A Visualization Enhanced Hypertextbook for Computer Science Education

Curt Hill
Department of Mathematics and Computer Science
Valley City State University
Valley City, ND, USA
Curt.Hill@vcsu.edu

Brian Slator
Department of Computer Science
North Dakota State University
Fargo, ND, USA
slator@cs.ndsu.edu

ABSTRACT
An online system is described that has been used for fully online instruction as well as a supplement for face to face courses. The student uses a web browser as the interface. The system was originally described as the Virtual Textbook, but its capabilities have increased to include delivering interactive educational experiences, monitoring student progress and giving individual assignments.
It has many of the features of a Learning Management System as well as additional features. Unlike an LMS, much of its behaviour may be altered with programming. It is also freely available under the GNU GPL.

KEY WORDS
Online learning, Computer Science education.

1. Introduction

A MOO Education Platform (MEP) is an object-oriented, Visualization Enhanced Hypertextbook[1] that has been shown to be suitable for online instruction as well as augmenting classroom instruction. A MEP may deliver content, record student progress, give surveys, tests or quizzes, record the scores and give assignments based upon the lesson material a student has completed.
The MEP is based upon common off-the-shelf software, such as a MOO[2] server, web server and client software. The flexibility of the MEP is derived from the programmability of the objects within the system.
Since a MEP is object-oriented this paper will examine some of the important objects in a top-down approach.

2. Important Objects of a MEP

The organization of a MEP resembles many other educational endeavours. At the very highest level there is a course, which is populated by students, taught by an instructor and there are agents which provide assistance for the students. Clearly, the instructor and students are real people who login to the system using client software. The instructor owns the course. This ownership allows the instructor to determine the lessons that are to be taught, the assignments to be given, the tests to be taken, among other things. The instructor connects to the system and is represented by an instructor object.
Like everything in a MOO, the course is another object. It references the instructor that owns the course, the student objects that are enrolled in the course, the lessons that make up the content, a grade book, as well as any other necessary objects.
An instructor may own several courses and each one is open or closed. An open course is still being used and each student in the course is linked to the course. When a course is closed, the student object continues to exist but the connection to the course is absent. A closed course may be retained indefinitely as a record of the lessons and students involved in it.
A course is largely a collection of lessons that a student must complete. A lesson is a collection of educational opportunities and contains requirements that a student must satisfy in order to complete the lesson. A course has a list of lessons, so there is a strongly implied order of how the lessons are to be completed. However, the student does have some freedom to complete the lessons in different order.
A MOO (the acronym is MUD Object Oriented) has an inherent room and exit model characteristic of the dungeon and dragon type of game from which it was derived. In the MEP a room is also known as an exhibit. The client used on a MEP displays the exhibit in a way similar to a web page, with some small differences. The exits are shown as links in the bottom as are any other objects present in the room. A display of an exhibit is shown in Figure 1.
The course may have an optional gradebook object attached. Scores from multiple choice test questions will be automatically stored in this object. Scores from any type of question, such as an essay, that requires manual intervention may also be entered manually. Lesson completion may also trigger an automatic grade update

and this may be decreased if the completion does not occur prior to a lesson-specific date.

