MUPPETS: Multi-User Programming Pedagogy for Enhancing Traditional Study: An Environment for both Upper and Lower Division Students

Andrew M. Phelps¹, Christopher A. Egert², and Kevin J. Bierre³

Abstract - This paper discusses the pedagogy and use of MUPPETS (Multi-User Programming Pedagogy for Enhancing Traditional Study) for both introductory and advanced students. MUPPETS is a desktop collaborative virtual environment (CVE) that allows students to learn introductory programming skills through the creation of objects and avatars. The goals of the MUPPETS system are twofold. First, the MUPPETS system provides a complex, interactive, collaborative playground in which introductory students can learn the fundamental principles of objects-first programming, which places emphasis upon encapsulation, inheritance, and polymorphism over traditional constructs such as selection and repetition. Objects created by introductory students can be shared with peers and upper division students. Second, the MUPPETS system provides a mechanism through which upper division students can contribute to the success of introductory students. Upper division students are responsible for the development of MUPPETS extensions as well as complex artifacts intended for use within introductory courses.

Index Terms – Virtual Environments for Computing Education, Introductory Programming Pedagogy, Bridging the Lower Division / Upper Division Divide.

INTRODUCTION

Over the last decade and a half, there has been a fundamental shift in the way instructors teach introductory programming at the college level. Programming, as an area of introductory study, has evolved from imperative languages and techniques to languages and techniques that stress object-oriented design and implementation. The latest endorsement of object-oriented techniques at an introductory level is offered by the ACM/IEEE Computing Curriculum 2001 Computer Science [42] guidelines, which recognizes that an introductory sequence that is object-oriented should begin with fundamental concepts of objects and inheritance and progress later to traditional control structures as long as they serve the purposes of object-oriented design and implementation.

Unfortunately, despite such an endorsement, the actual uses of objects-first techniques in the introductory classroom have been met with limited success. Instructors and researchers have cited several reasons why they believe an objects-first approach is either difficult or infeasible. Concerns cited (but not always agreed with) in the literature include:

- difficulties in the teaching and development of objects-first appropriate course materials [11,19,32].
- the lack of experience by instructors in regards to objects-first techniques [11,32].
- the belief that object-oriented thinking is not natural for the types of problems addressed in the introductory classroom [22].
- that introductory students do not have the appropriate framework to grasp object-oriented programming [28].
- the leap from non-existent program to realized application with independent objects and relationships [19].
- the proper understanding of the term “objects-first” as it applies to programming [23].
- the instruction of imperative techniques must coincide with the development of objects-first techniques, thus rendering it too difficult for students to understand [11,23].
- the addition of complex libraries and graphical interfaces overwhelms the introductory student [9,29].
- that object-oriented techniques are not always reinforced in upper division courses [11].

However, despite these concerns, many practitioners have encountered tremendous success in deploying an objects-first curriculum. Keys to success include:

- an understanding that new students are familiar with gaming systems as computing devices and have difficulty relating to simple examples that support textual interaction [2,16,46].
- the understanding that students should be exposed to design concepts first and coding syntax second [1,46].
- the understanding that students making small changes to complex environments and recognizing those changes might be more effective than a student creating objects and a complete program from scratch [30].

¹ Andrew M. Phelps, Computer Mediated Experience Group, Information Technology Department, Rochester Institute of Technology, amp@it.rit.edu
² Christopher A. Egert, Computer Mediated Experience Group, Information Technology Department, Rochester Institute of Technology, cae@it.rit.edu
³ Kevin J. Bierre, Enterprise Computing Group, Information Technology Department, Rochester Institute of Technology, kjb@it.rit.edu
• a realization that students respond better to immediate feedback as a way to internalize cause and effect when developing programs [2].
• the presentation of object-oriented concepts (encapsulation, inheritance, and polymorphism) early in the student’s introductory programming experience [1,2,10,19,46].
• exposure for students to relationships between classes such as composition, association, and dependency [1,46].
• early introduction of design patterns as a way to discuss design choices in the construction of programs [1, 17,46].
• the understanding that object-oriented design concepts can apply to later courses in the curriculum if given a chance [46].

