

**DEVELOPMENT TOOLS FOR CONTENT CREATION IN  
VIRTUAL ENVIRONMENTS**

**A Paper  
Submitted to the Graduate Faculty  
of the  
North Dakota State University  
of Agriculture and Applied Science**

**By**

**Guy Eric Hokanson**

**In Partial Fulfillment of the Requirements  
for the Degree of  
MASTER OF SCIENCE**

**Major Department:  
Computer Science**

**November 2010**

**Fargo, North Dakota**

North Dakota State University  
Graduate School

---

Title

**DEVELOPMENT TOOLS FOR CONTENT CREATION**

---

**IN VIRTUAL ENVIRONMENTS**

---

By

**GUY ERIC HOKANSON**

---

The Supervisory Committee certifies that this *disquisition* complies with North Dakota State University's regulations and meets the accepted standards for the degree of

**MASTER OF SCIENCE**

---

North Dakota State University Libraries Addendum

To protect the privacy of individuals associated with the document, signatures have been removed from the digital version of this document.

Date

Signature 

## **ABSTRACT**

Hokanson, Guy Eric, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, October 2010. Development Tools for Content Creation in Virtual Environments. Major Professor: Dr. Brian M. Slator.

Immersive Virtual Environments (IVEs) for education are designed and implemented to enable students to learn complicated concepts in an exploratory and inquiry based manner. The environments are constructed so that multiple users can interact with the educational simulation and learn to think like a scientist.

Developing these IVEs requires a multi-disciplinary development team that consists of more than just software engineers. It requires content experts to provide the information needed to create the best and most interactive lessons possible. While some content experts have a strong interest in technology and are capable of working with the development environment directly, many are more interested in their fields of expertise and would prefer to leave the programming and technical details to others. This presents a logistical problem as the experts have to somehow transfer their knowledge to the programmers who encode it into the IVE.

In order to increase productivity it is suggested that web based, content editors would alleviate this development bottleneck. These tools would need to be cross platform, accessible anywhere Internet is available and not require the installation of any special software. This paper describes the design and implementation of a principled set of tools; Bot Conversation Editors used to create agent conversations, Task Editors to create and manage player tasks, and Help Editors to manage educational content in in-game reference materials.

## **ACKNOWLEDGEMENTS**

I would like to express my appreciation to my advisory committee: Dr. Brian M. Slator, Dr. Donald P. Schwert, Dr. Anne Denton, and Dr. Juan (Jen) Li. A special thanks to my thesis advisor, Dr. Slator for his encouragement and the occasional prodding that got me through the entire process.

Thanks to the Computer Science students who have worked with me over the years, particularly Otto Borchert for his suggestions, constructive criticism, and putting up with my sometimes inane questions.

I would also like to thank those members of the World Wide Web Instructional Committee that I had the opportunity to work with: Dr. Brian M. Slator, Dr. Donald P. Schwert, Dr. Bernhardt Saini-Eidukat, Dr. Lisa Daniels, Dr. Jeff Terpstra, and Dr. Jeffrey Clark.

Finally, I would like to thank my father who bought me my first computer long before they became ubiquitous.

# TABLE OF CONTENTS

ABSTRACT .....	iii
ACKNOWLEDGEMENTS .....	iv
LIST OF FIGURES .....	vii
1. INTRODUCTION .....	1
1.1 Motivation .....	2
2. BACKGROUND .....	4
2.1 Technology Background .....	4
2.2 IVE Background .....	6
2.2.1 On-A-Slant Background .....	6
2.2.2 Geology Explorer Background .....	7
2.2.3 eGEO Background .....	8
2.3 Bot Background .....	8
3. LITERATURE REVIEW .....	11
3.1 Content Development Tools developed at NDSU .....	11
3.1.1 The ZeleCon or the Zelenak Conversation Constructor .....	11
3.1.2 Virtual Entity Tool .....	13
3.2 Other Content Development Tools .....	14
4. IMPLEMENTATION .....	16
4.1 Bot Conversation Editors .....	17
4.1.1 Using the BCE .....	17
4.1.2 Bot Editor Implementation .....	22
4.1.2.1 List of Agents .....	22

4.1.2.2 Add New Bot.....	27
4.1.2.3 Edit Bot.....	29
4.1.2.4 Dump the XML.....	30
4.2 Task Editors .....	32
4.2.1 Using the Task Editor.....	34
4.2.2 Task Editor Implementation .....	38
4.2.2.1 List of Tasks .....	38
4.2.2.2 Add New Task .....	40
4.2.2.3 Edit Tasks.....	41
4.2.2.4 Dump the XML.....	44
4.3 Dictionary and Help Editors .....	46
4.3.1 The Geology Explorer Help Editor .....	46
4.3.2 The On-A-Slant and eGEO Dictionary Editors .....	47
5. CONCLUSION.....	51
6. REFERENCES .....	55

## LIST OF FIGURES

Figure 1. The conversation network of a software agent named Red Blossom .....	16
Figure 2. The list of agents .....	18
Figure 3. The Bot conversation interface .....	20
Figure 4. Database tables that store software agent conversations. ....	25
Figure 5. Bot Editor system diagram.....	26
Figure 6. A sample XML bot conversation template file.....	28
Figure 7. The bot conversation XML file for the Dog Travois Women .....	31
Figure 8. The list of tasks.....	35
Figure 9. The edit task interface. ....	37
Figure 10. Database tables that store task information.....	39
Figure 11. (a) The <i>task_questions</i> database table.....	40
Figure 11. (b)The <i>task_answers</i> database table.....	40
Figure 12. A sample XML task template file. ....	41
Figure 13. The XML task file for the Hide Scraper task. ....	45
Figure 14. Dictionary database tables. ....	48
Figure 15. The eGEO Dictionary Editor .....	49
Figure 16. The On-A-Slant Dictionary Editor .....	50
Figure 17. A screen capture from the On-A-Slant Virtual Village IVE .....	53
Figure 18. A screen capture from the eGEO IVE.....	54

# 1. INTRODUCTION

Many will argue the future of the Internet lies in user-generated content. Koster (2006) writes eloquently about the issues involved with user-created content on the web – the efforts are unfocused and incomplete. He concludes that the best hope is for good editing tools to allow content providers to work easily and well.

At North Dakota State University (NDSU), the World Wide Web Instructional Committee (WWWIC) conducts research into intelligent educational media. These systems are educational games that cover subjects such as cellular biology, geology, environmental science, micro-economics, computer science, and archaeology/anthropology.

These Immersive Virtual Environments (IVEs) for education are designed and implemented to enable students to learn complicated concepts in an exploratory and inquiry based manner. The environments are constructed so that multiple users can interact with the educational simulation and learn to think like a scientist.

Developing these IVEs requires a multi-disciplinary development team that consists of more than just software engineers. It requires content experts to provide the information needed to create the best and most interactive lessons possible. Typically, these content experts have little experience with programming and are unable to translate their knowledge directly into the code. This presents a logistical problem as the experts have to somehow transfer their knowledge to the programmers who encode it into the IVE. This is frequently done through various



forms of messaging or face-to-face meetings but this can be inefficient. It is often hard to describe exactly what is wanted in a message and having two people working on a single task is inefficient and often frustrating. Also, waiting for messages to arrive or setting up a convenient meeting time results in delays and loss of productivity.

## **1.1 Motivation**

For many years WWWIC has been developing tools to assist in building Immersive Virtual Environments. Most of these were built as necessity demanded. This paper describes the design and implementation of a principled set of tools, leveraging from the experience we have developed in this area.

During the development on the On-A-Slant Virtual Village IVE, it became apparent that the current state of information flow between content experts and programmers was an impediment to the development process. Considerable time was lost waiting for content to be delivered to the programmers and entering and editing text is not an efficient use of the programmer's skills. Also, the review and edit cycle was extremely inefficient.

Once information was entered by the programmers they would notify the content experts, usually by email. The content experts would then review the information and send their comments and changes back, again usually by email. This process would often take days for something that could be completed by one person alone in a few minutes, if afforded the appropriate tools.

While training the content experts in the skills necessary to modify the necessary parts of the IVEs source files is a possibility, it is not an efficient use of

content expert time and many would find it a daunting task. There is also the possibility that the content expert, with limited training, could make unintentional changes to other parts of the IVEs source files, causing further lost effort.

In order to increase productivity it is proposed that simple, web based, content editors will alleviate this development bottleneck. These tools will enable content experts to add their information directly into the necessary areas of the IVEs without programmer assistance. Furthermore, the use of these tools will help prevent errors, especially when dealing with complex data and knowledge structures.

To be fully functional, these tools would need to be cross platform and accessible anywhere the Internet is available. Further, they should not require the installation of any special software.

## **2. BACKGROUND**

This section provides background information on the technologies used to develop the tools covered in this paper and the IVEs in which they were implemented.

### **2.1 Technology Background**

The tools described in this paper were developed using free and open-source software. This collection of software is essentially the components needed to build a viable general purpose web server and is often referred to as LAMP, an acronym for Linux, Apache, MySQL, and PHP or Perl.

LAMP is an open source Web development platform that uses Linux as the operating system, Apache (Apache Software Foundation, 2009) as the Web server, MySQL (Oracle Corporation, 2010) as the relational database management system and PHP (Hypertext Preprocessor; PHP Group, 2010) or Perl (Perl.org, 2010) as an object-oriented scripting language. In our case PostgreSQL (PostgreSQL Global Development Group, 2010) was used instead of MySQL because of its more advanced SQL (Structured Query Language) support and significantly more flexible licensing.

Linux is a free open source computer operating system that provides basic functionality and the ability to run additional software and services, such as a web server or database server.

Apache is a web server that runs on top of the Linux operating system that listens for requests from other computers and either translates that request into a

filename, and sends that file back over the Internet, or into a program name, and then runs that program and sends its output back over the Internet.

PostgreSQL is an open source object-relational database management system that stores, retrieves and processes data.

PHP & Perl are server side scripting languages that can be used to create dynamic web pages and applications. In this context, they facilitate the interaction between HTML web pages, a LambdaMOO server and a PostgreSQL database.

LambdaMOO (Curtis, 1997) is a database server that provides a network-accessible, multi-user, programmable, interactive system well-suited to the construction of text-based adventure games, conferencing systems, and other collaborative software and at the same time provides a programming language for writing the simulation and customizing the IVE. The LambdaMOO server has three main purposes in IVEs: 1) It is the repository for user data so a player can connect using various instances of the client and still maintain their user history; 2) It handles messaging for broadcast and inter-client communications; 3) It also manages the representations of all non-interface game objects such as software agents, players, tasks and the other multitude of environmental objects that make up a virtual environment.

A Concurrent Versions System (CVS; Price & Ximbiot, 2006), is an open-source, network-transparent program that allows developers to manage different development versions of source code files by recording changes made to each file.

The Eclipse (Milinkovich, 2010) Integrated Development Environment (IDE) is an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across its lifecycle.

## **2.2 IVE Background**

The educational systems currently at our disposal are the Geology Explorer (Saini-Eidukat et al., 2001) and the Geology Explorer 3D (eGEO; Schwert et al., 2010), the Virtual Cell (White et al., 1999), Dollar Bay and Blackwood (Slator et al., 2001), the On-A-Slant Virtual Village (Hokanson, et al., 2008), and the ProgrammingLand MOOseum of Computer Science (Slator and Hill, 1999). All of the IVEs are of the 'desktop VR' variety, where students join an immersive simulation using a personal computer and then explore the 'virtual space' in a goal-directed manner, assuming a role and learning the content by actively participating in the problem-solving context. These have been rigorously tested in formal educational environments and have shown to be both engaging for students and highly effective (McClellan et al., 2001) in a range of controlled studies conducted over several years.

The three IVEs currently using the tool set described in this paper are the On-A-Slant Virtual Village, the Geology Explorer, and eGEO.