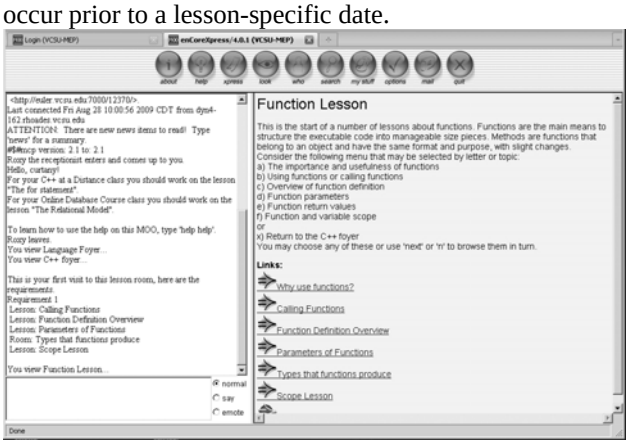

Figure 1. Entry into a lesson room

The illustration in Figure 1 shows a client display of an exhibit. There are several buttons for common commands on the top of the browser window. On the upper left pane are messages sent by the MOO. Commands may be typed into the lower left pane, although students rarely type in commands. The right hand pane is the HTML display. This is where most content is displayed. The icons on the bottom of this pane are adjacent rooms that may be viewed. If any other people or any interactive objects were present in this room their icons would be shown at the bottom as well.

2.1 Lessons

There are several types of room and the lesson is just one of these. A room displays a modest amount of text and graphics. The room display is just displayed HTML. A lesson room is usually the entryway to a collection of other rooms of related topics.

Each lesson has a set of requirements to be satisfied. There are a several items that could be required. The simplest is the visitation of a room. There is no guarantee that a student understood the content displayed in a room, but the system records that this student visited the room, so that it may be a requirement.

Lessons may also be hierarchically arranged. A high level lesson may require the completion of a lower level lesson and any of these lower level lessons may require others as well. If an instructor wants the student to be orderly in the perusal of the course lessons, each lesson may require completion of the prior lesson.

There are also a number of interactive objects that may be required as part of a lesson. Most of these are Java applets that appear in another tab or window of the browser. Several of these will be discussed later.

A lesson may have several sets of requirements. If a student completes any set, this constitutes completing the lesson. The completion of a set of requirements is to satisfy each requirement of the set. These sets may have overlapping requirements but only one of the sets needs to be satisfied.

Each lesson room has a default set of requirements and these are what are generally used. However, two instructors may disagree as to what constitutes proper lesson requirements. The instructor may then put alternative requirements on this lesson. These apply only to the course that this instructor teaches.

If a student is about to leave a lesson – that is to move from any room in the lesson to any room outside the lesson – the system checks the progress towards satisfying the lesson. If all that is lacking is the visitation of simple rooms, a quiz may be offered. An agent appears before the student, tells the student that they have not completed the lesson and offers the opportunity to take a quiz to show mastery. If the student accepts they are transported to an otherwise inaccessible room and a lesson quiz is generated.

Each lesson quiz contains five multiple choice questions. They are culled from the unvisited, but required rooms, then are randomly selected and ordered, as well as the multiple choice answers randomly arranged. The quiz questions and answers are attached to the room object, but they are never shown to a student viewing the room. If the student answers at least four of the five correctly, the lesson is completed. When the quiz is complete the student is given the results and then transported to whichever room was originally intended. If a student misses a question the correct answer is shown. To prevent question depletion, a student who fails a lesson quiz twice will not be offered the quiz option for that lesson again. Moreover, if there are insufficient prepared questions attached to the rooms the quiz will also not be offered.

2.2 The Student Experience

The process for a student is simple. A Java enabled browser is directed towards the system. The student provides a login and password so that the system can activate the correct student object and connect it. The student then meanders through the rooms reading the content material. Unbeknownst to the student the system is recording which exhibits are visited and which interactive objects are used.

Should the student need to be refreshed as to what needs to be done the @course command may be given. This command will show every high-level lesson in any open courses in which the student is enrolled. If the lesson has been completed it will show that fact. The student finds the first uncompleted lesson and begins work on that.

The first time a student enters a lesson room the lesson will show the all the requirement sets present for that exhibit. In Figure 1 the requirements are displayed at the bottom of the upper left pane. If any of these are complete, that is also stated. At any subsequent visit to this room, the student may enter the @requirements command (possibly abbreviated to @req) and an updated list is displayed.

