A Multi-user Desktop Virtual Environment for Teaching Shop-keeping to Children

Brian M. Slator, Harold Chaput (a), Robert Cosmano,
Ben Dischinger, Christopher Imdieke, Bradley Vender

Computer Science Department
North Dakota State University
Fargo, ND 58102 USA

(a) The University of Texas at Austin
Computer Science Department
1 University Station
Austin, Texas 78712

Contact: slator@cs.ndsu.edu

A Multi-user Desktop Virtual Environment for Teaching Shop-keeping to Children

Abstract

Virtual role-playing environments can be a powerful mechanism of instruction, provided they are constructed such that learning how to play and win the game contributes to a player's understanding of real-world concepts and procedures. North Dakota State University (NDSU) provides students with environments to enhance their understanding of geology (Planet Oit), cellular biology (Virtual Cell), programming languages (ProgrammingLand), retailing (DollarBay), and history (Blackwood). These systems present a number of opportunities and an equal number of challenges. Players are afforded a role-based, multi-user, ‘learn-by-doing’ experience, with software agents acting as both environmental effects and tutors, and the possibilities of multi-user cooperation and collaboration. However, technical issues and one important cultural issue present a range of difficulties. The Dollar Bay environment, its particular challenges, and the solutions to these are presented.

Keywords: role-base learning systems, multi-user learning systems, software agents, intelligent software tutoring agents, agent-based economic simulation

1. Introduction

Dollar Bay is a fictitious seaside town simulated in an interactive, multi-user, Web-based environment intended to teach the principles and practices of retailing. To join the game, a player creates a character and becomes a storeowner. Their character is then assigned retail space and a starting budget. The player's goal is simple: make more profit than the other store owners in Dollar Bay. However, the simulation presents a formidable and invigorating challenge. The economic environment is sensitive to a number of factors, and players must adapt to changing market forces. Perceived demand changes as other players enter the market and the game simulates seasonal affects on consumer purchasing trends among other things. Dollar Bay players must anticipate these and other trends along with socioeconomic factors in order to adjust their business and keep it thriving. Depending on the success of their business decisions, a business might fail or succeed and be inducted into the Hall of Fame (Mack et al., 2003). Figure 1 shows the Dollar Bay game at work, giving players all the functions of a retail business, and a view into the store itself.

>> INSERT FIGURE 1 AROUND HERE <<

An educational game should be both engaging and informative. Players should acquire concepts and skills as a consequence of playing the game, and this learning should transfer to contexts outside the game. The challenge then is to construct a game of sufficiently interesting complexity that is consistent with the subject it attempts to teach. When the player acts in the simulated environment, the environment must re-act in coherent and plausible ways. Without this consistency the game will fail the ultimate test: students will not play it (Slator and Chaput, 1996).

2. Context: Playing the Dollar Bay Game

At North Dakota State University (NDSU), the World Wide Web Instructional Committee (WWWIC) is engaged in research aimed at developing virtual education environments to assist in the education and growth of students (Slator et al., 1999). Some of the key factors that lead to the success of these environments are a) the theory of role-based environments on which they are based, b) the use of graduate and undergraduate students in the development process, c) the use of the environments in actual classes, and d) the application of knowledge from one environment to the others. One of the major goals of WWWIC research is to find ways to provide tutoring agents to communicate expert advice to students as they progress through the environment. These agents monitor the student and send advice as needed while being careful to never insist upon or block any course of action (Slator, 1999).

As players join the Dollar Bay game they are assigned a location and must decide what to sell, what level of service to offer, how much to spend on advertising, how much to stock, who to buy from, and what prices to set in order to attract customer agents, among other things. In order to simulate an economic environment, time is divided into 'virtual weeks' (a week of simulated game time, not to be confused with a week of actual game play or 'real weeks.') At the beginning of each virtual week, simulated customer agents are given a shopping list representing a week's worth of demand for various products representing an economic group. These agents spend the virtual week purchasing the items on the list from the stores in Dollar Bay. After each virtual week has concluded and the shopping lists are exhausted, each agent assigns new attractiveness ratings to each store based upon the past week's experience (Borchert et al., 2001). Players are also charged for their weekly expenses (such as rent and advertising) and their case files are updated with a record of the week's activity. Customer agent motivations are recalculated based on this new information, and new shopping lists are created for the upcoming week.

At the end of a player's life, they are retired to the Hall of Fame. The Hall of Fame is a place where players are moved when they graduate from the game either by reaching a profit goal, going bankrupt, or being inactive for a long period of time. Upon a player's retirement from Dollar Bay, their store's active case file is archived to the historical cases for future reference by the case-based tutor (Regan and Slator, 2002).

