Frameworks in CS1 – a Different Way of Introducing Event-driven Programming

Henrik Bærbak Christensen
Department of Computer Science
University of Aarhus
8200 Aarhus N, Denmark
hbc@daimi.au.dk

Michael E. Caspersen
Department of Computer Science
University of Aarhus
8200 Aarhus N, Denmark
mec@daimi.au.dk

ABSTRACT

In this paper we argue that introducing *object-oriented frameworks* as subject already in the CS1 curriculum is important if we are to train the programmers of tomorrow to become just as much *software reusers* as *software producers*. We present a simple, graphical, framework that we have successfully used to introduce the principles of object-oriented frameworks to students at the introductory programming level. Our framework, while simple, introduces central abstractions such as *inversion of control*, *event-driven programming*, and *variability points/hot-spots*. This has provided a good starting point for introducing graphical user interface frameworks such as Java Swing and AWT as the students are not overwhelmed by all the details of such frameworks right away but given a conceptual road-map and practical experience that allow them to cope with the complexity.

Categories and Subject Descriptors

D.1.5 [**Programming Techniques**]: Object-oriented Programming; D.2 [**Software**]: Software Engineering; D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.13 [**Software Engineering**]: Reusable Software; I.3 [**Computing Methodologies**]: Computer Graphics; K.3 [**Computing Milieux**]: Computers and Education

General Terms

Design, Human Factors

Keywords

CS 1 Curriculum, Event-driven Programming, Frameworks

1. INTRODUCTION

We are presently teaching a CS1 course with an objects-first approach using Java. The curriculum covers four central subjects:

- Jump start: classes, objects, methods, control flow, parameterization
- Basic object-oriented programming: state, behavior, information hiding, modeling, UML to Java
- Algorithmic patterns: sweep, loop invariants, searching, merging, divide-and-conquer
- Advanced object-oriented programming: polymorphism, interfaces, specifications, design-by-contract, class invariants, frameworks, GUI-programming

As part of the advanced object-oriented programming subject, we teach the principles of programming using *frameworks* (not building them) and, based upon this treatment, graphical user interface frameworks. While the subject of frameworks is considered an advanced and complex topic, we have decided to include it in the CS1 curriculum nevertheless for several reasons.

The most important reason is the realization that the programming context most programmers are facing today is radically different from the one that existed, say, 10-15 years ago. Few modern programs (or "systems") are monolithic entities; instead they draw upon functionality from many different sources: third party components, object-oriented frameworks, dynamic link libraries, etc. Thus programming today is more a matter of "gluing" application code together with one or several frameworks and components than to be able to write a large monolithic program that does it all by itself. In this changed context we find it natural that students at an early stage are exposed to sound principles for making their code cooperate with third party code, and we find object-oriented frameworks to be a good vehicle for demonstrating these principles.

Another reason is pinpointed by Culwin [6]—today's students are confronted exclusively with graphical user interfaces as the interface to programs. To demonstrate programming only using text IO is not very motivating.

We also strongly think that we as teachers have a prime responsibility to educate *software reusers* just as much as *software producers*. Systematic software reuse [8, 10] is at the moment our best cure against the "software crisis". Forcing students to program using frameworks is the right step towards producing software reusers—and the result of their efforts look much more professional than mere sequences of, say, prime numbers in a text box.

In this paper, we describe the introductory framework and the exercises associated with it. We then discuss how the terminology introduced by the simple framework is used in our introduction to Java AWT as representative of modern object-oriented graphical user interface frameworks. Finally we summarize and discuss some of our experiences.

2. A TWO-STEP LEARNING PROCESS

We have adopted a *two-step* learning process for introducing graphical user interface frameworks in order to lessen the learning curve [3].

In the first step we teach the students a basic understanding of the principles underlying frameworks, using a concrete, simple, yet flexible, framework example. The example has nevertheless the fundamental characteristics of a framework:

- *Inversion of control*: The framework defines the control flow and collaboration patterns of the objects in the final application, instead of the usual "driver" program that the students write themselves.

- *Hotspots* [11]: The provided framework is abstract and needs to be specialized to the particular domain of the final application. Abstract classes that must be subclassed define the hotspots of our concrete framework.