If a student knows from the @course command that a particular lesson should be mastered, yet does not know how to find the lesson, the lesson map may be of help. The lesson map is a series of web pages outside of the MEP system that is built by program. In it is the linked name of each lesson. The link takes one to a lesson description page. This page describes a lesson, including the shortest path to the lesson, subordinate rooms, subordinate lessons and links back to the lesson map and

a lesson index. A typical lesson page is shown in Figure 2.

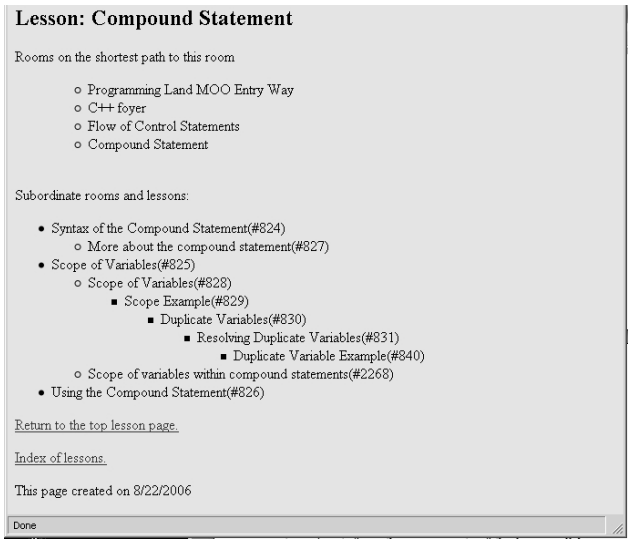

Figure 2. A Lesson Description Page

2.3 Agents

There are a variety of agents that populate the system that attempt to assist the student. The quiz agent has already been mentioned, the roving goalie, lost and aimless agents will now be discussed.

The roving goalie is an agent that is sent to a student, once a goal has been achieved. As previously mentioned, every time a student leaves a room within a lesson to enter a room that is outside of the lesson, the progress towards satisfying the requirements of the lesson will be checked. If the requirements are not satisfied the student might be offered a quiz. If they are complete the student object is augmented with notification of lesson completion, which in turn may be used in other lesson's requirements. High level lessons – those specifically listed in the course object – may also have assignments attached to them. When the lesson completion is attached to the student object another object, the dispatcher, is also notified. If this is one of those lessons with an assignment a roving goalie is dispatched to visit the student.

The roving goalie appears in the exhibit in the same way another person might appear. However, they tell the student they have completed the lesson and that they now have an assignment. Although the student will not necessarily be aware of the fact, but an assignment object has been connected to the student object. At any later time the student may display the text of assignment. After the roving goalie agent has greeted the student and described the assignment, it will leave.

The roving goalie has a list of assignments, which it processes in a round robin fashion. If there are more assignments than students there will be no duplication. However, even with fewer assignments than students the likelihood of plagiarism is greatly reduced. Of course, the instructor has access to which assignment was given to which student. The roving goalie may give any type of assignment. The assignment object contains text, but it may reference web pages or any other type of material that could be supplied through a web server. Since the system has mostly been used in programming classes, programming assignments are the most common type.

The name roving goalie is used because this agent is derived from a (stationary) goalie. In early uses of the system a goalie was placed in a room and the students had to engage in a scavenger hunt to find it. When a student did then the goalie gave them an assignment.

The lost agent has a completely different purpose. Whenever a student passes through any lesson room, this is noted. This event is then compared with the lesson list from the course object. If the student is not within one of his or her first two uncompleted lessons, the lost agent concludes that the student is lost and seeks to help.

A MOO has a typically short horizon. All that may be observed is the contents of the current exhibit as well as the list of exits and the names of the room to which they lead. A student who is classified by lost may have simply forgot the path taken to this location or may be driven by curiosity into areas not yet needed.

Once the lost agent classifies a student as lost, the agent moves into the same room as the student and then offers to help. If the student is actually lost they will be transported into their first uncompleted lesson. On the other hand if the student is just exploring the system, the agent will exit the room and leave the student to continue to explore. In addition, the agent will not bother the student again for some time.

