Role Playing in an Object-Oriented World

Steven K. Andrianoff
David B. Levine
Computer Science Department
St. Bonaventure University
St. Bonaventure, NY 14778
{ska, dlevine}@cs.sbu.edu

Abstract

Role playing exercises are one of many teaching techniques commonly employed to drive home lessons about computer science. Most of the specific role playing described in the literature, however, relates to algorithm or hardware design. More recently, the Pedagogical Patterns Project has published patterns involving role playing in a general sense. In this paper, we draw on three separate role playing exercises that we have developed to show that scripted role playing is a natural and effective way to introduce concepts of object-oriented design.

1 Context

The literature suggests that role playing has been a part of computer science education for at least fifteen years [12]. Anecdotal evidence of role playing abounds and many textbooks suggest role playing exercises as a method of building intuition. Various papers have suggested role playing to teach topics such as hardware design [12], formal methods [8], or backtracking [9]. More generally, it has been suggested that it can help with building intuition regarding understanding of object-oriented design [6, 13].

In general, activities that involve students directly (as opposed to as passive viewers or listeners) result in better retention of material. Indeed several of the patterns identified by the Pedagogical Patterns Project [14] require students to take an active role in creating understanding; Group Card Sorting (#39) and Simulation Game Workshop (#22) are two. When combined with Physical Analogy (#16), they provide students with a way to absorb new concepts within familiar settings. Role playing can combine all of these, and has been described in various guises within this community [2, 3, 11].

Role playing exercises as described in the literature vary in the degree of freedom that they give to their actors. In some cases, the actions to be taken are left deliberately vague – in which case the exercise is also an example of experiential learning; in other cases, the actions are fairly well or even completely specified. In this case, the dramatization provides the essence of learning. It is the latter form of role playing, perhaps better called *scripted role playing*, that we address in this paper.

In the next section of the paper, we discuss some of the key concepts from object-oriented design/programming with a particular eye as to how these concepts might be related to role playing. Immediately after, we describe three separate scenarios for which we have written scripts demonstrating these concepts. The remainder of the paper discusses how these exercises relate to current thinking about pedagogy, the successes we have had, and external factors that influence those successes.

2 Object-orientation and Role Playing

Sebesta [16] says that, "Object oriented methodology begins with data abstraction, which encapsulates processing with data objects and hides access to data, and adds inheritance and dynamic type binding." Implicit in this description is that the objects intercommunicate in manners that are more complex than the traditional function call of imperative programming languages.

Bergin et. al. [3] state that, "In teaching object systems, a good metaphor is human beings. People are autonomous, encapsulated actors, who interact with other similar objects."

We have combined these two notions and created three different scripts for the purpose of introducing concepts of object-oriented programming. The first of these scripts is a whimsical play that can be enacted as early as the first day of an introductory course; it emphasizes the encapsulation

SIGCSE'02, February 27- March 3, 2002, Covington, Kentucky, USA.

and data hiding ideas. The second is a fairly faithful rendition of the current Advanced Placement Computer Science Case Study, the Marine Biology Case Study [1]; it demonstrates intercommunication of objects within a moderately complex system. The third is a simulation of a game of Chips, a Nim-like game; it demonstrates inheritance and run-time polymorphism.

2.1 First Day Role Play

One of the requirements for any exercise undertaken on one of the first days of class is that the students need to be able to begin the exercise without much preparation. The role play that we use on the first or second day of classes has very simple scripts which students can readily master. The goal of this role playing exercise is to demonstrate the basic mechanism of message passing. The instructor plays the role of the main program, asking the students to perform various tasks. One sample student script is shown below:

> You are an **Acrobat**
>
> When you are asked to **Clap**, you will be given a number. Clap your hands that many times.
>
> When you are asked to **KneeBend**, you will be given a number. Stand up and sit down that many times. Note that if you are told "2", then you will stand up twice AND sit down twice
>
> When you are asked to **Count**, you will reply (verbally) with the total number of exercises you have done. Note that Clap-ping four times counts as four exercises, and KneeBend-ing twice counts as two. If you have done these things (and only these things) then your reply should be "6".

The instructor will illustrate message passing by addressing a student who is an Acrobat with, "John, Clap 5" or "Mary, KneeBend 2" or "John, Count". The class can observe the student doing so. Other student actors are asked to perform various tasks. The tasks chosen emphasize the following points:

- Message passing protocols, i.e. naming an object and then giving it a request.
- Parameter passing – receiving zero, one, or more parameters.
- Return values – both void and non-void returns are seen.
- State of an object/hidden data – e.g., an Acrobat's **Count** or a Dice's **NumberOfRolls**.
- The difference between class names and object names (there are multiple instances of the Acrobat and Blackboard objects).
- Overloaded method names – some roles have two tasks with the same name, but different parameter lists.
- Non-existent methods – an Acrobat is asked to do a **TripleBackFlip**.
- Objects communicating with other objects – a LazyCalculator gets a Calculator to do the dirty work when it is asked to **Add**.
- Multiple objects of the same class, e.g. two Acrobats who have independent exercise counts.
- "Improperly" functioning objects – a Bamboozler does not **Subtract** in the "normal" manner.