### **2.2.1 On-A-Slant Background**

On-A-Slant Virtual Village is a virtual reconstruction of a sedentary Native American village located along the Missouri River near Mandan, North Dakota, USA. The On-A-Slant village was established in the second half of the sixteenth

century and was abandoned around 1781, more than two decades before the Lewis and Clark expedition would explore that region for the United States. The environment is a “learn by doing” simulation based on a 3D reconstruction of the archeologically important Mandan village. Students explore the site, discover artifacts, and develop an interpretation of the relationship between the archeology and society by interacting with the visualized context.

In the game, students are sent back in time to explore the village, learn about the food, family, lifestyle, education, and other cultural elements of the Mandan people before the full impact of Euro-American expansion. At the same time students are taught the methods and logic of anthropology and archaeology at an introductory level.

### **2.2.2 Geology Explorer Background**

The Geology Explorer is a goal-oriented computer game in which students learn about geology by acting like scientists exploring a new world. Within this virtual world, students “travel” to an imaginary Planet Oit, in order to gather geologic data about this newly discovered planet. Students act like geologists by performing various tests in order to identify unknown rocks and minerals and create a geologic map, which serves as an interpretation of the underlying geology of the area.

In the game, students are transported to the planet's surface and acquire a standard set of field instruments. Students are issued an “electronic log book” to record their findings and are assigned a sequence of exploratory goals. These goals are structured using a scaffolding learning strategy (Vygotsky, 1986) and are

intended to motivate the students to view their surroundings with a critical eye, as a geologist would. The students make field observations, conduct small experiments, take note of the environment, and generally act like geologists as they work towards their goals. A scoring system has been developed, so students can compete with each other and with themselves.

### **2.2.3 eGEO Background**

eGEO is a 3D immersive virtual environment (sometimes referred to as the Geology Explorer 3D) with a primary focus to interest students in careers in the geosciences. As geology is rarely taught at the high school level, student interest in careers in the earth sciences tend to be lower than that of the more often taught biology or chemistry. The eGEO environment is based on the same concepts as the original Geology Explorer but combines math and chemistry concepts along with the geologic goals. In this way, teachers in high school mathematics, chemistry, and environmental science can use eGEO in their classrooms, in hopes of increasing interest in a career in the geosciences.

### **2.3 Bot Background**

The method that transforms an IVE into a learning experience is enculturation within the virtual conditions. Enculturation in this sense refers to learning through social observation and interaction (Spindler & Spindler, 2000).

Learning through enculturation is a social process. That is, it requires two or more actors. One actor is the student and the other actor can be another student, a tutor, or other software agent. Without this social interaction, students

are compelled to learn through rote memorization or other nonsocial interactions that can slow down, frustrate, or even skew knowledge. Furthermore, social interaction is fundamental to the real world, at least in the physical and social sciences, and as such bears directly on the student's ability to begin approaching problems in ways found in real-world disciplines (Borchert et al., 2010). Thus, in the role-based scenario of the virtual world, the amount of social interaction bears directly on the effectiveness of the learning environment for the student. In other words, the more the student interacts with objects and persons, the greater informal guided discovery and through that, the diffusion of knowledge.

In Immersive Virtual Environments, software agents (Bots for short) are implemented to exhibit authentic behavior(s) of the following types (Slator, 1999):

- **Atmosphere agents** that simply lend color to the IVEs. These do not directly effect game play but provide animation and interest without causing distraction. For example, in an urban simulation there might be a street magician, a street vendor, a beat cop, a street sweeper, and so forth; in a museum simulation there might be visitors wandering the exhibits or vendors selling popcorn; on a virtual planet perhaps animals roaming the desert.
- **Infrastructure agents** who contribute in some way to the game play. In the On-A-Slant Virtual Village simulation, there are villagers who answer questions and demonstrate various aspects of village life. In a museum simulation, one might expect a tour guide; on a virtual planet, another kind of guide.
- **Tutoring agents** are unobtrusive but proactive software agents that provide assistance to students in the course of their diagnostic reasoning within the



scientific problem solving required to accomplish their goals. They provide help if needed, remediate on student actions, and point students in the right direction. Diagnostic tutors work from knowledge of the materials found in the environment, the student's history, and knowledge of the experiments needed to complete a student's tasks (Slator et al., 2003). They monitor a student's progress in completing a task and identify when a student may be having difficulty. Depending on the results of this analysis, tutors may decide to remediate on the spot, or to defer remediation until the student begins to show an identifiable pattern of behavior (Slator et al., 2006).

For example, the On-A-Slant Virtual Village has forty interactive software agents, several of each type, with which players can interact.

### **3. LITERATURE REVIEW**

In order to appreciate the value of the described content development tools, it is necessary to understand what other tools currently exist. The goal of this section is to provide relevant background information to show the technological gaps that these tools were created to fill.

#### **3.1 Content Development Tools Developed at NDSU**

Creating IVEs is an intensive process in terms of design, knowledge engineering, and software development. Over the years, WWWIC has gained experience in the crafting of these systems and has designed and developed an integrated library of software tools to substantially streamline the development of future IVEs. These tools primarily support simulation and agent building with the ultimate aim of moving development into the hands of content specialists, teachers, and curriculum developers, rather than computer programmers.

##### **3.1.1 The ZeleCon or the Zelenak Conversation Constructor**

Some WWWIC IVE's use conversation networks for agents within the environment. A conversation network is a vector of 3-tuple response nodes containing an identifying question, a list of responses, and a list of follow up questions. Each node is unique based on its identifying question and one node is designated as being the root or "init" node of the network. The "init" node is the starting point of the conversation and contains only the follow up questions that will be loaded into the conversation at startup. Each follow up question in a node points to an identifying question in another node. Every identifying question has a

list of possible responses and each response has an associated follow up question.

These conversation networks give depth to the character of the agent. However, building multilevel conversation networks directly in LambdaMOO, which is one of the programming languages used to create virtual environments, is tedious and error prone. In addition, the direct coding approach does not allow the user, who is building the virtual world, to visualize the conversation network and test the traversal of the conversation network.

To solve this problem the ZeleCon or the Zelenak Conversation Constructor (Zelenak, 1999a) was built for the construction of multilevel conversation networks for the agents with an easy to use graphical user interface. The tool is built in Java and integrates three independent programs under centralized control. The independent programs are: 1) the Visualization Program for visualizing the topology of the conversation network, 2) the Response Constructor for creating nodes within the conversation network and control the automatic readjustment through several options such as scrambling, shaking, randomly exciting, and freezing the conversation structure, and 3) the Testing Program for testing the traversal of conversation network. There is also a fourth program called the Telnet Window, which allows for direct connection to the educational simulation and can be used by an advanced user to submit commands to the server or for a log of the session.

The ZeleCon Controller integrates these components under a centralized control window, provides a connection to the educational simulation environment

as well as uploading and downloading of the conversation structure from the simulation environment.

Conversations are displayed in the Visualization Program in graph form, with nodes on the graph indicating something an agent would say while links in the graph indicate student choices. Upon building a conversation, that conversation can be virtually tested and eventually exported to the MOO for use in the game (Zelenak, 1999b).

The ZeleCon was developed for a different generation of IVEs and the extensive modifications needed to make it work with the On-A-Slant and eGeo projects made its use impractical. The ZeleCon is also a Java applet which requires the installation of the Java Runtime Environment (JRE) plug-in, putting it outside the “No special software” installation constraint.

### **3.1.2 Virtual Entity Tool**

The Virtual Entity Tool (Xinhai, 2000) employs an entity template system with a form-filling interface to enable creation of multiple instances of a category. For example, they define a template for minerals that specifies the properties indigenous to minerals, and ranges of values associated with each property. Then, a content specialist will create new minerals, quartz, tourmaline, talc, etc., with a graphical form-filling interface where values such as color, texture, and hardness can be quickly and easily selected from menus. This tool is general in that any category of entity (animal, vegetable, or mineral) can be constructed with it (Slator et al., 1999).

### 3.2 Other Content Development Tools

A number of organizations that develop virtual environments for education have produced tools for use by content experts with limited or no programming skills. These tools typically come in two categories; those for editing visual content and those for creating a complete environment.

Creating and manipulating 3D models and graphics in a virtual environment requires an expert in visual content. Tools for this purpose range from commercial stand alone modeling suites like Maya (Autodesk, 2010) and Blender (Blender Foundation, 2010) to environment specific tools like Unreal Technology's UnrealEd (Epic Games, 2010) and Second Life's Build tool set (Linden Research, 2010).

The second category of content creation tools is closer to what is described in this paper. They are not specifically designed to split the IVE creation task between programmers and content experts but they do allow non-programmers to construct educational virtual environments.

Alice is an educational software package developed at Carnegie Mellon University (Alice, 2010). It is designed to be an interactive graphical programming environment that allows novice users to create 3D virtual environments without programming knowledge. Alice uses a drag-and-drop interface to manipulate graphical tiles to assemble a program where the instructions correspond to standard statements in a production oriented programming language, such as Java, and C++. Alice is explicitly designed to teach programming to children and its customizability is insufficient for WWWIC's use in building content rich IVEs.

The HI FIVES (the Highly Interactive Fun Virtual Environments in Science) project at North Carolina State University has created Virtuoso, a development environment created using the Half Life game engine. This tool makes it easy for non-programmers to create educational games in a multiplayer setting. They have a number of games in development for students in grades 6-9 across a wide variety of disciplines (North Carolina State University, 2008). As mentioned, HI FIVES requires the Half Life game engine which also puts it outside the “No special software” installation constraint.

LearnLab is a joint collaboration between Carnegie Mellon University and the University of Pittsburgh designed to provide expert help in the area of the learning sciences (Pittsburgh, 2008). In particular, they have developed a number of tools for creating advanced learning technologies, including the Cognitive Tutor Authoring Tool (CTAT) and TuTalk (Jordan, Rigenberg, and Hall, 2006). They claim no programming skill is required but the system is based on Macromedia/Adobe Flash authoring tools which requires its own set of specific skills as well as a software installation.

## 4. IMPLEMENTATION

All of our IVE's contain some combination of tutoring, atmosphere, and infrastructure agents. The On-A-Slant IVE, for which the Bot Conversation Editor was originally developed, presents most of its educational content through the interactions (i.e. conversations) between students and infrastructure agents.

Figure 1 shows a simple conversation between a student and a software agent that provides some basic information on gardening and the use of a bison scapula as a hoeing instrument.

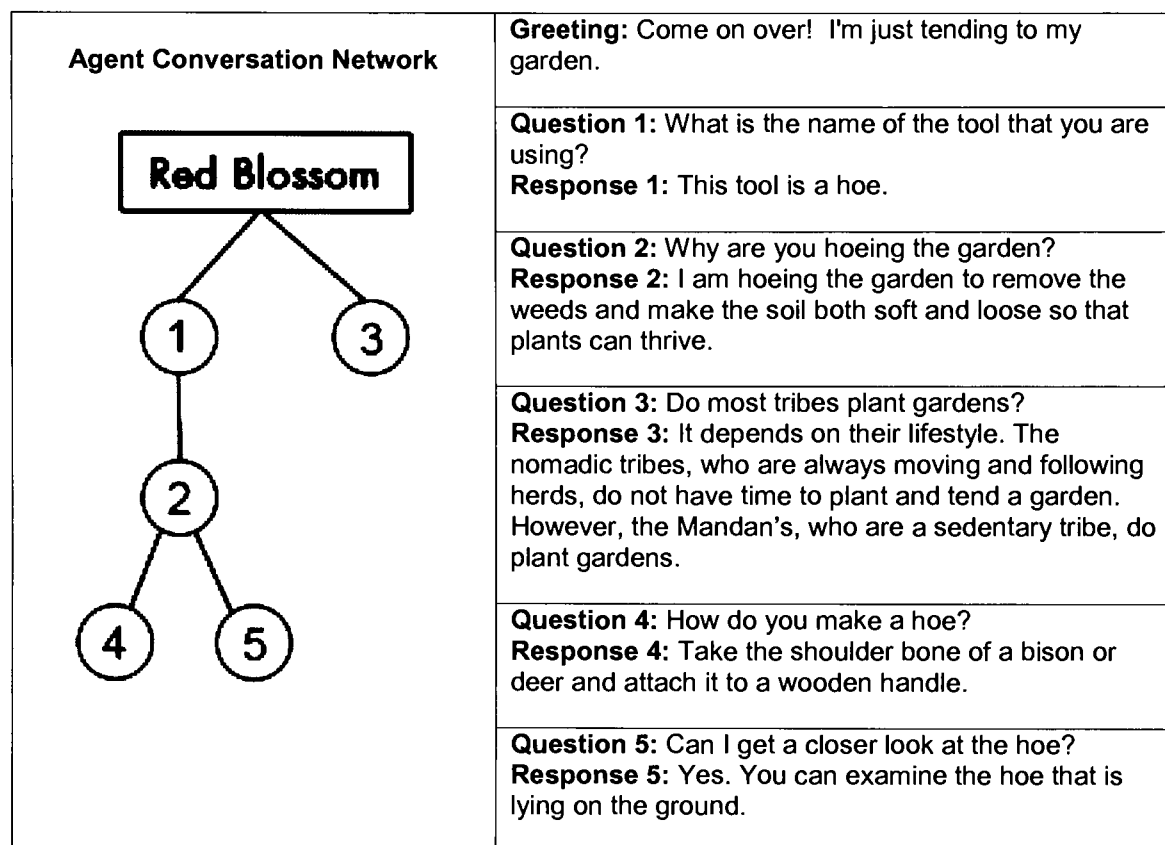


Figure 1. The conversation network of an On-A-Slant software agent named Red Blossom which represents a Mandan woman working in her garden. The graph on the left represents the question graph. Questions 1 and 3 can be asked at any time. Question 2 can only be asked after question 1 and questions 4 and 5 can only be asked after question 2.