The aimless agent has a similar task to that of the lost agent. Instead of a student moving in an un-required part of the MEP the student is in one of the first two lessons still needing completion. Despite moving around a lesson that is needed, the student is not making progress towards the goal of satisfying the lesson requirements. Like the lost student this might or might not be a concern. The student may be reviewing exhibits to clarify concepts and understanding. It is also possible the student may not be able to find what is needed to complete the lesson.

The aimless agent must keep track of all students' progress towards their current goal. When the student is in the correct lesson but not making progress, the aimless agent moves in and offers to move the student to a better location. This could be an exhibit that has not been visited, a subordinate lesson that has not been completed or a room that contains an interactive object that is required to be used. The aimless agent also asks if assistance is needed in a way similar to the lost agent.

2.4 Required Interactive Objects

Although agents are software objects sent to interact with students, they are never part of the requirements of a lesson. They are helpful but they give no course content nor provide an educational experience. Instead there are several objects able to provide experiences that will be beneficial to the students. The generic one is the MOOApplet, but the CodeApplet and Presentation are common as well.

The MOOApplet consists of two objects, one inside the MOO and one outside. One MOOApplet is an object inside the MOO. This object is designed to start a Java

applet and then receive confirmation that the user of the applet has done enough to receive credit for the exercise. On the outside another object named MOOApplet is a Java class, a descendent of the Java Applet class. The intent of this pair of objects is to allow any useful Java Applet to be converted into an applet that performs some educationally desirable action and that the completion of the applet can be recorded inside the MOO. Figure 2 shows a simplified computer simulator applet that has been converted into a MOOApplet.

The procedure for conversion is generally much easier than writing the applet itself. First a useful Java applet with available source code needs to found or developed. (In the former case the permission of the author should also be obtained.) The applet source code then needs to be modified in two different ways. It must be made a descendent of MOOApplet, rather than Applet. The MOOApplet ancestry provides the means to communicate with the MOO and since a MOOApplet is in itself a derivation of Applet, no capabilities are lost. The second part is to determine what constitutes adequate use of the applet by the student. When this adequate use has been reached then a method defined in MOOApplet is called and this sends notification back to the MOO. The MOOApplet object, the one within the MOO, then records on the student object the completion of the task.

As may be inferred from the previous discussion, the tricky part of this task is determining "adequate use." Many applets are simulations, how much simulation is required for the student to have learned what is needed? This is where the educational expertise of the instructor is needed. What typically happens is that the applet has inserted code which counts actions or notes suitable changes in state. When sufficient numbers of these occur the applet signals its approval to the MOO. However, the student may continue the simulation after this message is sent.

The MOOApplet object inside the MEP identifies the applet name that is required, any parameters that may be needed and generates the needed code to start the applet. The applet itself may be started in the client itself, within the right hand pane, or it may be started as a separate browser window or tab. Figure 2 shows a descendent of MOOApplet running in a new tab, while Figure 3 shows a CodeApplet, which is another descendent of the MOOApplet, running in the client pane. Either configuration will still communicate with the MOO to signal completion of the task. This information will then be recorded on the student object for checking lesson completion.

The CodeApplet is also a descendent of the MOOApplet, but is specialized for the original intent of the courses in the MEP. Since the first courses in the MEP were programming courses, the CodeApplet was used to display, explain and trace through snippets of programming language code. It superseded an earlier MOO object that performed this action, but without any graphical assistance.

