Exploring a Framework for Goal-Driven Collaboration through Serious Gaming

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Thomas Cheyney Loffing

Graduate Program in Computer Science and Engineering

The Ohio State University

2010

Master's Examination Committee:

Rajiv Ramnath, Advisor

Ola Ahlqvist

Jay Ramanathan
Abstract

Serious gaming is gaining momentum as a legitimate tool in education, occupational training, physical fitness, military simulations, and many other domains. However, little work has been done to generalize the structure and development of serious gaming applications.

This thesis describes the establishment of object and runtime models that can be used to enable goal-driven collaboration. This idea is explored through simple and rapid conceptualization of serious gaming applications. It begins with a background in serious gaming that includes examples and studies of their effectiveness. Then a description of GeoGame, a GIS-based multiplayer game that is used as a pilot serious gaming application, is provided. Next, the object model and runtime models are described along with the course of their development. Disparate scenarios are specified in terms of these models in order to validate them. An accompanying architecture is described that can be used to implement the models. Next the process of converting GeoGame to adhere to these models is laid out. An analysis of the model-based version of GeoGame describes how it compares to the original purpose-built application in terms of performance and other factors. The broader applications of a serious gaming framework are also explored, along with other future work.
Dedication

To my family
Acknowledgments

I would like to thank Dr. Jay Ramanathan for all her direction and support. She convinced me to pursue a thesis and provided the motivation I needed throughout the process. I would like to thank Dr. Rajiv Ramnath for all of the insight that was vital in guiding my research, especially when I ran into technical challenges. I would also like to thank Dr. Ola Ahlqvist for providing an interesting project that was the key inspiration for my work.
Vita

June 2004 ........................................ West Liberty-Salem High School

June 2008 .......................................... B.S. Computer Science and Engineering,

The Ohio State University

September 2008 to present .................... Graduate Teaching/Research Associate,

Department of Computer Science and

Engineering, The Ohio State University

Fields of Study

Major Field: Computer Science and Engineering
Table of Contents

Abstract .................................................................................................................................................. ii

Dedication ............................................................................................................................................. iii

Acknowledgments ................................................................................................................................. iv

Vita ......................................................................................................................................................... v

Table of Contents ................................................................................................................................ vi

List of Tables .......................................................................................................................................... ix

List of Figures ........................................................................................................................................ x

Chapter 1: Introduction ........................................................................................................................ 1

Serious Gaming ...................................................................................................................................... 1

The Effectiveness of Serious Gaming .................................................................................................. 2

Outline of Thesis .................................................................................................................................... 3

Chapter 2: Related Work ....................................................................................................................... 4

Chapter 3: GeoGame ............................................................................................................................. 6

Concept .................................................................................................................................................. 6

Application ............................................................................................................................................ 7
Architecture .......................................................................................................................... 11

Project Darkstar .................................................................................................................. 13

Keyhole Markup Language (KML) ...................................................................................... 14

NASA World Wind ............................................................................................................. 15

Deployments and Effectiveness ......................................................................................... 16

Chapter 4: A Conceptual Model for Serious Gaming Applications ................................. 19

Object Model ...................................................................................................................... 19

Runtime Model .................................................................................................................. 25

Chapter 5: The Architecture to Support the Models ......................................................... 28

Knowledge Capture .......................................................................................................... 28

Web Ontology Language (OWL) ....................................................................................... 28

Keyhole Markup Language (KML) .................................................................................. 29

Runtime Execution ........................................................................................................... 29

Java Expert System Shell (Jess) Rules Engine ................................................................. 29

Chapter 6: GeoGame and the Conceptual Models ............................................................ 31

Conceptualization ............................................................................................................. 31

Implementation ............................................................................................................... 35

Coding Changes .............................................................................................................. 36

Defining the Ontology ..................................................................................................... 38
List of Tables

Table 1: GeoGame’s Student Survey Statements .............................................................. 17
Table 2: GeoGame’s Student Survey Results ................................................................. 18
Table 3: Motivating Scenario Specifications .................................................................. 24
Table 4: GeoGame’s OWL Ontology ............................................................................. 40
Table 5: GeoGame’s KML ......................................................................................... 41
Table 6: GeoGame’s Jess Rules .................................................................................... 42
Table 7: Processing Times for GeoGame’s Round Processing Procedure ................. 67
List of Figures

Figure 1: GeoGame’s Object Model ................................................................. 7
Figure 2: The GeoGame Lobby ................................................................. 8
Figure 3: A GeoGame Simulation ............................................................. 9
Figure 4: GeoGame’s Architecture .......................................................... 12
Figure 5: Project Darkstar’s Architecture [11] ....................................... 13
Figure 6: Object Model ........................................................................ 21
Figure 7: Runtime Model ...................................................................... 26
Figure 8: GeoGame’s Object Model ....................................................... 32
Figure 9: GeoGame’s Round Processing Procedure .............................. 34
Figure 10: GeoGame’s Base OWL Ontology .......................................... 39
Figure 11: GeoGame’s Round Processing Performance ....................... 45
Figure 12: GeoGame’s Purchase Procedure ........................................... 58
Figure 13: GeoGame’s Trade Procedure ............................................... 59
Figure 14: GeoGame’s Fate Procedure ................................................... 59
Figure 15: Round Processing Times for the Hard-Wired Version of GeoGame ............ 69
Figure 16: Round Processing Times for the Model-Based Version of GeoGame .......... 70
Chapter 1: Introduction

Before describing the details of a conceptual model for serious gaming applications it is necessary to motivate the discussion with an introduction to serious gaming in general. This introduction contains a definition of serious gaming, examples of serious games, and analysis of the effectiveness of serious gaming in various domains.

Serious Gaming

Quite simply, a serious game can be described as any game aimed at a purpose other than pure entertainment. Often these purposes include education, occupational training, physical fitness, and military simulations, although new domains are continually arising. The common element across all serious gaming efforts is the use of entertainment in order to increase the efficacy of conveying the primary objectives.

Examples of serious gaming are numerous and varied. For the average American, perhaps the most well-known instance of serious gaming is the Nintendo Wii, which uses games like Wii Fit and Wii Sports to improve physical fitness, coordination, and dexterity. In the corporate world IBM has had success with INNOV8, a simulation game that teaches IT and business professionals about process management. This success led to CityOne, a game about creating a cleaner, more efficient planet. The armed forces make extensive use of simulations for personnel training as well as recruitment. Emergency management is another popular application for serious gaming, where
simulated terrorist attacks or natural disasters are used to test the readiness of response
teams.

The Effectiveness of Serious Gaming

Serious gaming’s effectiveness is built largely on the idea of using entertainment
to convey an objective, whether it is to teach elementary school students the state capitals
or help a patient regain motor skills after a serious injury. This section will describe
studies that highlight serious gaming’s effectiveness throughout two domains in which
serious gaming has made significant inroads: education and medicine.

Serious gaming is gaining popularity in the education domain as entertainment
has shown promise in improving the learning experience. Consider, for example, the E-
Junior project – a serious virtual world intended to teach children about natural science
and ecology. When compared to normal classroom learning, students taught using E-
Junior attained the same comprehension level, but reported that they enjoyed themselves
more, felt more engaged, and had greater intentions to participate [1]. Another
educational game, Statecraft X, is meant to teach teenage students about citizenship by
allowing them to govern regions of a fantasy world [2]. It was developed as an iPhone
application and shown to be both meaningful and enjoyable through student surveys.

Today’s medical field also make significant use of serious gaming applications,
most notably is physical fitness and therapy. The preliminary results from one study in
therapeutic “exergaming” found that supplementing traditional exercise with gaming
components made therapy more enjoyable and interesting for patients, perhaps increasing
the motivation that is vital to successful rehabilitation [3]. Serious gaming may also be
effective in helping patients recover from serious medical incidents. Elinor is a game designed to encourage patients to perform rehabilitation activities following a stroke [4]. Early results indicated a high level of patient enjoyment and motivation, and even an improvement in overall motor function for one patient.