In the second step we introduce Java AWT to the students through the context and terminology introduced by the first step—what are the hotspots of AWT and how do we tailor them to our needs? The main point is that AWT/Swing is large and complicated and thus confusing to the beginner, and you simply must master the underlying concepts and principles in order not to be overwhelmed by the sheer number of classes and methods.

3. FIRST STEP: PRESENTER FRAMEWORK

Our requirements of the framework were the following aspects: It should illustrate the basic principles of frameworks (inversion of control and hotspots); it should be simple for students to use; it should be flexible in the sense that a number of sensible instantiations should be possible; it should be fun, challenging, and visual.

The result is a *presenter framework*. The presenter framework facilitates construction of multi-media presentations of a domain where the compass-directions are a suitable metaphor for user navigation. (So far "multi-media" is limited to images and text but it is straightforward to extend it to movies and sound.)

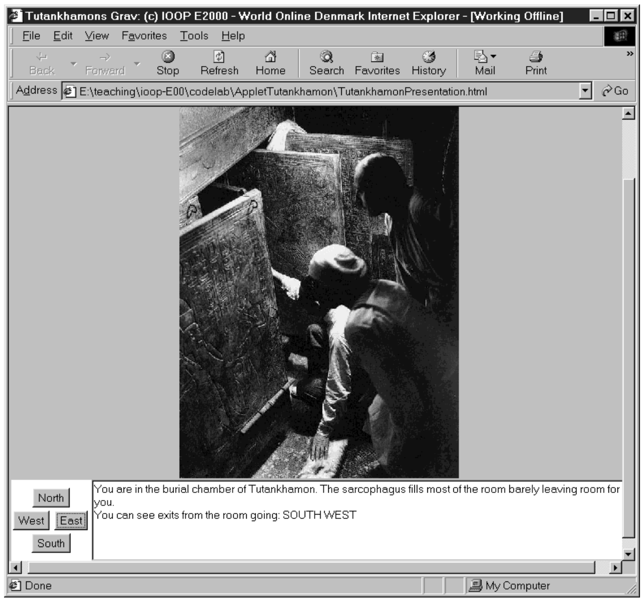
In our *first step* lecture we introduce the presenter framework through a specific instantiation, namely a multi-media presentation of the tomb of Tutankhamen, the pharaoh whose tomb was miraculously found rather intact in 1922 by Howard Carter [4].

In fig. 1 is shown a screen snapshot of the Tutankhamen tomb presentation. The presenter framework is an applet thus the presentation and later the student exercises can be run in a web browser.

Using the four buttons marked with the compass directions the user can navigate around the chambers of the tomb. In each chamber the user is presented with a picture taken during the original opening of the tomb along with some explanatory text.

It is our experience that the concrete instantiation—moving around a tomb with pictures from the original opening—grabs the imagination of the students.

The Tutankhamen's tomb instantiation also allows us to underline an important software engineering principle, namely separating model/domain code and user interaction code. We build a small object-oriented model of the domain with classes: *chamber* (having exits, an image and a description) and *visitor* (having an association with a specific chamber and a `move` method). As the user interaction code is completely defined by the framework, it is simply impossible for the students to mix UI and model code except through the well-defined hotspots provided by the framework.


Tutankhamons Grav: (c) IOOP E2000 - World Online Denmark Internet Explorer - [Working Offline]
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address E:\teaching\ioop-E00\codelab\AppletTutankhamon\TutankhamonPresentation.html
Go
North
West
East
South
Done
My Computer

You are in the burial chamber of Tutankhamon. The sarcophagus fills most of the room barely leaving room for you.
You can see exits from the room going: SOUTH WEST

Figure 1: The presenter framework instantiated to present Tutankhamen's tomb.

3.1 Design

The presenter framework provides the application programmer with a simple interface (in practice the interface is split into two, as described in the next section):

```
public abstract class ImagePresenter
{
 public void showImage(String filename)
 {...}
 public void showText(String text) {...}

 public abstract void northButtonPressed();
 public abstract void eastButtonPressed();
 public abstract void southButtonPressed();
 public abstract void westButtonPressed();
}
```

An instance of `ImagePresenter` is an applet that provides the graphical user interface: a large area for displaying images, a smaller one for displaying text, and the four compass direction buttons that respond to user clicks.