3. Context: Software Agents and Intelligent Tutoring

The overall goal of intelligent tutoring is to implement context-sensitive advice within multi-user distributed simulations to help provide effective learning experiences (Slator 1999). Examples of diagnostic tutoring may be seen in Planet Oit (Saini-Eidukat et al., 2002). For example, the science tutor looks at the decision making process that a player follows while trying to properly identify a material, and what experiments were performed on the material. The tutor is able to both guide students having difficulties and identify students who have made lucky guesses and let them know that they did not follow the proper process in getting to their answer.

Rule-based tutoring in Dollar Bay functions by maintaining a simple set of rules about the domain, monitoring student action for any indication of breaking one of the rules, and then visiting the student to present a warning. For example, one of the rules concerns whether a student has set their prices to an excessive markup. In such an instance, the tutor sends a message to the student informing them that they may be setting their prices too high (Slator and Farooque, 1998).

The most recent innovation is the Dollar Bay case-based tutoring system which provides an analysis of student behavior based on selected attributes, and a classification and subsequent advice based on comparisons with previously stored student records. The case-based retrieval attends to attributes such as product spread and advertising quotients. This system provides the means to generate personalized lessons for each student participating in the Dollar Bay environment (Regan and Slator, 2002).

>> INSERT FIGURE 2 AROUND HERE <<

4. Background: Agent Based Economic Simulation

In order to survive the open competition the player must set a competitive price, hire the appropriate staff, and select appealing products while always keeping an eye on the competition. The player must think about who their likely customers are, what goods these customers want, and how they can use their advertising dollar most efficiently to reach these customers. The strategy of carefully selecting consumer groups is often called targeting by marketing and advertising experts, and targeting is the key teaching goal (Slator and Chaput, 1996).

The pedagogical goal of Dollar Bay is to teach a wide set of skills associated with running a retail business by allowing the student to control a simulated store in a simulated economy. Therefore, the economic simulation must be authentic and complex not only to effectively teach shop-keeping concepts, but also to preserve the player's interest over an extended period.

The Dollar Bay economic model (Hooker and Slator, 1996) assumes rational, cost-minimizing consumers. Consumers, the shopping agents described above, consider travel costs, search costs, service benefits, and product quality as well as price when making buying decisions. For example, the simulation assumes that advertising increases sales by reducing the search cost to consumers in finding information about desired products. In this way advertising helps consumers find the best value for goods by reducing their cost (as opposed to their price).

Dollar Bay models the entire consumer population by defining it in terms of consumer groups defined by psychographic segmentation, a strategy employed by many advertisers and marketers. Psychographic segmentation, as opposed to demographic factors, is the classification of a population into groups that share similar values, attitudes, and lifestyles (Rice 1988; Piirto 1990). The premise is that persons with similar values and lifestyles will exhibit similar buying behavior. Psychographic segmentation is a well established method in marketing, for it promises insight into the emotional and lifestyle factors that motivate consumer's buying behavior.

5. Results: Evaluation and Feedback

The Dollar Bay game in its original implementation has been informally tested a number of times using NDSU Governors School students. These are bright high-school students who spend a summer on the NDSU campus studying math, science, and business, in a concentrated daylong format for six weeks. Since the summer of 2000, twenty of these students have spent an hour a day playing Dollar Bay just after their lunch hour (see http://dbay.ndsu.edu/ ~mooadmin/ DollarBay/ scoreboards/ scoreboards.html).

Surveys of user satisfaction were administered to the students at the end of this period. These surveys cover a range of variables, and allow students to comment on the game in free-form text (the survey instrument is online at http://wwwic.ndsu.edu/ wwwic/ docs/ inteval/ DB-interface-eval.txt). These surveys were used primarily to gather user feedback leading to new interface design.

A set of controlled studies of the case-based tutor in Dollar Bay were conducted during June, 2002. Results of this activity were reported in Regan (2002) and summarized in Regan and Slator (2002). Details are available online at http:// dbay.cs.ndsu.nodak.edu/ ~mooadmin/ DollarBay/ scoreboards/ exper-02.html.

This study showed that case-based tutoring had positive effect, with no harm caused in any case. The tutor advice either proved to be highly beneficial, or showed no real difference in the course of this short study. Indeed, the tutoring had significant effect in some cases. It is interesting to note that the first place and last place stores pursued the same strategy in the same neighborhood, and that the Tutored store proved to be 459% better ($89,812, or nearly 4 standard deviations).

