# A Middleware Platform for Providing Mobile and Embedded Computing Instruction to Software Engineering Students

Chris A. Mattmann, *Senior Member, IEEE*, Nenad Medvidović, Sam Malek, *Associate Member, IEEE*, George Edwards, *Member, IEEE*, and Somo Banerjee

***Abstract*—As embedded software systems have grown in number, complexity, and importance in the modern world, a corresponding need to teach computer science students how to effectively engineer such systems has arisen. Embedded software systems, such as those that control cell phones, aircraft, and medical equipment, are subject to requirements and constraints that are significantly different from those encountered in the standard desktop computing environment. For example, embedded systems must frequently address challenges that arise from severe resource restrictions (e.g., low memory and network bandwidth), heterogeneous hardware platforms, and safety-critical operations. Software architecture has been shown to be an effective means for coping with such issues, yet traditional courses on embedded software development rarely focus on software architectural abstractions. Instead, they have concentrated on lower-level issues such as programming languages and hardware interfaces. Since 2005 at the University of Southern California, Los Angeles, a unique course has been developed that affords students the opportunity to gain experience and insights on developing software architectures for embedded systems. At the heart of the course is a middleware platform, Prism-MW, that helps students use software architectural principles to construct embedded systems and understand the important challenges of the embedded systems domain. This paper describes this course through the explanation and evaluation of four years of class projects, weaving together the course, the middleware platform, and the relationship of each to three key pedagogical goals that drove the formulation of the course curriculum.**

***Index Terms*—Glide, mobile computing education, Prism-MW, software architecture, software engineering.**

Manuscript received July 12, 2011; revised October 07, 2011; accepted December 01, 2011. Date of publication January 27, 2012; date of current version July 31, 2012. This material is based upon work supported by the National Science Foundation under Grants CCR-9985441 and ITR-0312780. The effort was also supported by the Jet Propulsion Laboratory, managed by the California Institute of Technology under a contract from the National Aeronautics and Space Administration.

C. A. Mattmann is with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109 USA, and also with the Instrument and Science Data Systems Section, NASA Jet Propulsion Laboratory, and the Computer Science Department, University of Southern California, Los Angeles, CA 90089 USA (e-mail: mattmann@jpl.nasa.gov; chris.a.mattmann@jpl.nasa.gov).

N. Medvidović is with the Computer Science Department, University of Southern California, Los Angeles, CA 90089 USA (e-mail: neno@usc.edu).

S. Malek is with the Department of Computer Science, Volgenau School of IT and Engineering, George Mason University, Fairfax, VA 22030 USA (e-mail: smalek@gmu.edu).

G. Edwards is with Blue Cell Software, Los Angeles, CA 90069 USA (e-mail: george@bluecellsoftware.com).8359 Fountain Avenue

S. Banerjee is with Software Engineering, CarsDirect.com, El Segundo, CA 90245 USA (e-mail: s.s.banerjee@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TE.2012.2182998

## I. Introduction

IN THE first decade of the 21st century, the world of computing has moved from large, stationary, desktop machines to small, mobile, handheld, and embedded devices. The methods, techniques, and tools for developing software systems that were successfully applied in the former scenario are not as readily applicable in the latter. Software systems running on networks of embedded devices must necessarily exhibit properties that are not always required of more traditional systems: near-optimal performance, robustness, distribution, decentralization, dynamism, and mobility. Consequently, software engineering for embedded systems is distinctly different from generalized software engineering, and the subject demands specialized courses to be given adequate treatment.

Since 2005, a class titled "Software Engineering for Embedded Systems" has been taught at the University of Southern California (USC), Los Angeles, that examines the key properties of software systems in the embedded, resource-constrained, mobile, and highly distributed world; assesses the applicability of mainstream software engineering methods and techniques (namely, software architecture and middleware) to the embedded systems domain; and exposes students to real-world embedded technology and cutting-edge research in the field. While not the primary focus of the class, some enabling advances in other areas (e.g., embedded, real-time operating systems, wireless networking, the Internet) are also studied from a software application development perspective.

A crucial element of the course is a class project that gives students hands-on experience in engineering software for embedded systems, either through implementing software development infrastructure or developing applications in this important domain. The key enabler of the project is a middleware platform for mobile and embedded computing instruction. The platform, Prism-MW [1], has several important characteristics that make it possible to achieve three pedagogical goals around which the class as a whole is centered. The first goal (G1) is for students to understand and appreciate the unique requirements, constraints, and challenges of the embedded systems domain. The second goal is for students to understand: 1) how and when to apply generalized software engineering methodologies and tools to embedded systems (G2a); and 2) how and when to apply engineering techniques that are specialized for embedded systems (G2b). The third goal (G3) is to provide students hands-on experience with emerging examples of embedded technology.

TABLE I
PEDAGOGICAL GOALS FOR PRISM-MW AND FOR OUR COURSE

| Pedagogical Goal | | Description |
|---|---|---|
| **G1** | Understand Embedded Systems Domain | Discern challenges in embedded systems (such as low memory footprint; frequent disconnection, etc.) and how to design around them. |
| **G2a** | Decide when to use software engineering tools and techniques. | Understand how and when to apply software engineering tools and techniques for the problem. |
| **G2b** | Decide when to specialize. | Decide when specialized embedded systems techniques and approaches are useful and where they fit into the architecture. |
| **G3** | Provide hands-on experience | Provide a challenging and useful practical project for the students to implement giving them experience on an embedded systems focused software problem. |

Prism-MW allows these goals to be achieved more effectively than would otherwise be possible, given the relatively limited time and resources of a one-semester university course. The goals are summarized in Table I.

The decision to use Prism-MW as an instructional tool was based on two key insights: First, software architecture is an effective means of embedded system design; second, middleware is an effective means for embedded system implementation. Prism-MW is an extensible middleware platform that supports the efficient implementation of software architectures in the embedded and mobile setting. To achieve this, Prism-MW is specialized in a number of important ways. First, Prism-MW addresses the specific challenges posed by embedded systems engineering. For example, Prism-MW abstracts heterogeneous hardware interfaces and is exceptionally lightweight. Second, Prism-MW is both extensible and easy to learn, which makes it particularly well suited to the classroom environment. Finally, and perhaps most importantly, Prism-MW provides programming language-level constructs for implementing software architecture-level concepts such as component, connector, configuration, and event (similar to other frameworks such as *ArchJava* [2] as will be discussed in Section II-D). This last property allows students to rapidly implement software architectures for embedded systems, thereby gaining hands-on experience without getting mired in extraneous issues. Furthermore, students are expected to modify and enhance the Prism-MW platform itself to successfully complete the class project, which exposes them to an even wider range of development challenges.

The approach taken by this course stands in stark contrast to the typical embedded systems engineering course found in many universities. Such a course regularly focuses on very low-level issues (e.g., code optimization) rather than high-level strategies (e.g., how architectures are developed, evaluated, implemented, and evolved). By focusing on the principles of software architecture, the middleware platform described here provides a unique opportunity for students to gain experience with, and insights into, the process of developing software for embedded devices, from design through implementation. Ultimately, the Prism-MW middleware platform has been found to be invaluable in students' development into effective embedded software systems engineers. In this paper, the authors describe their experience teaching the class over a five-year period, focusing on the middleware implementation platform as the key enabler of their novel approach to this difficult topic, and on three unique years of course projects that demonstrate Prism-MW's unique ability to provide effective architectural training in the embedded software systems domain.