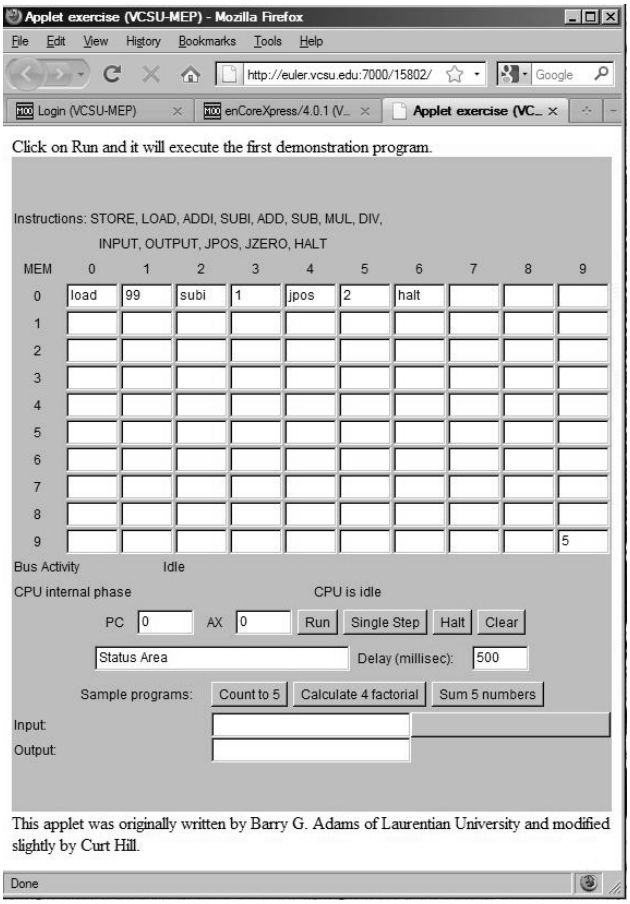

Figure 3. An example MOOApplet

The CodeApplet will display one or more lines of a program. These could be copied out of the window and pasted into an editor, but the normal use is to explain or run the code. In an explanation, each line is shown with some explanatory comments. This may discuss the syntax, semantics, how it is used in the program or any other information that the instructor would like to convey.

A trace is somewhat more dynamic. It is not the actual execution of the lines of the program, but a preformatted trace of how the code would execute. Each line is executed as it would normally. The line that is executed is highlighted. Variables may be changed or outputs displayed. A line in a conditional may be skipped due to the truth conditions and lines within a loop may be executed multiple times.

The trace may be a passive exercise, but it may be interactive as well. The CodeApplet may start the trace of a line and then ask the student the new value of a variable that is to be changed by that statement. This forces the student to enter a value so that they are engaged in the process. Figure 3 shows a CodeApplet executing.

In Figure 3 one line is highlighted. This is the line that has just been completed. At the bottom is a dialog box where the student should enter the value that is to be computed by the next statement. Partially obscured by this dialog box is another box containing variables that have been used, their current and last values. A student can observe the statement to be executed, the current variable values and make a prediction of the value about to be computed.

The system also tells the student whether the value they typed is correct.

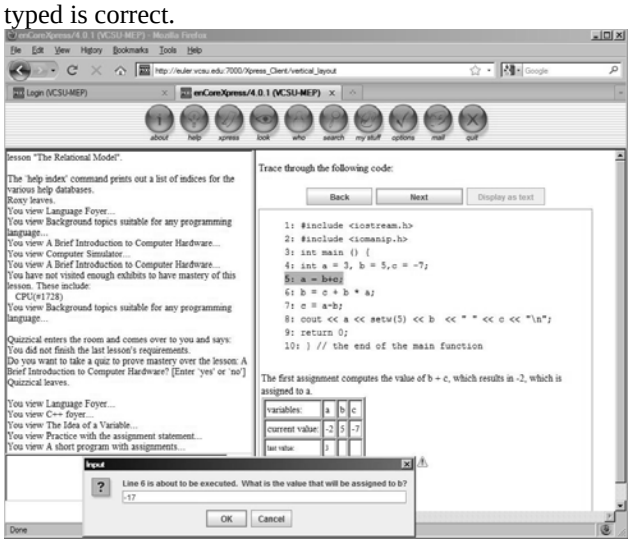

Figure 4. A CodeApplet running

The CodeApplet is parameterized by three strings. The first of these are the lines of code to be displayed. The second is the explanation for each line. The third is a form of the simulated trace, which may include the questions and answers. The student may move forward or backward through either a trace or explanation. However, a question on a variable's value will only be asked the first time through the line. Either a trace or explanation may be a requirement of a lesson. The student must complete the trace or explanation to receive credit.