Outline of Thesis

The rest of this thesis is organized as follows. Chapter two provides a brief discussion of related work. Chapter three gives an overview of GeoGame, a serious game used as a pilot application to motivate the models being developed. Chapter four describes the conceptual object and runtime models that are at the heart of this work. Chapter five discusses the software architecture that is used to implement the conceptual models. Chapter six demonstrates how GeoGame can be specified in terms of the model while chapter seven analyzes the process of converting GeoGame from its original standalone version to the model-based version. Finally, chapters eight and nine provide concluding thoughts and a look at possible future work.
Chapter 2: Related Work

Much work has been devoted to serious gaming in the past several years. However, most of this research focuses on different aspects of the problem than that of this research, such as education-specific games or development toolsets. This section presents a brief collection of work that in some way motivated the conceptual models to be presented later.

Yusoff, et al. established a framework for serious games that is intended to help in both the development of the game and in measuring the game’s effectiveness in conveying its objectives [5]. This work focuses primarily on defining the game in terms of its objectives. Thus the framework includes concepts like instructional content and learning activities. The concentration on educational games was not a goal of this thesis, but the work was still useful as a reference for creating a serious gaming framework in general.

EMERGO is a methodology and toolkit aimed specifically at development and delivery of serious games for higher education [6]. It differs from this work in that it specifies the course of development to be taken, which is based upon the Unified Process and accompanying development phases (analysis, design, implementation, etc.). EMERGO also provides an evaluation phase which determines if an implemented game meets its educational objectives. Such an evaluation phase is not included in this research but would make for interesting future work.
The MDA framework defines games in general, not just serious games, by breaking them into mechanics, dynamics, and aesthetics layers [7]. This model can be used not only in iterative development of games, but also in obtaining a fundamental understanding of games that is useful in gaming research and criticism. The MDA framework acts as a good foundation for any gaming framework.

The Ontology-based Edutainment Development Framework (OEDF) pursues the “made by teacher” concept for game development [8]. OEDF seeks to simplify creation of educational games enough so that teachers with no background in game development will be able to successfully construct games for use in their classrooms. A major advantage of this is that games created by teachers would be potentially more effective than those created by traditional game designers who are much less familiar with the pedagogical field.

The work presented in this thesis combines elements of the above related research, among other sources. The object model seeks to define the static structure of serious games in a way similar to the work of Yusoff, et al., although it focuses on their implementation and runtime structure rather than any underlying educational goals. Also included in this work is a set of tools to facilitate game development. As part of future work, this toolset will be simplified and unified to more closely resemble something like the EMERGO toolkit. The models presented here strive to be minimal and highly flexible, much like the MDA framework. Finally, the software architecture behind the models will be ontology driven, similar to the OEDF.
Chapter 3: GeoGame

GeoGame – Green Revolution is a GIS-based multiplayer game intended to teach college-level geography students about the Green Revolution, which involves introducing modern farming materials and techniques to developing countries.

In GeoGame each student manages their own virtual farm which is tied to real-world GIS data. This involves management of land, crops, and other assets to best provide for the player's virtual family. Several factors, such as weather and random fate events, play into the success or failure of each farm. Through this process students become familiar with the types of decisions that farmers in developing countries face on a regular basis. In deciding between genetically modified or land race seed, artificially- or naturally-irrigated land, and fertilized and non-fertilized crops, students gain hands-on experience in the impacts, positive and negative, of the Green Revolution.

Concept

The GeoGame concept was developed by Dr. Ola Ahlqvist, Assistant Professor in the Department of Geography at The Ohio State University. It is modeled after a Green Revolution simulation created by Dr. Ricardo Salvador while he was an Associate Professor at Iowa State University [9]. That simulation was later adapted by Engineers without Borders, Canada [10] whose work also served as a valuable reference in the creation of GeoGame.
Application

The original version of GeoGame was a purpose-built application that was developed specifically for use in Geography 200 at The Ohio State University. A simplified object model of the game is shown in Figure 1.

As is evident from the figure, the lobby and simulation are the two central components of GeoGame’s simplified object model. The lobby is used to manage and monitor simulations. Numerous players may be logged into the lobby at any one time and may chat with each other via a communications channel. The lobby also maintains references to all active simulations. Each simulation has a number of players, a communications channel, a banker that controls market prices, and an oracle that manages weather and fate events.

Figure 2 is a screenshot of the lobby, which is displayed once the user successfully logs in.
Simulation creation is carried out using the upper left panel. The creator must specify a name for the simulation and select a KML file (that must be uploaded to the server beforehand) that specifies where the simulation will take place and what land parcels will be used. In order to assist GeoGame administrators in classroom deployment, simulation creation may be made privileged functionality only available to a subset of users.

Once a simulation is created it will be shown in the active listing in the upper right panel. This panel allows the user to join, delete, or gather statistics about
simulations. Again, simulation deletion and statistics gathering may be made privileged commands.

Finally, the lower panels display what players are currently in the lobby and allows players to chat with each other.

Players typically reside in the lobby for only a short period of time before joining a simulation. Upon joining, the lobby is hidden and the simulation window (Figure 3) is displayed.

![Figure 3: A GeoGame Simulation](image_url)
The simulation window consists of four main areas: the GIS panel, the summary cards panel, the chat panel, and the actions panel. The GIS panel acts as the “game board” for the simulation. It is used to display the area in which the simulation is played and shows each land parcel and its current owner, if any. A tabbed display on the left allows the visibility of individual GIS layers to be toggled and also allows the player to add additional layers that are reachable through a URL. Below this panel is an icon indicating the weather report from the previous round.

The summary cards panel on the right provides a snapshot of the status of each player’s farm. The top card always displays the status of the logged-in player while the bottom card can be cycled to display the status of every other competitor.

The chat panel on the bottom right is a tabbed display that aggregates the bottom two panels of the lobby. One tab shows the players currently logged into the simulation while the other allows for chat messages to be sent to those players.

The actions panel on the bottom left is where most of the simulation’s decisions are made. The market tab displays the resources available for purchase from the market with their associated prices for the given round. The transactions tab allows players to trade resources among themselves or to donate resources to needy players. The score sheet tab displays a more in-depth analysis of the logged-in player’s farm than does the score sheet. This information includes yield and consumption values, resource quantities and expenses, and weather and price histories for each round. The news tab alerts the player to pertinent events throughout the game, such as natural disasters, births, deaths, or
governmental action. These events are referred to as “fate” and occur randomly throughout the game. The discussion tab contains a threaded discussion forum that provides players with a more persistent form of communication that the chat panel. This is useful to negotiate trades and exchange game experiences. Finally, the information tab informs the player of important simulation parameters (e.g. how many acres of land an ox can work in a round), and displays alerts related to the farm’s status (e.g. when the labor on hand is insufficient to cover the farm’s acreage).

Architecture

GeoGame was implemented as a Java-based client-server application. The client was made as “thin” as possible, meaning that all significant game logic is implemented by the server. The client performs two main tasks. It processes messages from the server, updating the user interface as necessary. It also converts user actions into appropriate messages that are then sent to the server. Figure 4 shows this architecture.
Client-server communication is handled by a gaming framework called Project Darkstar (described below). On the client side, game logic mainly consists of sending, receiving, and processing messages and updating the user interface as appropriate. NASA World Wind (also described below) drives the GIS display. On the server side the game logic makes up the vast majority of the application. This logic manages the lobby, simulations, and players and handles the myriad of requests that come in from each client. Project Darkstar is used significantly on the server side. Darkstar uses a Berkeley Database server to persist all objects while a separate MySQL database contains login credentials and statistics for all players.
Project Darkstar

Project Darkstar is a framework that enables rapid development of highly scalable multiplayer games. Features provided by Project Darkstar include data persistence, client-server communication, user authentication, and transaction management, all of which are used significantly throughout GeoGame. Figure 5 illustrates the basic architecture of the Project Darkstar framework.

