Event-Driven Computing Projects for Software Engineering Education

Marjorie Skubic and James Laffey
Computer Engineering and Computer Science Department / School of Information Science and Learning Technologies
University of Missouri-Columbia
skubicm@missouri.edu / laffeyj@missouri.edu

Abstract

There is a growing need in the software industry for the development of systems with a dynamic, event-driven behavior, such as interactive human-computer interfaces, client-server architectures, and embedded systems. Developing successful, event-driven software requires a paradigm shift from traditional program development, and new curriculum approaches are needed to help computer science and engineering students develop competencies. In this paper, we describe an effort to address this problem through hands-on projects that provide experience in developing dynamic, event-driven systems and let the students physically see the results of their efforts. We describe our project testbed and exercises, based on the smart home theme, and report our experiences with using the testbed in an actual course setting. Although the proof of concept is being evaluated in a software engineering course, the project theme and testbed could be used in other computer-related courses, such as object-oriented programming, embedded systems, or even a first level computer science course.

Introduction

There is a growing need in the software industry for the development of systems with a dynamic, event-driven behavior, *e.g.*, interactive human-computer interfaces, client-server architectures, and embedded systems[1]. Developing successful, event-driven software requires a paradigm shift from the traditional, sequential program development. Unlike sequential programs, event-driven software must work correctly in an environment with concurrent processes, uncertainties, and dynamic, external entities; indeed, the program may execute differently each time because of these outside effects.

New curriculum approaches are needed to develop programming competencies for event-driven software [2]. Students do not typically get significant experience in studying or developing event-driven software development. As a result, they have difficulties learning behavioral models for dynamic software systems and understanding the concepts necessary to design, program, and test such systems. To prepare students for their future careers as software engineers, they need experience in realistic projects that develop event-driven software [3].

In this paper, we describe an effort to address this problem through hands-on projects that provide experience in developing dynamic, event-driven systems, and specifically, projects that let the students physically see the results of their efforts. Following the theme of the smart home, we have been developing a project testbed (hardware and software) and corresponding project exercises to teach software development of event-driven systems and to introduce the idea of embedded "appliances". We hope to motivate students by making the projects fun and interesting. At the same time, the concepts are better illustrated through hands-on projects, which have been shown to yield increased learning. We simulate a real world project by providing incremental, project-related exercises that together comprise a semester-long "project", covering the critical elements in a software development process lifecycle. As a proof of concept, the idea is being tested in an existing software engineering course, which has been required for all computer science (CS) and computer engineering (CE) undergraduates at MU.

Many education projects have focused on related topics, but target different student groups, *e.g*., teaching electrical engineering (EE) and CE students how to co-design hardware and software for embedded systems or teaching mechatronics and/or robotics to mechanical engineering (ME) and EE students. In contrast, we are focusing on software modeling and development for the more general category of event-driven systems, targeting CS and CE students, and including software engineering related issues. It is also our hope that the smart home testbed and exercises will be attractive to a broader group of students that tend to be excluded from the more hardware-oriented courses.

Similar efforts in teaching event-driven software development principles to CS students include Stein's Rethinking CS101 project [2]. However, in this introductory course, the projects are structured as smaller, isolated projects and have been software only. Other related efforts include courses at the University of Victoria (Canada) [4] and Rowan University [5]. Li's course on Software Models for Embedded Systems has similar objectives but focuses specifically on embedded systems [4]. The course at Rowan University is a CS1 course (focusing on object oriented programming) but does include some hardware; students are reported to like the projects[5].

In the remaining sections, we discuss our educational plan, the project theme and testbed implementation, assignments using the testbed, and include comments regarding the evaluation of the effort.

Educational Plan

The target group for this effort includes primarily undergraduate students majoring in CS and CE. To include new directions in the curriculum, it is tempting to introduce yet another course; however, we decided instead to incorporate these project exercises into an existing course. The software engineering course was chosen because it has many similar objectives and has, in the past, introduced students to state-driven models of dynamic systems. However, our observation is that students have not readily grasped the concepts of dynamic, event-driven systems, perhaps in part because they have had little experience in programming such systems. Our intent was to use the new project exercises to increase students' learning in the analysis,

modeling, and development of event-driven software, and, thus, better prepare them for careers as software engineers.