The rest of the paper is organized as follows. Section II discusses relevant related work and background on software engineering education for embedded systems. Section III discusses in detail the objectives of Prism-MW, as stated briefly above. Section IV describes the three years of class projects. Section V concludes the paper with final thoughts and insights.

## II. BACKGROUND AND RELATED WORK

This section begins with background describing the type (graduate versus undergraduate) and number of students, as well as other key properties of the course. Then follows discussion of four academic areas that are most related to the teaching experience using Prism-MW. First, relevant courses in the general area of Software Engineering Education are identified, and their curriculum and pedagogical goals contrasted with that of the authors' course at USC. Second, some related courses are surveyed, which instruct students in mobile technology. Then, courses are identified, which cover design and implementation aspects of embedded systems software, and finally a similar architectural middleware implementation technology is mentioned.

### A. *Software Engineering for Embedded Systems at USC*

The USC course "CS589: Software Engineering for Embedded Systems" is a lecture-style course available to graduate students at any level, with an average enrollment between 2005–2010 of 24 students. Most graduate students take the course in their first or second years.

The course requires students to review research literature (relevant selected papers in embedded systems, software architecture, mobile computing, wireless grids, etc.) and to select one paper to present in a selected week over the 15-week semester. Student presentations last 20 min, with 5 min reserved for explicit questions, and two or three presentations are typically scheduled per class, leaving about 30 min for open discussion at the end of the class on the day's presentations. The class presentation is worth 15% of the overall grade.

A written exam tests the students on the papers, and on the presentation materials, and represents 25% of the overall grade. Two or three times during the semester, students are given quizzes consisting of short questions about the readings and/or the prior week's presentations. The total percentage value for the quizzes is 10% of the overall grade. Class participation amounts to 10% of the grade.

Students are required to perform a team-based final project using Prism-MW that is worth the remaining 40% of the grade. This project has allowed students to expand upon Prism-MW core functionality directly by extending the core classes, and also to treat Prism-MW as immutable and to build applications on top of the Prism-MW core. This will be discussed further when describing the class projects in Section IV.

Section II-B compares this course to other relevant courses in the field.

## B. Software Engineering Education

Most software engineering courses offered as part of the computer science curriculum in various universities deal with the broad and general aspects of software engineering [3]–[6]. The pair of courses taught by Boehm *et al.* [5], [6] deal with application of software engineering concepts, models, and management approaches for design and development of large software systems, and the projects that have been done as part of these courses over the past years reflect this approach [6]. The course offered by Arms [7] is also of a similar nature and imparts instruction on software engineering's various aspects including feasibility, requirements, design, implementation, testing, and delivery. Anderson, in his course [8], also addresses the same issues while additionally talking about software engineering tools and performance evaluation. Chen and Poon [9] use a sophisticated classification tree method (CTM) to instruct students in "black-box" testing as a method of software testing and verification. In the USC course, on the other hand, the focus is more on software engineering as applicable to embedded, mobile technology, which, as far as the authors are aware, is a novel approach to address cross-cutting issues in the fields of software engineering and embedded, mobile, resource-constrained systems.

Emmerich's course "Advanced Software Engineering" [10] is a natural compliment to the software modeling and design that is emphasized in the USC course. The stated goals of Emmerich's course are to "widen the understanding of software engineering by considering it in a broader context." Topics within the course include instruction in software engineering process, architecture, and advanced modeling. The course described here differs from Emmerich's in that it is focused on the above course topics within the domain of mobile, embedded technology. However, Emmerich's course considers software engineering in the larger context of distributed applications. Additionally, Emmerich's course does not include a course project and involves only examinations and assignments. Projects are an important part of software engineering education [3], [4], and they have been incorporated into the USC course in order to allow the students to encounter, understand, and tackle the embedded systems-related software engineering issues discussed in class in a more realistic and practical scenario.

Kogut's course [11] also revolves around team projects where teams of four or fewer students have to choose one of the two project scenarios. The difference between this course and the USC course is that while in Kogut's course the students are just required to do the project management planning, requirement analysis, and design of the project, in the USC course students are required to do the actual development and implementation of the required application on top of Prism-MW.

## C. Courses in Mobile Technology

Some courses [12], [13] focus on the application and design of mobile technology, without the supplementary constraints levied by the embedded systems domain. An example of this is Nixon's course [13], which has the goal of instructing students in the design, implementation, and evaluation of mobile applications. Mobile middleware, context-aware applications, and models of mobile systems implementation and design are all highlighted and listed as key concepts that the students should learn. Nixon's course differs from the USC course in that the latter requires students to learn mobile technology with the additional constraints of embedded software (low memory, low processor power, and so on). Additionally, the USC course requires students to read and understand several seminal and current research papers defining the problem space of software engineering for embedded systems. Nixon's course only focuses on the design and implementation of mobile technology. Another example of the this trend is the course by Myatt and Starner [12], titled "Mobile and Ubiquitous Computing," which has three main objectives: 1) *understand important historical precedents*; 2) *identify critical technologies*; and 3) *obtain necessary research skills*. The course further requires students to demonstrate their understanding of the research material and also to conduct research through the exploration and proposal of an advanced research project in ubiqutious and mobile computing. Students submit a project proposal, perform the research, and are evaluated on their design and its implementation. Myatt and Starner's course, however, focuses more on the applications and use of ubquitous mobile computing, rather than its design, and evaluation. Additionally, the USC course focuses on interface design (such as middleware interfaces), whereas Myatt and Starner's course neglects to consider interface enabling technologies (although another related course in their curriculum is identified as addressing this topic).

One important aspect of the USC course is its emphasis on a learner-based learning scenario where the students are encouraged to learn the course content by teaching it to themselves, rather than by just listening to it in class lectures. While Ohlsson *et al.* [4] and Bruce *et al.* [14] attempt to achieve this, mainly by introducing projects as part of the curriculum where the student plays an active role, the course described here takes it a step further. Apart from a project that requires the students to form teams and play different roles, the students are also responsible for reading an assortment of selected papers addressing various software engineering issues and concerns in the context of embedded systems. They are then required to present a paper of their choice using presentation tools of their choice.

Moreover, they are also required to actively participate in discussions based on these papers and the student presentations. Similar presentation-centric pedagogical approaches have also been used in other courses [11], [15], [16]. Kogut's course [11] provides the students with a list of topics. The student is required to choose one of these topics. The instructor then provides the student with a publication on this topic, which the student has to analyze, dissect, and present. The authors' approach differs from Kogut's in that the list of publications is already provided to the students to see and read before they choose their topic. This not only allows the student to read and understand beforehand what issues the paper tries to address, but also encourages her/him to delve further into a topic that she/he likes.