6. Problems with the Original Dollar Bay

The original implementation of Dollar Bay was built by building a graphical user-interface onto a MOO ("MUD, Object-Oriented", where MUD stands for "Multi-User Domain"). MUDs are typically text-based electronic meeting places where players build societies and fantasy environments, and interact with each other (Curtis 1992). Technically, a MUD is a multi-user database and messaging system. The foundational components, though, support a basic spatial metaphor for navigating the database and communicating with other players. The root object classes of the database (in addition to the most generic root 'object' class) are 'player' (representing the user), 'container' (an object that can hold other objects), 'room' (an object that can hold players) and 'exits' (an object that lets players move from one room to another). A MUD supports the object management and inter-player messaging that is required for multi-player games, and also provides a programming language for customizing the MUD with new objects (such as stores), objects with independent behaviors (such as shopper agents), and an environment with complex properties (for example, a simulated economy).

While the original implementation of Dollar Bay was a success, it suffers from shortcomings that hamper its distribution outside the laboratory and into classrooms around the world. Many of these issues are technical in nature, stemming from the use of LambdaMOO as a server. And there is a vital social issue that needs to be addressed as well.

LambdaMOO was the breakthrough technology that allowed educational games like Dollar Bay to exist. The Multi-User Domain programs introduced the ideas and techniques that served as the design foundation of these games, and the LambdaMOO server itself has served this project for many years. Significantly, LambdaMOO focused on dynamic, programmable systems that could be changed and reprogrammed on the fly using an interpreted language also called LambdaMOO but often referred to as MOOCode. These dynamic features allowed for quick and painless prototyping and experimentation with the economic simulation and the environment. But there was a (classic) trade-off of flexibility for speed, and Dollar Bay's complex economic simulation could easily overwhelm the LambdaMOO system. Releasing this program to the public, where thousands of students might connect to the LambdaMOO server at once, would only make things worse.

Compounding this issue were some of the implementation choices made in LambdaMOO. For one, the object storage is a memory resident system, meaning that the entire contents of the database need to be stored in memory constantly. Another is that client/server interaction is limited to lines of text. Again, these choices support rapid prototyping and work great in the lab, or with text-based clients, but they cannot handle the strain of a graphical client in wide release. Plus, there is no mechanism in LambdaMOO to extend its functionality, such as using an SQL database or binary network communication.

Finally, LambdaMOO has a very rudimentary authentication system. This is adequate in a lab of trusted peers, but this project's target audience is in the age range of 9 to 13 years. Other Internet services which appeal to this age range and have weak authentication have become subject to malicious and sometimes predatory behavior (chat rooms are a notorious example). This could lead to situations where youthful players were playing in a simulation also inhabited by stalkers and pedophiles – in educational software, this simply will not do. In order to completely remove this threat, a strong authentication system is needed that would guard against users posing as something they are not (or simply being anonymous).

7. Solution: the JavaMOO DollarBay Project

To address these many technical and cultural issues, the MOO substrata needed to be re-implemented from the ground up with these concerns in mind. Of course, the issues mentioned above are not new issues, and are important to anybody interested in developing an industrial-strength networked application. Fortunately, these problems are addressed by Java, an application platform written by Sun Microsystems for network applications. Programs written in the Java programming language using the Java API can be compiled to run on a Java Virtual Machine (JVM), itself an application available on a variety of operating systems including Linux, Microsoft Windows and Macintosh OS X. (This is often abbreviated by saying that Java is platform-independent.)

The Java API offers a full palate of networking functions, including client/server communication, user authentication and threaded execution. On the server side, Java has a database access library (JDBC) that can be used to integrate a wide variety of industry standard databases. On the client side, the Java Swing API provides a rich user interface toolbox that uses native UI widgets on each platform. Finally, since Java is compiled into microcode, it is efficient, compact, and runs fast enough to support a real time, detailed economic simulation.

The game support architecture, both server and client, were re-implemented on the Java platform, which is called JavaMOO. The JavaMOO Platform was built to support any number of game designs, including the afore-mentioned games that had already been implemented on the earlier architecture. The JavaMOO Platform provides an API for future game developers to easily realize their ideas in this new architecture. As a proof-of-concept, Dollar Bay was ported to the new JavaMOO Platform.

JavaMOO is a three-tiered platform: client, server, and persistent data store with communication between clients and the server accomplished though Remote Method Invocation (RMI), which allows the client to invoke procedures which are transparently executed on the server, and vice versa. This technology allows native data structures to be sent between the client and server by simply passing them as arguments. Persistent objects are created on the server and are automatically stored, creating a transparent persistence. See Figure 3 for a schematic.