The general goals of the software engineering course are listed below:

- Introduce an organized and systematic approach to developing software
- Provide practice in requirements specification, analysis, design, implementation, and testing of a software system
- Provide experience in software development within a team setting

Within this context, topics include software process models, software analysis and modeling techniques (including models for event-driven systems), design methods, software testing, and some project management. Object oriented methodologies are emphasized. To reinforce the topics, a semester project is assigned. Students are grouped into teams for the project and progress through the entire development lifecycle from requirements analysis to testing.

In the past, we have tried both assigning a specific project to all students and also allowing students to propose their own projects. Although it is advantageous to allow each group to develop a different project (and ideally with a real client), this presents practical difficulties. In many academic environments, there is limited support for teaching assistants and lab assistants. The additional burden tends to make instructors reluctant to assign real-world projects. In addition to the difficulties in supporting and managing the diverse teams, disparity in project difficulty level can make assessment awkward. More importantly, however, the student-chosen projects will not necessarily include all elements that are needed to satisfy the course objectives. In particular, we have observed from previous classes, that student projects do not generally contain a sufficient dynamic behavior to provide adequate illustration of the event-driven software concepts, possibly because students do not feel confident developing such software.

An alternative is to assign a similar project to all teams, but ideally, one that can provide some flexibility in design and implementation; this is the strategy we have taken here. We have developed a testbed for exploring event-driven systems and developed a project concept that teaches the required principles, but yet is open-ended enough to permit flexibility and encourage creativity. The goal is to address the educational objectives but at the same time make the projects manageable so that the developed project units can be utilized broadly in a wide variety of educational institutions. The project exercises are structured as a set of incremental assignments that build on each other. The last assignment culminates in a completed system.

Although the target group is CS and CE undergraduates, the software engineering course also attracts graduate students, especially first year grad students with degrees from other disciplines, and some undergraduates outside of the major. Generally the CS and CE majors take the course in their junior or senior year. The total class size currently runs 70-75 students. The students form their own groups, typically 4-5 students in a team.

Project Theme

We started the effort with the "smart home" concept but have since generalized it to include work environments. The description of the *building automation* project is provided to the students at the beginning of the semester as shown below:

A PC-based, building automation computer system is to be developed for monitoring and controlling devices in a home or work environment that use energy or other resources (e.g., lights, heating, air conditioning, stereo, TV, VCR, computer, pool, yard sprinklers). The user should be able to specify conditions (e.g., turn on light, set the temperature to 72 degrees, turn on lawn sprinklers at 6 a.m.) and observe the results via display on the computer. The user should be able to configure the system as desired (i.e., the devices should not be hard-coded). A logging function should also be included so that the user has a record of the status and control events.

The description is intentionally general to allow room for creativity and flexibility. Students can choose the specific devices they want to control and how they interact. Many teams have chosen to implement a smart home; however, some of the more interesting projects have resulted from automated work environments.

Project Testbed

To facilitate the smart home and smart workplace concept, several commercially available devices were investigated, with the goal to make the system easy to use, flexible and accessible, and to use off-the-shelf equipment as much as possible so that the testbed could be duplicated at other institutions. For ease and simplicity, a generic PC platform is used as the testbed base. The X10 protocol [6] was chosen as a basis for communication with external devices. This standard utilizes existing electric power lines to transmit information so it has a wireless "feel". Devices such as lamps and household appliances are plugged into the power lines via an X10 module that allows addressing through X10 commands. A lamp module is shown in Figure 1. Devices are assigned both a house code (A-P) and a device number (1-16), which provides addressing for 256 devices. A variety of X10 devices are readily available [7]. There has been some concern of noise and reliability problems with X10; however, we have not noticed any substantial problems in our lab.

Within the X10 world, we explored several approaches for connecting devices to a PC. The first prototype used a two-way computer interface, the CM11A[8], which connects to the PC

Figure 1. X10 2-Way Lamp Module[8], addressable with a house code (A-P) and device number (1-16) for 256 possible addresses.

directly via the serial port. The low-level java code to formulate the X10 commands through the CM11A was deemed overly complicated and resulted in timing delays. Also, we were not able to get adequate input signals for sensor readings using this interface and thus needed another mechanism for reading sensor inputs.