One way to minimize the negative concerns and maximize the positive observations associated with object-oriented and objects-first programming is to provide students with a rich, interactive, collaborative virtual environment that supports the programming experience. This paper examines the role of the Multi User Programming Pedagogy for Enhancing Traditional Study (MUPPETS), a collaborative virtual environment developed at RIT for the express purpose of supplementing programming education. MUPPETS is currently being used with both introductory students as well as with students further along in their academic careers. This paper will discuss the goals of the MUPPETS project, the impact of MUPPETS on pedagogy, and the use of MUPPETS with both introductory and advanced students.

BACKGROUND

The use of interactive virtual environments for programming is not a new concept. The research literature notes attempts by researchers and instructors to provide languages, libraries and complete environments that act as “cognitive playgrounds” for new learners.

One of the most fundamental ways to provide students with interactive programming experiences is to create programming languages specifically geared to the tasks.

For example, in the Logo programming language [25], learners controlled a visual “turtle” displayed on the screen. The turtle would respond to programming commands such as move forward, rotate, and draw. Students were able to see an immediate effect when changing their code, and quickly learned to appreciate the ability to create functions and use parameters when constructing complex shapes.

Another such attempt was the creation of the Karel the Robot [34] programming language. Based upon Pascal syntax, Karel allowed students to program interactions of an agent within a virtual world. When a Karel program was executed, the programmer would receive immediate visual feedback as to whether Karel could perform the described task or not. In the Karel language, students would control an agent to do tasks such as navigation, obstacle avoidance, and interactions with objects.

Along with programming languages, another technique is to provide students with a rich set of libraries that present students with pre-fabricated units of computation as well as simplify existing libraries for graphics construction.

For example, the “Nice Graphics Package” (NGP) [31], developed at Brown University presents students with a set of drawing tools that can be used “out of the box” with the Java programming language. Without much overhead, students can quickly create primitive graphic shapes, and control them through timers and event handlers. Since NGP complements Java, learning NGP can be used to reinforce object-oriented design techniques and NGP knowledge scales well to the use of traditional graphics libraries in Java such as AWT and Swing.

The third approach is the creation of complete virtual environments in which students add on to a complex, existing structure and observe the effect.

For example, Moose Crossing [39] allows students to add scripting to a text-based collaborative environment in order to create avatars (representations of the self) and artifacts that respond to spoken commands. Students can construct interesting objects, add them to the world, and observe how their presence changes interactions with users in the Moose Crossing community.

CVE-VM [18] allows students to collaboratively create virtual museums and other educational spaces using a combination of VRML 2.0 graphical objects and the Extended Authoring Interface (EAI), an extension that allows VRML and Java to interact.

Alice [8,9,29], a project at CMU, allows students to interact with a wide range of 3D objects as a means to develop programming skills. In Alice, students create programs by using a drag-and-drop visual interface. Students can construct new objects on their own, or transfer objects by swapping model files or by providing downloads over the web. Alice provides a level of abstraction such that introductory programmers are not exposed to the low-level nuances of graphics programming.

THE MUPPETS SYSTEM

As instructors within the Information Technology Department at the Rochester Institute of Technology, the authors of this paper have been moving towards an objects-early and, more recently, an objects-first methodology for the introductory programming courses. To help students overcome challenges attributed to objects-first approaches, the authors have begun to deploy the MUPPETS system.

The MUPPETS system [4,35,36] is a collaborative virtual environment designed to support community-based construction of complex, interactive worlds. The MUPPETS system was designed to match the pedagogical approaches utilized in the deployment of an objects-first course. Students using MUPPETS must quickly learn how to embrace concepts such as encapsulation, inheritance, and class relationships. MUPPETS, as shown in Figure 1, also provides students with advanced code creation tools including a fully functional integrated development environment (IDE) and controls to expose created objects and object definitions to the rest of the MUPPETS community.
MUPPETS is also utilized in upper division courses as a means of bridging the gap between first-year and senior students. The gains for both groups are mutually beneficial. Introductory students have a voice in the overall community and can learn about experiences that await them as they continue their academic careers. Senior students are able to act as role models for introductory students and learn how to effectively communicate with those who have novice levels of computing and programming experiences.