The RecordedPresentation[3] is another derivation of MOOApplet. This particular object is somewhat more general than the CodeApplet, at least in regards to subject matter. The idea is to deliver to the students the slides of a Powerpoint® presentation along with recorded audio. The presentation is contained in compressed archive files as graphic files and sound files. Any graphic or sound file that a Java applet may play are acceptable so the original form of the presentation does not need to be Powerpoint. It could be any presentation manager that is able to export its slides to a JPEG or other common format. The audio may be captured in giving the presentation within a class environment or may be recorded privately.

A Java applet, which is also a descendent of MOOApplet, provides the player. It displays each slide and then plays the audio clip. It notifies the MOO when the presentation is complete, but only gives this notification if each slide was displayed and each audio clip is played in its entirety. A presentation could be lengthy, so the player notifies the MOO of each completed slide. Should a student be forced to stop the player before completion and start it again later it will start at the next slide, not necessarily the beginning. Figure 4 shows the player and its controls.

3. Experiences

The system has been used as a resource for a face to face class, often functioning as a virtual textbook. However, the system has been designed so that students may work asynchronously with minimal interaction with the instructor. Students may login and work towards their current goal at any time. If two or more are logged in at the same time they can be aware of each other as well as the instructor. Every room in a MOO is a chat room, so communication is easy. Each student may find out what needs to be done next, receive individual assignments and browse through the exhibits at their own pace. There is no requirement that a class of students needs to progress at the same rate.

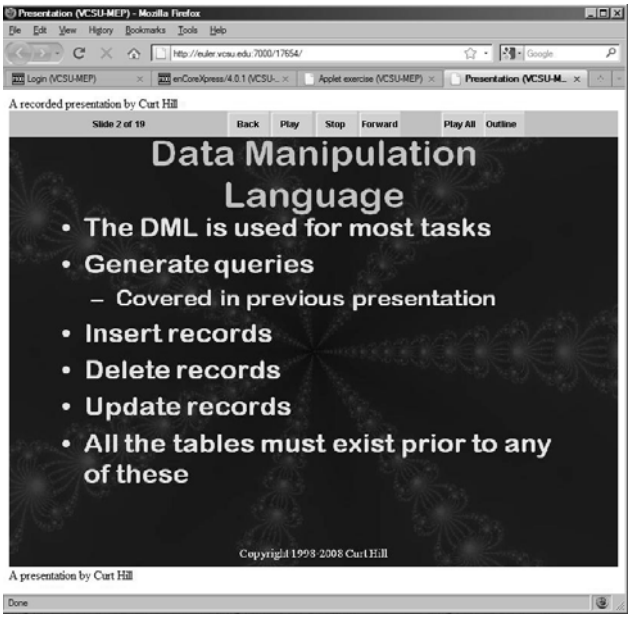

Figure 5. A RecordedPresentation.

There is a provision for local email within the MOO, but normal email is generally preferred. Any instructor with online courses needs to provide a quick response to email questions, so this is the usual communication method for fully online students.

Unlike, Blackboard[4] or other Course or Learning Management Systems there is no means to load a substantial amount of material quickly and easily. Instead the creation of content is comparable with the writing of text book. Creating material also requires some knowledge of the MOO paradigm. It is not just writing the material but also arranging it into the separate exhibits.