## **4.1 Bot Conversation Editors**

Creating and maintaining these software agents is a joint task involving content experts and programmers. The content experts create a written design that describes the agent, its conversation network, and any tasks it can accomplish. This design is then given to the programmers to implement. If the agent needs any modifications or adjustments to the agent, the content experts have to contact the programmers to make the changes.

The Bot Conversation Editor (BCE) was created so that anyone on the development team, content experts in particular, could create or update the software agents using a simple web based interface. Conversations are built using a form layout that provides options for inserting, deleting, editing and ordering of conversational statements and responses in hierarchal order.

### **4.1.1 Using the BCE**

The BCE consists of three interfaces; a list of agents, a form for adding agents to the IVE, and a form for editing agent conversations.

The list of agents (Figure 2) displays all the active software agents in the environment, their job title, and their location. Clicking on one of the agents in the list brings up a form for editing the agent's conversation. There are also two buttons at the bottom of the page. The 'Add New Bot' button brings up a form where you can enter information necessary to create a new agent. The 'Dump XML' button updates the CVS repository to reflect any changes made using the editor. Its operation is discussed later.



The interface for adding agents collects the minimal information needed to create a new agent (agent name, agent's job or title, and agent location), creates the agent object on the MOO and adds an empty XML bot conversation template to CVS. The conversation is entered later using the editing interface.

**Bot Conversation XML Editor**
On-A-Slant Home | Maint Home | Editor Home

name	job	obj#	location
<a href="#">Corn Silk</a>	Final Construction Woman	1124	Village
<a href="#">Crow Woman</a>	Winter Lodge Woman	2544	Village
<a href="#">Crows Heart</a>	Horse Man	1010	Ceremonial Grounds
<a href="#">Exterior Post Animation</a>	Exterior Post Animation Man	959	Village
<a href="#">Lone Cloud</a>	Central Post Man	2542	Village
<a href="#">One Buffalo</a>	Rafter Woman	2543	Village
<a href="#">Owl Woman</a>	Hide Scraping Woman	2992	Ceremonial Grounds
<a href="#">Prairie Flower</a>	Exterior Post Overseer Woman	2478	Village
<a href="#">Sage</a>	Squash Slicing Woman	2935	Ceremonial Grounds
<a href="#">Small Horn</a>	Age-Graded Societies (Men)	2052	Village
<a href="#">Stays at Home</a>	Scaffold Woman	1011	Ceremonial Grounds
<a href="#">Yellow Blossom</a>	Age-Graded Societies (Women)	1964	Village
<a href="#">Eternal Bloom</a>	Fire Pit Woman	1117	Earth Lodge B
<a href="#">Rippling Water</a>	Horse Corral Woman	1663	Earth Lodge B
<a href="#">First Daughter</a>	Cache Pit Woman	2662	Earth Lodge C
<a href="#">Little Bluff</a>	Storytelling Boy	2089	Earth Lodge D
<a href="#">New Arrow</a>	Games Man	1686	Earth Lodge D
■ { Additional Bots removed from image to conserve space. } ■			
<a href="#">White Star</a>	Bison Butchering Woman	348	Lakota Village
<a href="#">Bluebird</a>	Lookout Girl	1794	Gardens
<a href="#">Red Blossom</a>	Gardening Woman	2924	Gardens
<a href="#">Spotted Eagle</a>	Fishing Man	2477	Garden Area Plains
<a href="#">Green Leaf</a>	Windscreen Woman	2130	Earth Lodge A
<a href="#">Snowbird</a>	Bed/Basket Woman	2173	Earth Lodge A
<a href="#">Professor Slator</a>	On Bus Tutorial	2623	Tutorial Room

Last XML.jar dump: 12:11 PM August 29, 2008

+ [View database maintenance trace.](#)

Figure 2. The list of agents displays the active software agents in the environment, their job title, and their location. Clicking on one of the agents in the list brings up a form for editing the agent's conversation. There are also two buttons at the bottom of the page. The 'Add New Bot' button brings up a form where you can enter information necessary to create a new agent. The 'Dump XML' button updates the CVS repository to reflect any changes made using the editor. The column of object numbers is included for easy reference between the content experts' and programmers' view of the software agent's program object.

The edit conversation interface (Figure 3) allows the construction of a conversation between a player and an agent. It is accessed by clicking an agent name in the list.

A conversation begins with an initial greeting message. For example, when a student playing the On-A-Slant Virtual Village game first encounters the agent, 'Little Bluff' (the software agent who talks about the importance of storytelling to the Mandan people) exclaims "Hi, you're just in time, Grandfather is about to tell me a story!"

The other parts of a conversation are questions you can ask of an agent and each question's associated answer. Questions can be added by clicking on the 'Add new question' link and filling out the form that opens. Existing questions can be edited by clicking on one of the questions in the list.

Questions are organized in a hierarchical manner by declaring their dependence on other questions (see the graph in Figure 1). This means that questions farther down the hierarchy can not be presented to the player until the higher level questions have been asked. This is achieved by entering a list of dependent questions into a dependency list. This list is a comma delimited string of question ID numbers, where ID number is the order that questions are displayed in the editing interface. Question display order can be manipulated using 'move up' and 'move down' buttons. Unwanted questions can be removed using the 'delete' button. If a deleted question has dependents then those dependents are promoted up the hierarchy. This may not be the optimal solution in every case and the intent was to have the content experts maintain the integrity of the dependency list using the BCE.



### Rippling Water (Server obj #1663)

+ intro: Welcome!

+ Add new question.

+ 1) What are you doing?

+ 2) How do you clean a horse corral?

- 3) Why do you keep horses inside of the earth lodge?

question:	Why do you keep horses inside of the earth lodge?
answer:	We keep my husband's favorite horse inside at times of bad weather or when there is a threat from another tribe.
verb call:	#1663:start_animation()
depends:	1
<input type="button" value="save changes"/> <input type="button" value="delete"/> <input type="button" value="move up"/> <input type="button" value="move down"/> <input type="button" value="view entry"/>	

+ 4) Do you keep any other horses inside?

+ 5) Where are the rest of your horses?

+ View dictionary links.

Figure 3. The Bot conversation interface allows a content expert to construct a conversation between a player and an agent. Clicking on 'Add new question' or on an existing question opens the question editing form where the question's text and the agent's response are entered. The verb call field is where a function call can be entered. Questions can be moved up and down the list using the move up and move down buttons. The view entry button displays a preview of how the question will look in the IVE. The 'Delete Bot' button removes the agent from the IVE.

Sometimes the content experts may want the asking of a question to trigger a specific event such as assigning the student a new task, displaying a movie, or starting an animation. To facilitate these events, a function call element can be associated with each question. This requires interaction outside of the BCE between content experts and the IVE's programmers to design the event's actions. A function call is then provided to the content experts who associate it with the question of their choice by placing it in a question's function call element.

The 'Delete Bot' button at the bottom of the page removes the agent from the IVE. Agent data is archived in the PostgreSQL database so the agent can be restored if desired.

If a question contains terminology that may not be familiar to a student, there is an option to insert a dictionary link into any element of the agent's text.

After creating or editing a conversation, changes are saved to the database and when the content experts click the 'Dump XML' button, a PHP script extracts the agent conversation entries from the database, converts them into an XML file and commits the XML file to the CVS repository.

An earlier instance of the On-A-Slant IVE was deployed using Java Web Start. At that time the PHP script also added the new or modified XML file to the deployment package, and client installations were automatically updated upon connection. The Java 1.6 release redefined the Web Start cache directory structure making this impractical and necessitated the switch to an installed application.

## 4.1.2 Bot Editor Implementation

The Bot Editor is a collection of PHP scripts located on a remote web server. Accessing the editor in a web browser runs scripts that dynamically build the HTML pages that are viewable to the user.

### 4.1.2.1 List of Agents

The 'list of agents' web page, shown in Figure 2, is the Bot Editor's start or 'home' page. Accessing this page runs a PHP script that does three tasks.

First, the script opens a socket connection to the LambdaMOO server and writes the `get_bot_list()` function call string to the connected file stream. All the IVE's software agents are stored as objects on the LambdaMOO server. They are children of the generic avatar object and contain the hierarchical conversation tree.

The following PHP code is used to connect to the LambdaMOO server to request a list of agents.

```
1.  $fp = fsockopen ($ip_address, $port, &$errno, &$errstr, 30);
2.  if (!$fp) {
3.      echo "$errstr ($errno)<br>\n";
4.      exit();
5.  } else {
6.      $line = fgets ($fp, 100);          // Purge Buffer
7.      fputs ($fp, "get_bot_list\n");
8.      $line = fgets ($fp, 100);
9.      list($keyword, $status, $bot_list_str) = split ('\\|', $line, 4);
10.     fclose ($fp);
11. }
```

Line 1 of the code establishes a connection to the LambdaMOO server. Line 2 verifies that the connection is established. If the connection is successfully created, the code continues by purging the file stream buffer of any residual garbage data in line 6 and sending the function call in line 7. A string

representation of a 2-dimensional list of agents is passed back to the script in line 8. Line 9 separates the agent list (\$bot\_list\_str) from the function call's status message. Line 10 closes the connection.

When the LambdaMOO server receives the get\_bot\_list() function call, it retrieves a list of the active agents in the IVE, determines the agents' id, name, location, and job description and assembles that information into an array of 4-tuples defined as (bot id, bot name, bot location, bot job). In order to pass this information back to the PHP script, the 2-dimensional array is converted into a nested character delimited string. In this case the string is a semi-colon delimited list of comma delimited lists.

The following LambdaMOO code is used to assemble a list of active software agents and return that list to the calling PHP script.

```
1.  try
2.    bot_list = {};
3.    bots = this:sort_by_area($object_utils:leaves($g.gen_avatar));
4.    for bot in (bots)
5.      if ($object_utils:isa(bot.location, $g.room))
6.        bot_str = tostr(tonum(bot), ",", bot.name, ",",
7.                      bot.location.name, ",", bot.job);
8.        bot_list = {@bot_list, bot_str};
9.      endif
10.   endfor
11.  except V (ANY)
12.    "There is a problem. Log it.";
13.    $g.bug_db:log_error(V);
14.    notify(player, "#BOT_LIST|ERROR|O|");
15.    return;
16.  endtry
17.  notify(player, tostr("#BOT_LIST|OK|",
18.                      $string_utils:from_list(bot_list, ";"), "|"));
19.  return;
```

Line 3 retrieves a list of all agents in the IVE and sorts the list according to agent location (bots). Line 4 starts the iteration through the list of bots. Line 5 determines if the current bot is an active agent by determining if it is in a valid

game location (\$g.room). Line 6 creates the 4-tuple of agent information and line 7 adds the tuple to the list of active agents (bot\_list). Lines 10-14 handle an error situation, and line 16 creates and returns the nested character delimited string of agent info.