FIGURE 1: THE MUPPETS SYSTEM AND INTEGRATED IDE.

MUPPETS GOALS

The MUPPETS system was designed with a number of goals in mind. First and foremost, the system was designed under an agreement with other faculty that we would not modify the core Java language in any way. Thus, when programming in MUPPETS, the entire standard J2SE is available (and in fact if one so desired, the J2EE package could be incorporated). This is in contrast to most graphical engines available today, which generally use custom scripting languages, or more esoteric languages such as Ruby or Lua [15,45].

Second, the system is designed as a fully featured graphical engine by today’s standards in games and massively-multiplayer environments. It was deemed very important that while the Java to control graphical objects be easily available and usable to first-year students, the core of the graphics system have capabilities extending all the way to cutting-edge effects found in games today. Support for shading languages such as Cg and GLSL are included, and accessible from the Java layer. We are seeking to capitalize on the notions of ‘glory’ found throughout players of online graphical games, and a large portion of that pride of accomplishment comes from the look-and-feel of the reward [3,40,43]. When students create a MUPPETS object, they should enjoy the results of that object graphically, and systems that can only represent a crude graphical representation are deemed, by students, to not be worth the effort.

The third and final goal of the MUPPETS system was that it attracts students at several levels, and fosters communication between them. MUPPETS is designed as a shared virtual world specifically so that students can share their objects with one another across course sections and programming levels. Thus, a senior can create advanced objects that are seen in a freshman world as inspiration and a source of ideas, and this in turn fosters communication amongst the student body. Allowing several forms of interaction and collaboration within the system, while still maintaining the ability for all users to have control of their own work, was a tenet that flows throughout the implementation of the system.

Towards this end, a sub-goal in this area is that the system operates as a help system outside of class. By establishing a world and populating it with students at all levels, in addition to formal tutors and faculty, the system can serve as a secondary help system outside of class. In our existing pedagogy, we observed that students are rarely motivated to seek help outside the classroom, and in particular the barriers to asking for help from anyone other than other students in their section are quite high. By pursuing assignments in a CVE that already has a helpful and knowledgeable community available, the design of the system seeks to reduce the threshold at which students will seek assistance, particularly the divide between lower- and upper-division students.

MUPPETS IMPACT ON THE OBJECTS-FIRST CLASS ROOM

The use of MUPPETS to augment the traditional classroom impacts how an objects-first curriculum is taught, and has stronger repercussions towards students’ attitudes regarding education and programming as a whole. This section will examine MUPPETS ability to 1) support the objects-first’s methodology of defining class and object early, 2) support fundamental principles of encapsulation, inheritance, and polymorphism, 3) support class relationships, 4) support active and constructivist learning, and 5) promote affective learning outcomes along with cognitive learning outcomes.

Differentiation between Class and Object

The process of differentiating classes and objects can be difficult for introductory objects-first programmers. In MUPPETS, the difference between class and object is visually perceptible. As students design their classes, they can iteratively test their creation by creating an instance in a MUPPETS world. The class can be shared with other programmers and the object can be made to persist as interaction continues. The student can modify the class and see changes that occur when the next instance is created. In the MUPPETS case, classes provide the blueprints for creating tangible items in the collaborative world and objects are realized when the class is instantiated. Since feedback is immediate, the programmer can create multiple instances and manipulate each independently and interactively.

Encapsulation, Inheritance, and Polymorphism

Encapsulation, inheritance, and polymorphism are central concepts in MUPPETS. When a programmer or user introduces a MUPPETS object into the world, encapsulation reinforces with users that interaction and control must be performed through the object’s exposed methods as well as

October 19 – 22, 2005, Indianapolis, IN
interfaces provided. This is true for both programs and the interactive control environment alike.