Corner's course [16] is the closest in principle to the USC course in that it presents a list of publications from which the student chooses the one that she/he would like to present in class. Similarly, a major emphasis is on a semester-long project. There are some differences, though between Corner's course and the USC course. First, Corner's course is a seminar course and does not have examinations and assignments. The USC course also includes weekly assignments based on the readings scheduled for that week, and one exam that encompasses all the readings. This ensures that the students read all the papers and are hence better prepared to participate in class during the presentations and discussions. The written exam focuses on the general principles discussed in class across the presentations and discussions. Another difference is that while Corner's course allows the students to pick a project of their choice, which had to be approved by the instructor, the USC course provides the students with a realistic project problem that tries to address the salient features of software engineering issues in the context of mobile, resource-constrained embedded systems. In some cases, though, students have been allowed to work on a project of their choice if they could convince the instructor of the suitability and relevance of the chosen project to this course. Here again, the pedagogical issue of learner-based learning is addressed by providing enough flexibility for students to experiment freely within the broad umbrella provided by the course content.

## D. Courses in Embedded Systems

Maher's course [17] titled "Hardware and Software Engineering for Embedded Systems" has the objective of instructing students how to understand real-time, operating systems. It involves a course project dealing with deployment of a real-time, embedded software system on an embedded processor. While Maher's class deals with software engineering issues such as embedded systems' implementation (using the C language), it fails to focus on mobile technology. Additionally, Maher does not focus on design-level software engineering issues such as software architecture, separation of concerns, and evolvability. The USC course, however, focuses on all of these design issues in addition to the implementation aspects of mobile, embedded systems. Furthermore, because Maher's class is primarily geared toward undergraduates, the class fails to focus on the research aspects of embedded, mobile systems. Zalewsk and colleagues have described similar approaches to the study of real-time systems [18], [19], and there have also been related workshops focused on real-time systems education [20].

Vahid taught a course in 1998 called "Software and Hardware Engineering of Embedded Systems" [21]. The goals of Vahid's courses are closest to those of the course described here. In Vahid's class, students are required to read research papers on the design and implementation of embedded systems and also to present a short 10-min summary outlining their comprehension of the underlying material. Vahid notes in his syllabus that embedded and mobile computer systems are becoming pervasive, and that their design is an often-overlooked pedagogical goal in the current curriculum. Vahid's course differs from USC's in that Vahid's course is a seminar course and does not involve a course project. The USC course also gives the students an opportunity to relate embedded systems design and research to hands-on experience (via the course project). Hsiung's course [22] "Embedded Software Engineering" gives students the opportunity to learn how to design efficient and concurrent embedded software systems. The students are required to formulate a course research project, perform the research and implementation work, and then report on the results in a formal presentation to the rest of the class. Hsiung's course, like that described here, focuses not only on embedded systems' implementation issues, but also on system design (both software and hardware architecture). The USC course differs from Hsiung's in that it also requires students to become familiar with topological research studies in embedded systems through the reading and discussion of research papers on embedded systems. Furthermore, Hsiung's course does not focus on embedded technology for mobile devices, as is the case in here.

The course by Schawn *et al.* [15] is titled "Embedded Systems" and is focused around three major course projects. The first involves the implementation of real-time priorities and dynamic adaption of tasks within a real-time operating system. The second involves the design and implementation of a wireless MPEG delivery system. A third option available to the students is to come up with a project of their choice that fits the nature of instruction and the course content. While the course is different from the USC course in that the USC Prism-MW-based projects are focused on the application layer, above the OS, the notion of a wireless MPEG delivery system is similar to the course project in Year 4 (see Section IV-C). In addition, even though students *can* choose their own project in the USC course, this was not the norm, and the students were encouraged to follow the general project description and guidelines.

## E. Relationship to ArchJava

Another related architectural middleware similar to Prism-MW is the ArchJava [2] project. ArchJava is an extension to the Java programming language, allowing the explicit declaration and inclusion of architectural constructs within implementation code. ArchJava focuses on the introduction of new keywords and constructs to Java that allow students and developers to explicitly declare components, connectors, ports, and other architectural constructs in code, akin to Prism-MW. Though Prism-MW does not extend the Java language to do this, it simply provides a framework for constructing architectures via classes, methods, parameters, and so forth. Unlike Prism-MW, though, ArchJava is not strictly focused on architectural styles and does not provide explicit support for modeling styles in implementation or code (though it does support this architectural capability explicitly). Both frameworks are useful pedagogically and have similar teaching utility for those who desire to teach software engineering and architecture in the embedded systems domain. ArchJava has been leveraged in several courses taught by Aldrich *et al.*, including the Spring 2011 CSE 403 Software Engineering course at the University of Washington, Seattle [23].

## III. Prism-MW: Objectives and Capabilities

As outlined in Section I, USC's Software Engineering for Embedded Systems course is centered around the application of software architecture to embedded systems' design. Therefore, the effective use of software architectural constructs was made integral to the class project. However, the selection of a development platform for such a project presents a number of seemingly conflicting requirements. On the one hand, the platform should expose students to the inherent difficulties of the embedded systems domain, while at the same time allowing them to focus on core architectural concepts [24]. Similarly, the development platform needs to be simple and intuitive enough to learn in a short time period, while also challenging students intellectually and providing them with opportunities to pursue interesting and novel project extensions and engineering solutions. The unique design and capabilities of Prism-MW resolve these potential conflicts, and hence provide an effective middleware platform for course projects that supports the original pedagogical goals. This section details the reasons why Prism-MW is particularly advantageous as a design and implementation tool for this embedded software course.

### A. Addressing the Challenges of Embedded Systems

In order to achieve G1, the first pedagogical goal—for students to understand the unique challenges of the embedded systems domain—Prism-MW provides effective solutions to several important embedded software engineering problems. Prism-MW deals with drastic resource constraints imposed in embedded environments through its efficiency and compact memory and complexity footprint. Additionally, Prism-MW transparently mitigates hardware platform heterogeneity and interface heterogeneity that is common among embedded systems. Because successful completion of the class projects requires the development of extensions and enhancements to Prism-MW, students become proficient in implementing these solutions.

*1) Resource-Constrained Environments:* Devices on which embedded applications reside may have limited power, network bandwidth, processor speed, memory, and display size and resolution. These constraints demand highly efficient software systems in terms of computation, communication, and memory footprint. They also demand more unorthodox solutions such as "off-loading" nonessential parts of a system to other devices [25]. Prism-MW imposes minimal overhead on an application's execution and enables efficient execution of applications on platforms with varying characteristics [1]. To illustrate, Prism-MW's core, recorded at the time of architecture initialization, is 2.3 kB. Prism-MW incorporates several optimization techniques, including event routing based on architectural topology and a centralized event queue with a configurable thread pool per address space. These properties make Prism-MW a suitable platform for highly resource-constrained devices (e.g., cell phones, PDAs, robotic rovers, etc.).