If the function calls status message is "OK", the script parses the nested character delimited string representation of the list of agents into a 2-dimensional array. If the return status is not "OK", then the value returned in the message field has a description of the problem that occurred with the function call.

The second task performed by the PHP script is to compare the list of active agents with entries in a PostgreSQL database and update the database if discrepancies are found.

As far as the IVE is concerned, software agent conversations are stored in XML files, one file for each agent, and on the LambdaMOO server, one object for each agent. This is inconvenient for editing outside the development IDE, especially in a web browser, as each edit would require:

- Retrieving the relevant agent information from the MOO
- Checking the associated XML files out of CVS
- Reading in the XML file and parsing out the relevant information to display in the editor
- Combining the information from the XML file with the information from the MOO
- Maintaining this information between post actions (either by writing directly back to the MOO and XML files or storing the information in a temporary file
- Updating the information on the MOO when finished with an agent
- Writing the changes to the XML file when finished with an agent
- Checking the XML file back into CVS

In order to simplify the editing process, software agent conversation info is stored in a PostgreSQL database (Figure 4). The database contains three tables:

1) bot\_list which contains the software agent's ID and name, 2) bot\_intro that contains an agent's intro message, and 3) bot\_questions that stores the elements of the conversation.

A conversation element is composed of a question, its answer, the order the question appears in the conversation box, its place in the conversation hierarchy, and a function call if the asking of a question is going to trigger an event. PostgreSQL's advanced query functions are used to manipulate data while editing.

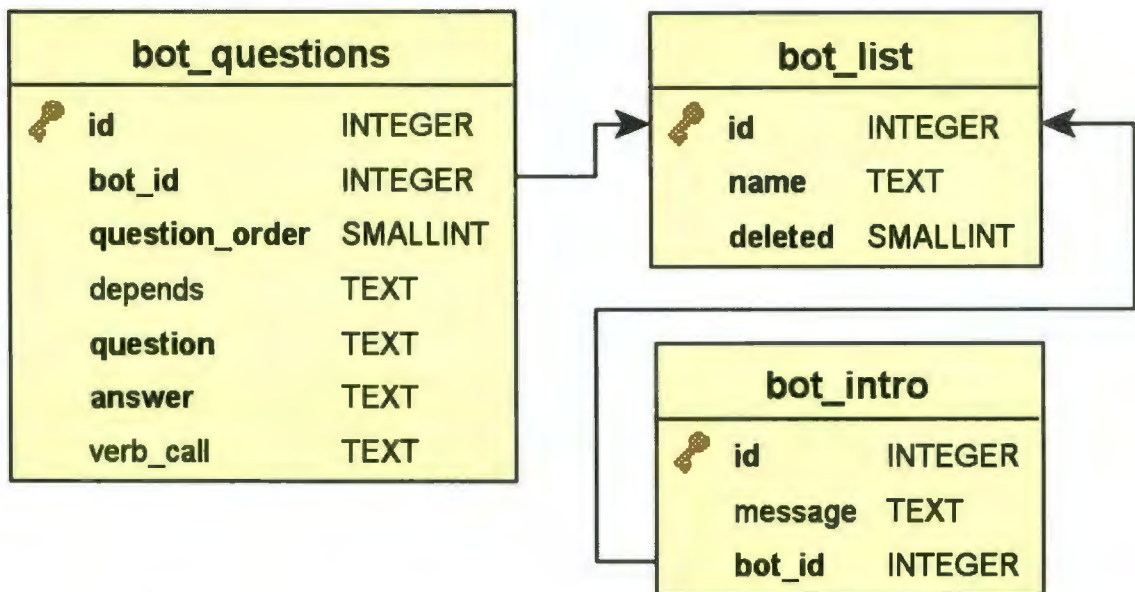


Figure 4. Graphical representation of the relationships among the three database tables that store software agent conversations.

When the Bot Editor's home page is accessed, the PHP script retrieves the list of agents from the LambdaMOO server and iterates through this list comparing the information with the agent's corresponding entry in the database. If an entry for the agent is found in the database, the script compares the data and updates the database entries if they are different. The reason the database is updated and not



the MOO is because the developers have agreed to use the Bot Editor for maintaining the software agent's conversations. Thus we make the assumption that the current state of the MOO is always correct. This could lead to problems if the developers failed to follow convention but it was decided that the minimal risk outweighed the effort needed to add a routine to resolve concurrent version issues.

If an entry for an agent is not found in the database, one is inserted. Correspondingly, if there is an agent entry in the database and not one in the agent list, the entry is removed.

The final task of the script is to create the table of agents and assemble the HTML web page displayed in Figure 2. A system diagram for the Bot Editor can be found in Figure 5.

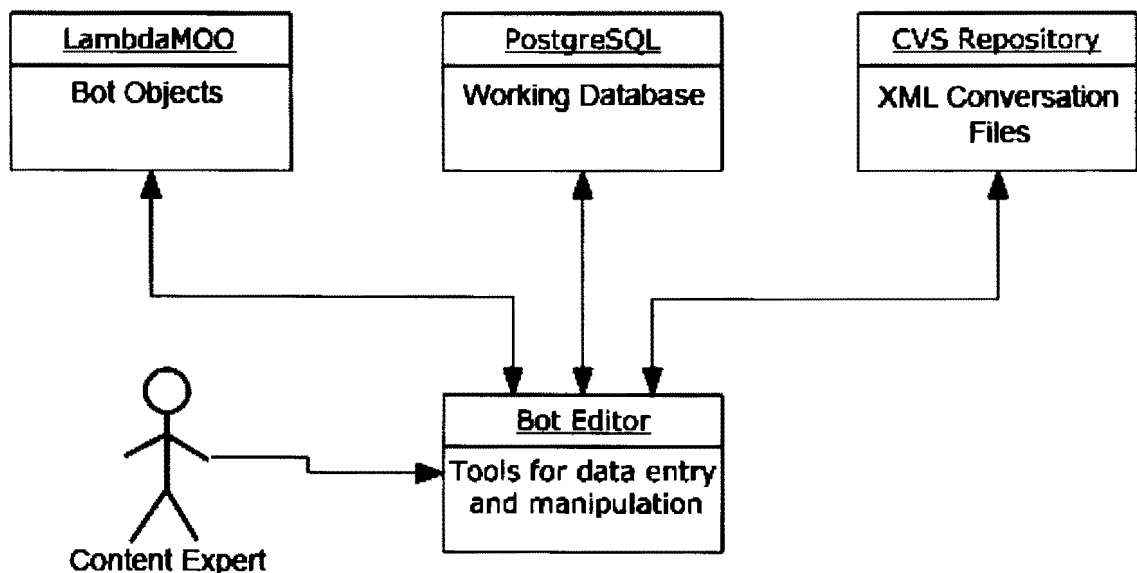


Figure 5. Bot Editor system diagram. The editor retrieves a list of active software agents from the LambdaMOO server and combines them with agent information extracted from XML files stored in the CVS Repository. The combined information is stored in the PostgreSQL database for fast access and to make use of PostgreSQL's query toolset.

#### 4.1.2.2 Add New Bot

Clicking on the Add New Bot button on the Bot Editor's home page, opens up an HTML form that collects the information needed to create a new agent. This information is bot name, bot job (roll in the environment), and bot location. Before submitting, form data are validated using JavaScript. Each element is checked, and if found invalid or incomplete the JavaScript routine notifies the user to correct the problems. Users are not able to continue until all the problems have been addressed. Notification is handled on the client side without any information being submitted to the web server.

The form posts to a PHP script on the web server that collects the element information, opens a connection to the LambdaMOO server, and writes the `create_new_bot()` function call to the connected stream. The procedure for sending the `create_new_bot()` function call is identical to that of the `get_bot_list()` function call except that it has three parameters.

```
1. fputs($fp, "create_new_bot\n");
2. fputs($fp, "$bot_name\n");
3. fputs($fp, "$bot_job\n");
4. fputs($fp, "$bot_location\n");
```

The function call returns a string containing the new agent's object ID and a status message. If the returned status message is "ERROR", execution of the add bot script halts with an error message notifying the user that the new agent was not created. A traceback of the LambdaMOO error event is automatically stored in a bug tracking database (Green, 2000) for later examination by the developers. If the function call's status message is "OK" the script writes an empty XML agent

conversation template file and calls a Perl script to add the new XML file to the CVS repository (Figure 6).

```
<?xml version="1.0"?>
<bot_questions>
  <bot id="2623" name="New Agent Name">
    <intro></intro>
  </bot>
</bot_questions>
```

Figure 6. A sample XML bot conversation template file.

The following Perl script was used instead of an equivalent PHP script to perform the CVS manipulation functions because the verbose output from the CVS commit command would cause PHP shell functions to fail. No explanation for this was ever found, and with the implementation of the Perl workaround a solution was never pursued.

```
1.  #!/usr/bin/perl
2.  use lib '/var/www';
3.  use CGI;
4.  use CGI::Carp qw(faltsToBrowser);
5.  $q = new CGI;
6.  $id = $q->param("bot_id");
7.  $path = $q->param("file_path");
8.  $cvspath = $q->param("cvspath");
9.  chdir($path);
10. system("cvs -Q update");
11. system("/bin/cp " . $cvspath . "/bot_$id.xml .");
12. system("cvs -Q add bot_$id.xml");
13. `cvs -Q commit -m "Update from Bot Editor web site."`;
14. print $q->redirect(".");
```

Lines 6-8 read in the variables passed from the PHP script. Line 9 changes the current directory to the location of the checked out CVS repository files. Line 10 updates the checked out files, and line 11 copies the new XML file into the repository directory. Line 12 adds the new file to the repository, and line 13 commits the changes to the CVS repository. Line 14 returns control to the Bot Editor's home page.

### 4.1.2.3 Edit Bot

Clicking on one of the entries in the list of agents brings up the edit conversation interface (Figure 3). The interface is created by a PHP script that queries the PostgreSQL database for the agent's initial greeting message and a list of all the agent's conversation questions. The script then builds a table containing three types of forms. It first creates a form for editing the initial greeting message, then a form for adding a new question, and these are followed by a series of forms for editing questions; one form for each question. Form elements are populated with the information retrieved from the database.

Having a large number of form elements displayed on a page can be confusing to the user so to clean up the interface, a JavaScript routine is used to dynamically collapse inactive forms. This can be seen in Figure 3 where the form for question 3 is active and the forms for questions 1, 2, 4, and 5 are collapsed as are the forms for the initial greeting message, dictionary links, and adding a new question.

Each type of form in the edit conversation interface submits to a separate PHP script that makes the appropriate changes to the database. After making the changes to the database, each script redirects back to the edit conversation interface.

The 'Delete Bot' button found on the bottom of the edit conversation interface doesn't actually delete the agent from the IVE. It archives the agent by moving its object to an archive area and marks its database entry as deleted. This

allows the restoration of agents that may have been deleted accidentally or are currently not in use.

#### 4.1.2.4 Dump the XML

Clicking on the 'Dump XML' button executes a PHP script that builds XML conversations files and updates the MOO objects for each of the active software agents in the IVE.

The first step to building the XML files is to query the PostgreSQL database for a list of agents. Working linearly through this list, the script progressively builds an XML file for each agent using double iterative loops.