The entire exercise can be completed in fifteen minutes, leaving plenty of time for either traditional first-day activities (e.g. roster/syllabus) or for a debriefing discussing the role play.

2.2 Marine Biology Case Study

As part of the Advanced Placement Computer Science curriculum, students are required to be familiar with a particular case study. "A case study is a document that includes the statement of a problem, one or more programs that solve the problem and a written description of one expert's path from problem statement to solution program(s)." [7] Currently, the Marine Biology Case Study (MBCS) [1] is used. MBCS uses seven interacting classes and represents a larger program than most students (and some teachers!) have previously encountered. The second role playing exercise is designed to give participants a better understanding of the complex interactions between the various classes. The scripts are much more complex than the First Day Role Play and it takes substantially longer to go through the process. The script for (one of) the Fish object(s) is shown below:

> You are a **Fish**
>
> Before the role play begins, you will be given an ID and a location, remember them.
>
> • When asked for your **ID**, respond by stating the number aloud.
>
> • When asked for your **Location**, respond by stating your location aloud.
>
> • When asked if you **AreDefined**, respond, "true".
>
> • When asked to **ShowMe**, respond with the IDth letter of the alphabet, e.g. if your ID is 3, say "C".
>
> • When asked to **Move**, you will be given an Environment
>
> 1. **Construct** a Neighborhood that will serve as your list of empty neighbors.
> 2. For each of North, South, East, and West (in that order)
>   a. Ask the Environment if that location **IsEmpty**.
>   b. If so, ask the Neighborhood to **Add** it.
> 3. Ask that Neighborhood its **Size**.
> 4. Ask the RandGen to pick a **RandomInteger** from 0 to <The integer from Step 3>-1.

5. Ask the Neighborhood to **Select** passing the integer from Step 4. (You will be given a location in response)
6. Set your location to the one from Step 5.
7. Tell the Environment to **Update** itself based upon you along with your former location.
8. Tell your Neighborhood to destroy itself.
9. Acknowledge that you are done.

In addition to reinforcing the earlier concepts, this exercise emphasizes (or in some cases reemphasizes):

- A much greater level of inter-object communication than the previous role play. In fact, the role of the main program is very minor indeed.
- The difference between class names and object names (there are multiple instances of the Fish objects).
- The difference between object names and internal data that may be unique (such as ID number).
- The hiding of private data.
- The ability of an object to refer to and pass itself as a parameter to other objects (see Step 7 in the **Move** action above).

This exercise takes about fifty to seventy-five minutes to complete, depending upon the preparation of the students and the degree to which "short cuts" are allowed.

2.3 Chips

Chips is a Nim-like game usually featuring two players and a pile of chips. The players take turns removing chips from the pile; whoever removes the last chip wins. The first player may remove any number – but not all – of the chips. Each subsequent player must remove between 1 and twice the number of chips removed by the preceding player.

While there are many possible software designs for this game, the example we show features a ChipsGame class, a TextDisplay class, and an abstract class Player. Different concrete player classes are derived from Player, e.g. OnesPlayer (always removes one chip), RandomPlayer (chooses a random, but legal, move), and HumanPlayer (gets input from the user). Each of the Player sub-classes extends the parent class, keeping the same implementation of the name() method, but supplying different implementations of the getMove() method.

While larger in scope than the First Day Role Play, this exercise is considerably smaller than the MBCS Role Play. Its primary intention is to demonstrate inheritance and polymorphism. A script for the (abstract) Player appears below:

You are a **Player**

- Just before you are **constructed**, you will be "claimed" by some other more specific kind of player. When you are claimed you are to move next to the player that claimed you. The two of you will be partners for the remainder of the role play. If your partner nudges you, take over for him/her and fulfill the request that he/she was given. When you are **constructed**, you will be given your name as a string. Remember it. Acknowledge that you are done.
- When you are asked for your **Name** via a nudge, simply state it aloud.
- If you are ever asked to **GetMove** via a nudge, you will probably be given a ChipsGame. Ignore it. Simply explain that, "I know I should be able to do this, but no one ever taught me how".

and this is the script for the OnesPlayer:

You are a **OnesPlayer**

- When you are **constructed**, you will be given your name as a string. You need to find a (generic) **Player** and "claim" him/her. This generic Player should move next to you. He/She will be your partner for the remainder of the role play. Begin by asking your Player to **construct** him/herself using the name you were given. From this point forward, if you cannot do anything you are asked to, you should ask your Player for help by visibly nudging him/her; DO NOT SAY ANYTHING ALOUD.
- When you are asked to "**GetMove**" you will be given a ChipsGame. Ignore it. Simply respond, "1".