The `showImage` and `showText` methods are methods that provide services for the application programmer (the students are well versed in object oriented thinking at this point in the course).

Thus, to instantiate the tomb presentation is a matter of overriding the `..ButtonPressed()` methods as e.g. in:

```
public void northButtonPressed() {
  visitor.move(NORTH);
}
```

where the move method of visitor must test for an exit leading north and invoke the `showImage` and `showText` methods with appropriate parameters.

The new technique the students must adopt is that in order to provide application specific functionality that reacts on user interaction, they have to subclass the abstract `ImagePresenter` to define the actions to perform when the user presses the buttons on the user interface. This raises discussions on the central points in frameworks as outlined below.

3.2 Inversion of control

In their previous programming experience from example code and exercises, there are always a number of interacting objects and a single ‘driver’ that does the setup and defines the main control flow. Now the control flow is dictated and controlled by the presenter framework instead. The application code comes into play only when the overridden `...ButtonPressed()` methods are called. This is a simple variant of event-driven programming and illustrates the inversion of control principle.

3.3 Hotspots

Frameworks define core functionality, control flow and object collaboration patterns. Application programmers refine frameworks to specific domains by adding code at well-defined points denoted hotspots (also called hooks or variability points). Hotspots can be defined using a number of different techniques: callback methods, objects that implement interfaces, subclassing, etc. We have adopted the subclassing technique as we find it the simplest and as it also demonstrates yet another use of polymorphism and specialization.

4. ELABORATION

We found that the framework could be used in more contexts by introducing a higher level of abstraction: A presenter that does not demand that the central graphical area is an image. Thus we split the framework into providing a `Presenter` class and a more specific subclass `ImagePresenter`, the latter being the one used for the tomb instantiation. The `Presenter` only demands that the graphical centre component is a Java AWT component and provides an abstract factory method [7] for subclasses to define the concrete instance.

Thus, the real framework classes are:

```
public abstract class Presenter
  extends java.applet.Applet
  implements ActionListener
{
 public abstract java.awt.Component
          createCenterComponent();
 public void showText(String text) {...}
 public abstract void northButtonPressed();
 public abstract void eastButtonPressed();
 public abstract void southButtonPressed();
 public abstract void westButtonPressed();
 ...
}
public abstract class ImagePresenter
       extends Presenter
{
 public void showImage(String filename){...}
 public Component createCenterComponent() {
   // return a Canvas instance
   // that can display images
 }
}
```

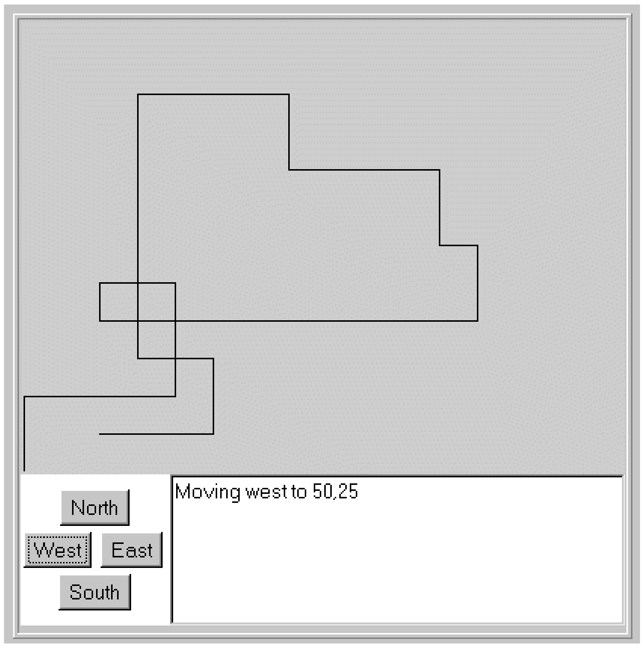
5. STUDENT EXERCISES

Several interesting, yet simple, instantiations can be made from the Presenter and ImagePresenter frameworks.

