Event-driven Programming can be Simple Enough for CS 1

Kim B. Bruce
kim@cs.williams.edu

Andrea P. Danyluk
andrea@cs.williams.edu

Thomas P. Murtagh
tom@cs.williams.edu

Department of Computer Science
Williams College
Williamstown, MA 01267, USA

Abstract

We have recently designed a CS 1 course that integrates event-driven programming from the very start. Our experience teaching this course runs counter to the prevailing sense that these techniques would add complexity to the content of CS 1. Instead, we found that they were simple to present and that they also simplified the presentation of other material in the course. In this paper, we explain the approach we used to introduce event-driven methods and discuss the factors underlying our success.

1 Introduction

We recently implemented a major update of our CS 1 course. The new version of the course is based on Java (replacing Pascal), takes an objects-first approach, uses labs incorporating graphics, and introduces event-driven programming techniques including threads quite early in the course.

Our decision to introduce our students to GUI programming and to use event-driven techniques was influenced by two arguments. The first, expressed by Culwin and others [2, 8, 10], is that "our current students need to be prepared for practicing their skills in the context of software development in the twenty first century." [2] Teaching them techniques appropriate for teletype terminals and card readers not only fails to prepare them with the skills required to understand and construct modern systems, but may burden them with techniques that later need to be unlearned.

Others have cited student motivation as their incentive for incorporating elements of GUI programming and event-driven techniques in introductory courses [3,5]. Students accustomed to working with programs with flexible, graphical interfaces have little interest in laboratory assignments based on line by line text input and output.

There are also those opposed to this approach. Several authors have expressed concern that event-driven techniques are either beyond the grasp of students in introductory courses or would require so much additional class time that other, more fundamental topics would be displaced. Wolz and Koffman [4] have developed their "simpleIO" library to enable students to construct applications with the feel of a GUI but "without the complexity of non-sequential models." Reges advocates a "conservatively radical" approach in which the instructor provides students with implementations of the GUI components of an application and then asks the students to construct classes that interact with the instructor's [6].

Even some that advocate the introduction of event-driven techniques, do so with the assumption that these techniques are advanced. Although Stein seems to be quite radical in her call to introduce students to a computing model based on interacting communities of processes, event-driven techniques do not appear until relatively late in her course. In one draft of her text, event-driven programming is not covered until chapter 15.

It is reasonable to be concerned that event-driven programming is too complicated to be presented in an introductory course. We debated the issue extensively before deciding to incorporate the event-driven approach from the very beginning of our course. Since then, experience offering the course has convinced us that event-driven programming techniques can be presented to students in a way that makes them intuitive. Furthermore, we discovered that the use of an event-driven model facilitates the clear presentation of several other, more standard topics.

2 The Evidence

From the very first lab in our course, we have students construct programs in a framework based on the Java Applet class. Their first programs are defined as extensions of a WindowController class from our library that itself extends the Java Applet class. Our WindowController creates a specialized Canvas in which the students can place graphical objects, and it allows

them to handle mouse events by defining event-handling methods with names like onMouseMove and onMouseDown. These methods are very similar to the mouse event-handling methods of the Java AWT. The main difference is that they expect simpler parameters. Most of our event-handling methods receive a single parameter describing the coordinates where the mouse event occurred, rather than a more complex "Event" object. Our library also provides students with a set of primitives to simplify the display of graphics. The graphic features of our library are similar to those described by Roberts and Picard [7].

In the first lab, students use this environment to construct a simple drawing that changes in response to mouse clicks. By the second lab, they have learned conditionals, allowing them to construct a program that lets users drag objects around the screen to implement something similar to a shape-sorter. Next, they learn to write programs involving several classes. For the third lab, they again write a program that allows users to drag objects around the screen, but this time there are several objects and they interact like magnets. We then introduce loops and a simple form of threads which they use in lab to create an animated game. After about 6 weeks, we can move beyond the simplified interface provided by our library and explain standard Java event-driven programming mechanisms. At this point, students also learn how to program with buttons, choice items, scrollbars (sliders), and text fields.