*2) Heterogeneous Hardware Platforms:* Embedded computing environments distinguish themselves with proprietary operating systems (e.g., PalmOS, Symbian), specialized dialects of existing programming languages (e.g., Sun's Java KVM, Microsoft's Embedded Visual C++, or EVC++), and device-specific data formats (e.g., prc for PalmOS, efs for Qualcomm's Brew). To adapt to these different environments, Prism-MW is extensible and configurable. For instance, Prism-MW supports multiple architectural styles, awareness, mobility, dynamic reconfigurability, security, real-time support, and delivery guarantees [1]. Prism-MW has been implemented in the Java JVM. Subsets of the described functionality have also been implemented in Java KVM, C++, EVC++, Python, and Qualcomm's Brew; they have been used in sample applications and in evaluating Prism-MW.

### B. Providing a Flexible and Intuitive Learning Platform

The use of Prism-MW as a teaching tool imposes special considerations on its design. Given the relatively short time period available to students to learn the middleware, Prism-MW must be simple and easy to understand. This allows students to concentrate on the fundamental concepts around which Prism-MW is constructed, rather than idiosyncrasies of the middleware itself. Also, Prism-MW must have an open and extensible structure that allows students to experiment with novel approaches to engineering problems. The course project is designed in such a way that many different implementations and solutions are possible, and students are required to extend the middleware platform itself. By incorporating open-ended issues into the class projects, students are encouraged to think creatively and formulate original solutions. Descriptions of the projects are given in Section IV.

*1) Usability:* Prism-MW has a very simple core, with only 11 classes and four interfaces (under 900 SLOC), allowing students to learn to use the framework in a short time period. Because of this, the students can rapidly advance to the important problem-solving elements of the course project. Furthermore, the design of the core (and the entire middleware) was, in so far as possible, kept highly modular by limiting direct dependencies among the classes via abstract classes, interfaces, and inheritance as discussed below.

*2) Extensibility:* Prism-MW has been designed to be easily extensible, allowing students to enhance the existing middleware while abiding by the constraints provided by the embedded, mobile environment. The design of Prism-MW's core supports this objective by providing extensive separation of concerns via explicit architectural constructs and use of abstract classes and interfaces. The core constructs (Component, Connector, Port, Event, and Architecture) are subclassed via specialized classes (ExtensibleComponent, ExtensibleConnector, ExtensiblePort, ExtensibleEvent, and

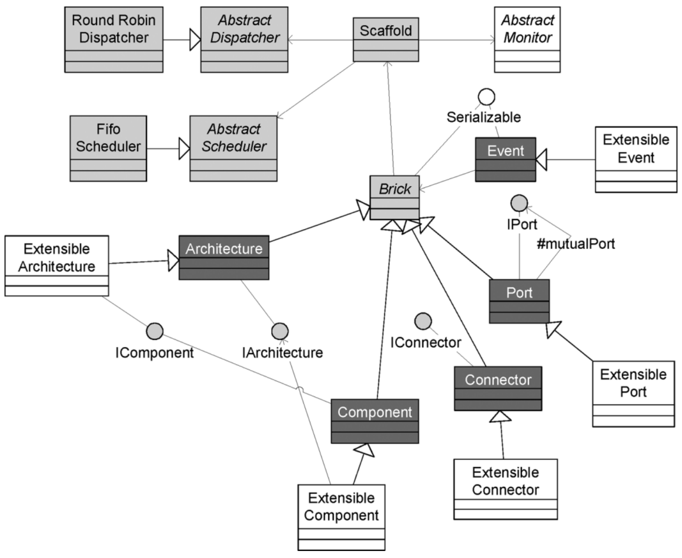

Fig. 1. UML class design view of Prism-MW. Core classes are highlighted.

ExtensibleArchitecture), each of which has a reference to a number of abstract classes (AbstractExtensions). Each AbstractExtension class can have multiple implementations, thus enabling selection of the desired functionality inside each instance of a given Extensible class. If a reference to an AbstractExtension class is instantiated in a given Extensible class instance, that instance will exhibit the behavior realized inside the implementation of that abstract class. Multiple references to abstract classes may be instantiated in a single Extensible class instance. In that case, the instance will exhibit the combined behavior of the installed abstract class implementations.

## C. Supporting Architecture-Based Software Development

The core elements of a software architecture are *Components*, the units of computation; *Connectors*, models of the interactions between the Components; and *Configurations*, arrangements of Components and Connectors, and the rules that guide their composition [26]. Prism-MW provides direct implementation of software architectures in embedded environments through the contribution of programming language-level constructs for implementing the aforementioned software architecture-level atomic elements, including implementation-level support for arbitrary architectural styles (ingredients for architecture found to be effective independent of a particular software domain). Prism-MW allows students to transfer architectural decisions directly into implementations [1].

*1) Design:* Fig. 1 demonstrates the class-level view of Prism-MW where *software classes* are named boxes, and where unfilled arrow heads and lines represent inheritance, e.g., class B is a child of class A if there is a directed arrow from B to A, with the endpoint of the arrow unfilled and pointing at A. Filled arrow heads on directed lines indicate an association relationship—for example, class A has an instance of class B via association if there is a directed line from A to B, with the filled arrow head pointing at B at the end of the line.

Lightly shaded classes constitute the middleware core. The core is a minimal subset of the system that allows for architecture-based implementations that run locally in a single address space. Only the dark gray classes of Prism-MW's core are relevant to the application developer, meaning the application developer need only learn six classes and four interfaces to begin writing Prism-MW-based applications. The goal was to keep the core compact and highly modular; direct dependencies among the classes were limited by using abstract classes, interfaces, and inheritance as discussed below.

Brick is an abstract architectural building block. It aggregates commonality of its subclasses (Architecture, Component, Connector, and Port). Architecture records the configuration of its constituent components, connectors, and ports and allows them to be dynamically modified possibly at system runtime. A distributed application is implemented as a set of interacting Architecture objects.

Events represent architectural communication. An event is comprised of a name and payload. An event's payload includes a set of typed parameters for carrying data and meta-level information (e.g., sender, type). An event type is either a request or a reply.

Ports are the loci of interaction in an architecture. A link between two ports is made by "welding" them together. A port can be welded to at most one other port.

Components compute and maintain their own internal state. Each component can have an arbitrary number of attached ports and can interact with other components by exchanging events via their ports, directly, or indirectly, via connectors.

Connectors are used to control the routing of events among the attached components. Each connector can have an arbitrary number of attached ports. Components attach to connectors by creating a link between a component port and a single connector port. Connectors may support unicast, multicast, or broadcast delivery. In order to support the needs of dynamically changing applications, each Prism-MW component or connector is capable of adding or removing ports a system runtime. Coupled with event-based interaction, dynamic addition of ports has allowed us to dynamically reconfigure software systems, a key behavior in the embedded domains that must respond to constant environmental change.

Each subclass of the Brick class has an associated interface. The IArchitecture interface allows two ports to be "welded" together, and it also exposes send and handle methods used for exchanging events. The IConnector interface provides a handle method for routing of events. The IPort provides the setMutualPort method for creating a one-to-one association between two ports.