Inheritance is also enforced in the MUPPETS environment. All objects introduced into the collaborative world extend a base-level MUPPETS object class. The inheritance chain in MUPPETS allows us to derive specific functionality for static objects, dynamic objects, and avatars respectively.

Polymorphism is also a powerful concept in MUPPETS. As groups of developers create new objects for use in a shared world, they will employ polymorphism through interface or polymorphism through inheritance as the interaction context of the object changes. Student can use polymorphism to use objects in clever new ways within their MUPPETS creations.

**Support for Class Relationships**

Early on, students are taught the importance of composition, association, and dependency as critical relationships between classes. Traditionally, such ideas are visualized through the use of graphical notation languages such as the UML class diagram. Often, students do not see the impact of such relationships early on when designing and implementing their classes.

In MUPPETS, relationships are critical. In order to build a complex component out of subcomponents, the principle of composition must be taught early on. For example, Figure 2 demonstrates the design of an end user’s avatar as an exercise in composition. It is clearly illustrated that a avatar is made up of other parts, such as a head, body, legs, and arms.

![Figure 2: Student Receives Composition Feedback on the Class Layout of an Avatar.](image)

Association is also a key concept in the MUPPETS system. Upon construction, MUPPETS objects can establish a “knows-of” relationship with each other to perform complex interactions. The interactive feedback provided by MUPPETS clearly demonstrates when a class relationship fails. In the case of composition, body parts do not follow along with the rest of the avatar. With association, interactions with one object will not change the state of another in the scene.

**Support for Active and Constructivist Learning**

Lately, there are a number of computing education papers describing the positive effects of active and constructivist learning in the traditional classroom for both individual and collaborative work [6,7,21,26,37,38]. This movement is categorized by a shift from lecture-style, regulated, “by the numbers” interactions to those where the student learns through investing in projects and activities that have personal meaning [33,37].

The use of MUPPETS as a programming environment helps by supplementing the classroom with an environment that is geared towards the exploration and learning through personal investment. Students can be given exercises that discuss the nature of what to build without specifically describing the content.

For example, one possible MUPPETS assignment is for students to create a simple world that relays a story to the user. The story is presented via interactions with objects throughout the world. In such an example, students often have different visions as to what these worlds should be. Some students choose holiday theme worlds, where others choose historical stories, myths, and science fiction scenes. Anecdotal evidence suggests that these students have much more personal investment in the construction of their introductory projects.

**Affective as well as Cognitive Learning Outcomes**

In the literature for computing education, there is a shift to describe and assess learning outcomes in computing courses effectively. Computing educators are increasingly turning to Bloom’s Taxonomy [5] for the cognitive domain as an effective way to assess learning [24,41,44]. Bloom’s taxonomy classifies cognition into six discreet categories: 1) recall of facts and data, comprehension of information, 3) application of knowledge, 4) analysis of information, 5) synthesis of information, and 6) evaluation. Although our goal is to teach students to utilize all categories of Bloom’s Taxonomy when designing and implementing programs, success at these levels is not always enough to guarantee student interest.

More recently, the literature has been turning towards affective domain outcomes that supplement the cognitive domain objectives [12,13,27]. The affective domain [20] focuses upon the student’s beliefs and internalization process for knowledge. In the affective taxonomy, categories include 1) receiving, 2) responding, 3) valuing, 4) organization, and 5) characterization. Preliminary investigations reveal that affective outcomes may be correlated with the nature or utility of the problem being solved, and the degree of active and
collaborative learning that occurs in the classroom. Denton and McKinney [12] go even further to suggest that affective factors could account for gender effects seen in the classroom.

The MUPPETS system, by nature, supports affective domain goals. By utilizing a complex system from the start, MUPPETS encourages behaviors found at the lower part of the affective scale. Students quickly learn “cause and effect” relationships between their actions, thus increasing interest and creating a causal relationship between value and effort. As the student processes, interactions with the community help to refine the middle affective outcomes. The student’s value system is shaped by interactions between the instructor, fellow students, and advanced students (and even alumni!) alike. The student’s self perception as a valuable and contributing member of a diverse programming community may also impact affective outcomes.