The ease with which our students learned to program using event-driven techniques surprised even the most optimistic of us. Students with no previous programming experience found the model completely natural. Those with previous experience had no problem adapting to the approach. In both cases the change had consequences that made other components of the process of learning to program easier. Most strikingly, the event-driven style nearly eliminated the tendency novice programmers have to construct long, complex procedure bodies, and the agony students suffer trying to debug such complex code.

At the end of each semester, students are asked to complete a standardized evaluation form that includes two questions relating to the difficulty of the course. The questions ask the students to rate “the amount of work expected” and “the level of difficulty.” Student responses to both questions after the first offering of the new course were 3.7 on a scale from 1 to 5. These responses fall at the 75th percentile among the workload/difficulty rankings for all courses in the college. More importantly, these rankings were similar to those of a recent offering of the previous course taught by the same instructor. For that class, the ratings were both 3.6 out of 5.

In addition to using event-driven techniques in our CS 1 course, we have introduced this programming model in an introductory course we offer for non-majors. Many of the students enrolled in our non-majors course have technical skills considerably weaker than those who take CS 1. Nevertheless, the use of an event-driven model has also been quite successful in this non-majors course. Since there was no previous version of the non-majors course, we can not offer any quantitative comparison of student perceptions of difficulty.

Between our CS 1 course and our non-majors course, we have observed the reactions of over 200 students to an introduction to programming based on an event-driven model. This experience has strongly confirmed our hope that this model could provide a simpler rather than a more complex approach to programming for the novice.

3 Some Explanations

Our experience has been so contrary to the expectations of others that it is appropriate to explore this contrast in more detail. We believe that two factors are responsible:

(1) We and others have tended to overestimate the complexity of the event-driven model;

(2) The actual support current systems provide for event-driven programming is more complex than necessary.

We suspect that the reason many overestimate the difficulty of event handling is that they themselves did not encounter the subject until they were already experienced programmers. It is only natural to assume that something one learns late in one's training must be advanced.

In reality, the absence of event-driven techniques in the training of current instructors was not related to their complexity. It resulted from the fact that situations where event-driven techniques are appropriate, such as programs based on interactive GUIs, were not common when most current instructors learned to program. In today’s world full of interactive programs, it is appropriate to expose students to the programming techniques that best suit the construction of such programs.

We must remember that event-driven techniques were not devised to make programming harder. They were developed because they can actually simplify both the details and the underlying conceptual model required to construct interactive programs.

To make this concrete, imagine a simple, interactive program that is supposed to merely draw a dot at the current mouse position each time the mouse is clicked. In event-driven form using our library, the solution to this problem involves a single, short method that is invoked by the system each time the mouse is clicked. The body of the method contains just the code to draw the dot using the method's parameters to determine the mouse position. The complete Java code required for this program in our environment is shown below:

```
class DrawDots extends WindowController {
    public void onMouseClick( Coords mouseLocation ) {
        new FilledOval( mouseLocation, 1, 1, canvas);
    }
}
```

A more traditional solution to this problem would require a loop that would be executed once per click. Its body would consist of nested loops that repeatedly do nothing until the mouse is depressed and released followed by code to draw the dot. The code for this approach, appropriate for the environment we used in the previous version of our CS 1 course, Pascal and Quickdraw, is shown below. The inner loops could be replaced by a call to an "awaitClick" procedure with an appropriate library.

```
program drawdots;
   var mouseLocation: point;
begin
 while true do
 begin
    repeat until button;
    repeat until not button;
    GetMouse( mouseLocation );
    PaintOval(mouseLocation.x, mouseLocation.y, 1, 1)
 end;
end.
```

If you run a program like this and ask students with no programming experience what the computer is doing before you press the mouse, they will give a fairly obvious answer: "Nothing." The event-driven solution mirrors the students' natural perception of what the computer does while it is waiting. Doing nothing requires no code. In the traditional version, however, one must write explicit code to tell the machine to do nothing.