First the outer loop collects the information unique to each agent. ID, name, and job already exist in the list of agents. The initial greeting message is retrieved from the `bot_intro` table in the database. The outer loop then queries the database to retrieve the questions associated with the agent currently being processed. This information is found in the `bot_questions` table. Then an inner loop iterates through this list of questions to assemble the text, answer, and function call entries. A dependency list for each question is also created at this time but is stored in a 2-dimensional array until all the agent's questions have been retrieved. When all the agent's questions have been processed the outer loop writes the XML file to disk and connects to the LambdaMOO server to write the agent's dependency list from the array. This is done by converting the array to a nested character delimited string and writing the `set_depends_list()` function call to the connected stream.

```
1. $depends_str = implode("|", $depends_list);
2. fputs($fp, "set_depends_list\n");
3. fputs($fp, "$bot_id\n");
4. fputs($fp, "$depends_str\n");
```

The `set_depends_list()` function on the LambdaMOO server then converts the dependency list string back to a 2-dimensional array and sets the agent's MOO object's question tree property.

After creating an XML file and setting the dependency list for each agent the script was used to "jar up" the XML files, sign the jar file, and move the jar file to the Web Start directory. As mentioned earlier, changes in the Java 1.6 Web Start cache directory structure negated individual package updates making this impractical, so these functions were removed.

Similar to adding a new agent, the final task when dumping the XML files is to call a Perl script that commits the new XML files to the CVS repository.

Figure 7 contains a sample agent conversation XML file. The hyperlinked term 'travois', in the first answer, points to a travois dictionary entry.

```
<?xml version="1.0"?>
<bot_questions>
  <bot id="341" name="(Dog Travois Woman)">
    <intro>Hi!</intro>
    <question id="1">
      <text>What is attached to this dog?</text>
      <answer>It is called a <a
href=http://onaslant.ndsu.edu/dictionary/Dictionary.php?method=def&id=187>travois</a>. These two poles are harnessed to the dog's chest. A basket is attached between the poles for carrying items.</answer>
    </question>
    <question id="2">
      <text>What are you carrying on the travois?</text>
      <answer>This dog is carrying the butchered parts of the bison. She can pull up to 60 pounds. When we get to camp, I will unpack the load and we will go back for another.</answer>
    </question>
  </bot>
</bot_questions>
```

Figure 7. The bot conversation XML file for the On-A-Slant Virtual Village Dog Travois Women software agent. For comparison, see Figure 6. The `id` field in the `<bot>` tag refers to the agent's object number on the LambdaMOO server while the `id` field in the `<question>` tag refers to the question's display order.

## 4.2 Task Editors

Learning in IVEs is guided by the completion of goals and tasks. Students are encouraged to explore the world, perform experiments, and learn concepts. By following a set of goals, students learn how to do tasks that are important to the field under study.

Goals make up the overall focus of the IVE's teaching element, and tasks are the strategically ordered components students must realize and undertake to reach their goals. Using the constructivist proposition of scaffolded learning (Vygotsky, 1986), students are given goals and must undertake a set of tasks to reach those goals. Task models are specific to the discipline under study. Tasks force the students into learning by doing and can be understood as rehearsals for real-world problems. Furthermore, the tasks are the principal place in which the software tutoring agents in the IVE can monitor and assess students' performances for correction and assessment (Slator et al., 1999; Brandt et al., 2006).

As an example, one of the goals of the eGEO IVE is to teach students how to calculate the velocity of a river. This is done by dropping an orange into the virtual river and measuring how long it takes to travel a measured distance.

In order to accomplish this goal a student needs to complete five tasks.

1. Get Distance Wheel
2. Measure Distance
3. Get Stopwatch
4. Measure Orange Travel Time
5. Calculate Orange Velocity

Tasks are also important for assessment, so that the instructor can tell if a student is capable of completing a particular task from the subject material. Goals also act as important scaffolding tools. By giving students tasks that are initially easy and contain many hints, they gain initial success, increasing the odds of them continuing to play the game. The goals, however, are generally designed to get progressively harder, resulting in more challenging learning moments as time progresses.

For example, when a student receives the 'Hide Scraper' task they are shown a picture of the bone part of an artifact found while excavating. They are asked to explore the village to locate the item and discover its purpose. Artifacts are usually located near software agents who have knowledge about the items. Interacting with these agents provide useful information about the artifacts. When a student has completed the task's activities, in this case locating the hide scraper artifact and discovering its purpose, they are given a set of formative assessment questions intended to help guide students toward their learning objectives.

Similar to managing software tutors, creating and maintaining these tasks is a joint endeavor between content experts and programmers. Before the implementation of the content creation tools, content experts would create a written design that describes the task, its assessment questions, and any sub tasks it can assign. This design was then given to the programmers to implement. If the task needed modifications or adjustments, the content experts would have to contact the programmers to make the changes. Similarly, if the programmers had questions they would have to contact the content experts for a resolution.



The Task Editor was created to allow anyone on the development team to create or edit task descriptions and associated assessment questions, using a web based HTML interface. The editor also allows for the creation of multiple choice, short answer, or essay assessment questions using dynamic forms.

#### **4.2.1 Using the Task Editor**

Similar to the Bot Conversation Editor, the Task Editor consists of three interfaces; a list of tasks, a form for adding tasks, and a form for editing task content and associated formative assessment questions.

The list of tasks (Figure 8) displays all the active tasks in the environment. Clicking on one of the tasks in the list brings up a form for editing the task's description and assessment questions. There are also two buttons at the bottom of the page. The 'Add New Task' button brings up a form where you can enter information necessary to create a new task. The 'Dump XML' button updates the CVS repository to reflect any changes made using the editor.

The interface for adding new tasks collects the minimal information needed to create a new task (task name), creates the task object on the LambdaMOO server and adds an empty XML task template to CVS repository. Additional task descriptions and assessment question information is entered using the editing interface.

The edit task interface (Figure 9) allows the content expert to enter the task description. The description is the text and/or images that a student receives when they are assigned a task and an icon image that is used as a visual representation of the task.

task name	object #
<a href="#">Visit Slate</a>	2929
<a href="#">Laptop Quiz</a>	2547
<a href="#">Talk to Jade 1</a>	1996
<a href="#">Initial Water Testing</a>	1995
<a href="#">Calculate Discharge</a>	2312
<a href="#">Find Gold Task</a>	2433
<a href="#">Visit Slate (Post)</a>	874
<a href="#">Laptop Quiz (Post)</a>	368
<a href="#">Get Water Current Meter</a>	2341
<a href="#">Measure River</a>	2344
<a href="#">Send Discharge Report</a>	2345
<a href="#">Mining Plant</a>	2097
<a href="#">Geo</a>	2103
<a href="#">Get Distance Wheel</a>	2365
<a href="#">Measure Distance</a>	2367
<a href="#">Get Stopwatch</a>	2380
<a href="#">Measure Orange Distance</a>	2382
<a href="#">Calculate Orange Velocity</a>	2383
<a href="#">Get Water Quality Test Kit</a>	2385
<a href="#">Create Line Graphs</a>	2421
<a href="#">Water Quality Report</a>	2464
<a href="#">Gather Eutrophication Water Samples</a>	890
<a href="#">Play Aqua Survivor</a>	6708



Last XML.jar dump: 4:48 PM September 14, 2010.

**+ View database maintenance trace.**

Figure 8. The list of tasks displays active tasks in the environment. Clicking on one of the tasks in the list brings up a form for editing task descriptions and assessment questions. There are two buttons at the bottom of the page. The 'Add New Task' button brings up a form where you can enter information necessary to create a new task. The 'Dump XML' button updates the CVS repository to reflect any changes made using the editor. The column of object numbers is included for easy reference between the content expert's and programmer's view of the task's program object.

This description includes a small set of activities that the student needs to complete in order for them to progress to the next step in the completion of their current learning goal. The editor also allows formative assessment questions to be attached to the task. Assessment questions are added by choosing the 'Add new question' option and filling out the presented form. The form allows the content expert to choose the number of possible answers to the question. A limit of fifteen answers was chosen to be a practical upper limit for the number of answers. Selecting zero answers implies an essay question, one answer implies a short answer question, and two or more answers imply a multiple choice question. For multiple choice questions, check boxes are provided to indicate which answers are correct. There is the ability to select multiple correct answers and areas to enter tutoring responses for correct and incorrect answer submissions. For questions with multiple correct answers, responses can be for all answers correct, all answers incorrect, or a combination of correct and incorrect answers.

Once a question has been added to a task they can be edited by clicking on one of the questions to bring up the edit question form. Question display order can be manipulated using 'move up' and 'move down' buttons. Unwanted questions can be removed using the 'delete' button.

After creating or editing a task, changes are saved to the PostgreSQL database and when a content expert clicks the 'Dump XML' button, a PHP script extracts the task's entries from the database, converts them into an XML file and commits the XML file to the CVS repository. A list of question ID's is also sent to the LambdaMOO server so player objects know which questions are available to be answered.

### Hide Scraper (Server obj #1914)

We have just found an artifact while excavating. We're sending a picture to you now.



+ description:

What information can you discover about this? You may have to explore the village to locate the artifact. It may look slightly different than the picture. Once you've found it, you'll need to record your findings, using the field book, and send it back here for examination.

+ Add new question.

- 1) What is the artifact?

Q: This is a stone object. What is it?

update question    delete question    move up    move down    add answer

1) Spoon

update answer    delete    up    down

A: 2) Squash Knife

update answer    delete    up    down

3) Hide Scraper

tutoring responses

all correct: You are correct; this artifact is a hide scraper!

all incorrect: After careful examination of the artifact, we found that your answer is incorrect. Explore the village for this artifact and ask questions of anyone using it.

first correct:

second correct:

+ 2) For what purpose was the artifact used?

+ 3) What else have you discovered about this artifact?

+ View dictionary links.

Figure 9. The edit task interface allows the content expert to enter the task's description and assessment questions.

## 4.2.2 Task Editor Implementation

The Task Editor is a collection of PHP scripts located on a remote web server. Accessing the editor in a web browser runs a PHP scripts that dynamically build the HTML pages that are viewable by the user.

### 4.2.2.1 List of Tasks

Like the software agents mentioned earlier, Tasks in IVE's are objects on the LambdaMOO server. They are children of the generic task object and contain additional functions for assigning tasks to students and calculating point values for completing tasks. Opening the Task Editor's 'list of tasks' runs a PHP script that has three functions.

This process mirrors the bot conversation process. First the script opens a socket connection to the LambdaMOO server and writes the `get_task_list()` function call string to the connected file stream. When the LambdaMOO server receives the function call it retrieves a list of active task IDs and names and assembles that information into an array of 2-tuples defined as (task id, task name). In order to pass this information back to the PHP script the 2-dimensional array is converted into a nested character delimited string. In this case, a pipe delimited list of semi-colon delimited lists.

If the function calls status message is OK the script parses the nested character delimited string representation of the list of tasks back into a 2-dimensional array. If the return status is not OK, then the value returned in the message field has a description of the problem that occurred with the function call.

As with software agents, IVE task information is stored in a combination of XML files and LambdaMOO object properties. One XML file and one object per task. So to simplify the editing process, task information is stored in a PostgreSQL database. Thus, like with the Bot Editor, the second function of the PHP script is to compare the list of active tasks with entries in a PostgreSQL database and update the database if discrepancies are found.

The database contains four tables (Figure 10): 1) `task_list` which contains the task's ID and Name, 2) `task_description` that contains a task's description and icon information, 3) `task_questions` (Figure 11a) that stores the text of the formative assessment questions and tutoring messages for correct and incorrect responses, and 4) `task_answers` (Figure 11b) that contains all the possible answers for a question.