**MUPPETS Introductory Programming Experiences**

In deploying the system at the entry level, we have first experimented by integrating a MUPPETS enabled project in the third course of our first-year programming sequence. The culmination of this course is the final project, the end result of one academic year of study in programming concepts using the Java language. The project is more complex than typical homework assignments and requires planning to successfully complete the project. The typical team size varies from quarter to quarter, but is usually between two and four students.

Prior to implementing the final project, the primary programming experience for students is through homework and class assignments. Because of time limitations, these assignments are generally fairly short, focusing on new material covered during class or in the readings for the week. The final project covers a wider range of material, and requires skills from all three courses in the sequence. The assignment has documentation requirements, and design information is kept in a journal, and there is a post-mortem at the end of the project where this information is shared amongst the class.

Implementations of the final project prior to the MUPPETS based implementation have included client-server chat applications from scratch, simple multi-user games, voting machine simulations, and competitions in IBM’s Robocode 1, a framework for programming virtual tanks that fight each other in an arena. All of these projects shared several commonalities. Each of them provided immediate feedback when testing the application, there was generally some element of cooperation or competition, and the program had to adhere to some standard interface. In addition, each of the projects was large enough that students were forced to plan and cooperate in order to complete the assignment.

Of all of the previous implementations, students seemed to enjoy the Robocode tournament the best. This project worked exceedingly well the first time it was implemented, but unfortunately later implementations suffered, as existing Robocode source was available around the campus. In addition, IBM made a significant amount of code available online, and we started to see tanks that were created by copying code from other sources, rather than through ground up development. As a result, we no longer use Robocode, but as MUPPETS became available the idea of a Robocode style project was revived. There were several reasons for this, chief among them was the fact that in a MUPPETS implementation we have control over the arena, and can thus require new coding styles by adding obstacles or modifying the interface. Control of the arena also meant we could change certain popular tactics that were almost guaranteed to place a student’s tank high in the tournament scoring (“Wall crawling” was a popular tactic and most of the winning tanks from the Robocode project used this approach).

In addition, MUPPETS addresses many of the overall goals of the final project in a more complete way than any of the previous incarnations. Since MUPPETS allows full access to the underlying JDK, we are free to use any portion of it in testing concepts from the class, from threading and networking to GUI development, and everything in between. We were able to point at the entire MUPPETS system as an example of class-based inheritance, and place emphasis on code re-use and object oriented design. We have found thus far MUPPETS succeeds in its goals of supporting team-based work and multi-user programming, as students quickly warmed to the concepts of online interaction and simultaneous development.

In deploying the MUPPETS implementation of the final project, we first tested it with a handful of students to get their reaction. Anecdotal evidence suggests that they prefer this system to any of our previous final projects. Students noted that the graphical system was a key to their enjoyment that they could create objects that “looked like games”. Also noted was the way in which the IDE was directly integrated into the environment, with particular emphasis placed on the color-coding and auto-complete features as being highly desirable. That users did not have to exit the environment to recompile and re-test their work was generally described as a huge improvement, particularly in comparison to the Robocode project. A few students also noted the complete “Javadoc” help system for MUPPETS, and have requested that this be exposed within the world as well (and we are now implementing this feature). In previous presentations, one criticism raised was that our sample materials were too “male-looking” and that an alternate color scheme and default view should be provided so as not to give the impression that the system is only capable of displaying things that “look like a first-person shooter”. This is certainly a very real issue of perception, and one that we will address in our next classroom implementation.

**MUPPETS Advanced Student Experiences**

MUPPETS is also currently in use with upper-division students, namely juniors and seniors in Information Technology and Computer Science. There have been several
projects supported by the system, including experimentation in Virtual Theatre [14], molecular visualization (one of our students will be studying at Brookhaven National Laboratory because of his involvement with MUPPETS), file-system visualization, and game engine design. The system serves as an example graphics engine in our 3D Graphics Programming course, as well as for the Java Graphics portion of our Application Programming course. It also serves as an example of the use of the Java Native Interface in those courses.