![Project Darkstar's Architecture](image)

Figure 5: Project Darkstar’s Architecture [11]

Each Darkstar server instance has a cluster of nodes, each of which runs a copy of the game logic on top of the Darkstar stack. This stack includes services that manage
transactions, communications, persistence, and other core functionality. A number of meta-services perform load-balancing and additional coordination tasks to ensure that maximum parallelism is used to increase performance while maintaining synchronization and data integrity. This architecture is highly scalable in order to support massively multiplayer games.

Project Darkstar was chosen because it is an open-source, highly scalable way to allow rapid deployment of multiplayer games. Its core services are easy to use, which allows the majority of the programming effort to focus on game-specific logic. Other solutions briefly considered include Microsoft’s XNA toolset and Electroserver. The most common reasons for rejecting alternative solutions were the use of technologies that were less familiar to the developer and a lack of free software.

Unfortunately, Project Darkstar was canceled on 2 February 2010 shortly after Sun Microsystems was purchased by Oracle. The last stable release of Darkstar was version 0.9.11. However, since the project was open source, members of the community were able to pick up development. Project Darkstar was quickly rebranded as RedDwarf Server, whose latest stable release is version 0.10.1 [12]. The majority of GeoGame development was completed before this rebranding so the port to RedDwarf Server was not completed. This port is a high priority for future work.

*Keyhole Markup Language (KML)*

Keyhole Markup Language (KML) is an XML-based language for representing GIS data [13]. It associates features, such as polygons, images, and 3D models, with a
latitude, longitude, and altitude, along with other data. KML is endorsed by the Open Geospatial Consortium (OGC) [14].

KML was selected because it is a simple representation scheme that can be created with a basic XML editor as well as most GIS packages. It is also displayable by nearly all GIS packages. While it may not be suitable for some very data-intensive GIS tasks (e.g. calculating a detailed irrigation model based on a drainage network and spatial proximity), it satisfies the needs of most serious gaming applications, which typically need nothing more than simple rendering, modeling, and routing.

**NASA World Wind**

NASA World Wind is an open source GIS package [15]. It is used to display and manage a rendering panel that is used as the “game board” of GeoGame. Players use the World Wind interface to view and interact with farm parcels. They may also navigate around the globe and load their own GIS layers from the Internet.

One minor limitation of NASA World Wind at the time of GeoGame’s initial implementation was its lack of KML support. However since World Wind is an open source project, a community solution called “wwj-kml” was available for use [17]. This solution does not render the more advanced components of KML, such as 3D models, but is able to display polygons and place markers effectively, which is all that is required for land parcel display.

NASA World Wind was chosen mainly due to its Java SDK. The drawbacks of World Wind compared to Google Earth (which was initially targeted as GeoGame’s GIS
package), such as low-resolution imagery and limited format support, were outweighed by its ease of integration, especially since most major drawbacks had workarounds.

Deployments and Effectiveness

GeoGame was deployed twice in classroom settings. The first deployment took place in autumn 2009 while GeoGame was still in a relatively early beta stage. As a result a number of problems arose during the deployment, such as difficulties installing the client on untested platforms and server malfunctions due to scalability issues. Considering these complications, student response was quite positive. A second deployment took place in spring of 2010 with a more robust version of GeoGame. Feedback from both deployments was gathered using a survey that presented the statements in Table 1, among others. These statements are a subset of EGameFlow, a scale used to measure the level of enjoyment a player feels while playing an educational game [16].
Table 1: GeoGame’s Student Survey Statements

<table>
<thead>
<tr>
<th>Category</th>
<th>Average of:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CONCENTRATION</strong></td>
<td>- Most of the gaming activities were related to the learning task.</td>
</tr>
<tr>
<td></td>
<td>- Generally speaking, I could remain concentrated in the game.</td>
</tr>
<tr>
<td></td>
<td>- Workload in the game was adequate.</td>
</tr>
<tr>
<td><strong>FEEDBACK</strong></td>
<td>- I received feedback on my progress in the game.</td>
</tr>
<tr>
<td></td>
<td>- I received immediate feedback on my actions.</td>
</tr>
<tr>
<td><strong>CHALLENGE</strong></td>
<td>- The game provides “hints” as part of the interface that help overcome challenges.</td>
</tr>
<tr>
<td></td>
<td>- The game provides “online support” that help overcome challenges.</td>
</tr>
<tr>
<td></td>
<td>- The difficulty of challenges increased as my skills improved.</td>
</tr>
<tr>
<td><strong>SOCIAL INTERACTION</strong></td>
<td>- The game supports cooperation and competition.</td>
</tr>
<tr>
<td></td>
<td>- The game supports social interaction between players.</td>
</tr>
<tr>
<td><strong>IMMERSION</strong></td>
<td>- I forgot about time passing while playing the game.</td>
</tr>
<tr>
<td></td>
<td>- I became unaware of my surroundings while playing the game.</td>
</tr>
<tr>
<td></td>
<td>- I temporarily forgot worries about everyday life while playing the game.</td>
</tr>
<tr>
<td></td>
<td>- I felt emotionally involved in the game.</td>
</tr>
<tr>
<td><strong>KNOWLEDGE</strong></td>
<td>- The game increased my knowledge.</td>
</tr>
<tr>
<td></td>
<td>- I tried to apply my knowledge in the game.</td>
</tr>
<tr>
<td></td>
<td>- The game motivates the player to integrate the knowledge taught.</td>
</tr>
<tr>
<td><strong>GOAL CLARITY</strong></td>
<td>- Overall game goals were presented clearly.</td>
</tr>
<tr>
<td><strong>AUTONOMY</strong></td>
<td>- The game interface and mechanics was easy to learn and use.</td>
</tr>
<tr>
<td></td>
<td>- I felt a sense of control over the game (interface, objects, input).</td>
</tr>
</tbody>
</table>

Students were asked to rate their level of agreement with each statement, where a rating of one means that the student strongly disagrees and a rating of seven means that the student strongly agrees. The results of this survey are presented in Table 2.
Table 2: GeoGame’s Student Survey Results

<table>
<thead>
<tr>
<th>Game dimension</th>
<th>GeoGame Survey Mean (SD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concentration</td>
<td>4.99 (1.4)</td>
</tr>
<tr>
<td>Feedback</td>
<td>4.23 (1.67)</td>
</tr>
<tr>
<td>Challenge</td>
<td>4.09 (1.44)</td>
</tr>
<tr>
<td>Social interaction</td>
<td>5.15 (1.57)</td>
</tr>
<tr>
<td>Immersion</td>
<td>2.5 (1.51)</td>
</tr>
<tr>
<td>Knowledge improvement</td>
<td>4.17 (1.65)</td>
</tr>
<tr>
<td>Goal clarity</td>
<td>4.96 (1.6)</td>
</tr>
<tr>
<td>Autonomy</td>
<td>4.08 (2.05)</td>
</tr>
</tbody>
</table>

The results show that the deployments of GeoGame excelled in concentration, social interaction, and goal clarity. This indicates that students easily understood the objectives of the game and were able to pursue those goals while effectively communicating with their peers. The shortcomings of these deployments were in the immersion and autonomy categories. This isn’t very surprising as the features creating an immersive environment, such as advanced map interaction, have not yet been fully developed. Also, given the complexity of the interface, the students experienced a learning curve which likely contributed to the low autonomy score. Overall the two deployments provided positive results in the two areas that were focused on by the instructors, knowledge improvement and social interaction, and justified further work on GeoGame.
Chapter 4: A Conceptual Model for Serious Gaming Applications

The goal in creating models for serious gaming applications is to enable rapid conceptualization of games from any domain. That is, using the models, a serious gaming application can be quickly and easily created by defining the problem domain in terms of an object and runtime model. These two models are presented below.