>> INSERT FIGURE 3 AROUND HERE <<

There are three parts to any server programmed within JavaMOO: 1) the JavaMOO core, 2) the application package, and 3) the communication package. The JavaMOO core contains the basic underlying functions of the server such as connection management, database connectivity, and object persistence. All applications will have this core, which enables programmers to concentrate on other aspects of design. The application package is a class library built for a particular project on top of the JavaMOO core. In this library the programmer defines the underlying structure and functions of the application with the ability to utilize persistent objects. The economic simulation within Dollar Bay, including shoppers, employees, products, and stores, are implemented here. Finally the communication package contains the RMI mechanism and all of the objects which are passed between server and client.

7.1 JavaMOO Issues: Execution Speed

The Java language is compiled into byte-code that is then executed on the JVM (on whichever platform is desired). The JVM is very quick and efficient, and many operations – especially mathematical operations – are as quick as those written and compiled in C or C++. In addition, because Java is compiled, the compiler can perform optimizations to the program before the code is ever executed. Compiled Java is more than adequate to host Dollar Bay's economic simulation, as well as handle thousands of connected users.

There is a serious trade-off though. Because the program is compiled, changes cannot be made to a program which is currently executing (as opposed to the situation with LambdaMOO). This doesn't mean that the data – such as parameters, new users, and product prices – are static. But the code that is executing and performing the calculations of the simulation cannot be altered in the running system. To make these changes, a new program must be compiled, and the old program must be replaced.

This means that server upgrades can no longer be handled on the fly (as opposed to LambdaMOO). To upgrade the server, the old server must be taken down and everybody must disconnect from it. This makes JavaMOO less attractive for rapid prototyping, since even the smallest change would be very disruptive.

However, this is an appropriate trade-off. If Dollar Bay, or any other educational game, were in widespread use by thousands of students, one would want to discourage changes to the server while the game is being played. The required disruption in game play serves as a notice that the underlying software has changed. This is one of the elements that changes as a system moves from the lab into more general use.

7.2 JavaMOO Issues: Memory Management

In the previous architecture, LambdaMOO served as both the database manager and the simulation executor, which means that it both manipulated and stored data objects. LambdaMOO's object storage method is rudimentary, in that it retains all objects in memory and periodically dumps all of memory to a file (known as "checkpointing"). This approach has the advantage that all object manipulations are very fast. But the drawbacks outweigh this advantage in production.

In a large system, keeping the entire database resident in memory can be an enormous waste of resources. Additionally, checkpointing the entire contents of memory can take a noticeably long time. Consequently, altered objects are not stored until the next scheduled checkpoint, which leaves open the possibility of lost data or even a corrupt database. A system was needed where objects could be independently accessed, manipulated and stored, with all the necessary checks on locking and database coherence.

This is what a standard database is designed for, and the JavaMOO platform was designed to allow game developers to use any number of SQL databases using Java's ODBC database access library (JDBC), thus allowing connections to a wide variety of SQL databases. The server handles the storage and retrieval of objects via JDBC. Objects are retrieved from the database when they are needed, and written to the database whenever they are modified. When any part of the program needs to access an object, JavaMOO first checks a resident object table to see if the object is resident. If it is, the program uses the existing object. Otherwise, the object is read from the database.

The Java platform uses garbage collection for memory management, meaning that objects that are no longer being used are de-allocated and their memory is reclaimed. This memory management method conflicts with JavaMOO's object allocation system because all retrieved objects are stored in the resident object table, which would register to the Java garbage collector as "in use" and thus would never be garbage collected. To solve this problem, the resident object table uses Java's Weak References, allowing objects to be garbage collected when they are in the table but otherwise unused.

Given the normal mode of operation, an educational game, it is likely that an object will be altered many times in a row. This would result in a sequence of redundant store operations, which would slow down a system where object storage is now more time-consuming. This eventuality was ameliorated by using an object store queue. In this strategy, an object is put on the store queue whenever it is changed. If an object is changed, but it is already on the store queue, it is not put on the queue again: the pending store operation will store both object alterations. The queue is periodically processed, which has the effect of optimizing the number of database queries.

These techniques allows the use native Java objects, a desirable outcome, and results in an object management system that is as flexible and as fast as LambdaMOO's, but far more robust and efficient.

7.3 JavaMOO Issues: Client/Server Communication

LambdaMOO was built for textual interaction, not unlike the original text adventures from which these MUD engines first drew their inspiration (like 'Adventure', an exploration game to explore the 'Colossal Cave' distributed with early version of Unix). . Users would log in with a text client (or, if you were using a Unix shell, just from the command line), and interact with the virtual environment by typing simplified English commands and receiving prose feedback.