We traditionally run the Chips Role Play twice – once using a OnesPlayer vs. a RandomPlayer and once with a OnesPlayer vs. a HumanPlayer. We use small pile sizes (to create short games) in both instances. Among other points, the exercise emphasizes:

- The notion that a subclass has all of the capabilities of the parent class available to it; even if it chooses not to use them.
- How a method invocation is passed "up the inheritance hierarchy" to the nearest ancestor class that can handle it.
- That when an object is constructed its parent class is also constructed – as part of the construction of the original.
- Run-time binding – until a ChipsGame asks a competitor to GetMove, it is impossible to know which code will execute.
- A Model-View-Controller architecture.
- In an optional scenario, students can see what happens if you instantiate a class that is not

complete, i.e. that still contains some abstract methods.

3 More Than Just Scripts

Socrates said, "The unexamined life is not worth living" [15]. More contemporarily, budding lecturers are often told, "Tell them what you are going to say; then say it; then tell them what you just said." Experience has shown that this advice is particularly apt for the use of scripted role playing exercises in the classroom.

The benefit of each of these exercises is greatly enhanced if the experience is discussed afterwards and referenced frequently as students learn more. We always follow the actual role playing exercises up with a formal debriefing in which the bulleted points above (along with several others) are discussed by the class as a whole. One part of the debriefing includes handing out a "Cast of Characters" so that everyone is aware of who played which role; this greatly enhances the discussion.

Any human modeling of a computer process is bound to be a simplification at some level. These role playing exercises are no exception. One of the main jobs of the debriefing is to discuss these simplifications. (An early version of the MBCS role play attempted to stay more closely aligned with the actual C++ code of the case study. Although the modeling was thus more accurate, the exercise bogged down and participants were more frustrated and learned less than with the current, more simplified version.)

Since most novice students are expected to write code in some language as part of their coursework, one aspect of the discussion concerns the relationship between program entities and role playing entities. (Also, an interesting side discussion arises about the difference between a Human, i.e. flesh and blood, and a HumanPlayer, i.e. a software entity.)

We also explicitly discuss the various methods we used to capture the essence of certain concepts. For instance, we often have students write their private data on the back of the name tags that we issue. In this case, we point out that while the associated actor can see the data, others can only get at it through the public interface. (This is perhaps a less graphic example of the "What Did You Eat For Breakfast?" pedagogical pattern [5].)

Similarly, in the case of inheritance, we point out not only why an object is next to one of its super class (to pass message requests up the hierarchy), but also why this communication does not involve speaking as do other requests (since this action is not a separate method call but rather simply a deferral of responsibility).

In general, we use the debriefing session to set up future discussions and lessons. We have found that as new topics are introduced, students often refer back to the role playing exercises to demonstrate/solidify their understanding. In this sense, one can view the role playing exercise as an example of the Consistent Metaphor pattern [4].

4 Benefits and Reactions

These exercises have been field tested to varying degrees. Most popularly, the MBCS Role Play has been used by at least eight different instructors – either in a class of their own or in a teacher-training workshop. We have actively solicited feedback and have learned (and incorporated) much through this process. Each of the exercises exhibits all of the benefits traditionally ascribed to role playing exercises. More specifically, they serve well to illuminate the concepts discussed above. In the case of the MBCS role play, numerous teachers have written to state that, "[The role play] really enables students to get their minds around the case study".

In general, the role playing exercises (at least the last two) help illustrate the flow of control in a somewhat complex object-oriented system. Even students who are more accustomed to the imperative paradigm obtain a rudimentary understanding of the differences in a short amount of time. Similarly, they illustrate the dynamic behavior of such a system – which is difficult to do in a static forum.

There is a lasting effect of these exercises as well. We have noticed that people who have gone through the exercises tend to discuss general object-oriented design in role playing terms. We recently ran a workshop for teachers, including the three exercises over the first three days. On the fourth day, as they were discussing their own design for a separate (video store) application, the participants started using the language from the role play to illuminate design points. In one case, they referred back to specific portions of one exercise to clarify a point. Throughout the discussion, they used the role playing exercises as touchstones for several different object-oriented concepts. As a result of this (and our other) experience(s), we will be introducing formal role playing exercises into our software design course starting this fall.