To create these models a number of different gaming scenarios had to be referenced. In addition to GeoGame, two others were identified. The first is a scavenger hunt game. The goal of the game is to allow incoming college freshmen to become familiar with their surroundings by satisfying various goals around campus. For example, goals could include eating at a local ethnic restaurant, attending a sporting event, or checking out a book from the library. The gaming application presents the goals, tracks player progress, and facilitates player interaction. The second scenario that was as a basis for the models was a suite of health care games. One example game from this collection is a healthy living game where the player guides a virtual presence through life by making various decisions that may help or hurt their overall health. The player makes decisions like when to exercise, how long to sleep, and what food to eat.

Object Model

The purpose of the object model is to provide game designers a way to easily define their game concept in terms of fundamental building blocks. The model was built by examining the commonalities between the three example scenarios (the GeoGame,
scavenger hunt, and health care scenarios) and extracting the most basic elements that were common throughout. Figure 6 shows the result of this effort.
Figure 6: Object Model
This model was built using a “minimal but complete” philosophy meaning that fairly complex serious gaming applications can be built on top of this model which consists only of the following six classes:

- **Entity**: An entity is a basic object in a game. These typically include players, locations, and objects that are manipulated through user and system actions.

- **Role**: In order to create events, an entity must first assume a role. For example a player may have to assume the “Administrator” role before creating an event to remove another player. Roles are assigned to specific entities.

- **Agent**: An agent is responsible for generating events. There are two types of agents: a role-entity agent and an interaction agent.

- **Event**: An event is a basic occurrence in a game. These can occur as the result of user actions, system actions (e.g. a timer firing), or random actions.

- **Rule**: A rule links an event to an interaction. There are two parts to a rule. The first part is a pattern matching clause that activates the rule when a certain situation is satisfied. The second part indicates what interaction(s) should occur when the rule is activated.

- **Interaction**: An interaction is an action that is taken by the application in response to an event. Interactions are fired by rules.

The driving force behind this model was the event-rule-interaction triplet. This can be easily compared to the popular event-condition-action (ECA) paradigm widely used in event-driven architectures. This methodology was chosen because it is easily
understandable and lends itself well to a rules engine, which is a main component of the software architecture to be presented later.

To better understand the object model it is useful to look at how it is applied to the three example scenarios described earlier. Table 3 shows these applications.
Table 3: Motivating Scenario Specifications

<table>
<thead>
<tr>
<th>Scenario</th>
<th>GeoGame</th>
<th>Scavenger Hunt</th>
<th>Healthy Living</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entity</strong></td>
<td>Player</td>
<td>Player</td>
<td>Avatar</td>
</tr>
<tr>
<td></td>
<td>Resource</td>
<td>Goals</td>
<td>• Health</td>
</tr>
<tr>
<td></td>
<td>Farm</td>
<td>Proof</td>
<td>Activity</td>
</tr>
<tr>
<td></td>
<td>• Family</td>
<td>• Images</td>
<td>Food</td>
</tr>
<tr>
<td></td>
<td>• Resource</td>
<td>• Video</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Land parcel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Weather</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fate event</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GIS layer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Round timer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Discussion forum</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Purchase</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Trade</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Role</strong></td>
<td>Player</td>
<td>Scavenger</td>
<td>Player</td>
</tr>
<tr>
<td></td>
<td>• Administrator</td>
<td>Jury</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Farmer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Agent</strong></td>
<td>Role-entity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• System-round timer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Farmer-player</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Event</strong></td>
<td>Round expiration</td>
<td>Submit proof</td>
<td>Eat</td>
</tr>
<tr>
<td></td>
<td>Purchase request</td>
<td>Accept proof</td>
<td>Sleep</td>
</tr>
<tr>
<td></td>
<td>Trade request</td>
<td>Reject proof</td>
<td>Exercise</td>
</tr>
<tr>
<td></td>
<td>Discussion posting</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chat posting</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Rule</strong></td>
<td>Process round</td>
<td>Process submission</td>
<td>Process activity</td>
</tr>
<tr>
<td></td>
<td>Process purchase</td>
<td>Process approval</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process trade</td>
<td>Process rejection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process discussion</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process chat</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Interaction</strong></td>
<td>Calculate yield</td>
<td>Notify scavenger</td>
<td>Affect health</td>
</tr>
<tr>
<td></td>
<td>Calculate consumption</td>
<td>Notify jury</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accept/reject purchase</td>
<td>Mark goal achieved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accept/reject trade</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Post discussion</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Post chat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Even though these scenarios are not highly complex, it is still helpful to know that they can be fully specified in terms of the object model. The way in which GeoGame fits into the object model will be examined in greater detail in Chapter 6.

Runtime Model

Once a gaming scenario has been statically defined, there is still the question of how the game will operate during execution. The specification of this operation is the goal of the runtime model, shown in Figure 7.
There are two phases of runtime: instantiation and execution. During instantiation the game specification is parsed by the engine. The roles and entities determine the objects in the game while the agents, rules, and interactions determine how they can interrelate. During the instantiation phase, the game logic is responsible for taking the
specification and using it to place the game in a state that is ready for execution. This process will vary widely between different applications.

After instantiation is complete, the game enters the main execution loop. This loop involves generating events, capturing events with rules, and responding to events through interactions. This process of creating and handling events continues until the game terminates. Notice that an element of recursion is involved in execution. An event is captured by a rule that fires an interaction. This interaction may in turn lead to another event being posted, continuing the cycle.
Chapter 5: The Architecture to Support the Models

Before the models presented in the previous chapter can be validated, the architecture behind the models must be defined. This architecture can be split into two concerns: knowledge capture and runtime execution. Overall this architecture is vaguely reminiscent of the one presented in [18]. Here OWL and KML fulfill the roles of the location, object, and role builders while Jess is similar to the scenario builder.

Knowledge Capture

The knowledge capture part of the architecture is responsible for supporting the game specification that was created using the object model. The technology selected must be able to capture instances of entities, roles, and agents, as well as the relationships between these instances.

Web Ontology Language (OWL)

Web Ontology Language (OWL) is a language based on RDF and XML that is used to represent knowledge that must be processed by applications, rather than read by humans [19]. As the name implies, OWL was developed with a special focus on Semantic Web applications, but has been widely adopted in many other domains where complex domain knowledge must be captured and processed.

OWL was selected because it is more expressive than XML, the technology originally targeted for domain knowledge capturing. Also, OWL is a World Wide Web Consortium (W3C) standard that has been widely adopted and for which a variety of
tools exist. These tools include Protégé [20] which was used throughout this work to manipulate ontologies.

Keyhole Markup Language (KML)

KML was selected for the model’s architecture for the same reasons it was selected for the initial GeoGame implementation. It has a simple representation scheme that can be created with a basic XML editor, and has almost universal support among GIS packages. Its specification, while simple, can support the GIS needs of typical serious gaming applications.

Runtime Execution

As previously described, the runtime of this model can be broken into two phases: instantiation and execution. The execution phase is a continual process of generating, capturing, and responding to events. This type of execution model is perfectly suited to a rules engine.

Java Expert System Shell (Jess) Rules Engine

The Java Expert System Shell (Jess) is a rules engine for the Java platform based on CLIPS [21]. Jess uses the Rete algorithm, which increases the speed of pattern matching queries by representing rules and their conditions as nodes in trees. These nodes are augmented with information about the facts in the working memory of the engine [22].

In choosing a rules engine the decision eventually came down to Jess versus Drools Expert, an expert system that is part of the larger Drools platform offered by JBoss. Both are freely available: Jess has free academic licensing and Drools is open-
source. Ultimately the decision to use Jess was made because of its extensive
documentation and ease of integration with the existing GeoGame application.
Chapter 6: GeoGame and the Conceptual Models

To validate the object and runtime models developed in the previous two chapters, as well as their architecture, a new implementation of GeoGame was created that employed the models. This process took the first version of GeoGame, which was built in a “hard-wired” fashion (i.e. with all entities, roles, etc. defined within the code) and converted it to a more flexible, declarative implementation. The domain knowledge that was originally contained in Java code, such as the types of resources to be traded and the numbers of players in a simulation, was pulled out into an external OWL file. Also, rules and interactions that were originally hard-coded, such as the yield and consumption calculations, were placed in an external Jess rules file that could be easily modified. This new implementation would not only validate the object and runtime models presented earlier, but also allow GeoGame administrators much more control over how the game would be played.