Finally, Prism-MW's core associates the Scaffold class with every Brick. Scaffold is used to schedule and queue events for delivery (via the AbstractScheduler class) and pool execution threads used for event dispatching (via the AbstractDispatcher class) in a decoupled manner. Prism-MW's core provides default implementations of AbstractScheduler and AbstractDispatcher: FIFOScheduler and RoundRobinDispatcher, respectively. These extension points allow Prism-MW to independently select the most suitable event scheduling, queueing, and dispatching policies for a given application.

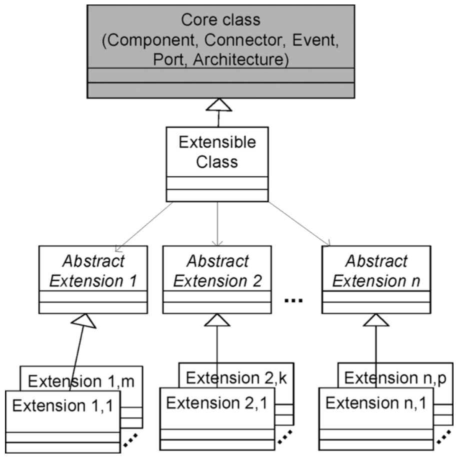

Fig. 2. Prism-MW's extensibility mechanism.

Furthermore, they allow independent assignment of different scheduling, queueing, and dispatching policies to each architectural element, and possibly even to change these policies at runtime. Scaffold also directly aids architectural awareness [27] by allowing probing of the runtime behavior of a Brick via the AbstractMonitor class.

*2) Using Prism-MW:* Prism-MW's core allows developers to construct complex applications, so long as they rely on the default facilities (e.g., event scheduling, dispatching, and routing) and stay within a single address space. First, a developer creates the application-specific portion of each component by subclassing the Component class and providing custom functionality in its handle and start methods. Next, the developer constructs an Architecture class and adds Components (and Connectors if needed). Instances of Ports for components (and Connectors if they are used in a given architecture) are then associated with their container Components (or Connectors), and finally the developer attaches components and connectors via their ports into a configuration, which is achieved by using the Architecture class's weld method.

Developing an application that leverages extensions is done similarly. The statement will be illustrated by describing the method for developing a remote communication extension. First, the developer needs to create a distribution enabled port. To create a distribution enabled port that communicates over TCP/IP, the developer needs to override the default implementation of Prism-MW's core port. Thus, an implementation of the appropriate communication protocol (in this case TCP/IP) is instantiated and associated with an instance of an ExtensiblePort. Since each Extensible class instance is subclassed from one of Prism-MW's core classes (as is shown in Fig. 2), the middleware core (more specifically the thread dispatching, event handling, and routing) is not affected by the new behavior.

Section IV describes the use of Prism-MW in three sample course projects between 2005–2010. After the description of each project, the project and its use of Prism-MW will be related to the pedagogical goals for the class.

## IV. Description of Class Projects

The projects described here are selected from three representative years, starting from the second year of the course. In the first year of teaching the course, the projects were not well developed, and the results were thus not altogether satisfactory. Years two, three, and four are described as they present sound examples of course projects that fall in line with the stated goals of the course. Each of the projects demonstrates goal G3 from Table I, providing the students with hands-on experience in the embedded systems domain.

The challenges faced by students in developing mobile and embedded solutions for their projects are discussed as is the way in which the basic core of Prism-MW was leveraged and extended by the students as appropriate to meet these challenges. Students had access to the following resources throughout the class: 1) a private lab consisting of 10 mid-range wireless-enabled PCs with Intel Pentium IV 1.5-GHz processors and 256 MB of RAM running JVM 1.4.2 on Microsoft Windows XP; 2) 30 PDAs of type Compaq iPAQ H3800 with 200-MHz processors, 64 MB of RAM, WL110 Wireless PC cards, and running Jeode JVM on WindowsCE 2002; and 3) a dedicated network leveraging a dual-band wireless 2.4-GHz router.

### *A. Year Two*

The "Dynamic Service Discovery" project was created during the second year of the course offering. The goal of the project is to develop an application that supports a graceful degradation of services in a distributed system having mobile devices that connect and disconnect unpredictably. This directly illustrates goal G1 from Table I by demonstrating one of the important challenges from the embedded systems domain: disconnected operation. Each device provides and requires different sets of services, and each service is provided and required by multiple devices. Sets of provided and required services may change during runtime. All devices serve as both mobile servers and clients, providing their own local services to other devices and/or using local services provided by other devices. In addition, clients of services provided by devices that become disconnected are required to search for no longer available service(s) that are provided by other "live" device(s). The described setting is supposed to simulate a distribution that integrates mobile devices and other devices, sometimes termed spontaneous networking [28], providing the following key features: 1) easy connection to a local network; 2) easy integration with local services, such as automatic discovery of connected devices, with no special configuration actions by the user; and 3) limited connectivity. Furthermore, the project requirements are also in accordance with two major discovery service interfaces: 1) registration service, and 2) lookup service.

The following describes constraints of the project assignment. Compaq iPAQ PocketPC H3800 Series devices with WL110 Wireless PC cards were used as mobile devices. Devices communicated with each other in an ad hoc wireless network using IEEE 802.11 standard. The application was required to be implemented on top of the Java version of Prism-MW. Furthermore, each PocketPC was equipped with JeodeRuntime as a Java virtual machine implementation of Sun's Personal Java 1.2 (JDK 1.1.8) specification. There were eight teams of three students. As a sample application for the specified project goal, students were required to implement a simple calculator application with various arithmetic operations serving as services. The correctness and performance of the solutions were determined based on results obtained from a mediated competition among teams. Therefore, the requirements specification included protocols for proper communication among devices belonging to different teams, such as protocols for device and service discovery. The development of such protocols allowed the students to see examples of software engineering specific tools and approaches (the development of protocols, or *connectors* for communication), helping to illustrate goal G2a from Table I.

Furthermore, various gauges were used to measure the performance, such as resource awareness (CPU speed and network throughput) or device load (CPU load and network load per socket). Each solution was required to meet the following criteria of being able to:

1) access required services locally;
2) access required services from a known remote host;
3) discover and access services in a decentralized setting;
4) discover and access services in a decentralized setting but most efficiently (based on resource awareness and/or device load);
5) discover and access services in a decentralized setting but for multiple devices simultaneously;
6) discover and access services in a decentralized setting but with trusted devices being added;
7) discover and access services in a decentralized setting, but with potentially nontrusted (the opponents') devices being added.

The winners were determined based on the following two criteria: 1) smallest number of nonserviced requests due to disconnection, and 2) fastest execution time.

## B. Year Three

In the third year, 14 teams of three students were created. The project was intended to empower students with the ability to build support for dynamic software architecture adaptation in a distributed and embedded environment, focusing in part on goal G2a from Table I. The focus of this project was on the deployment and migration of software components between devices. The project was broken down into four tasks and an extra credit portion.

1) Build support for component deployment and migration:
    a) ability to deploy software components from a centralized server (i.e., a desktop PC) onto mobile hosts (i.e., PDAs);
    b) ability to migrate a component from one mobile host to another mobile host;
    c) ability to link remote software components to other remote components and connectors;
    d) ability to start and execute remote components.