Some of the comments from other instructors who have used these particular exercises include:

- It really humanizes the programming.
- Now I understand inheritance.
- Puts [the student] in the mindset of a designer.
- I plan to use your ideas this year.

Other instructors have found that much of the value in the scripts is that they can serve as a base for other modifications – either in the scripts, or in the stage instructions. For instance, one teacher we know displays all of the private data on the blackboard, but is strict about not letting the actors look at it (though the "audience" can). Other teachers have added physical representations of call stacks to them. Still others are experimenting with

different ways to exhibit inheritance in a role playing situation. These modifications can only lead to better teaching.

5 Access

Complete scripts along with the "stage instructions" for each of the three exercises can be found at http://web.sbu.edu/cs/RolePlay.html. We will certainly be developing (and collecting) more of these exercises in the future.

6 Conclusions

Role playing exercises have been used (and suggested) for use in computer science education for quite some time. The emergence of object-oriented design and its emphasis on the interactions of independent actors only increases the appropriateness of this pedagogical technique. In this paper, we have discussed three particular scripted role playing exercises that have been effectively used to introduce the essential concepts of object-oriented programming. The exercises serve as a base which instructors can modify to emphasize whatever points they wish. Modifications of the scripts or the stage instructions can and should be made; any presentation is improved when it is tailored to its audience.

Most important, however, is the idea that role playing exercises such as these are effective in building the mind set needed for learning object-oriented software development.

7 Acknowledgements

We wish to thank Joe Bergin, Nancy Mahosky, and Roselyn Teukolsky for helpful comments on early versions of this paper. In addition, we wish to thank all those who have generously provided us with feedback about the actual scripts that our exercises use; without this feedback, the project would be nowhere nearly as successful as it is.

References

[1] Astrachan, O., Clancy, M., and Matsuoka, C., The Marine Biology Case Study, 2000. College Board. Also available WWW: http://www.collegeboard.org/ap/computer-science/html/case_study.html

[2] Bellin, D. ROLE PLAYING. Online. Internet. [August 22, 2001] Available WWW: http://www-lifia.info.unlp.edu.ar/ppp/pp5.htm

[3] Bergin, J., J. Eckstein, M.L. Manns, and E. Wallingford, Patterns for Gaining Different Perspectives, in *Proceedings of PLoP 2001*, 2001.

[4] Bergin, J. CONSISTENT METAPHOR. Online. Internet. [August 22, 2001] Available WWW: http://wol.pace.edu/~bergin/PedPat1.3.html#consistentmetaphor

[5] Brown, K. WHAT DID YOU EAT FOR BREAKFAST?. Online. Internet. [August 22, 2001] Available WWW: http://www-lifia.info.unlp.edu.ar/ppp/pp12.htm

[6] Cockburn, A., Object-Oriented Analysis and Design, Part I, *C/C+ Users Journal*, Online. Internet. [August 22, 2001] Available WWW: http://www.cuj.com/articles/1998/9805/9805b/9805b.htm

[7] College Board, Advanced Placement Course Description: Computer Science, 2001. Online. Internet. [August 22, 2001] Available WWW: http://cbweb2s.collegeboard.org/ap/pdf/cd_computer_science_02.pdf

[8] Dean, N. and M. Hinchey, Introducing Formal Methods Through Role Playing, *SIGCSE Bulletin*, vol. 27, no. 1, 1995, pp. 302-306.

[9] Dorf, M.L., Backtracking the Rat Way, *SIGCSE Bulletin*, vol. 24, no. 1, 1992, pp. 272-276.

[10] Eckstein, J. INCREMENTAL ROLE PLAYING. Online. Internet. [August 22, 2001] Available WWW: http://www-lifia.info.unlp.edu.ar/ppp/pp7.htm

[11] Fayad, M., D Schmidt, and R. Johnson. Empowering Framework Users. In: Building Application Frameworks: Object-Oriented Foundations of Framework Design. John Wiley & Sons 1999. Chapter 22, p. 505.

[12] Jones, J. Participatory Teaching Methods in Computer Science, *SIGCSE Bulletin*, vol. 19, no. 1, 1987, pp. 155-160.

[13] Levine, D. "Helping Students Through Multiplicities", *The Journal of Computing in Small Colleges*, Vol. 15, No. 5, May 2000, pp. 285-291.

[14] The Pedagogical Patterns Project, Home Page. Online. Internet. [August 22, 2001] Available WWW: http://www.pedagogicalpatterns.org.

[15] Plato, *The Apology* (38a), in Plato – The Collected Dialogs, E. Hamilton and H. Cairns, ed., Pantheon Books, 1963.

[16] Sebesta, Robert. Concepts of Programming Languages, 4th Edition, Addison Wesley, 1999, Chapter 1, p. 23.