This is the only interaction channel with LambdaMOO. When the original Dollar Bay was built, the graphical client used this text interface to communicate with the simulation. Information was sent back and forth using 'out of band' text directives, a vector of delimited strings that needed to be constructed by the sender and parsed by the receiver.

However, there are problems with this approach. Sending all client-server communication via text is very limiting. All non-textual data must be converted into text, and then back from text once received. Structured data, such as objects, also need to be flattened and reconstituted. On top of this, there was a 256 character limit on the directive itself, resulting in single directives sometimes being broken up into many lines.

As an alternative, Java provides Remote Method Invocation, an elegant solution for client server communication. Objects can be transparently streamed between the client and the server by simply passing them as arguments to remote methods. The Java RMI libraries handle the transmission and reception of these objects automatically. This technique allows full-fledged objects to be served between programs over any network connection.

7.4 JavaMOO Issues: Client-Server Separation

The economic simulation in Dollar Bay was re-written using Java objects. These objects hold all the information that the client needs to build whatever representation is desired, so it is tempting to use the same objects on both the client and the server. However, there is information in server objects (such as database access) that is not used in the client. And there is information needed in the client (such as client specific user interface data) that would not be used on the server. More importantly, any reference to a server object on the client would mean that the entire server would need to be linked into the client for the program to compile. Direct sharing of the simulation objects would not work. Consequently, a strict separation of client objects and server objects was enforced that could not be breached.

So then how would the client and server talk to each other? For this purpose, a set of common event objects were created that could be used in both the client and the server, but could not make reference to any objects in either. These event objects allow the server to tell clients of important changes in the state of the simulation, and let the client reflect those changes to the user. Conversely, clients use event objects to notify the server of actions performed by the user that would alter the simulation, allowing those changes to be stored and disseminated.

To implement this, the communication package provides a ‘Mailbox’ object which exists on the server but can be accessed by the client via an RMI stub. The stub looks and acts just like a local object, but when the client calls a method on the Mailbox stub, the method actually gets invoked on the server. Further, the client then automatically packages all of the arguments to these methods and send them over the network to the server transparently. So when the client wants to notify the server of a user action, the client simply calls a method on the Mailbox object with an appropriately filled-out event object. The server then processes this event object as necessary. Similarly, the client provides the Mailbox object with a callback object, which the server uses to notify clients of changes in the server state. All of this communication is handled quickly and efficiently, and it is effortless for the game developer to employ.

7.5 JavaMOO Implementation Issues: Authentication

One of the problems identified in user testing is the fear that unauthorized individuals will be able to stalk the young players. This is a serious cultural concern for anyone in the business of producing multi-user simulations for education. These issues are addressed as follows.

Java supports state-of-the-art authentication and authorization standards that can be used to match each server message with the client's host, and each person with their account on JavaMOO. RMI supports strict authentication to allow only authorized clients to connect to the server. Client access can also be restricted by their host or subnet. In addition, each JavaMOO user has an account that must be used to access the online simulation. Java supports secure password verification between clients and servers.

By using these industrial-strength authentication strategies, access to the system can be strictly controlled, and users' access and actions can be logged in detail. With this system of accountability, game administrators can spot any wrongdoing– such as system abuse or illegal access – and associate it with a particular user. More importantly, though, an on-line security system that is well advertised will discourage any wrongdoers in the first place.

8. Case Study: JavaMOO Dollar Bay

As a test case, Dollar Bay was reimplemented using the new JavaMOO platform. The client and the server were programmed in Java using the JavaMOO API. After a few weeks of programming the system had about 80% of the functionality of the original Dollar Bay. The following months were spent transferring the original Dollar Bay content to the new JavaMOO server. While extensive stress testing of the new system is underway, the execution and communication time is noticeably much faster, and data storage is quick, reliable and seamless. The client is also much improved. Since the clients use native interface widgets on each platform, the client not only looks appropriate for each platform but is also much more responsive.

9. Conclusion

The JavaMOO Platform is a development API for widely distributed multi-user educational simulations. It's highly customizable and extensible, it includes SQL database access, Remote Method Invocations, secure authentication and platform independence, and it uses compiled code with rapid execution.

The Dollar Bay application was re-implemented on the JavaMOO Platform quickly and easily without sacrificing any of the original game functionality, and providing additional features that would allow for Dollar Bay's widespread distribution to young students. This new system and its first test case will pave the way for the porting of other educational games to the new platform, as well as the development of new educational games with more sophisticated content and engaging interfaces.