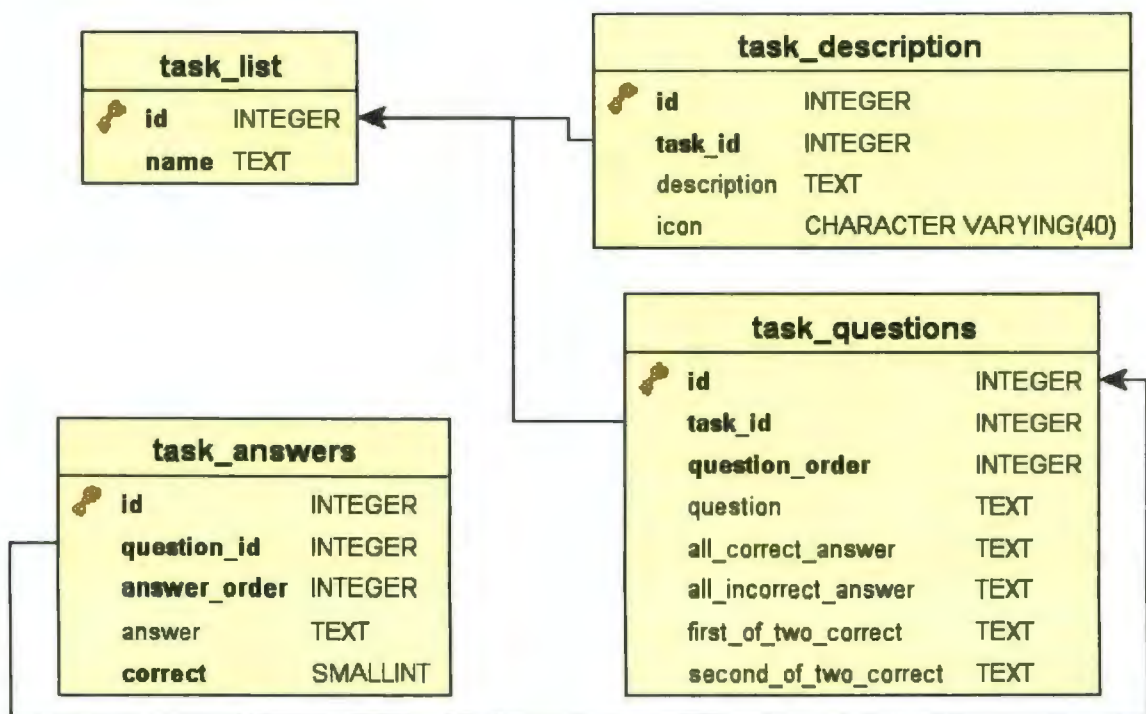




Figure 10. Graphical representation of the relationships among the four database tables, which store task information.

task_questions	
 <b>id</b>	INTEGER
<b>task_id</b>	INTEGER
<b>question_order</b>	INTEGER
question	TEXT
all_correct_answer	TEXT
all_incorrect_answer	TEXT
first_of_two_correct	TEXT
second_of_two_correct	TEXT

(a)

task_answers	
 <b>id</b>	INTEGER
<b>question_id</b>	INTEGER
<b>answer_order</b>	INTEGER
answer	TEXT
<b>correct</b>	SMALLINT

(b)

Figure 11. (a) The *task\_questions* database table. *id* is a unique question identifier. *task\_id* is a foreign key referring to the task to which the question is associated. *question\_order* is the order the question is displayed in a list of questions. *question* contains the text of the question. *all\_correct\_answer* contains the agent's response if all parts of the question are answered correctly. *all\_incorrect\_answer* contains the response if all parts of the question are answered incorrectly. *first\_of\_two\_correct* contains the response if the first part of the answer is correct and the second part is incorrect. *second\_of\_two\_correct* contains the response if the second part of the answer is correct and the first part is incorrect. (b) The *task\_answers* database table. *id* is a unique answer identifier. *question\_id* is a foreign key referring to the question to which the answer is associated. *answer\_order* is the order the answer is displayed in a list of answers. *answer* contains the text of the answer. *correct* indicates the answer is one of the correct answers for a question.

The final task of the script is to create the table of tasks and assemble the Task Editor web page, displayed in Figure 8.

#### 4.2.2.2 Add New Task

Clicking on the Add New Task button opens up a HTML form that collects the information needed to create a new task. Before submitting, form data are validated using JavaScript. Each element is checked and if found invalid or incomplete, the JavaScript routine notifies the user to correct the problems.

Notification is handled on the client side without any information being submitted to the web server.

The form posts to a PHP script on the web server that collects the element information, opens a socket connection to the LambdaMOO server, and writes the `create_new_task()` function call to the connected stream. The procedure for sending the function call is identical to that of the `get_bot_list()` function call except that it has one parameter.

1. `fputs($fp, "create_new_task\n");`
2. `fputs($fp, "$task_name\n");`

The function call returns the new task's object ID as a return value. If the function call's status message is "OK" the script writes an empty XML task template file (Figure 12) and calls a Perl script to add the new XML file to the CVS repository.

```
<?xml version="1.0"?>
<tasks>
  <task id="186" name="New Task Name" icon="">
    <description></description>
  </task>
</tasks>
```

Figure 12. A sample XML task template file.

#### 4.2.2.3 Edit Tasks

Clicking on one of the entries in the list of tasks brings up the edit task interface (Figure 9). The interface is created by a PHP script that queries the PostgreSQL database for the task's description and assessment questions along with the answers for those questions and the associated remediation responses. The script then creates a table containing four types of forms. There is a form for editing the task's description, a form for adding a task icon, and a form for adding



a new assessment question. These three are followed by a series of forms for editing assessment questions and their associated answers. Form elements are initialized with information retrieved from the database and each type of form submits to a separate PHP script that makes the appropriate changes to the database and redirects control back to the edit task interface when finished.

Since assessment questions can have a variable number of answers, from an essay answer question with no answers and a short answer question with one answer, to a multiple choice question with as many as fifteen, forms for adding new questions are dynamically created using a JavaScript routine. The following script generates the required elements for a new question form containing, n, answers.

```
1.  function createForm(n) {
2.    data = "<table>";
3.    inter = "";
4.    if(navigator.platform == "MacPPC") // A hack to fix for Macs
5.      size = 67;
6.    else
7.      size = 87;
8.    if (n < 16 && n > -1) {
9.      for (i=1; i <= n; i++) {
10.         data = data + "<tr><td>" + i + "<\td><td>" + "<input
type='text' size='" + size + "' name='" + inter + "a" + i + inter +
"><input type='checkbox' name='" + inter + "c" + i + inter +
"><\td><\tr>";
11.      }
12.      data = data + "<\table>";
13.      if (document.layers) {
14.        document.layers.cust.document.write(data);
15.        document.layers.cust.document.close();
16.      } else {
17.        document.getElementById('cust').innerHTML = data;
18.      }
19.    } else {
20.      window.alert("Please select up to 15 entries.");
21.    }
22.  }
```

When editing an assessment question, the content experts may want to delete or insert an answer to a specific question. Deleting an answer is as simple as deleting its entry from the PostgreSQL database. Inserting an answer requires adding data, which has yet to be entered into a form element, to the database. To accomplish this, without causing a page refresh, a JavaScript routine was created to dynamically add the elements required for an additional answer to the existing question form. The following script adds a form element to the question identified by the parameter `id`.

```
1.     function addRow(id){
2.         var tbody =
document.getElementById(id).getElementsByTagName("TBODY")[0];
3.         var tde1 = document.createElement("TD")
4.         var tde2 = document.createElement("TD")
5.         var row1 = document.createElement("TR")
6.         var td1 = document.createElement("TD")
7.         var in1 = document.createElement("INPUT")
8.         in1.setAttribute("name", "new_answer");
9.         in1.setAttribute("type", "text");
10.        if(navigator.platform == "MacPPC")    // A hack for Macs
11.            in1.setAttribute("size", "67");
12.        else
13.            in1.setAttribute("size", "87");
14.        td1.appendChild (in1);
15.        var in1c = document.createElement("INPUT")
16.        in1c.setAttribute("name", "new_correct");
17.        in1c.setAttribute("type", "checkbox");
18.        td1.appendChild (in1c);
19.        row1.appendChild(td1);
20.        row1.appendChild(td1);
21.        var row2 = document.createElement("TR")
22.        var td2 = document.createElement("TD")
23.        var in2 = document.createElement("INPUT")
24.        in2.setAttribute("name", "option");
25.        in2.setAttribute("type", "submit");
26.        in2.setAttribute("class", "button");
27.        in2.setAttribute("value", "add answer");
28.        td2.appendChild (in2)
29.        row2.appendChild(tde2);
30.        row2.appendChild(td2);
31.        tbody.appendChild(row1);
32.        tbody.appendChild(row2);
33.    }
```

Lines 3 through 14 add a text box to the table of answers, lines 15 through 20 add a checkbox, and lines 23 through 32 add a submit button. Lines 10 through 13 in the script are not necessary for the functioning of the script but have been added to adjust the text box element's appearance for the Mac platform.

Similar to the Bot Editor, a JavaScript routine is used to clean up the appearance of the Task Editor by dynamically collapsing inactive forms.

#### **4.2.2.4 Dump the XML**

Clicking on the 'Dump XML' button executes a PHP script that builds XML task question files and updates the MOO objects for each of the active software agents in the IVE.

The functioning of the XML dumping procedure is almost identical to that of the Bot Editor. An outer loop queries the PostgreSQL database for information unique to each task. In this case it is task id, task description and task icon. The outer loop then queries the database for a list of the task's assessment questions. An inner loop iterates through this list of questions to assemble the text and remediation responses for each question. An additional level of looping is needed to query the database for the set of answers for each assessment question.

The other functional difference is with the data sent to the LambdaMOO server. A `set_question_list()` function call sends a 2-tuple list of questions and their correct answers to the MOO.

```
1. fputs($fp, "set_question_list\n");
2. fputs($fp, "$task->id\n");
3. fputs($fp, "$question_list_str\n");
```

As with the Bot Editor, the final teas when dumping the XML files is to call a Perl script to commit the new files to the CVS repository.

Figure 13 contains a sample task XML file for the Hide Scraper artifact identification task.

```
<?xml version="1.0"?>
<tasks>
  <task id="1914" name="Hide Scraper" icon="hide_scraper.gif">
    <description>We have just found an artifact while excavating. We're
sending a picture to you now. <br><br> What
information can you discover about this? You may have to explore the
village to locate the artifact. It may look slightly different than the
picture. Once you've found it, you'll need to record your findings,
using the field book, and send it back here for
examination.</description>
    <mc_question id="1" text="What is the artifact?">
      <answer id="1" text="Spoon"></answer>
      <answer id="2" text="Squash Knife"></answer>
      <answer id="3" text="Hide Scraper"></answer>
      <answer id="4" text="Hoe"></answer>
      <answer id="5" text="Ladle"></answer>
    </mc_question>
    <mc_question id="2" text="For what purpose was the artifact used?">
      <answer id="1" text="To remove hair from the hide of the
animal."></answer>
      <answer id="2" text="To remove fat and flesh from the hide of an
animal"></answer>
      <answer id="3" text="For ceremonial purposes"></answer>
      <answer id="4" text="Used as a cooking utensil"></answer>
      <answer id="5" text="Used to draw water from the river"></answer>
      <answer id="6" text="Used to catch wild game"></answer>
      <answer id="7" text="Used for food gathering"></answer>
      <answer id="8" text="Used in the preparation of food"></answer>
      <answer id="9" text="Used to separate the hide from the
carcass"></answer>
      <answer id="10" text="Used to slice"></answer>
      <answer id="11" text="To remove weeds"></answer>
      <answer id="12" text="To loosen the soil"></answer>
    </mc_question>
    <essay_question id="3" text="What else have you discovered about
this artifact?">
      </essay_question>
    </task>
  </tasks>
```

Figure 13. The XML task file for the Hide Scraper task. . The id field in the <task> tag refers to the task's object number on the LambdaMOO server. The id field in the <mc\_question> and <essay\_question> tags refers to the question's display order. The id field in the <answer> tag also refers to the answers display order.

### **4.3 Dictionary and Help Editors**

Many of our games contain terms and concepts with which a player may not be familiar with. Embedded dictionary databases are therefore provided to allow players easy access to definitions and explanations. Originally these databases were managed by the software developers or a content expert with some familiarity with database management. This arrangement was inefficient as delays or miscommunication could occur as information was passed from various content experts to those managing the databases.