The first exercise is to make a virtual tour of a museum or gallery; a layout of a number of locations in a gallery is defined and a painting is associated with each location. The buttons can be used to move around the gallery and see the various paintings. This exercise is deliberately similar to the tomb instantiation. In another exercise only the “north” and “south” buttons are used to run through a list of images, essentially making the presenter a slide-show application.

The basic directional navigation metaphor also lends itself naturally to “classic” adventure games. We have an extension of the framework to include the ability to show two scrollable lists of images, one on either side of the center image. The application programmer can then program these so that one list represents an inventory of objects (images) carried by the user and the other list represents an inventory of objects in the visited location. A click-event on an image in a list is a hotspot of the framework that the student can refine to mean that objects are moved between the two inventories.

In other exercises we base ourselves on the *Presenter* class that takes any `java.awt.Component` as center component. Our course uses an object-oriented variant of turtle graphics to introduce people to programming and object-oriented thinking [5]. We therefore ask the students to make a demonstration of the turtle where the turtle moves some distance in the direction corresponding to the compass direction that the user clicks. A snapshot of the turtle instantiation is shown in fig. 2.



Moving west to 50,25

Figure 2: The presenter framework instantiated to demonstrate turtle graphics.

After having introduced the students to AWT, a slightly more advanced exercises in instantiating the presenter framework is to make a 4x4 slide-puzzle by defining a grid of buttons marked with the numbers 1–15 and an empty button denoting the “hole”. The “hole” is then moved by pressing the compass buttons so the user can try and solve the puzzle by arranging the numbers in the right pattern in the grid.

In summary we find that though the provided functionality of the framework is limited and simple, there are a number of intriguing exercises to be made based upon the framework that forces the students to negotiate the basic principles of inversion of control and refining hotspots.

6. SECOND STEP: JAVA AWT

The next step in the learning is introducing a real GUI framework. We restrict ourselves to AWT instead of Swing: the principles are the same but Swing contains even more detail that may blur the picture for the students.

We have many indications that the students are helped by the presenter framework as they learn AWT. They have seen the inversion of control principle; they have seen the principle of refining hotspots and can now concentrate on the particular technique used in AWT for doing this refinement; finally, they are acquainted with the underlying concepts and principles of framework design.

7. EXPERIENCE

At the time of writing our CS1 course has been taught seven times. While we have made many changes in the course material over the years, the `Presenter` framework has been taught with success every time. As the framework has been used ever since we started teaching this course we have no comparative evaluations of the advantages and drawbacks of our approach compared to other ways of introducing graphical user interfaces. However, we have a number of experiences; though they are not rigid scientific evaluations, they do illustrate key aspects of our approach.

First of all the students generally value the approach: The exercises are reported as “fun” and not too hard, the students value the visual appearance of their programs, and most importantly they value that the framework terminology they have learned is used to ease and enhance their understanding of the much more complicated AWT.

At the exams the students demonstrate adequate performance on the topics of frameworks and graphical user interfaces, but of course it is very difficult to measure curriculum quality from exams.

Finally we also find that the impact of teaching frameworks must be measured on a long-term scale. We feel that even to students that “just don’t get it” at this early stage, we have still planted a small but important seed that will ease their learning of framework theory, reuse techniques, design patterns, and software architecture at a later stage.

8. RELATED WORK

To the best of our knowledge our approach is novel and has not been reported elsewhere. However, several authors have reported and discussed approaches for teaching how to program graphical user interfaces at an early point in the CS curriculum. Common to most approaches is the desire to shield students from the underlying complexity (through the use of design patterns such as adapter and wrapper) more than to provide the conceptual tools to understand the complexity.

Woodworth et al. [14] describe how migrating from console- to event-driven models can be eased by introducing a module that acts as an adapter between the event-driven user interface and the domain classes. This way the adapter behaves like the program driver the students are used to from the console driven model. Wolz et al. [13, 12] describe an approach that reduces the complexity of GUI programming by wrapping the underlying user interface toolkit in simpler abstractions.

Bruce et al. [2] describe an interesting approach where event-driven models are introduced right at the start of the course and report their approach to be successful.

Common to most approaches is that the main goal is to teach programming *GUI toolkits* in CS1 and the event-driven model is the obstacle to be handled (by wrapping it, adapting it, or otherwise simplifying it). Our focus is radically different. Our main goal is to teach *frameworks* and a GUI framework is just one type of framework (although an important one). Teaching frameworks is teaching inversion of control and how to refine hotspots, i.e. the event-driven model comes out as a special case of inherent framework behavior.