To begin this process GeoGame was conceptualized in terms of the object and runtime models. This showed that the transformation was possible and allowed the actual implementation to begin.

Conceptualization

The first step in creating the model-based version of GeoGame was to specify the game in terms of the object model. Figure 8 shows this specification.
Figure 8: GeoGame’s Object Model
The entities group defines the individual objects in GeoGame. There are three roles. The player can be a farmer (student) or administrator (instructor). There is also a system role for periodic and randomized events. The process of mapping events to interactions is clear. For example, the round expiration event is caught by the round processing rule that activates the yield and consumption calculation interactions. Note that the entire specification of GeoGame is relatively small. For larger, more complex games the task of specification may be more difficult. This issue will be addressed in future work surrounding scalability.

After GeoGame had been specified in terms of the static object model, its execution needed to be shown to conform to the runtime model. Figure 9 shows the most interesting of GeoGame’s runtime models – the round processing procedure.
This procedure is broken into two phases. During the instantiation phase, the OWL ontology is parsed and Java objects are created as necessary. The Jess rules file is also parsed and the rules engine is initialized. All of this occurs during simulation startup, after which the main execution phase begins. The execution of the round processing procedure begins when the round timer, an entity, assumes the system role and generates a round termination event. This event is loaded into the working memory of Jess and thereby activates the round processing rule. This rule initiates the yield and
consumption interactions. These interactions carry out the logic of the calculations, calling methods on Java objects from within the Jess code to do so.

In the actual implementation of the round processing procedure, the sub-interaction model shown above was not followed. Instead, round processing was carried out in one large interaction which took care of both yield and consumption calculations. This was done in order to minimize Jess overhead and ensure that the interaction would fit into the 100ms time limit for Project Darkstar transactions. However, the overall concept of both execution methods is the same.

Runtime diagrams for other GeoGame procedures are contained in Appendix A.

Implementation

The conceptualization process proved that GeoGame could be specified using the object and runtime models. The next step was to actually implement GeoGame following those models. The goal of this implementation was to show that the conceptual models could be realized in a concrete application. The decision was made to only implement a representative subset of GeoGame functionality rather than changing the entire application. There were several reasons for this decision, the main one being the project’s time constraints. Also, it wasn’t necessary for GeoGame to be entirely converted for the game’s stakeholders (mainly geography instructors) to realize the flexibility benefits they desired.

To reinforce the advantages this partial implementation approach, consider two parts of the game play: round processing and resource purchasing. The round processing procedure was changed from the original GeoGame implementation to fully comply with
the object and runtime models. This means that a round expiration event is generated and handled by a round processing rule. This rule fires yield and consumption calculation interactions. Since the interactions are specified in a declarative rules file they can be easily modified. This is convenient for GeoGame administrators who may need to tweak these calculations to adjust the difficulty of the game. Conversely, the resource purchasing process was not converted to the event-rule-interaction model but was instead left as it was in the first implementation of GeoGame, i.e. “hard-coded.” But since the purchase process is the same for every instantiation of GeoGame, administrators would gain nothing if this process were pulled out into a declarative rules file. Also, since round processing is the most complex procedure in the game, converting less complex procedures to the model-based version would not prove anything that the conversion of the round expiration process had not already shown.

The remainder of this chapter describes the implementation process. The majority of time was spent on coding changes. The rest of the work was in defining the OWL ontology, creating KML data, and writing Jess rules.

**Coding Changes**

The first step in modifying GeoGame’s implementation was to make a number of coding changes that would make the rest of the conversion easier. These changes included things like making all interactions (e.g. purchasing and trading) first-class objects and making roles and agents explicit. No new functionality was added during this process.
After these stylistic and other conceptual changes were made, work began on integrating the necessary technology to support the architecture of the models. The first technology was OWL API [22], which is used to load and parse the OWL ontology files for each simulation. When a simulation is created, OWL API is used to search for specific elements in the ontology and then create Java objects related to those ontology individuals.

Integrating the Jess rules engine was a much more complicated process than that of the OWL API integration. This complication arose because of GeoGame’s reliance on Project Darkstar. Typically, an application can load plain Java objects into the working memory of Jess and manipulate those objects by calling their methods. However, because of the way Darkstar persists objects, by writing an object to the database at the end of each transaction and recreating it in subsequent transactions, any object loaded into Jess’ working memory becomes out-of-date in as soon as the transaction that loaded it expires.

For example, when a player joins a simulation, their farm is loaded into working memory. At the end of this transaction the farm is written to the database. Then when the first round expires the farm needs to be updated to account for the yield and consumption of the elapsed round. Thus Darkstar begins the transaction by pulling a new copy of the farm out of the database. But since this is no longer the same object as the one in Jess’ working memory (even though they are equal), Jess is unable to update Darkstar’s new version of the farm. This problem was solved by loading updated copies of all needed objects into Jess’ working memory at the beginning of any transaction in
which rules were fired. This solution naturally has a negative effect of performance, which is discussed in Chapter 7.

*Defining the Ontology*

The GeoGame ontology was broken into two separate files. The base file defines all the types of objects and relationships that can occur in the ontology. The simulation-specific file imports the base file and contains the actual instances of the types. In OWL terms, the base file contains “classes” while the simulation-specific file contains “individuals.” The base ontology file is described in Figure 10.
Figure 10: GeoGame’s Base OWL Ontology

This separation between the base and simulation-specific ontologies takes advantage of the commonalities among simulation ontology files. For example, it is common for a GeoGame administrator to change the types of resources being purchased and traded, but much less likely for them to introduce entirely new object classes. A portion of the GeoGame OWL file is shown in Table 4.
Table 4: GeoGame’s OWL Ontology

```xml
<GeoGame:Family rdf:about="#DefaultFamilyDulai">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <GeoGame:hasSavings rdf:datatype="float">1700.0</GeoGame:hasSavings>
  <GeoGame:hasChildren rdf:datatype="integer">4</GeoGame:hasChildren>
  <GeoGame:hasAdults rdf:datatype="integer">5</GeoGame:hasAdults>
  <GeoGame:hasName rdf:datatype="string">Dulai</GeoGame:hasName>
  <GeoGame:hasFile rdf:datatype="string">dulai.gif</GeoGame:hasFile>
</GeoGame:Family>
```

Note that some of the notation has been simplified. For example an attribute like `rdf:datatype="float"` in Table 4 would actually read `rdf:datatype="http://www.w3.org/2001/XMLSchema#float"` in the OWL file. The `hasFile` relationship has also been simplified.

**Defining GIS Data**

Land parcel data is captured using KML. Each parcel is defined by a linear ring with a place marker in the center. The linear ring is used to delineate the boundaries of the parcel. The place marker is used to handle click events that either display the parcel’s information or select the parcel for purchase or trade. Each parcel also contains custom data such as area, a brief description, and a multiplier used in the yield calculation. This data is specific to GeoGame and not part of the KML specification. A portion of a GeoGame KML file is shown in Table 5.
Table 5: GeoGame’s KML

```xml
<Placemark>
  <name>Smith Farm</name>
  <styleUrl>#defaultStyle</styleUrl>
  <ExtendedData>
    <SchemaData schemaUrl="#ParcelTypeId">
      <SimpleData name="ParcelPrice">300</SimpleData>
      <SimpleData name="ParcelQuality">1.32</SimpleData>
      <SimpleData name="ParcelArea">1.5</SimpleData>
    </SchemaData>
  </ExtendedData>
  <Point>
    <coordinates>74.76172872406,29.95498620573,0</coordinates>
  </Point>
  <description>Smith Farm</description>
  <visibility>1</visibility>
</Placemark>
```

In addition to the parcel data, image overlays are used to cope with NASA World
Wind’s low-resolution imagery. By placing a high-resolution overlay underneath the
parcel layer, players are able to see how the parcels correspond to real-world farm land.