In addition to these formal projects and examples, the most important contribution that MUPPETS has had at the upper division is non-formal. Building this system has offered numerous students the opportunity to become engaged in helping to create the system that next-generation students will learn to program with. This has created an almost unbounded sense of accomplishment and pride in the development team, which has manifested itself as a cultural force within the Entertainment Technology Lab where the project was developed. Students working with the development of MUPPETS feel personally invested in the success of lower-division students, and often work with them directly either through the system or through the Entertainment Technology Lab with the system open around them. The formal projects that the system supports has led to numerous research opportunities for upper-division students, in applying the core to various use-cases and scenarios such as those mentioned above.

**MUPPETS EXPERIENCES BETWEEN INTRODUCTORY AND ADVANCED STUDENTS**

Lower division students, primarily the students in their first-year programming sequence, have much to gain by participating in the MUPPETS community. First, introductory students have access to a wealth of pre-made objects constructed by their peers and upper division students. Some of these objects serve as “best-practices” examples that introductory students can examine and alter. In essence, a rich set of complex objects creates a “computational toolbox” of objects for introductory students to access. Second, the set of objects and environments left behind by upper division students can act as scaffolding exercises for new students. Not only do such efforts provide an effective starting point, they also clearly demonstrate to introductory students that their upper division counterparts had similar experiences to their own. Third, the collaborative nature of MUPPETS means that introductory students have access to a community of students outside of the classroom. Students may interact with peers, advanced students, as well as the instructor when they run into difficulties. This allows for introductory students to continue their progress on assignments without waiting for office hours, tutoring hours, or the next class session. Finally, the community experience promotes lower division students to the same level of participation as their upper division counterpart parts. Even the most novice user can create objects that are of use to the community as a whole.

The overall impact on the student community is that MUPPETS transforms the student’s experiences on both sides of the pipeline. The MUPPETS learning community couples the concepts of personal investment through constructive processes and communication as a means to encourage students to invest in their own learning process. With future research, it will be possible to determine whether the construction of an online learning community in MUPPETS can help address concerns regarding first-year and long-term retention.

**SUMMARY**

In conclusion, the MUPPETS environment offers a rich interactive platform for students engaged in the process of learning object-oriented design and programming. The interactive environment provided by MUPPETS allows students to receive immediate feedback while constructing their code. As such, students can make iterative revisions and immediately see how their modifications change the environment as a whole. In addition, the MUPPETS environment is designed to support an objects-first pedagogy, due to its immediate utilization of concepts such as encapsulation, inheritance, and polymorphism. Furthermore, the peer-to-peer collaboration functions allow students to communicate with fellow students, upper division classmates, and instructors when they have difficulties with the design or programs. As such, the MUPPETS community can help correct introductory programming mistakes often encountered by introductory students during the formative stages of learning. Finally, by allowing the student to be both the producer of objects as well as the consumer of objects made by others, the student gains a greater appreciation of design and programming as it applies to real-world, complex systems. In other words, the student’s efforts are a contribution to a whole rather than an exercise in isolation.

The initial use of MUPPETS in the classroom has shown enormous potential. Students have responded positively to the use of the MUPPETS environment and have reacted well to the sharing, modification, and reuse of objects within the environment. Anecdotal evidence indicates that not only does the MUPPETS environment match to our cognitive learning outcomes specified in our introductory course syllabi, but it also addresses some affective learning outcomes as well. The authors feel strongly that the MUPPETS platform will support the pedagogy of introductory computing courses appropriately.

The future of the MUPPETS project is multifaceted in nature. First, the MUPPETS group plans to continue research into the improvement of the technology that drives the MUPPETS platform, which will provide capstone experiences for graduate students over the foreseeable future. Second, there are plans to creative extensive object libraries through a community process, which can be used in different courses across the curriculum. Third, the MUPPETS platform will provide research opportunities to explore the nature of collaboration and learning. Finally, MUPPETS will be used for interdisciplinary research projects. This last direction has
already been partially realized by researchers in the Department of Computer Science and the School of Design who have developed a platform for theatrical storytelling within the MUPPETS environment [14].

REFERENCES


