. Special issues on hybrid systems: Theory and applications. In *Proceedings of IEEE 88*, 7.

BRUSILOVSKY, P. 1998. Adaptive educational systems on the world-wide-web: A review of available technologies. In *Proceedings of Workshop "WWW-Based Tutoring" at 4th International Conference on Intelligent Tutoring Systems (ITS'98)*, San Antonio, TX, August 16–19.

BUTAZZO, G. C. 1997. Hard Real-Time Computing Systems, Kluwer Academic Publ., Boston, MA.

COGNITION AND TECHNOLOGY GROUP AT VANDERBILT 1993. Anchored instruction and situated cognition revisited. *Educational Technology 33*, 3 (Mar.), 52–70.

CORDRAY, D. S., PION, G. M., HARRIS, A., AND NORRIS, P. 2003. The value of the VaNTH Engineering Research Center. *IEEE Engineering in Medicine and Biology Magazine 22*, 47–54.

GOKHALE, A., BALASUBRAMANIAN, K., BALASUBRAMANIAN, J., KRISHNA, A., EDWARDS, G. T., DENG, G., TURKAY, E., PARSONS, J., AND SCHMIDT, D. C. 2005. Model driven middleware: A new paradigm for deploying and provisioning distributed real-time and embedded applications. *Journal of Science of Computer Programming: Special Issue on Model Driven Architecture*. Mehmet Aksit, ed. (to appear).

GRAESSER, A. C., LU, S., JACKSON, G. T., MITCHELL, H., VENTURA, M., OLNEY, A., AND LOUWERSE, M. M. 2004. AutoTutor: A tutor with dialogue in natural language. *Behavioral Research Methods, Instruments, and Computers 36*, 180–193.

HENZINGER, T. A., HOROWITZ, B., AND KIRSCH, C. M. 2003. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE 91*, 1 (Jan.).

HOROWITZ, B., LIEBMAN, J., MA, C., KOO, T. J., SANGIOVANNI-VINCENTELLI, A., AND SASTRY, S. 2003. Platform-based embedded software design and system integration for autonomous vehicles. *Proceedings of the IEEE 91*, 1 (Jan.).

HOWARD, L. 2002. CAPE: A visual language for courseware authoring. Second Workshop on Domain-Specific Visual Languages, Seattle, WA, November 4.

HOWARD, L. 2003. Adaptive learning technologies for biomedical education. *IEEE Engineering in Medicine and Biology Magazine 22*, 58–65.

IMS SIMPLE SEQUENCING SPECIFICATION. Version 1.0, Instructional Management Systems (IMS) http://www.imsglobal.org/simplesequencing/index.cfm.

KARSAI, G., AGRAWAL, A., AND SHI, F. 2003. On the use of graph transformations for the formal specification of model interpreters. *Journal of Universal Computer Science 9*, 11 (Nov.), 1296–1321.

KARSAI, G., LANG, A., AND NEEMA, S. 2004. Design patterns for open tool integration. *Software and Systems Modeling 4*, 2 (May 2005), 157–170.

KARSAI, G., NEEMA, S., ABBOTT, B., AND SHARP, D. 2002. A modeling language and its supporting tools for avionics systems. In *Proceedings of the 21st Digital Avionics Systems Conference, 2002*. 6A3-1–6A3-13.

KARSAI, G., SZTIPANOVITS, J., LEDECZI, A., AND BAPTY, T. 2003. Model-integrated development of embedded software. In *Proceedings of the IEEE 91*, 1 (Jan.), 145–164.

KOPETZ, H. 2003. The time triggered architecture. *Proceedings of the IEEE 91*, 1 (Jan.).

KOUTSOUKOS, X. AND ANTSAKLIS, P. 2003. Safety and reachability of piecewise linear hybrid dynamical systems based on discrete abstractions. *Journal of Discrete Event Dynamic Systems: Theory and Applications 13*, 3, 203–243.

LAVE, J. AND WENGER, E. 1990. Situated learning: Legitimate peripheral participation. Cambridge University Press, Cambridge.

LEDECZI, A., MAROTI, M., BAKAY, A., KARSAI, G., GARRETT, J., THOMASON, C., NORDSTROM, G., SPRINKLE, J., AND VOLGYESI, P. 2001. The generic modeling environment. *Workshop on Intelligent-Signal Processing, Budapest, Hungary*, May 17.

LEE, E. A. AND SANGIOVANNI-VINCENTELLI, A. 1998. A framework for comparing models of computation. *IEEE Trans. CAD 17*, 12 (Dec.).

MAROTI, M., KUSY, B., SIMON, G., AND LEDECZI, A. 2004. The flooding time synchronization protocol. In *ACM Second International Conference on Embedded Networked Sensor Systems* (Sen-Sys'04), Baltimore, MD (Nov.). 39–49.

MOSTERMAN, P. J. AND BISWAS, G. 1999. Diagnosis of continuous valued systems in transient operating regions. *IEEE Transactions on Systems, Man, and Cybernetics 29*, 554–565.

MURRAY, T. 1999. Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education 10*, 98–129.

NARASIMHAN, P. AND BISWAS, G. 2002. An approach to model-based diagnosis of hybrid systems. *Hybrid Systems: Computation and Control, Fifth Intl. Workshop Stanford, CA*. Lecture Notes in Computer Science, vol. LNCS 2289, C. J. Tomlin and M. R. Greenstreet, Eds. Springer-Verlag, Berlin. 308–322.

ROSELLI, R. J., HOWARD, L., CINNAMON, B., BROPHY, S. P., NORRIS, P., ROTHNEY, M., AND EGGERS, D. 2003. Integration of an interactive free body diagram assistant with a courseware authoring package and an experimental learning management system. In *Proceedings of the American Society for Engineering Education* (CD-ROM DEStech Publications) Session 2793: 10 pp.

SANGIOVANNI-VINCENTELLI, A. AND MARTIN, G. 2001. A vision for embedded systems: Platform-based design and software methodology. *IEEE Design and Test of Computers 18*, 6 (Nov.–Dec.), 23–33.

SCHMIDT, D. C., LEVINE, D. L., AND MUNGEE, S. 1998. The design and performance of real-time object request brokers. *Computer Communications 21*, 4 (Apr.), 294–324.

SZEMETHY, T. AND KARSAI, G. 2004. Platform modeling and model transformations for analysis. *Journal of Universal Computer Science 10*, 10, 1383–1406.

SZTIPANOVITS, J. AND KARSAI, G. 1997. Model-integrated computing. *IEEE Computer*. 110–112.

VAN ROSSUM, G. AND DRAKE, F. L. EDS. 2001. Python Reference Manual, PythonLabs, Virginia, USA, 2001. Available at http://www.python.org.

WANG, N., SCHMIDT, D. C., GOKHALE, A., RODRIGUES, C., NATARAJAN, B., LOYALL, J. P., SCHANTZ, R. E., AND GILL, C. D. 2003. QoS-enabled Middleware, in Middleware for Communications, Qusay Mahmoud, Ed. Wiley, New York.

Received December 2004; revised March 2005; accepted May 2005