Note that a simulation’s GIS data is completely decoupled from its ontology.
This allows the same simulation-specific ontology to be used for simulations all around
the globe by pairing it with different KML files.

**Writing Rules**

As described earlier, only a representative part of the GeoGame implementation
was made to conform to the object and runtime models. Specifically, the round
processing interaction was identified as a case study for which to write Jess rules. This
involved deciding what objects to load into the working memory of the Jess engine and
how those objects would be manipulated to fulfill round processing duties. Part of the
result of this effort can be seen in Table 6.
Table 6: GeoGame’s Jess Rules

(defrule process-round
  ; Get round number and banker, weather, and farm facts
  (Round {round >= 0} (round ?round))
  ?bankerFact <- (Banker)
  ?weatherFact <- (Weather)
  ?farmFact <- (Farm)

  =>

  ; Get Java objects from Jess facts
  (bind ?farm (?farmFact getSlotValue "OBJECT"))
  (bind ?banker (?bankerFact getSlotValue "OBJECT"))
  (bind ?weather (?weatherFact getSlotValue "OBJECT"))

  ; Calculate consumption and yield
  (calculate-consumption ?farm ?banker ?round ?scoreSheet)

  ; Broadcast score sheet
  (?scoreSheet broadcast)
)

; Calculate consumption for a farm and update savings accordingly
(deffunction calculate-consumption (?farm ?banker ?round ?scoreSheet)
  ; Only calculate consumption for each farm once per round
  (bind ?lastRoundProcessed (?farm getLastRoundProcessed))
  (if (> ?round ?lastRoundProcessed) then
    ; Calculate consumption expense
    (bind ?bushels
      (bushel-consumption
        (?farm getQuantity "Adult" ?round)
        (?farm getQuantity "Child" ?round)
      )
    )
    (bind ?consumption
      (*
        ?bushels
        (?banker getPrice "Wheat" (- ?round 1))
      )
    )
  )

  ; Round consumption expense to nearest cent
  (bind ?consumption (round-currency ?consumption))

  ; Deduct expense from savings
  (bind ?curr (?farm getQuantity "Savings" ?round))
  (?farm setQuantity "Savings" (- ?curr ?consumption) ?round)

Continued
This snippet from the rules file shows how Jess’ Lisp-style syntax is used to manipulate Java objects. The process-round rule is activated once per farm per round. Its interaction is carried out by calling functions to calculate consumption and yield for the elapsed round. The calculate-consumption function is shown for reference. For the complete Jess rules file, see Appendix B.

Before Jess rules were implemented for GeoGame one potential benefit that was identified was that GeoGame administrators could easily modify the way yield and consumption calculations were performed. In doing so the difficulty of the game could be changed without significant effort. However this depended upon administrators, who could not be assumed to have a programming background, being able to understand and manipulate the rules file. This has proven somewhat impractical. While most administrators can look at a rules file and make simple modifications, most don’t wish to put in the time it would take to fully understand the inner workings and be able to write a rules file from scratch. Future work will look into creating a toolset for the rules system that simplifies the process for non-programmers.
Chapter 7: Analysis of Model-Based GeoGame

This section analyzes the process and results of partially converting the hard-wired version of GeoGame to the model-based version.

Flexibility

A major benefit of the model-based version of GeoGame as compared to the hard-wired implementation is the flexibility afforded to the game’s administrators. After pulling domain knowledge out of Java code and placing it into an external OWL file, instructors were able to modify many of the entities of GeoGame. An instructor could change the price of wheat in a specific round, decrease the likelihood of a birth event, or create a new resource called “corn.” Further flexibility is granted by interactions that were pulled from the Java code and placed into an external Jess rules file. This allows instructors to change how yield and consumption are calculated.

This flexibility, however, depends upon the instructor’s ability and willingness to create or modify GeoGame’s OWL and rules file. In practice the motivation to do so is hampered by a number of factors. The easiest way to manipulate OWL files is to use Protégé (although a text editor may be used), which is unfamiliar software to most geography instructors, and thus poses a learning curve. The rules file, while not requiring anything beyond a simple text editor, is complex to anyone without a programming background. The future work section of Chapter 9 will discuss ways to potentially overcome these drawbacks.
Performance

When converting GeoGame from a hard-wired version to one that more closely implemented the conceptual object and runtime models, a decline in performance was expected. This is typical for most frameworks, where an increase in generality is gained only by sacrificing performance. Figure 11 shows the performance of the round processing procedure (chiefly composed of yield and consumption calculations) for the original hard-wired version of GeoGame versus the version based on the models.

![Round Processing Time - Model-Based vs. Hard-Wired](image)

Figure 11: GeoGame’s Round Processing Performance
In essence this comparison is between rules implemented in Java (hard-wired) and rules implemented in Jess (model-based). The rest of the process involves Project Darkstar code which is shared by both the hard-wired and model-based versions.

There are two things to note from this chart. First, regardless of the number of players, the time taken by the model-based version to process a round is always more than twice that of the hard-wired version. This is not surprising as the process of looping through a set of rules and activating individual ones is naturally slower than processing the round in straight-line Java code. Also, there is time consumed by the Jess-Darkstar interaction fix discussed in the “Coding Changes” section of Chapter 6. Second, performance of the model-based version declines more rapidly than that of the hard-wired version as more players are added. For example, the percent change from the time needed to process one player versus the time needed to process five players is about 106% for the model-based version. The percent change for the same player increase is only 49% for the hard-wired version. Also, the increase in processing time is much more consistent for the hard-wired version. Clearly the model-based version of GeoGame is less scalable than the hard-wired version.

Appendix C contains the data behind Figure 11 as well as more detailed auxiliary charts.

Development

A fundamental goal of the model-based approach was to enable rapid development of serious gaming applications. To address this goal, two factors were examined from GeoGame. First, the size of the rules file was compared to the Java
functionality it was replacing. This comparison consisted entirely of the round processing procedure. When converting this procedure from hard-wired to model-based, 281 lines of Java source code were replaced by 291 lines of Jess rules. This shows that there is no significant increase in the amount of work it would take to capture functionality in Jess as opposed to Java.

Another metric that was deemed useful was the amount of time it took to develop GeoGame from the ground up versus the amount of time it took build a model-based version. Unfortunately this exact metric could not be gathered for a number of reasons. The initial purpose-built version of GeoGame was built without regard to this research and therefore development time was not tracked accurately. Also, as described earlier, the conversion of GeoGame to the model-based version was not total, but rather it consisted of a representative subset of functionality. The best approximation to a development time comparison had to be culled from SVN commit dates throughout the project. Based on this data the amount of time to create a partially model-based version of GeoGame was about 25% of the time it took to build a hard-wired version of GeoGame from the ground up. Again this figure is only a very rough estimate. Also, given the large amount of time devoted to Jess-Darkstar integration issues and the decrease in developer availability in the conversion phase of the project, the 25% figure is likely to be much lower in future practice.
Chapter 8: Conclusion

This work began by creating a purpose-built serious gaming application called GeoGame that was deployed in an introductory college geography class with moderate success measured through student surveys. From this application, generic object and runtime models were developed to describe serious gaming applications in a flexible way. The software architecture to support these models used OWL, KML, and Jess as major components. After validating these models with alternate scenarios, GeoGame was partially converted to conform to these models. This model-based version dramatically increased flexibility but also incurred a significant performance overhead.
Chapter 9: Future Work

This section describes areas of further research that must be pursued in order to fully validate the models presented in this work.

Metrics

It would be helpful to create a set of metrics that could be used to quantify the success or failure of these models and their architecture. This would involve defining user surveys, development time statistics, and performance requirements that could be used for concrete, quantifiable evaluation. The end result could contain something like the question list in [24]. The metrics presented in this work were too few and qualitative in nature.