In order to provide the scalability for dissemination and security for children, a distributable server package will be developed next. This will allow system administrators of elementary and middle schools to install and run the simulation as a local network game, creating a safe and appropriate environment for children while diffusing the load from the current central NDSU Dollar Bay server.

10. References

Borchert, O., Bergstrom, A., Hockemeyer J., and the Worldwide Web Instructional Committee. (2001). Recent Advances in Immersive Virtual Worlds For Education. In: Proceedings of the 34th Annual Midwest Instruction and Computing Symposium (MICS-01), Cedar Falls, IA. April 5-7: [cd-rom: /papers/ borchert.pdf]

Curtis, P. (1992). Mudding: Social Phenomena in Text-Based Virtual Realities. In: Proceedings of the Conference on Directions and Implications of Advanced Computing (sponsored by Computer Professionals for Social Responsibility). Berkeley, April.

Hooker, R., Slator, B. M. (1996). A Model of Consumer Decision Making for a Mud Based Game. In: Proceedings of the Simulation-Based Learning Technology Workshop at the Third International Conference on Intelligent Tutoring Systems (ITS'96). Montréal, June 11: 49-58.

Mack, J., Slator, B. M., Boulais, K., Doan, V., Stanford, C., and the students of CS345 (2003). Learning by Earning: The Dollar Bay Project. Proceedings of the 36th Annual Midwest Instruction and Computing Symposium (MICS-01), Duluth, MN. April 5-7: [cd-rom: /papers/ mack.pdf]

Piirto, R. (1990) Measuring Minds in the 1990s. American Demographics, December.

Regan, P. M., Slator, B. M. (2002). Case-based Tutoring in Virtual Education Environments. Proceeding of the ACM Conference on Collaborative Virtual Environments (CVE-2002). Bonn, Germany. September 30 - October 2: 2-9. 2002

Regan, Patrick (2002), Case-based Reasoning in Virtual Environments. M.S. Computer Science. North Dakota State University, Fargo, ND, USA

Rice, B. (1988). The Selling of Lifestyles: Are You What You Buy? Madison Avenue Wants to Know. Psychology Today, March.

Saini-Eidukat, B., Schwert, D. P., and Slator, B. M. (2002). Geology Explorer: Virtual Geologic Mapping and Interpretation. Journal of Computers and Geosciences. 28(1): 1167-1176.

Slator, B. M. (1999) Intelligent Tutors in Virtual Worlds. In: Proceedings of the 8th International Conference on Intelligent Systems (ICIS-99). June 24-26. Denver, CO: 124-127.

Slator, B. M., Chaput, H. (1996). Learning by Learning Roles: a virtual role-playing environment for tutoring. In: Proceedings of the Third International Conference on Intelligent Tutoring Systems (ITS'96). Montréal: Springer-Verlag, June 12-14: pp. 668-676.

Slator, B. M., Farooque, G. (1998). The Agents in an Agent-based Economic Simulation Model. Proc. of the 11th International Conf. on Computer Applications in Industry And Engineering (CAINE-98) November 11-13, Las Vegas, Nevada USA: 175-179

Slator, B.M., Juell, P., McClean, P., Saini-Eidukat, B., Schwert, D.P., White, A., & Hill, C. (1999). Virtual Environments for Education at NDSU. Journal of Network and Computer Applications, 22 (4): 161-174.

11. Acknowledgments

Some portion of this NDSU Worldwide Web Instructional Committee (WWWIC) research was supported by funding from the National Science Foundation under grant EIA-0086142