In past semesters, situations like these have been stumbling blocks for our students. We have had to explicitly teach them to think like computers and take the view that nothing must be something. With hindsight, we feel it is clearly more appropriate to adopt programming tools that reflect natural human models of the tasks undertaken than to adjust the thinking patterns of our students to fit the limitations of the programming techniques we present.

In many ways, the argument over whether event-driven techniques are inherently complex is similar to the argument whether recursive or iterative programming techniques are simpler. Typically, those who are taught iteration before recursion find recursion mystifying at first. Once recursion is understood, however, it becomes obvious that there are problems for which the recursive solution is far more clear than any iterative solution.

There is an additional, more substantial reason to doubt the feasibility of presenting event-driven techniques in an introductory course. Mechanisms for event-driven programming in contemporary programming systems are not designed for beginners. As a reminder, here is what is necessary to do event-driven programming in standard Java:

-Define a class of objects that can act as listeners,

- Declare that the listener class implements the appropriate listener interface,

-Define the method(s) that the listener interface requires,

-Create one or more instances of the listener, and

- Pass an instance of the listener class as a parameter to the appropriate component's "addListener" method.

With all of this required, it is no surprise that instructors think that event-driven programming is too difficult for novices. If mechanisms for event-driven programming were designed for beginners, simplicity would be a design goal. Instead, the design goal is to provide the flexibility required by expert programmers.

When programming with our library, a student is provided with facilities that simplify the details of handling mouse events. Student programs are extensions of a class that catches mouse events and invokes simple methods students can override to handle these events. In the first weeks of our course, in fact, our students believe the only purpose for method declarations is to associate event-handling code with particular events. It seems perfectly natural to them that if you define a method named "onMouseClick" then the instructions inside its body will be carried out when the mouse is clicked. Most of these methods take a single parameter describing the point where the mouse event occurred. This parameter gives students easy access to just enough information about the events to construct interesting programs.

This "training wheels" version of event handling is necessary for the first few weeks of the course. Before long, however, we are able to expose students to most of the features of the underlying Java event-handling model.

4 Advantages of "Events-first"

Those who argue against including event-driven techniques in the first course frequently express concern that this new topic will displace other, more important topics. We had a very different experience. Introducing the event-driven model very early actually made it easier to teach other critical topics.

One of our goals was to teach students basics of object-oriented design. One can introduce Java programming by having students construct simple applications. In this case, students begin by placing all their code in a method named main. As the complexity of the code increases, students are taught to procedurally decompose their code into small, logically structured private methods. The problem with this approach is that in following it, one is teaching a procedural rather than an object-oriented approach. Students are led to think of methods as a mechanism to decompose larger tasks rather than as the means to describe the possible behaviors of an object.

In our course, our students never imagined the idea of a large main program. From the start, their programs were actually class definitions that consisted of lists of short

method definitions. At first, they thought of each method definition as an event handler. They were not conscious of the possibility of generalizing the use of the class construct to define new objects. They did, however, view methods as a means of describing how a single object (their program) should respond to outside stimuli. This view remained clear to them throughout the course. After years of pleading with students writing in Pascal to decompose their code into shorter procedures, it was shocking to find that this was simply no longer an issue.

In a paper describing experience teaching a course based on Stein's work, Weber-Wulff discusses the frustration of attempting to teach good design habits to students who enter a CS 1 course with what they view as considerable experience [9]. In our course, such students were thrilled by the power the event-handling model gave them to create interactive programs. They never even noticed that they were simultaneously being led to use good programming style. It was as if we offered them the tightest straight jacket we could find and they responded by happily asking us to help them put it on.