Scalability

As shown in Chapter 7, performance will be a huge issue in using these models and their software architecture in practice. Work must be done to minimize the performance degradation in the future. Since the majority of the observed performance impact was due to the Jess rules engine, this work might begin by first examining other rules engines that perform better than Jess.

In addition to investigating other rules engines, the models need to be employed by a complex, highly synchronous game. This will further define the scalability problems that exist and also demonstrate how difficult it will be to write ontology and
rules files for an application that is significantly more complex than GeoGame and the alternative validation scenarios presented here.

Design Complexity Reduction

One major benefit of the models is the flexibility that they provide. However, this flexibility depends upon laymen being able to modify the ontology and rules files upon which model-based serious gaming applications will be based. The current method of having users manually create and modify OWL and Jess files is not ideal. Instead, a simplified toolset should be developed that hides complexity while preserving functionality. This would likely involve building a simplified ontology editor that would not be as in-depth as Protégé, and a similar editor for rules files. Such a toolset would greatly increase the usability of this model-based approach, especially for users without an extensive technical background.

Extending to a Framework

One major goal of future work is to extend the conceptual models presented here, along with the software architecture needed to implement them, into a full-blown serious gaming framework. This framework would provide the core services needed to develop and deploy a serious gaming application that has been developed according to the conceptual models. The Mirror project currently underway at the Collaborative for Enterprise Transformation and Innovation (C.E.T.I) at The Ohio State University seeks to develop a cyber infrastructure that would fulfill the role of this framework as part of a larger initiative.
The Mirror Project

Mirror is a uniquely configurable geospatial cyber infrastructure that is intended to enable collaboration between citizens at a local level. This collaboration is often manifested in game-like scenarios and is intended to build capacity for research and effective local government and community services. Target areas for Mirror are diverse and include many domains that traditionally make extensive use of serious gaming, such as health and education.

The benefits that Mirror would provide to a serious gaming application are numerous. Its service-based architecture would allow rapid deployment since designers could tap into established robust functionality, rather than creating their own. These services could be used to increase the dynamism of a game. For example, Mirror services could be developed that would report current weather conditions and commodity prices. These services could then be fed into GeoGame so that weather and resource prices would reflect real-time, real-world data, instead of arbitrary values predefined by the game administrator. In addition to outside data, Mirror could also facilitate data sharing between applications (with the approval of the applications’ administrators). For example, the data stored by Mirror for an application intended to simulate the stock market could be used to drive market prices in GeoGame. Other potential benefits of Mirror include the availability of advanced data analysis tools and the establishment of an enterprise services bus to perform tasks such as automated data backup.

Mirror is currently in the early stages of development. Work is being devoted to the core services needed to get Mirror up and running. After this, the development of
external services will begin. During this stage Mirror will slowly develop the services necessary to act as a framework to implement the models presented here.
References


   history_-livelihoods%29. 4 May 2010.


   May 2010.


[16] Fu, Fong-Ling, Su, Rong-Chang, and Yu, Sheng-Chin. “EGameFlow: A 
   scale to measure learners’ enjoyment of e-learning games.” Computers & 


[23] “The OWL API.” University of Manchester.

http://owlapi.sourceforge.net/2.x.x/index.html. 5 May 2010.

Appendix A: GeoGame’s Runtime Diagrams

This appendix contains runtime diagrams of various GeoGame procedures. These diagrams are meant to accompany the runtime diagram of the round processing procedure (Figure 9) presented in Chapter 6.

Figure 12: GeoGame’s Purchase Procedure
Figure 13: GeoGame’s Trade Procedure

Figure 14: GeoGame’s Fate Procedure
Appendix B: GeoGame’s Rules Files

This appendix contains an example of a Jess rules file that would be used by a GeoGame simulation.

(import geogame.server.*)

; Global variables
; PARAMETER: An adult consumes nine bushels per year
(defglobal ?*adultConsumption* = 9)
; PARAMETER: A child consumes five bushels per year
(defglobal ?*childConsumption* = 5)
; PARAMETER: An ox can work four acres of land per year
(defglobal ?*oxCoverage* = 4)
; PARAMETER: An adult can work two acres of land per year
(defglobal ?*adultCoverage* = 2)
; PARAMETER: Four bushels of HYC seed cover one acre of land
(defglobal ?*hycBushelsPerAcre* = 4)
; PARAMETER: Irrigation covers one acre of land
(defglobal ?*irrigationCoverage* = 1)
; PARAMETER: The maximum amount of fertilizer per acre of land is 80 pounds
(defglobal ?*fertilizerPerAcre* = 80)

; Classes from Java
(defclass Round RoundWrapper)
(defclass Farm FarmStatus)
(defclass Banker ResourceCollection)
(defclass Weather WeatherCollection)
(defclass ScoreSheet ScoreSheet)

;/* ##### RULES ##### */
; Process round by calculating consumption and yield
(defrule process-round
 ; Get round number and banker, weather, and farm facts
 (Round {round >= 0} (round ?round))
 ?bankerFact <- (Banker)
 ?weatherFact <- (Weather)
 ?farmFact <- (Farm)
 =>
 ; Get Java objects from Jess facts
(bind ?farm (?farmFact getSlotValue "OBJECT"))
(bind ?banker (?bankerFact getSlotValue "OBJECT"))
(bind ?weather (?weatherFact getSlotValue "OBJECT"))

; Update score sheet with weather and wheat price
(bind ?scoreSheet (%farm getScoreSheet))
(bind ?wheatPrice (?banker getPrice "Wheat" (- ?round 1)))
(bind ?weatherDesc (?weather getWeatherDescription (- ?round 1)))

(?scoreSheet addValue
  (get-member ScoreSheet CONDITIONS_HEADER)
  (get-member ScoreSheet WHEAT_PRICE_ROW_TITLE)
  ?wheatPrice
  (- ?round 1) ; Set value for previous round
)

(?scoreSheet addValue
  (get-member ScoreSheet CONDITIONS_HEADER)
  (get-member ScoreSheet WEATHER_ROW_TITLE)
  ?weatherDesc
  (- ?round 1) ; Set value for previous round
)

; Calculate consumption and yield
(calculate-consumption ?farm ?banker ?round ?scoreSheet)

; Broadcast score sheet
(scoreSheet broadcast)

/* #### FUNCTIONS #### */
; Calculate consumption for a farm and update savings accordingly
(deffunction calculate-consumption (?farm ?banker ?round ?scoreSheet)
  ; Only calculate consumption for each farm once per round
  (bind ?lastRoundProcessed (?farm getLastRoundConsumptionProcessed))
  (if (> ?round ?lastRoundProcessed) then
    ; Calculate consumption expense
    (bind ?bushels
      (bushel-consumption
        (?farm getQuantity "Adult" ?round)
        (?farm getQuantity "Child" ?round)
      )
    )
    (bind ?consumption
      (*
        ?bushels
        (?banker getPrice "Wheat" (- ?round 1))
      )
    )
    ; Round consumption expense to nearest cent
    (bind ?consumption (round-currency ?consumption))
  ))
; Add consumption categories to score sheet
(scoreSheet addValue "Consumption" "Bushels" ?bushels (- ?round 1))
; Set value for previous round
(scoreSheet addValue "Consumption" "Expense" ?consumption (- ?round 1))
; Set value for previous round

; Deduct expense from savings
(bind ?currentSavings (farm getQuantity "Savings" ?round))
(farm setQuantity "Savings" (- ?currentSavings ?consumption) ?round)

; Broadcast new savings value to client
(farm broadcastUpdate)

; Mark this round as processed
(farm setLastRoundConsumptionProcessed ?round)

; Returns the number of bushels consumed by adults and children
(deffunction bushel-consumption (?adults ?children)
  (bind ?adultConsumption (* ?adults *adultConsumption*))
  (bind ?childConsumption (* ?children *childConsumption*))
  (return (+ ?adultConsumption ?childConsumption))
)