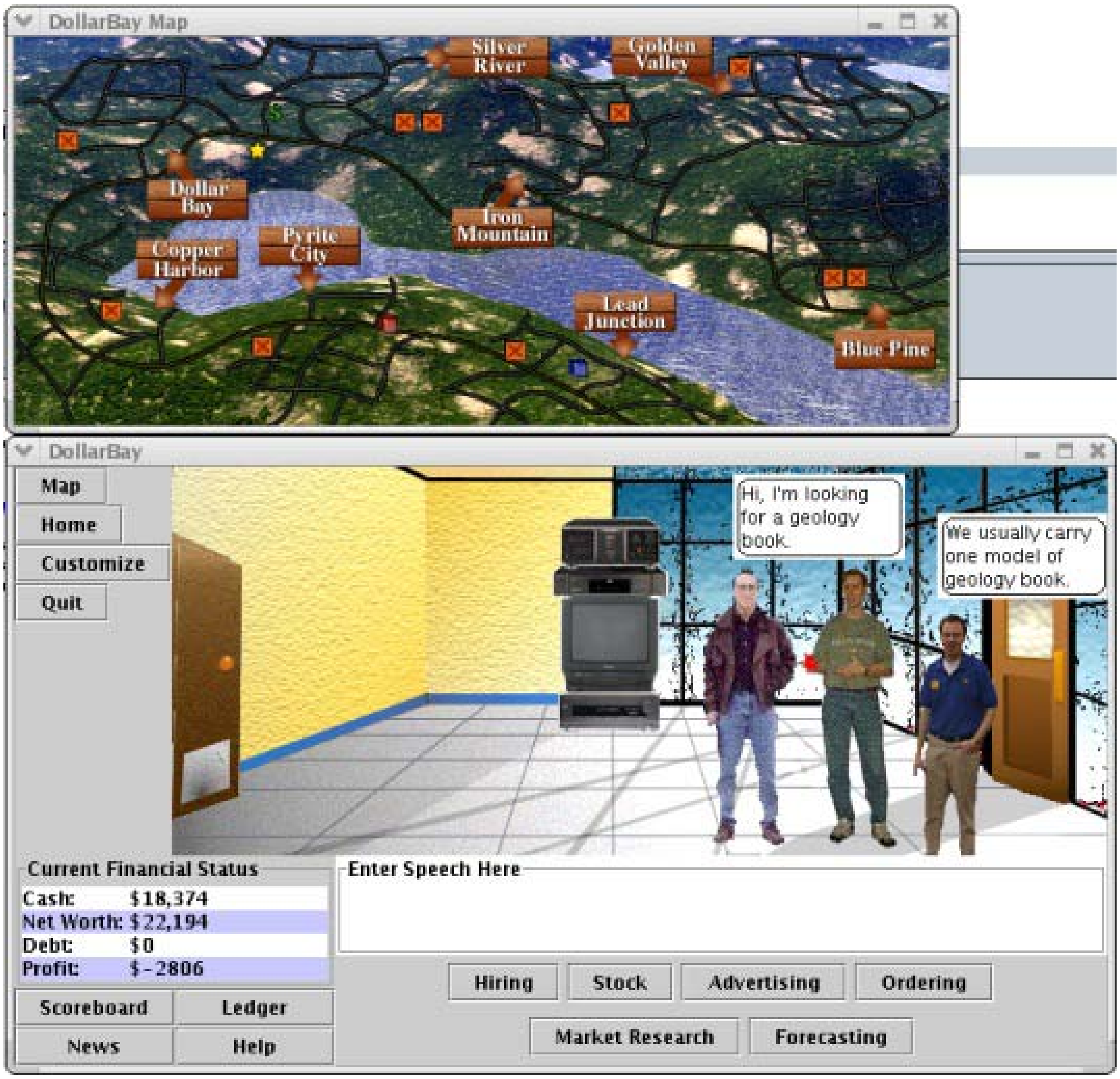

Figure 1: This is the main window of the Dollar Bay client. It contains all the controls of the game (hiring, advertising, etc.). This view shows the inside of a store with two shopper agents (below) one of whom is attempting to buy a geology book from a salesperson agent, and the map used for navigation (above).