    This feature of the project allowed the students to understand where it made sense to consider how Prism-MW's Architecture class (recall Section III-C.I) model can assist in determining the appropriate hardware hosts to migrate software components to. In addition, the project illustrates challenges in embedded systems (goal G1 from Table I) such as disconnected operation, low memory or bandwidth, and quality of service (QoS).

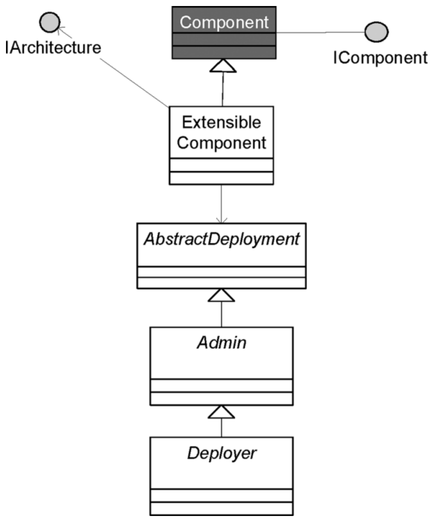

Fig. 3. Prism-MW's deployment extensions.

2) Create a software architectural model of the system: a) create and maintain local architectural model of the system on each mobile host; b) create and maintain a complete model of the system's architecture on the central server. This demonstrates goal G2a from Table I.
3) Display the system's software architecture; a) using the models, display the local architecture on each host; b) display the complete system's architecture on a centralized server. Again, this demonstrates goal G2a from Table I.
4) Maintain consistency: a) ability to maintain the software architecture models and displays; and b) automatically make the appropriate linkages when a component is migrated from one host to another host. This demonstrates both goals G2a and G2b from Table I.
5) Extra credit: Maintain consistency in a decentralized environment (i.e., when there is no centralized server to hold the entire system's architecture model). This portion highlights developing specialized embedded systems and architectural algorithms (goals G2a and G2b from Table I) for autonomic system behavior in an embedded environment.

Successful completion of the project required the students to leverage Prism-MW's support for meta-level components. The role of components at the meta-level is to observe and/or facilitate different aspects of the execution of application-level components. Meta-level components may be welded to specific application-level connectors to exercise control over a particular portion of the architecture. Alternatively, a meta-level component may remain unwelded and may instead exercise control over the entire architecture via its pointer to the Architecture object. Typically, a meta-level component is implemented as an ExtensibleComponent, which contains a reference to the Architecture object via the IArchitecture interface and allows the component's instances to effect runtime changes on the system's local (sub)architecture. The ExtensibleComponent class can also have references to abstract classes that provide specific (meta-level) functionality (see Fig. 3). The students had to develop the implementation of the two extensions shown in Fig. 3.

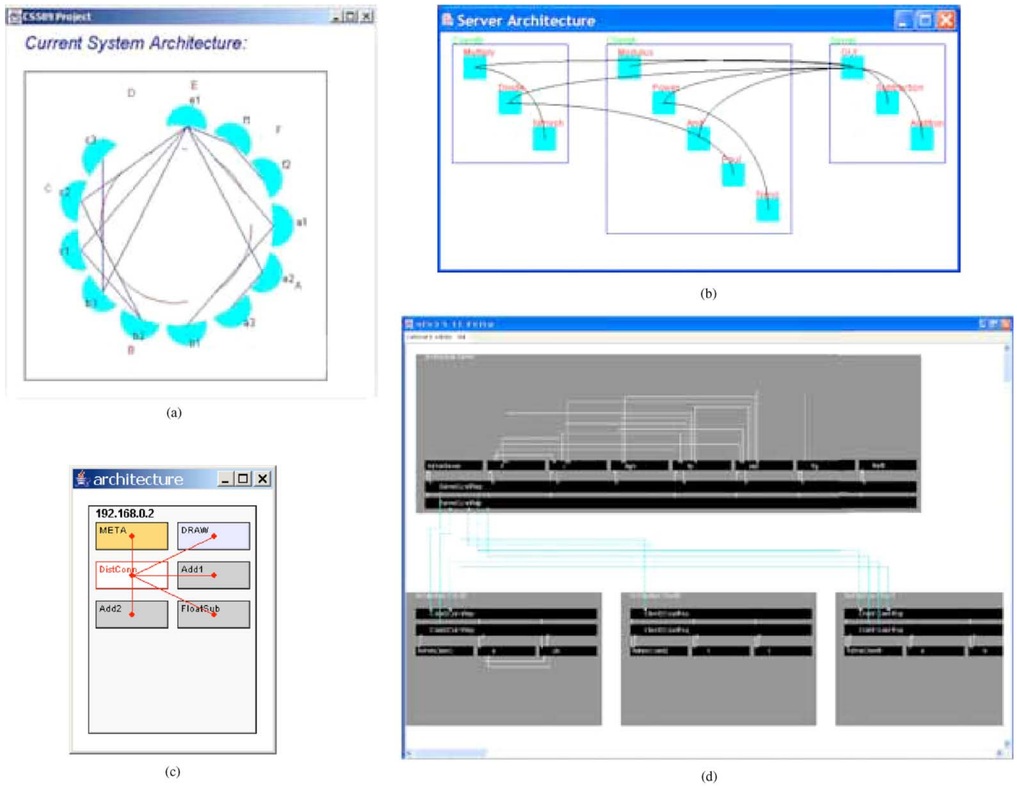

Fig. 4. Different visualizations of the system's deployment architecture by different teams. (a) Diagram inspired by flower petals, where the half-circles present in the upper left represent the components, the curve lines represent the connectors, and the straight lines represent component interdependencies. (b) Diagram that does not show the connectors. (c) Diagram that only shows the connection between components and connectors. (d) Very detailed depiction of system's architecture inspired by circuitry diagrams that shows the configuration of components and connectors.

The extra credit portion of the project (step 5 of the project) was much more challenging. The students were to devise techniques to ensure consistency between the hosts in a semidisconnected topology of hosts. This required each host to store some information about the architecture of other hosts. There were two challenges that students had to overcome: 1) determine the information they need from other hosts, and 2) determine how to acquire that information from the remote hosts.

To address the first challenge, students had to provide a mechanism for each AdminComponent to determine the interdependent points between the local architecture and other remote architectures. An example of such an interdependency is a component that makes requests to a remote component, in which case the host should store the location (host) of the remotely deployed component.

To address the second challenge, students had to implement a mechanism to communicate with hosts that were not directly connected. Students took various different approaches to address this challenge, such as relaying messages via neighboring hosts to a desired remote host or selective group broadcasting of messages.

Another challenge that the students had to face was the visualization of the system's architecture (step 3 of the project). Students had to be creative in rendering arbitrarily large deployment architectures on the PDA's small display monitor. Students had to make a tradeoff between the amount of information presented visually versus the clarity of images displayed. The above challenges were exacerbated by the fact that students had to develop their code in emulators before testing it on the actual PDA. Furthermore, they were limited to writing code that used a subset of JDK libraries for graphical rendering (subset of AWT package), which is supported by the version of JVM (Jeode) available on the PDAs. Fig. 4 shows the different visualization schemes created by the students. It shows the various tradeoffs and levels of detail that students chose to capture.