; Calculate yield for a farm and update savings accordingly
(deffunction calculate-yield (?farm ?banker ?weather ?round scoreSheet)
  ; Only calculate yield for each farm once per round
  (bind ?lastRoundProcessed (farm getLastRoundYieldProcessed))
  (if (> ?round ?lastRoundProcessed) then
    ; Calculate arable acreage
    (bind ?arableLand (calculate-arable-land ?farm ?round)))
  ; Add "Arable Land" yield category to score sheet
  (scoreSheet addValue "Yield" "Arable Land" ?arableLand (- ?round 1))
  ; Set value for previous round
  (bind ?seed (farm getQuantity "HYC Seed" ?round))

  ; Avoid division by zero for HYC acreage
  (bind ?hycLand 0)
  (bind ?hycBushels ?*hycBushelsPerAcre*)
  (if (<> ?hycBushels 0) then
    (bind ?hycLand (min ?arableLand (/ ?seed ?hycBushels))))
)
  (bind ?lrLand (- ?arableLand ?hycLand))

; Add "HYC Land" and "Land Race Land" yield categories to score sheet
(?scoreSheet addValue "Yield" "HYC Land" ?hycLand (- ?round 1)) ; Set value for previous round
(?scoreSheet addValue "Consumption" "LR Land" ?lrLand (- ?round 1)) ; Set value for previous round

; Calculate irrigation acreage
(bind ?wells (?farm getQuantity "Irrigation" ?round))
(bind ?irrigatedLand (min
    ?arableLand
    (* ?wells *irrigationCoverage*)
  )
)
(bind ?nonIrrigatedLand (- ?arableLand ?irrigatedLand))

; Add "Irrigate Land" and "Non-irrigated Land" yield categories to score sheet
(?scoreSheet addValue "Yield" "Irrigated Land" ?irrigatedLand (- ?round 1)) ; Set value for previous round
(?scoreSheet addValue "Yield" "Non-irrigated Land" ?nonIrrigatedLand (- ?round 1)) ; Set value for previous round

; Calculate total bushel yield

; Add "Bushels" yield category to score sheet
(?scoreSheet addValue "Yield" "Bushels" ?bushels (- ?round 1)) ; Set value for previous round

; Calculate total yield income
(bind ?income (*
    ?bushels
    (?banker getPrice "Wheat" (- ?round 1))
  )
)

; Round yield income to nearest cent
(bind ?income (round-currency ?income))

; Add "Income" yield category to score sheet
(?scoreSheet addValue "Yield" "Income" ?income (- ?round 1)) ; Set value for previous round

; Add income to savings
(bind ?currentSavings (?farm getQuantity "Savings" ?round))
(?farm setQuantity "Savings" (+ ?currentSavings ?income) ?round)

; Broadcast new savings value to client
(?farm broadcastUpdate)
; Mark this round as processed
(farm setLastRoundYieldProcessed ?round)
)
)

; Returns the number of arable land units on a farm
(defun calculate-arable-land (farm round)
  (bind manPower (+
    (farm getQuantity "Adult" round)
    (farm getQuantity "Laborer" round))
  )
  (bind oxenPower (farm getQuantity "Ox" round))
  (bind landOwned (farm getQuantity "Land" round))
  (bind workableLand (+
    (* oxenPower *oxCoverage*)
    (* (- manPower oxenPower) *adultCoverage*)
  )
  )
)

; Return arable land
(return (min
  ?landOwned
  ?workableLand
))
)

; Returns the number of bushels yielded by a farm
(defun calculate-bushel-yield (farm weather round hycLand lrlAnd irrigatedLand nonIrrigatedLand)
  ; Put land into categories (best to worst: HYC & irrigated, LR & irrigated, LR & non-irrigated, HYC & non-irrigated)
  (bind catOne (min hycLand irrigatedLand))
  (bind catTwo (- irrigatedLand catOne))
  (bind catThree (- lrlAnd catTwo))
  (bind catFour (- nonIrrigatedLand catThree))

  ; Get weather multipliers
  (bind weatherMultiplier (weather getWeatherMultiplier (- round 1))) ; Use previous round's multiplier
  (bind maxWeatherMultiplier (weather getMaxWeatherMultiplier))

  ; Get amount of fertilizer per land unit
  (bind fertilizer (farm getQuantity "Fertilizer" (- round 1))) ; Avoid division by zero for fertilizer per land unit
  (bind fertPerUnit 0)
  (if (<> arableLand 0) then
    (bind fertPerUnit (/ fertilizer arableLand))
  )
)
; Calculate bushel yield from HYC & irrigated (irrigated land
uses the max weather multiplier)
(bind ?catOneBushels
(*
  ?catOne
  (* 3.08 (** 1.01 ?fertPerUnit))
  (** 1.48 ?maxWeatherMultiplier)
)
)

; Calculate bushel yield from LR & irrigated (irrigated land
uses the max weather multiplier)
(bind ?catTwoBushels
(*
  ?catTwo
  (*
   (+
     (* 0.0002 (** ?fertPerUnit 2))
     (* 0.0046 ?fertPerUnit)
     15.08
   )
  (** 1.14 ?maxWeatherMultiplier)
)
)

; Calculate bushel yield from HYC & non-irrigated
(bind ?catThreeBushels
(*
  ?catThree
  (* 3.08 (** 1.01 ?fertPerUnit))
  (* 1.48 ?weatherMultiplier)
)
)

; Calculate bushel yield from LR & non-irrigated
(bind ?catFourBushels
(*
  ?catFour
  (+
   (* 0.0002 (** ?fertPerUnit 2))
   (* 0.0046 ?fertPerUnit)
   15.08
  )
  (** 1.14 ?weatherMultiplier)
)
)

; Return total bushels for all categories
(return (+ ?catOneBushels ?catTwoBushels ?catThreeBushels
?catFourBushels))
)
; Returns a number rounded to two decimal places
(deffunction round-currency (?amount)
  (return
    (/ 
      (round (* ?amount 100))
      100
    )
  )
)
)
Appendix C: Performance Analysis of GeoGame

This section contains the data and charts related to the performance analysis of GeoGame’s round processing procedure provided in Chapter 7.

Table 7: Processing Times for GeoGame’s Round Processing Procedure

<table>
<thead>
<tr>
<th># Players</th>
<th>Round</th>
<th>Model-Based</th>
<th>Hard-Wired</th>
<th>Round Processing Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>49</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>24</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>21</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>24</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>29.5</td>
<td>13.5</td>
<td>Model-Based</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>35.5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>24</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>19</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>19.5</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>24.5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>45</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>19</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>22</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>24</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>27.5</td>
<td>13.75</td>
<td>Model-Based</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>51</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>17</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>24</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>22</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>28.5</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

Continued
Table 7 Continued

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>60</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>38.5</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>37</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>30</td>
<td>16</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>41.375</strong></td>
<td><strong>15.75</strong></td>
<td><strong>Model-Based</strong></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>56</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>34</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>35.5</strong></td>
<td><strong>14.75</strong></td>
<td><strong>Hard-Wired</strong></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>50.5</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>37</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>29</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>34</td>
<td>18</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>37.625</strong></td>
<td><strong>16.75</strong></td>
<td><strong>Model-Based</strong></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>61</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>41</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>42</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>38</td>
<td>18</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>45.5</strong></td>
<td><strong>16.75</strong></td>
<td><strong>Hard-Wired</strong></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>67</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>47</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>52</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>52</td>
<td>18</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>54.5</strong></td>
<td><strong>19.25</strong></td>
<td><strong>Model-Based</strong></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>68</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>51</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>53</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>55</td>
<td>14</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>56.75</strong></td>
<td><strong>18.75</strong></td>
<td><strong>Hard-Wired</strong></td>
</tr>
</tbody>
</table>
Figure 15: Round Processing Times for the Hard-Wired Version of GeoGame
Figure 16: Round Processing Times for the Model-Based Version of GeoGame