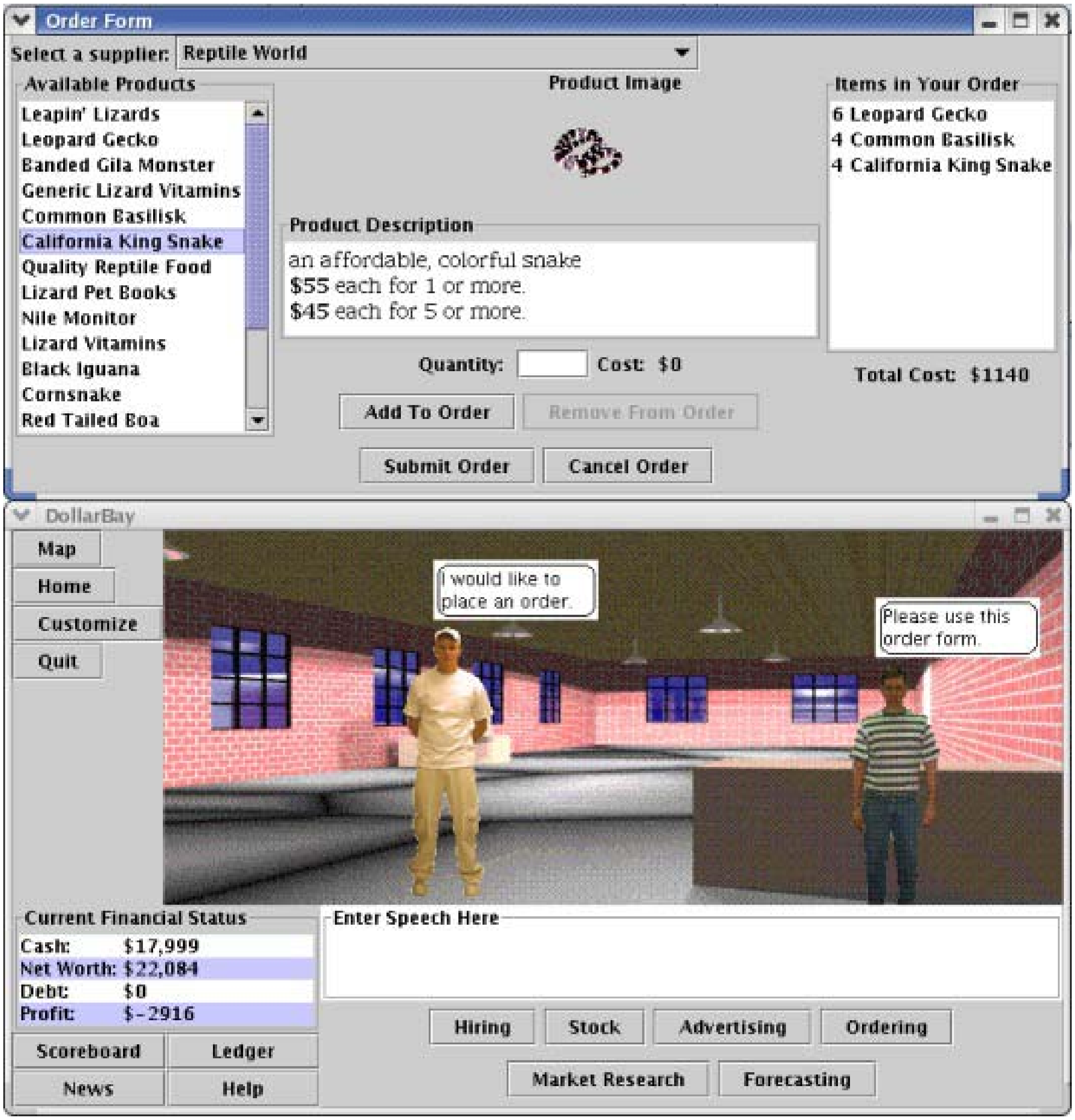

Figure 2: An example of a live player interacting with a simulated agent. Here, the player communicates with a wholesaler agent in order to stock his store with inventory (below). The Order Form (above) appears for the player to order items (a King Snake in this case).

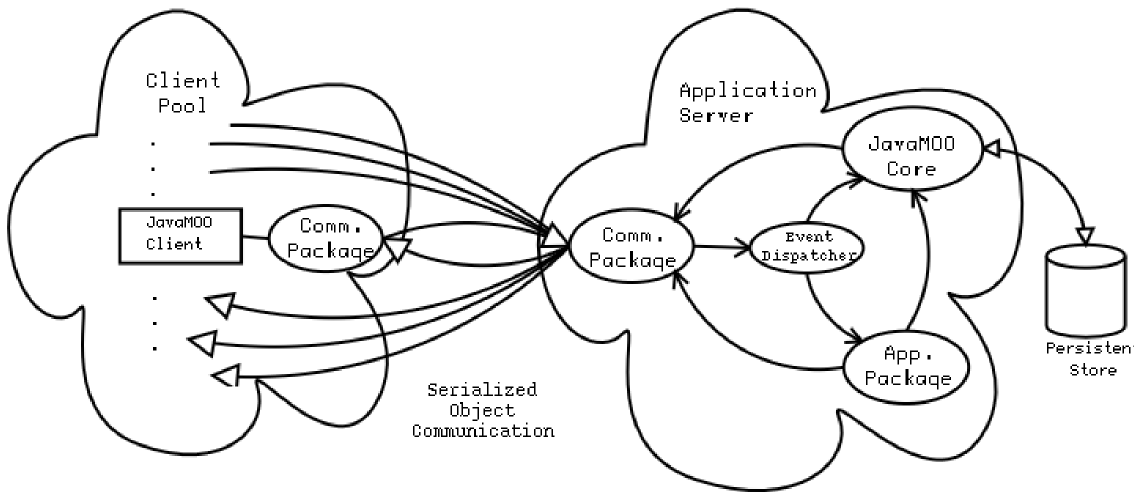

Figure 3: The JavaMOO Programming Model. The application server holds the JavaMOO core and the application package. These elements interact with each other and a communications package using dispatched events. The communications package is used by the server and the client to send information back and forth. All non-transient data is stored in the persistent store.