## C. *Year Four*

In the fourth year of projects, five teams of three students and two teams of four students were created. The project given to the students was intended to provide them with experience of the new paradigm of grid computing in the embedded systems realm. Each team was given the following scenario (shown in Fig. 5). A team of scientists on Mars would like to perform some science observations that require the use of computing resources across the team. The scientists are distributed, each have different kinds and amounts of data (and processing power) and need an infrastructure and methodology to reliably share data and computing resources in order to achieve the necessary science goals, e.g., "identify rock and take sample," or "communicate temperature measurement to science colleague." With this high-level scenario, the students were required to use replication and distribution to increase the system availability and ultimately aid the scientists in achieving their science goals. Students replicated data and computing resources across the coalition of scientists to increase data and compute availability, il-

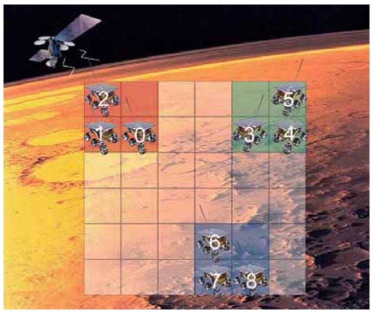

Fig. 5. Example GUI for the fourth-year project scenario: sharing resources on Mars.

lustrating goals G2a and G2b from Table I. Data and computing resources were replicated by distributing one or more resources from one (set of) PDA(s) to another. The project was broken down into four major components.

1) Develop a quantifiable measure of the system's availability based on the current state of compute and data resources. Then, increase the overall availability of the system and quantifiably show this increase. This illustrates goal G3 from Table I in coming up with an embedded system specific measurement of availability and how to show increase in it.
2) Create a generic, pluggable replica location service by extending the existing Prism-MW. The replica location service will determine where (i.e., which PDAs) to replicate resources to increase availability. This illustrates goal G2a from Table I in coming up with an architecture-based service in Prism-MW to replicate components from one hardware host to another.
3) Create a generic, pluggable replica distribution service by extending the existing Prism-MW. The replica distribution service will create and move the resource replicas to precalculated PDAs, which will result in an availability increase. In focusing on data replication, the students were asked to hone in on pedagogical goal G2ab from Table I.
4) Write an application that creates a heavy request and processing load to demonstrate increased availability. This illustrated all the goals—G1, G2a, G2b, and G3—from Table I.

Teams that successfully completed the project all implemented the replica monitoring, resource sharing, and replica actuation services described above. The replica monitoring component gathered data, such as the amount of times a particular resource was requested, which host requested it, and so on. The resource sharing component was responsible for providing access to, and cataloging compute and data resources available on, each host. Finally, the replica actuator provided a standard method for replicating different resources (such as files in a file system, rows in a database, or components running on a mobile host).

One of the major challenges of the project was leveraging the existing Prism-MW capabilities, along with a set of basic grid services, dubbed GLIDE [29], built on top of Prism-MW.

## V. Conclusion

This paper has presented the authors' experience from 2005–2010 teaching a course titled "CS589: Software Engineering for Embedded Systems" at the University of Southern California. This course distinguishes itself from others like it by focusing on architectural abstractions and design for embedded systems and mobile software technology, rather than just implementation-level detail and optimizations. The course teaches students how to effectively architect and design software systems suitable for deployment in embedded environments.

The course has three key pedagogical goals that are aided and abetted by an architectural middleware platform, "Prism-MW," which helps to teach the class. Students learn Prism-MW easily, and rapidly, allowing them to extend and use Prism-MW to build complex class projects; since 2005, students have conducted sound research and even provided novel solutions. The class is supplemented through the use of learning devices including student presentations, surveys of topical embedded software literature, and team-based, cooperative learning.

## References

[1] S. Malek, M. Mikic-Rakic, and N. Medvidovic, "A style-aware architectural middleware for resource-constrained, distributed systems," *IEEE Trans. Softw. Eng.*, vol. 31, no. 3, pp. 256–272, Mar. 2005.
[2] J. Aldrich, C. Chambers, and D. Notkin, "Archjava: Connecting software architecture to implementation," in *Proc. ICSE*, 2002, pp. 187–197.
[3] R. Dawson, "Twenty dirty tricks to train software engineers," in *Proc. ICSE*, 2000, pp. 209–218.
[4] L. Ohlsson and C. Johansson, "A practice driven approach to software engineering education," *IEEE Trans. Educ.*, vol. 38, no. 3, pp. 291–295, Aug. 1995.
[5] B. Boehm, E. Colbert, and A. W. Brown, "Software Engineering I, Fall 2004 syllabus," University of Southern California, Los Angeles, CA, 2004 [Online]. Available: http://sunset.usc.edu/classes/cs577a_2004/
[6] B. Boehm, "Software Engineering II, Fall 2002 syllabus," University of Southern California, Los Angeles, CA, 2002 [Online]. Available: http://sunset.usc.edu/classes/cs577b_2002/index.html
[7] W. Y. Arms, "Software Engineering, Spring 2004 syllabus," Cornell University, Ithaca, NY, 2004 [Online]. Available: http://www.cs.cornell.edu/courses/cs501/2004sp/syllabus.html
[8] K. Anderson, "Software Engineering Methods and Tools, Fall 2004 syllabus," University of Colorado, Boulder, CO, 2004 [Online]. Available: http://www.cs.colorado.edu/users/kena/classes/3308/f04/schedule.html
[9] T. Y. Chen and P. Poon, "Experience with teaching black-box testing in a computer science/software engineering curriculum," *IEEE Trans. Educ.*, vol. 47, no. 1, pp. 42–50, Feb. 2004.
[10] W. Emmerich, "Advanced Software Engineering syllabus," University College London, London, U.K., 2004 [Online]. Available: http://web.archive.org/web/20051221184427/http://www.cs.ucl.ac.uk/teaching/syllabus/ug/3c05.htm
[11] P. Kogut, "MCS580 Software Engineering, Fall 2004 syllabus," Drexel University, Philadelphia, PA, 2004 [Online]. Available: http://web.archive.org/web/20061010215752/http://www.mcs.drexel.edu/~pkogut/mcs580.html
[12] E. Mynatt and T. Starner, "Mobile and Ubiqutous Computing, Spring 2000 syllabus," Georgia Institute of Technology, Atlanta, GA, 2000 [Online]. Available: http://www.cc.gatech.edu/classes/AY2000/cs7470_spring/
[13] P. A. Nixon, "Mobile Software and Applications, Spring 2005 syllabus," University of Strathclyde, Glasgow, U.K., 2005 [Online]. Available: http://www.cis.strath.ac.uk/teaching/ug/syllabus/504.html
[14] J. W. Bruce, J. C. Harden, and R. B. Reese, "Cooperative and progressive design experience for embedded systems," *IEEE Trans. Educ.*, vol. 47, no. 1, pp. 83–92, Feb. 2004.
[15] K. Schawn, C. Pu, and L. Daley, "Cs4220: Embedded Systems, Spring 2002 syllabus," Georgia Institute of Technology, Atlanta, GA, 2002 [Online]. Available: http://www.cc.gatech.edu/classes/AY2002/cs4220_spring/
[16] M. Corner, "Mobile and Pervasive Computing, Fall 2003 syllabus," University of Massachusetts, Amherst, MA, 2003 [Online]. Available: http://www.cs.umass.edu/~mcorner/courses/691M/syllabus.pdf

[17] R. Maher, "Syllabus for EE475: HW&SW Engineering for Embedded Systems, Fall 2003," Montana State University, Bozeman, MT, 2003 [Online]. Available: http://www.coe.montana.edu/ee/rmaher/ee475_FL03/syllabus.htm

[18] J. Zalewski, "Real-time software architectures and design patterns: Fundamental concepts and their consequences," *Annu. Rev. Control*, vol. 25, pp. 133–146, 2001.