The definition of event-handling methods also has the advantage that the introduction of formal parameters is separated from the introduction of actual parameters. Students use formal parameters when defining event-handling methods, but don't have to worry about where the actual values come from. They accept the idea that the system somehow provides this information to their methods. At the same time, they are actively using actual parameters when invoking the constructors and methods of our graphics library. By the time we introduced the definition of new classes of objects, students are comfortable with both notions and are well prepared to deal with the idea that a formal parameter can refer to an actual parameter from another part of their program.

In conjunction with the use of objects through our graphics library, the event-handling style of programming provided an excellent way to prepare students for the introduction of classes. By the time we began asking students to define their own classes, they had already used all of the required language mechanisms. Instead of explaining parameter passing or class syntax, we could focus on the role of objects.

5 Comparison with previous work

Others emphasizing the advantages of event-driven programming include Conner et al [1] and Stein[8]. Conner et al advocate an object-first approach that also takes advantage of event-driven programming, though they suggest an event-driven model closer to that of Java 1.0. Behavior is associated with objects by defining a subclass of the GUI component. It would seem to be a large step to move from this model to the standard Java event model. In particular, this approach to event-driven programming unnecessarily mixes GUI appearance and behavior with the application's structure. Their event-handling techniques also require familiarity with more of Java's object-oriented features, and therefore they introduce these techniques slightly later than we do.

Stein[8] introduces event-driven programming late in her course, and moreover seems more interested in explaining the event queue and loop rather than introducing event-driven programming as a primitive notion.

Our experience has convinced us that early use of event-driven programming as a primitive notion provides an effective introduction to programming for novices.

References

[1] Conner D B, Niguidula D, & van Damm A, Object-Oriented Programming: Getting it Right at the Start, *Educator's Symp at the 9th Annual Conf. on Object Oriented Programming Languages, Systems, and Applications*, Oct. 1994, Portland, OR.

[2] Culwin F, Object Imperatives, *Proc. of the 30th SIGCSE Technical Symp. on Computer Science Education*, Mar. 1999, New Orleans, LA, pp. 31-36.

[3] Jimenez-Peris R, Khuri S & Patino-Martinez M, Adding Breadth to CS1 and CS2 Courses through Visual and Interactive Programming Projects, *Proc. of the 30th SIGCSE Tech. Symp. on Computer Science Education*, Mar. 1999, New Orleans, LA, pp. 252-256.

[4] Wolz U & Koffman E, simpleIO: A Java Package for Novice Interactive and Graphics Programming, *Proc. of the 4th Annual SIGCSE/SIGCUE Conf. on Innovation and Technology in Computer Science Education*, June 1999, Cracow, Poland, pp. 139-142.

[5] Mutchler D & Laxer C, Using Multimedia and GUI Programming in CS 1, *Proc. of the SIGCSE/SIGCUE Conf. on Integrating Technology in Computer Science Education*, 1996, Barcelona, Spain, pp. 63-65.

[6] Reges S, Conservatively Radical Java in CS1, *Proc. of the 31st SIGCSE Technical Symp. on Computer Science Education*, Mar. 2000, Austin, TX, pp. 85-89.

[7] Roberts E & Picard, Designing a Java Graphics Library for CS1, *Proc. of the 3rd Annual SIGCSE/SIGCUE Conf. on Integrating Technology into Computer Science Education*, August 1998, Dublin, Ireland, pp. 213-218.

[8] Stein L A, Beyond Objects, *Educator's Symp at the 12th Annual Conf. on Object Oriented Programming Languages, Systems, and Applications*, Atlanta, GA, October, 1997.

[9] Weber-Wulff D, Combating the Code Warrior: A Different Sort of Programming Instruction, *Proc. of the 5th Annual SIGCSE/SIGCUE Conf. on Innovation and Technology in Computer Science Education*, July, 2000, Helsinki, Finland, pp. 85-88.

[10] Woodworth P & Dann W, Integrating Console and Event-Driven Models in CS1, *Proc. of the 30th SIGCSE Technical Symp. on Computer Science Education*, Mar. 1999, New Orleans, LA, pp. 132-135.