The project testbed implemented as the second prototype is shown in Figure 2. Ultimately we decided to design our own micro-controller board, based on the Basic Stamp II [9]. This processor provides inputs for reading sensor signals and also includes capabilities for sending X10 commands through a power line interface such as the TW523 [8]. The micro-controller board connects to the PC via a serial port. A simple protocol was designed for reading sensor data and for sending commands to X10-addressed devices.

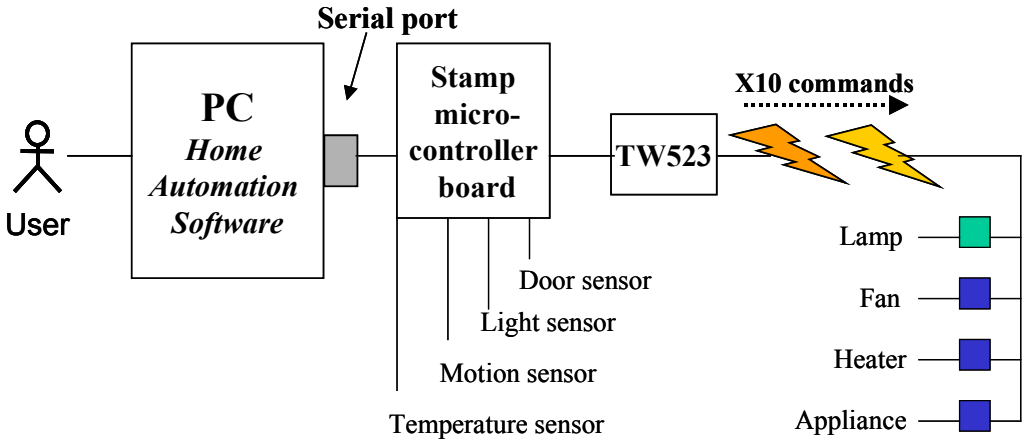

Figure 2. Project Testbed. X10 commands are issued to appliances via the Basic Stamp II [8] micro-controller. Lighting appliances are connected to the power line via an addressable X10 lamp module that allows for dimness levels. Other appliances are connected via X10 appliance modules that processe on/off commands. Sensors are connected to the micro-controller board directly.

The micro-controller board is shown in Figure 3. Analog sensors are connected via the ADC (analog to digital converter) connections on the left. Digital sensors and the output to the TW523 are made via the connections on the right. Each board is equipped with 4 sensors as shown in Figure 2. Furthermore, the flexible design allows different sensors and/or additional sensors to be connected. Eight micro-controller boards have been built in-house. The printed circuit boards were fabricated by MU Engineering Technical Services. The total cost of the board was about $120, which covers all parts, including the Basic Stamp II and the sensors.

To support the students' projects, 8 workstations were set up in one of the campus PC labs, each with the hardware shown in Figure 2. Lamps, fans, and heaters were connected via the appropriate X10 modules to provide a variety of output devices. In the future, other appliances can easily be connected via a basic appliance module. In addition, students could simulate the operation of other devices not available physically, such as sprinkler systems or a mailbox sensor. To support remote software development outside of the campus lab, a simulator was also developed for the micro-controller board so that software could be run and tested on virtually any PC.