[19] A. J. Kornecki, J. Zalewski, and D. Eyassu, "Learning real-time programming concepts through vxworks lab experiments," in *Proc. Conf. Softw. Eng. Educ. Training*, 2000, pp. 294–294.

[20] J. Madsen, "Embedded systems education for the future," *Proc. IEEE*, vol. 88, no. 1, pp. 23–30, Jan. 2000.

[21] F. Vahid, "Software and Hardware Engineering of Embedded Systems, Fall 1998 syllabus," University of California, Riverside, CA, 1998 [Online]. Available: http://www.cs.ucr.edu/~vahid/courses/269_f98/syllabus.html

[22] P. Hsiung, "Syllabus (Embedded Software Engineering)," National Chung Cheng University, Taiwan, 2004 [Online]. Available: http://www.cs.ccu.edu.tw/ pahsiung/courses/ese/info/syllabus.html

[23] J. Aldrich, C. Chambers, and D. Notkin, "ArchJava: Connecting software architecture to Implementation," University of Washington, Seattle, WA, 2011 [Online]. Available: http://www.cs.washington.edu/education/courses/cse403/04sp/lectures/ArchJava-4up.pdf

[24] M. Mikic-Rakic and N. Medvidovic, "Adaptable architectural middleware for programming-in-the-small-and-many," in *Proc. Middleware*, 2003, pp. 162–181.

[25] N. Medvidovic, M. Mikic-Rakic, N. R. Mehta, and S. Malek, "Software architectural support for handheld computing," *Computer*, vol. 36, no. 9, pp. 66–73, 2003.

[26] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Trans. Softw. Eng.*, vol. 26, no. 1, pp. 70–93, Jan. 2000.

[27] L. Capra, W. Emmerich, and C. Mascolo, "Middleware for mobile computing," University College London, London, U.K., Tech. Rep. RN/30/01, 2001.

[28] G. Coulouris, J. Dollimore, and T. Kindberg*, Distributed Systems: Concepts and Design*. Reading, MA: Addison-Wesley, 2001.

[29] C. A. Mattmann and N. Medvidovic, "The gridlite dream: Bringing the grid to your pocket," in *Proc. Monterey Workshop*, F. Kordon and J. Sztipanovits, Eds., 2005, vol. 4322, Lecture Notes in Computer Science, pp. 70–87.

**Chris A. Mattmann** (M'03–SM'10) received the Ph.D. degree in computer science from the University of Southern California, Los Angeles, in 2007.

He is a Senior Computer Scientist with NASA's Jet Propulsion Laboratory, Pasadena, CA, working on instrument and science data systems on earth science missions and informatics tasks. He is also an Adjunct Assistant Professor with the Computer Science Department, University of Southern California. He is the first NASA member elected to the Apache Software Foundation, the world's most prominent open-source software organization. His research interests are primarily software architecture and large-scale data-intensive systems.

Dr. Mattmann is a member of the ACM and ACM SIGSOFT.

**Nenad Medvidović** received the Ph.D. degree in information and computer science from the University of California, Irvine, in 1999.

He is a Professor with the Computer Science Department, University of Southern California (USC), Los Angeles. He is the Director of the USC Center for Systems and Software Engineering (CSSE). He is a coauthor of a textbook on software architectures. His research interests are in the area of architecture-based software development. His work focuses on software architecture modeling and analysis, middleware facilities for architectural implementation, domain-specific architectures, architectural styles, and architecture-level support for software development in mobile, resource-constrained, and embedded environments.

Prof. Medvidović is a member of the ACM and ACM SIGSOFT. He was the Program Co-Chair of the 2011 International Conference on Software Engineering (ICSE 2011). He is a recipient of the National Science Foundation CAREER Award, the Okawa Foundation Research Grant, the IBM Real-Time Innovation Award, and the USC Mellon Mentoring Award. He is a coauthor of the ICSE 1998 paper titled "Architecture-Based Runtime Software Evolution," which was recognized as that conference's Most Influential Paper.

**Sam Malek** (A'11) received the B.S. degree in information and computer science (*cum laude*) from the University of California, Irvine, in 2000, and the M.S. and Ph.D. degrees in computer science from the University of Southern California (USC), Los Angeles, in 2004 and 2007, respectively.

He is an Assistant Professor with the Department of Computer Science, George Mason University (GMU), Fairfax, VA. He is also a faculty member of the C4I Center, GMU. His general research interests are in the field of software engineering, and to date his focus has spanned the areas of software architecture, distributed and embedded software systems, middleware, autonomic computing, service-oriented architectures, and quality-of-service analysis. His research has been funded by NSF, DARPA, the US Army, and SAIC.

Dr. Malek is a member of the ACM and ACM SIGSOFT. His dissertation research was nominated by USC for the final round of the ACM Doctoral Dissertation Competition in 2007. He is the recipient of numerous awards, including the USC Viterbi School of Engineering Fellow Award in 2004 and the GMU Computer Science Department's Outstanding Young Faculty Research Award in 2011.

**George Edwards** (M'10) received the B.S. degree from Vanderbilt University, Nashville, TN, in 2003, and the M.S. and Ph.D. degrees from the University of Southern California (USC), Los Angeles, in 2006 and 2010, respectively, all in computer science.

He is the CEO and Chief Scientist with Blue Cell Software, LLC, Los Angeles, CA, a developer of advanced modeling, simulation, and analysis software. He formerly worked as a Researcher and Engineer with IBM, Yorktown Heights, NY; Boeing, Huntington Beach, CA; and USC. He has authored dozens of articles related to software engineering and distributed systems for scientific journals and industry magazines. He is listed as a co-inventor on two pending patent applications for novel software technology, and he has served as a software expert for plaintiffs in patent litigation.

**Somo Banerjee** received the M.S. degree in computer science, majoring in software architecture, from Louisiana State University (LSU), Baton Rouge, in 2004.

He is a Senior Java Developer with CarsDirect.com, Los Angeles, CA, an automotive e-commerce company. He has also worked as a Research Assistant with the University of Southern California, Los Angeles.

Mr. Banerjee was a recipient of LSU's Graduate Student Supplement Award in 2003.