#### **4.3.1 The Geology Explorer Help Editor**

The Geology Explorer (Slator, 1998) has an extensive database of help entries that need to be updated periodically. This used to be a time consuming and tedious task. A content expert would provide a list of additions and changes to someone with programming experience. The programmer then had to manually edit the help database on the LambdaMOO server, export said database to a file, insert the file into the CVS repository, build the application, and then upload the application files to the games live Web Start directory.

The Geology Explorer Help Editor was created so that anyone on the development team could add or update help entries using a web based HTML interface.

The Help Editor uses a series of CGI scripts and the LambdaMOO server's built in http service.

The help\_editor object on the LambdaMOO server creates a table of existing Geology Explorer help entries and displays them as a web page on the servers own http port. Clicking on one of the entries brings up an external cgi created HTML form that allows users to edit or delete the specific entry and its description. The forms were created using external scripts as a way to get around LambdaMOO's limited http request implementation. Changes are stored back to the LambdaMOO server until the entire database is dumped to the CVS repository.

The web page also provides the options to add a new help entry and to dump the help database.

Adding a new help entry is similar to the editing function mentioned earlier. Clicking on the Add New Entry link brings up a form where the user can enter a new entry and description.

The Dump the Help Database option exports the updated help entries to a file on the server and calls a CGI script to finish processing. The script merges the help entries into a java class file. Commits the java file to the CVS repository, compiles the java class file, and uploads it to the game's application directory where it can be accessed by the Applet and Web Start game clients.

#### **4.3.2 The On-A-Slant and eGEO Dictionary Editors**

In order to reduce network traffic and improve performance, the On-A-Slant and the eGEO clients were implemented with most non-user specific data stored client side. This resulted in a different implementation method than that of the Geology Explorer Help Editor.

The IVEs accesses the dictionary data from an XML file located in a resources directory. In order to make editing this file easy a copy of the dictionary's data is also stored in a PostgreSQL database (Figure 14). This allows easy access to the data and makes use of PostgreSQL's robust sorting, querying, and retrieval applications.

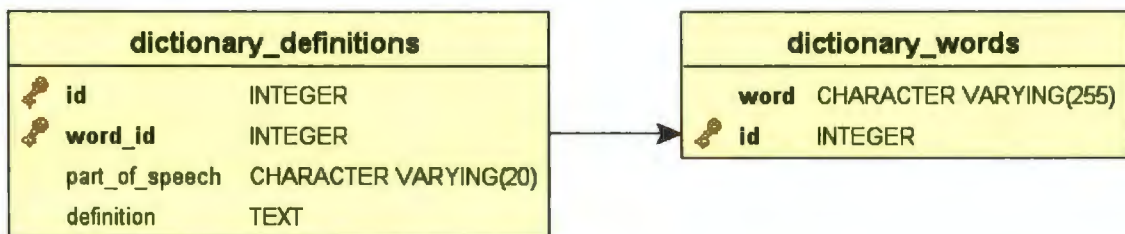


Figure 14. A graphical representation of the two tables that make up the dictionary database.

The Dictionary Editors (Figures 15 & 16) are a series of PHP scripts that retrieve dictionary information from the database, and present it in an easy to understand HTML form.

The form allows the user to view entries as they would appear in the game as well as to select entries to edit, delete, attach images to, or crosslink with other entries. Changes are saved back to the database.

Similar to the Geology Explorer Help Editor, there is an option to export the updated help entries to a file on the server and a call to a CGI script to finish processing. This script extracts the dictionary entries from the database, converts them into an XML file and commits the XML file to the CVS repository.

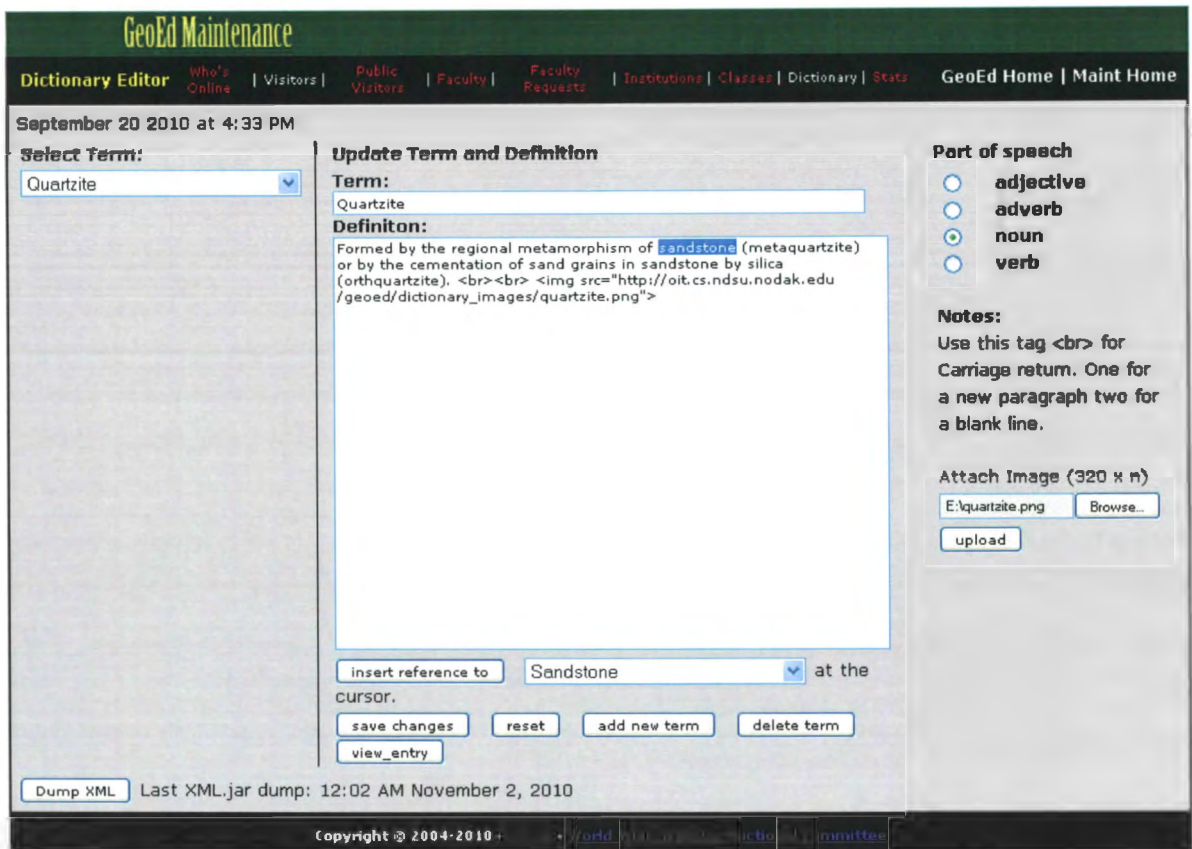


Figure 15. The eGEO Dictionary Editor. The dropdown box on the left lists the entries in the dictionary database. The text box and text area in the center are for editing a term and its definition. The 'insert reference to' button and dropdown box immediately below the text area, allows the cross linking of dictionary entries. For the instance shown here, clicking the 'insert reference to' button will link the highlighted word "sandstone" to the Sandstone dictionary entry. The buttons in the second row below the text area allow the user to add a new entry or to delete, preview or save changes to the current term. The attach image area in the lower right allows the insertion of an image at the cursor. The 'Dump XML' button updates the database and CVS repository to reflect any changes made using the editor.



Select an entry to modify:

Bioturbation

Dictionary Entry:

Bioturbation

Part of Speech:

adjective  adverb  noun  verb

Definition:

Disturbance at an archaeological site due to biological organisms, from roots to burrowing animals. Recognizing bioturbation is important as it creates secondary disposition in site layers. See also `<Reference wordID="103">Transformational Process</Reference>`.

Transformational Process

+ **Add new word.**

Last XML.jar dump: 2:25 PM June 25, 2008

Figure 16. The On-A-Slant Dictionary Editor showing the cross-link between the Bioturbation and Transformational Process entries.

## 5. CONCLUSION

Several successful content creation tools have been produced to facilitate the development of Immersive Virtual Environments. These tools meet the three requirements specified in the introduction. They are cross platform, accessible anywhere internet is available, and do not require the installation of any special software. The web browser they currently use is sufficient.

Three content experts have successfully used the tools to create and maintain a number of task and software agents. Figure 17 shows a screen shot of the On-A-Slant Virtual Village IVE with containing both a software agent and a task created using the content editors Figure 18 contains a similar screen shot from the eGEO IVE.

Even during periods of development where there are no content experts available, programmers have used the Bot Editor to create placeholder agents. Placeholder agents are created to fill a perceived role until they can be evaluated and modified by a content expert.

As with many software applications, there are enhancements that were discussed during and post development but were not feasible to implement due to time or difficulty constraints. The development of these tools will continue with the implementation of new features and enhancements.

The current version of the Bot Editor only allows the creation of conversations where the student asks an agent a question. A new version is in development that allows the integration of conversation parts where the agent asks the student a question.

There is the potential for concurrency problems to arise if multiple content experts try to edit the same conversation part or task question. Implementing atomic transactions within the editors will prevent this problem from occurring.

The ability to upload an image into the CVS image repository should be added to the Task Editor. Currently, images need to be added to the CVS repository manually and then linked by adding the proper image tag into the task description.

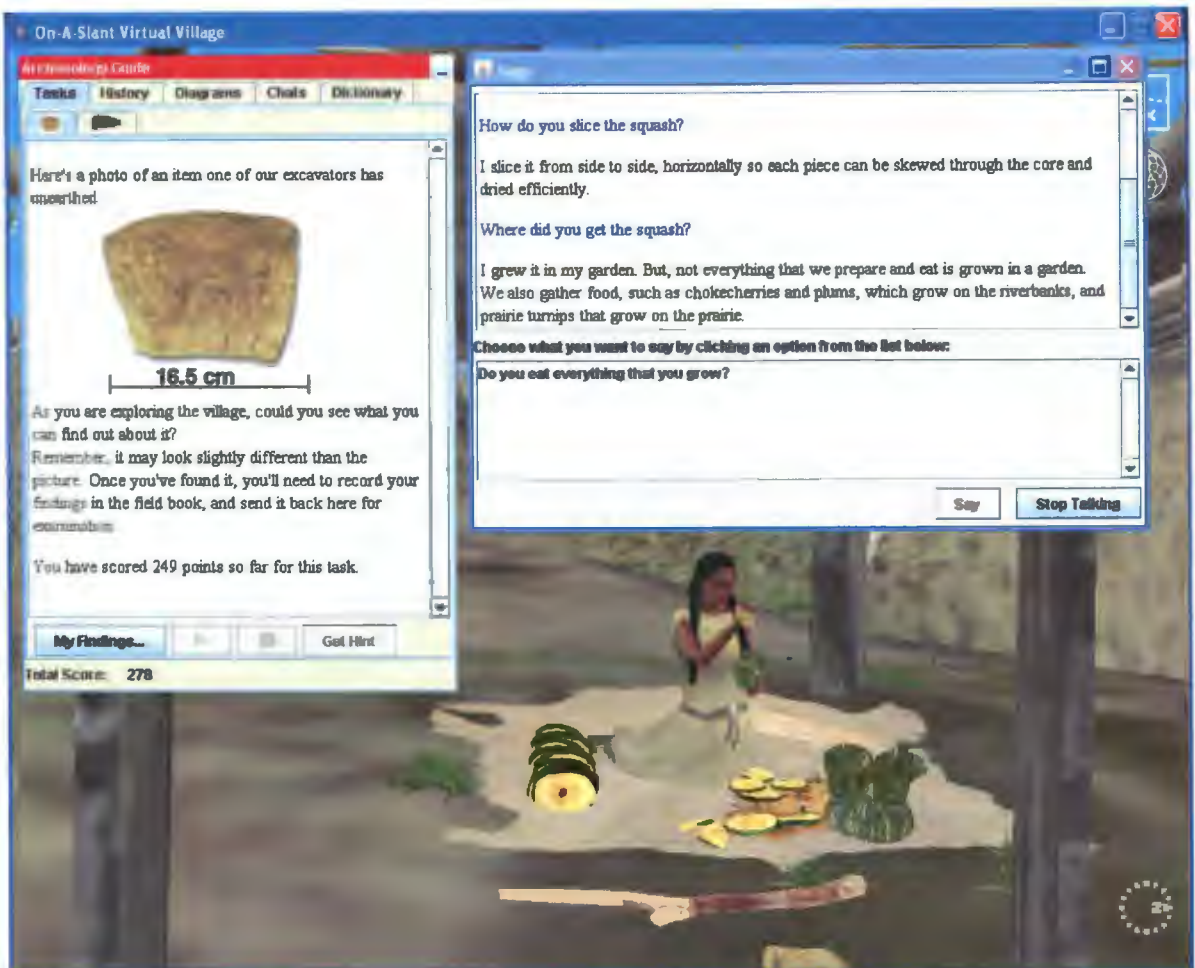


Figure 17. A screen capture from the On-A-Slant Virtual Village IVE. The Archaeology Guide pane in the upper left is showing the Tasks tab. Inside the tab is the content of the Squash Knife Artifact task. The box in the upper right is the conversation interface for the software agent Sage. Available bot questions are listed in the lower pane and the agent's responses are displayed in the upper pane. In the background is the 3D window showing Sage slicing squash with a bone knife.