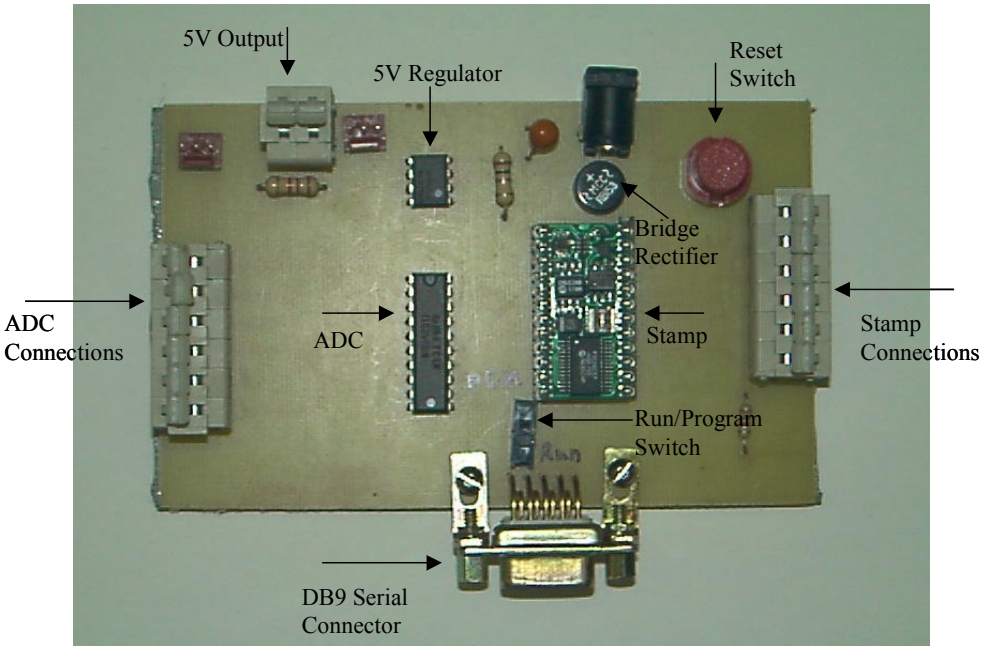

Figure 3. Layout of the micro-controller board with the Basic Stamp II [9]

Project Assignments

Project-related assignments are made in the form of four project milestones as well as the final project due at the end of the semester. To help prepare the students for the project milestones, 4 self-contained homework assignments are also given, interspersed with the project milestones throughout the semester. Not all of the project milestones and homework assignments are directly related to use of the smart home testbed; however, the full sequence culminates in the completed building automation project. Many of the assignments utilize the CASE tool Rational Rose[11] and the Unified Modeling Language[12] (UML) to incorporate modern practices in software engineering. The first homework assignment is completed by each individual student; all remaining homework assignments and milestones are done within the team setting. The sequence is summarized below:

Homework #1 *Introduction to use case diagrams and Rational Rose*
Each student identifies the functional requirements of a system and documents them with a narrative description and a use case diagram.

Milestone #1 **Requirements elicitation and project scope**
Students identify a person that can act as a client (*e.g.,* a relative or a friend) and meet with the client to discuss the system requirements. The requirements and project scope are documented with a narrative, use cases, and a use case diagram.

Homework #2 *Simulator of a heating, ventilation and air conditioning (HVAC) system*
Students are given source code for a simulation of a heating system. They must first reverse engineer the java code using Rational Rose, expand the program to include the full functionality of an automatic HVAC system, reverse engineer the modified code and finally demo it to a TA.

Milestone #2 **Analysis models**
Analysis models are created for the system, in the form of analysis class diagrams, collaboration diagrams, sequence diagrams, and UML statecharts.

Homework #3 *Integrating the HVAC functions with the micro-controller board*
Students modify their program from Homework #2, to communicate with the micro-controller board. The actual temperature is read from the sensor. Heating functions require turning on the heater, and cooling requires turning on the fan until the desired temperature is reached. Again, students reverse engineer their code and demo the program to a TA.

Milestone #3 **User interface prototype**
Students create a prototype of the user interface and demo it to their client, soliciting comments. Students also demo the interface to a TA and must discuss how their interface reflects the client's needs and what changes will be made to address the client's comments.

Homework #4 *Integration with a database*
Students create a database using, *e.g.*, Microsoft Access, for storing the X10 addresses of devices. The students demo a program that allows the user to select a device and then retrieves the X10 address from the database before sending the command.

Milestone #4 **Design models**
Students refine the analysis models as necessary and produce a design for the their system in the form of a subsystem design and UML design class diagrams.

Final project The completed system is "delivered" with a presentation, a live demo, and a written report containing the contents of the previous milestones as well as a testing plan and a reverse engineered model of the final code.

Grading

Grading for the software engineering course as described above is weighted heavily on the semester project. Project milestones comprise 25% of the semester grade and the final project accounts for 30% of the grade. Two small tests together also account for 30% of the grade. The homework assignments make up only 10% of the semester grade. A 5% class participation score is also included, based on in-class exercises and project reviews written on

other teams' projects. To encourage an even distribution of the workload among all students in a team, each student completes a peer evaluation of his teammates at the end of the semester. Project grades of individual students are adjusted if the peer evaluation shows a lack of participation.

Evaluation

Evaluation of the project testbed and exercises is ongoing; we offer a few comments and observations at this time. Regarding the students, there are two main questions that we would like to answer:

1. Do the project assignments engage the students in the learning process?
2. Are the students actually learning anything, both in terms of software engineering concepts and also event-based programming concepts?