Buck et al. [3] outline an inside/out pedagogical approach based on Bloom’s taxonomy for cognitive development. We find that our two-step approach is in line with their ideas as the introductory framework has relatively simple building blocks that allow students to comprehend the basic concepts before they are asked to apply them to build GUI interfaces themselves.

Our approach is an instance of the *early bird pedagogical pattern* by Bergin [1]. We find that frameworks, reuse, and reuse techniques are extremely important and must be presented at an early point in the careers of the students and reinforced throughout their studies.

9. SUMMARY

We have described our two-step approach for teaching the principles of object-oriented frameworks. In the first step, we introduce a simple framework that nevertheless has all main features of a full-blown framework. This allows us to concentrate on the main principles underlying frameworks without distracting details. In the second step, we expose the students to the AWT framework but can now draw upon their experiences with the concepts from the much simpler `Presenter` framework.

As outlined earlier, we cannot present rigid evaluations that demonstrate the strength of our approach. We do feel, however, that our argumentation in favour of the approach is strong and valid. The learning curve to climb for the students in order to tackle object-oriented graphical user interface frameworks is a steep one, and breaking it into smaller steps is essential to succeed. Our approach is one of many possible ways of providing such smaller steps but it has some unique benefits. It focuses on fundamental issues in frameworks and reuse techniques instead of concentrating narrowly on event-driven user interfaces. The students are introduced to the principles of object-oriented frameworks. In the `Presenter` framework they are forced to separate domain model code from their user interaction code, which is accepted as a superior architecture for designing interactive applications. They are taught that a programmer of today reuses code provided by others instead of building everything from scratch. Finally they are taught some of the central techniques for integrating reusable code with their own application code laying a strong basis for later courses that teach design patterns and software architecture.

The `Presenter` framework and sample instantiations can be obtained free of charge by contacting one of the authors.

10. REFERENCES

[1] Bergin, J. Fourteen Pedagogical Patterns. http://www.csis.pace.edu/~bergin/PedPat1.3.html.

[2] Bruce, K. B., Danyluk, A. P., and Murtagh, T. P. Event-driven Programming is Simple Enough for CS1. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE’01* (Canterbury, UK, 2001), pp. 1–4.

[3] Buck, D., and Stucki, D. J. Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development. In *Thirty-first SIGCSE Technical Symposium on Computer Science Education* (Austin, Texas, USA, mar 2000), pp. 75–79.

[4] Carter, H., Mace, A., and White, J. M. *The Discovery of the Tomb of Tutankhamen*. Dover Publications, 1985.

[5] Caspersen, M. E., and Christensen, H. B. Here, There and Everywhere — On the Recurring Use of Turtle Graphics in CS1. In *Proceedings of the Fourth Australasian Computing Education Conference, ACE 2000* (Melbourne, Australia, Dec 2000), pp. 34–49.

[6] Culwin, F. Object Imperatives. In Joyce [9], pp. 31–36.

[7] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reuseable Object-Oriented Software*. Addison-Wesley, 1994.

[8] Jacobson, I., Griss, M., and Jonsson, P. *Software Reuse: Architecture Process and Organization for Business Success*. ACM Press, 1997.

[9] Joyce, D., Ed. *Thirtieth SIGCSE Technical Symposium on Computer Science Education* (New Orleans, Louisianna, mar 1999).

[10] Karlsson, E.-A. *Software Reuse – A Holistic Approach*. John Wiley and Sons, 1995.

[11] Pree, W. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, 1995.

[12] Wolz, U., and Koffman, E. simpleIO: a Java package for novice interactive and graphics programming. In *Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education* (Krakow, Poland, jun 1999), pp. 139–142.

[13] Wolz, U., Weisgarber, S., Domen, D., and McAuliffe, M. Teaching introductory programming in the multi-media world. In *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE'96* (Barcelona, Spain, jun 1996), pp. 57–59.

[14] Woodworth, P., and Dann, W. Integrating Console and Event-Driven Models in CS1. In Joyce [9], pp. 132–135.