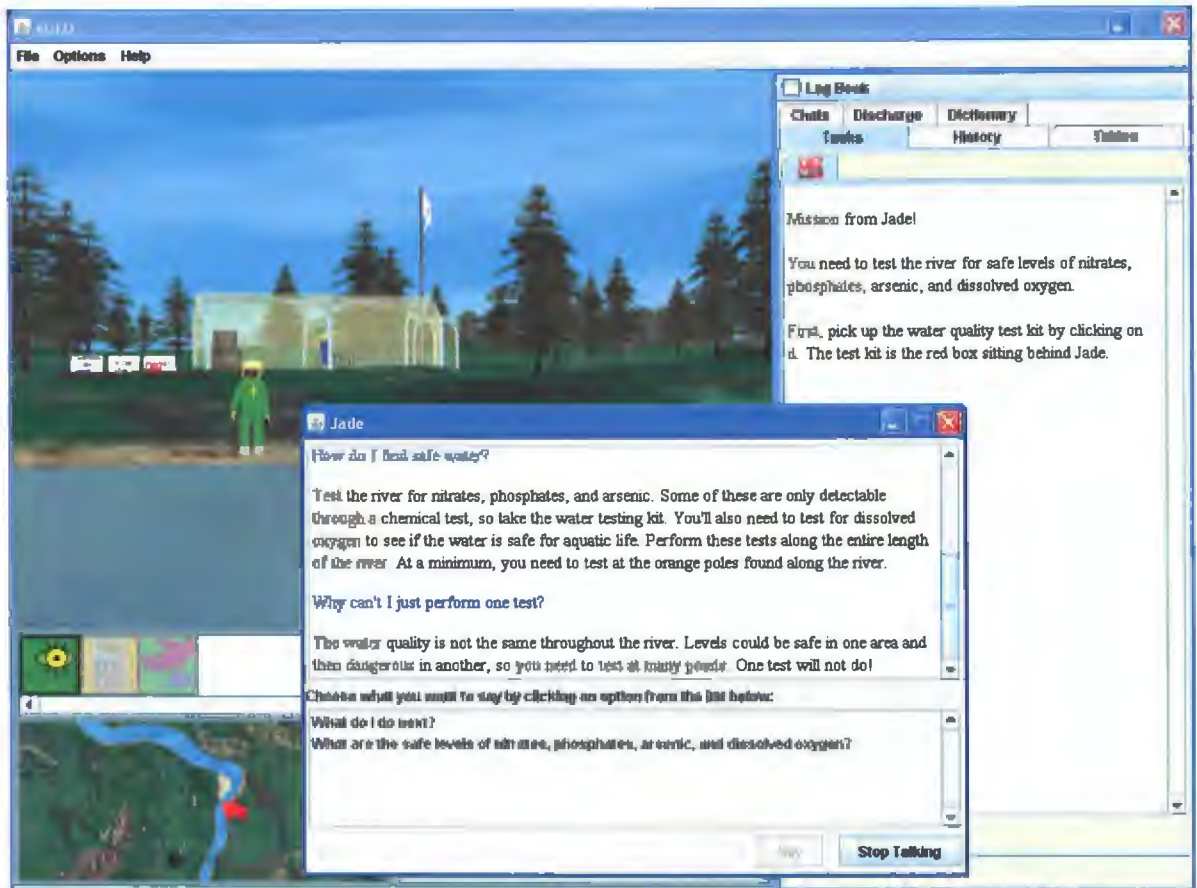


Figure 18. A screen capture from the eGEO IVE. The box in the middle is the conversation interface for the software agent Jade. Available bot questions are listed in the lower pane and the agent's responses are displayed in the upper pane. Behind the conversation pane and to the right is the Log Book pane showing the Tasks tab. Inside the tab is the content of the Mission from Jade! task. In the upper left is the 3D window showing the software agent Jade.

## 6. REFERENCES

Alice. (2010). Alice 2.2 [Software]. Retrieved from <http://www.alice.org>.

Apache Software Foundation, The. (2009). The Apache HTTP Server Project. Retrieved from [http://httpd.apache.org/ABOUT\\_APACHE.html](http://httpd.apache.org/ABOUT_APACHE.html)

Autodesk, Inc. (2010). Maya [Software]. Retrieved from <http://usa.autodesk.com/>

Blender Foundation (2010, Oct 14). Blender [Software]. Retrieved from <http://www.blender.org/>

Borchert, Otto, Lisa Brandt, Guy Hokanson, Brian M. Slator, Bradley Vender, Eric J. Gutierrez. (2010). Principles and Signatures in Serious Games for Science Education, in *Gaming and Cognition: Theories and Practice from the Learning Sciences* Edited by: Richard Van Eck. IGI Global. pp. 312-338.

Brandt, L., O. Borchert, K. Addicott, B. Cosmano, J. Hawley, G. Hokanson, D. Reetz, B. Saini-Eidukat, D. P. Schwert, B. M. Slator, S. Tomac. (2006). Roles, culture, and computer supported collaborative work on Planet Oit. *Journal of Advanced Technology for Learning*, 3(2), 89-98.

Curtis, Pavel. (1997). *LambdaMOO Programmer's Manual for LambdaMOO Version 1.8.0p6*. Xerox, San Francisco, CA. March 1997.

Epic Games, Inc. (2010). UnrealEd [Software]. Retrieved from <http://www.unreal.com/features.php?ref=editor>

Green, Nathan. (2000, July 26). Background Bug Logging for LambdaMOO. Unpublished manuscript.

Hokanson, G., Borchert, O., Slator, B. M., Terpstra, J., Clark, J. T., Daniels, L. M., Anderson, H. R., Bergstrom, A., Hanson, T. A., Reber, J., Reetz, D., Weis, K. L., White, R., & Williams, L. (2008). Studying Native American Culture in an Immersive Virtual Environment. *Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT-2008)*. IEEE Computer Society Press. Santander, Spain. July 1-5. Pg. 788-792.

Jordan, Pamela, Michael Ringenberg & Brian Hall. (2006). Rapidly Developing Dialogue Systems that Support Learning Studies. *Proceedings of ITS06 Workshop on Teaching with Robots, Agents, and NLP*.

Koster, Raph (2006). User Created Content. Raph's Website. Retrieved from <http://www.raphkoster.com/2006/06/20/user-created-content/>

Linden Research, Inc. (2009, June 4). Building Tools. Retrieved from [http://wiki.secondlife.com/wiki/Building\\_Tools](http://wiki.secondlife.com/wiki/Building_Tools)

Milinkovich, M. (2005, Nov 20). About the Eclipse Foundation. Retrieved from <http://www.eclipse.org/org/>

North Carolina State University. (2008, May 21). HIFIVES. Retrieved from <http://ced.ncsu.edu/hifives/games/index.html>

Oracle Corporation. (2010). About MySQL. Retrieved from <http://www.mysql.com/about/>

Perl.org. (2010). The Perl Programming Language. Retrieved from <http://www.perl.org/>

PHP Group, The. (2010, Sept 17). PHP: Hypertext Preprocessor. Retrieved from <http://php.net>

Pittsburgh Science of Learning Center. (2008). LearnLab. Retrieved from <http://www.learnlab.org>.

PostgreSQL Global Development Group. (2010, Feb 25). PostgreSQL: About. Retrieved from <http://www.postgresql.org/about/>

Price, Derek Robert, Ximbiot. (2006, Dec 3). Open Source Version Control. Retrieved from <http://www.nongnu.org/cvs/>

Saini-Eidukat, Bernhardt, Donald P. Schwert, & Brian M. Slator. (2001). Geology Explorer: Virtual Geologic Mapping and Interpretation. *Journal of Computers and Geosciences*. 27(4).

Schwert, Donald, Brian M. Slator, Guy Hokanson, Otto Borchert, Bernhardt Saini-Eidukat, Jeffery Terpstra, John Reber, and Lisa Daniels. (2010). Integrating Mathematics and Chemistry into a Virtual Environment for Geologic Education. 6th Quadrennial Conference of the International Geoscience Educators Organisation (IGEO). Johannesburg, South Africa. 30 August - 3 September.

Slator, Brian M. (1999). Intelligent Tutors in Virtual Worlds. *Proceedings of the 8th International Conference on Intelligent Systems*. Denver, CO. June 24-26, pp. 124-127.

Slator, Brian M., D. Schwert, B. Saini-Eidukat, P. McClean, J. Abel, J. Bauer, B. Gietzen, N. Green, T. Kavli, L. Koehntop, B. Marthi, V. Nagareddy, A. Olson, Y. Jia, K. Peravali, D. Turany, B. Vender, J. Walsh. (1998). Planet Oit: a Virtual Environment and Educational Role-playing Game to Teach the Geosciences. In the *Proceedings of the Small College Computing Symposium (SCCS98)*. Fargo-Moorhead, April. pp. 378-392.

Slator, B.M., P. Juell, P.E. McClean, B. Saini-Eidukat, D.P. Schwert, A. White, C. Hill. 1999. Virtual Environments for Education at NDSU. World Conference on Educational Media, Hypermedia and Telecommunications (ED-MEDIA 99), June 19-24, Seattle, WA.

Slator, Brian M., Lisa M. Daniels, Bernhardt Saini-Eidukat, Donald P. Schwert, Otto Borchert, Guy Hokanson, Richard T. Beckwith (2003). Software Tutors for Scaffolding on Planet Oit. Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies (ICALT). Athens, Greece, July 9-11, pp. 398-399.

Slator, Brian M., Richard Beckwith, Lisa Brandt, Harold Chaput, Jeffrey T. Clark, Lisa M. Daniels, Curt Hill, Phil McClean, John Opgrande, Bernhardt Saini-Eidukat, Donald P. Schwert, Bradley Vender, Alan R. White. (2006). Electric Worlds in the Classroom: Teaching and Learning with Role-Based Computer Games. New York: Teachers College Press. Columbia University. 192 pages.

Spindler, George and Louise Spindler. (2000). Fifty Years of Anthropology and Education, 1950-2000: A Spindler Anthology. Mahwah, New Jersey. Lawrence Erlbaum Associates.

Vygotsky, Lev. (1986). Thought and Language. Cambridge: MIT Press.

Xinhai Ye, M.S. (2001). Virtual Entity Tool: a Tool to Assist Content Specialists in Constructing and Maintaining Hierarchical Objects in Virtual Scientific Domains. Unpublished master's thesis. North Dakota State University. Fargo, ND.

Zelenak, Jozef, M.S. (1999a). Conversation Constructor for Agents in Educational Simulation Environments. Unpublished master's paper. North Dakota State University. Fargo, ND.

Zelenak, Jozef. (1999b). The ZeleCon Conversation Constructor [Software]. Retrieved from <http://wwwic.ndsu.edu/~moadmin/zelecon/>