To answer the first question, we consider observations over four semesters of teaching the software engineering course and incorporating the smart home project theme. For two of those semesters, no hardware was used and the students produced software simulations only. In fact, several students asked how they could incorporate real hardware into their systems; at the time, we did not have a feasible answer. The projects produced during these first two semesters were for the most part, bare minimum projects. That is, for the majority of the teams, the students produced the minimum that was required.

In the third semester, we introduced some aspects of the project testbed but were still adjusting the interface to the testbed and the content of the exercises. Also, the class size increased from about 50 students to over 70 students. Some student groups, but not all, became obviously more engaged, as the actual hardware was introduced.

In the fourth semester, the testbed and the exercises were complete, and the students had the benefit of the testbed from the beginning of the semester. In this semester, almost all of the teams produced more than the bare minimum required. Significantly more features were included in the final systems, and more sensors and devices were incorporated into the live demos. From this observation, we might conclude that the students were indeed more engaged in the learning process. This was the conclusion of everyone involved directly in the course (the instructor and the TA's) just by observing the students over the length of the semesters. Certainly the students as a whole spent more time in the lab working on their projects in the fourth semester.

To answer the second question objectively will require a more intensive look and comparison in the exams and final projects produced by the students over several semesters. However, our initial reaction is that, if indeed the students were more engaged in the learning process, they most likely did learn more as a result of their efforts.

Concluding Remarks

In summary, we developed a new project testbed for an existing software engineering course, to emphasize and illustrate the principles of both software engineering and event-driven

programming. The testbed, which consists of both hardware and software components, follows the theme of the smart home/smart workplace and was introduced to provide an interesting and engaging platform in which to pursue the educational objectives and explore the entire development lifecycle of event-driven systems. The project exercises developed thus far show potential in achieving the educational objectives. However, there is further potential in which to utilize the project theme and testbed that we have not yet tapped. With a diverse and interesting set of external entities, students can observe the effects of changes in the requirements (a critical aspect of software engineering). Also, using the testbed and project exercises, the instructor can introduce real-world problems in concurrency and uncertainty and illustrate the effects of dynamic external entities.

Acknowledgements
The authors would like to acknowledge the contribution of the students in designing and building the project testbed: graduate students Jeff Brooks and Ritesh Patel and undergraduates Pete Parker and Laura Heffernan.

References

1. Tennenhouse, D. (2000). "Proactive Computing", *Communications of the ACM*, 43(5):43-50.
2. Stein, L.A. (1998). "What We Swept Under the Rug: Radically Rethinking CS1", *Computer Science Education*, 8(2):118-129.
3. Lethbridge, T.C. (2000). "What Knowledge Is Important to a Software Professional?", *Computer*, 33(5):44-50.
4. Li, Kin (2000). SENG 440: Software Models for Embedded Systems, http://www.engr.uvic.ca/~seng440/
5. Stone, D., S. Bergmann, G. Baliga & A.M. Berman, "A CS1 Maze Lab, Using Joysticks and MIPPETs", *Proceedings of the 30th SIGCSE*, March, 1999, New Orleans, LA, also, http://www.rowan.edu/mars/compsci/CS1labs/CS1labs.htm
6. http://www.smarthome.com/aboutx10.html
7. http://www.smarthome.com
8. X10 (USA) Inc., Closter, NJ.
9. Parallax, Inc., Rocklin, CA. http://www.parallaxinc.com/
10. Course web page for Software Engineering at the University of Missouri-Columbia http://www.cecs.missouri.edu/~skubic/332/
11. Rational Software Corporation, Cupertino, CA. http://www.rational.com
12. Booch, G., Rumbaugh, J. and Jacobson, I. (1999). *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, MA.

MARJORIE SKUBIC
Dr. Skubic is an Assistant Professor in the Computer Engineering and Computer Science Department at the University of Missouri-Columbia. She received the Ph.D. in Computer Science from Texas A&M University in 1997. Prior to graduate school, she worked in industry for 14 years as a software engineer, specializing in real-time systems. Current research interests include sensory perception and robotics, especially human-robot interaction.

JAMES LAFFEY
Dr. James Laffey is an Associate Professor at the University of Missouri, Columbia. He is Director of the MU Center for Technology Innovations in Education and has been the principal investigator for several NSF awards. His teaching and research interests are in developing technology innovations to facilitate learning and support performance.