The RecordedPresentation is the most recent addition to the system and has become the means to rapid content development. Many classroom instructors have presentations that may be quickly converted into a RecordedPresentation object. This allows a lesson to be filled with content quickly. At a later time text-based exhibits may be created to complement these presentations.

The RecordedPresentation applet has applicability outside of the MEP environment as well. It may be used as an ordinary applet, with the setting of a parameter within the surrounding HTML. This parameter suppresses communication with the MOO and the expectation that

parameters that describe the student will be present. Since there is no recording of the results, there is also no reason to insist that a clip must be played in its entirety prior to moving on to the next slide. This facility was used in a face to face class using the following procedure. The presentation was given in a classroom and the audio recorded. Later the audio was separated into clips for use on each screen. The applet and needed information was placed on the web. This gave a facility for students to review the material a second time as well as students who missed the class to see it for the first time.

The MOO software allows for the development of any other objects that may be deemed useful. This requires some programming sophistication as does the creation of new Java applets. The modification of an existing Java applet into a MOOApplet is much less complicated and can usually be handled even by students with just a semester or two of Java programming.

4. Conclusion

The MEP system has been successfully used at two different institutions in computer science classes. This is a historical accident of the author's departments and teaching duties. This has certainly influenced the types of interactive objects, but the system could easily be used in other areas as well.

The system has been used both as an online course and as a supplement to a face to face course. Unlike a textbook, there is some student learning overhead required to use the system. Motivation is often an issue in online courses and the MEP system is no exception. The MEP system is somewhat less structured than other systems, which can cause problems. Objects such as the lost and aimless agents have been implemented to counteract such problems. Determined and motivated students have not found the overhead a problem.

The interface for students is somewhat easier than for instructors. The first author invariably uses an older, less graphical interface for development and the web client for testing. Other instructors are more likely to use the graphical interface for both.

The system may be freely distributed. Several MOO servers have been used and others exist. LambdaMOO[5] is the most common, but WinMOO[6] has also been used. Any web server should be acceptable. Apache[7] has been quite reliable and is used to serve the HTML generated by a RecordedPresentation, among other things. The original core for the MOO is due to enCore[8]. However, the core of MEP has not been brought up to the current level of that enCore. This is a project that will be investigated soon. Those interested in obtaining components not available from the cited web pages should contact the first author.

Acknowledgements

The MEP project has been supported by National Science Foundation Grant EIA-0313154 and by ND-EPSCoR through the FLARE program under EPS-9874802. The authors would also like to acknowledge the many students who used the system and gave helpful feedback as well as those who programmed some of the objects. In addition the applet authors such as Barry G. Adams who gave us permission to use and modify their applets.

References

[1] Curt Hill, Brian M. Slator, & Vijayakumar Shanmugasundaram, ProgrammingLand: A Visualization Enhanced Hypertextbook. *Proceedings IEE Frontiers In Education, 2007*, Milwaukee, WI, 2007, CD.

[2] P. Curtis, Mudding: Social Phenomena in Text-Based Virtual Realities. *Proceedings of the Conference on Directions and Implications of Advanced Computing. 1992*.

[3] Curt Hill, Captured Presentations for Online Learning. *Midwest Instructional and Computing Symposium*, Rapid City, SD, 2009, CD.

[4] Blackboard, Teaching and Learning. http://www.blackboard.com/Teaching-Learning/Learn-Platform.aspx Date accessed 4 Sept 2009.

[5] Ken Fox, Lambda MOO, MUD Server Software. http://www.moo.mud.org/ Date accessed 3 Sept 2009.

[6] Chris Unkel, WinMOO Frequently asked questions. http://www.stanford.edu/~cunkel/WinMOO/ Date accessed 3 Sept 2009.

[7] Apache HTTP SERVER PROJECT. http://httpd.apache.org / Date accessed 3 Sept 2009.

[8] enCore Open Source MOO Project. http://sourceforge.net/projects/encore/ Date accessed 4 Sept 2009.