Software Specification of MERTIS: Modifiable, Extensible Real-Time Interactive Simulation System

Frederick C. Harris, Jr., Leandro Basallo, Ryan Leigh, Regan Snyder, and Sam Talaie

Department of Computer Science and Engineering
University of Nevada,
Reno, NV 89557
{fredh, lbasallo, leigh, rsnyder, talaie}@cs.unr.edu

Abstract

Game and simulation development is a difficult process because there are many low level infrastructure concerns that need to be addressed. This is a barrier to development for inexperienced programmers and distracts from pure game (and simulation) design. MERTIS seeks to ease the development process by reducing the design to the important elements, removing software design from the simulation design. This is achieved by using an extensible scripting engine that allows all elements of a simulation to be specified outside the operation of the program and dynamically loaded at run-time. The only foreseen limitation at this time is that the graphics used in the script will be two-dimensional. This paper outlines the motivation for, and the development of, MERTIS. It further discusses the specific requirements, functional and non-functional, with the aid of the Unified Modeling Language (UML).

Keywords: Software specifications, UML, Modifiable, Extensible, Real-Time, Interactive, Simulation.

1. Introduction

MERTIS (Modifiable Extensible Real-Time Interactive Simulation) is a software system that will allow the user to design a simulation based on its 2D engine. The user will be able to create these simulations by writing simple XML script files [8] rather than writing full-scale programs.

What this means is that a MERTIS user does not have to be an experienced programmer who has prior knowledge of developing graphical simulations and games. If a programmer were to create a game from scratch, development would begin at a very basic level before any actual simulation development took place. A graphics engine must be created for rendering. Algorithms such as artificial intelligence and simulated physics must be implemented from scratch as well. Furthermore, this implies that the user must have strong high-level programming skills. When creating a game from scratch, the user must be very familiar with programming tools and techniques, while keeping in mind the speed and efficiency of the program.

MERTIS is a tool that will allow the development of games and simulations as quickly as possible and as easily as possible. The users can focus on game design and development, without concerning themselves with the issues of programming an entire game from scratch. MERTIS is designed to handle all those issues internally, and it serves as a palette for design and creativity, rather than a palette for technical skill and knowledge.

This paper presents MERTIS and describes its requirements via the use of the Unified Modeling Language (UML) through use case specification. UML is an open, extensible industry standard visual modeling language that establishes the notation for specification, design, and documentation of component based software systems [1].

We have organized the remainder of the paper according to the following sections: Section 2 presents general description of the software, Section 3 lists the functional and non-functional requirements, Section 4 includes the Use Case model, Section 5 displays the class diagram and finally, Section 6 covers the conclusion and future expansions of MERTIS.

2. General Description

MERTIS is a software application that runs 2D simulations defined by simple script files rather than full programs. MERTIS was designed so that users could create games without having to deal with the burden of most of the lower-level aspects of programming. The scripts that users will create are simple enough so that simulations can be created with little effort and basic programming knowledge, yet they will still have the power to define any aspect of the simulation.

The scripts allow the user to define objects, their behavior, as well as their interaction with other objects and the environment. The dynamic behavior of objects may be user defined with Lua code. According to [3], Lua is a lightweight programming language designed for extending programs.

Aside from Lua, other third party technologies are employed in MERTIS including OpenGL [4] and QT [5]. QT is used primarily for windowing and parsing of the script files while OpenGL is used for rendering the graphics. The key feature offered by these two technologies is that they are platform independent. This feature will make MERTIS available to a wider range of users, thus enabling the creation of simulations and games that are also platform independent.

The focus of MERTIS is gaming, so several higher-level built in functions and attributes for MERTIS developers will be included. This is to even further simplify game development by providing a simple MERTIS Application Program Interface (API). With built in functions and attributes, the user can invoke common events, access environment variables, and object properties. Users will also be able to map keys to objects and define functions for specific keys.

3. Requirements Specification

MERTIS will offer a relatively easy scripting language and a user friendly interface for loading those scripts. The *Functional Requirements* list the behavior of the system; while the *Non-functional Requirements* lists a number of specific properties the software should have [1].

3.1 Functional Requirements

MERTIS will be completed over several stages. These requirements provide a good overall picture of what MERTIS will look like and the functionality it will provide. For traceability during the software development process each functional requirement has a number and is denoted using the format <R#>.

R1 MERTIS shall read in a script file in order to load a simulation.

R2 MERTIS shall be able to create sprites.

R3 MERTIS shall be able to create and modify a 2D environment that will contain various user defined objects.

R4 MERTIS shall load images for skins of objects and images for textures.

R5 MERTIS shall run event-driven simulations.

R6 MERTIS shall allow the user to bind keyboard and mouse interaction with simulation events.
R7 Every object shall have a default (idle) event.
R8 Objects shall interact with each other (clip).
R9 Objects shall have alpha blending.
R10 Objects shall have a clipping mask.
R11 MERTIS shall validate XML files.
R12 MERTIS scripts shall be extensible.
R13 MERTIS shall have a manual.
R14 MERTIS shall display the contents of the script file in a hierarchical tree structure.
R15 MERTIS shall provide a means of pausing and restarting a simulation.
R16 MERTIS shall have audio playback for music and sound effects.
R17 MERTIS can be restarted at any time
R18 MERTIS shall cleanly exit at any time

3.2 Non-functional Requirements

The non-functional requirements for MERTIS describe many of the technologies that will be used in its development as well as some system constraints. Following is the list of these requirements, denoted using the format <T#>:

T1 MERTIS shall operate at 30 frames per second (fps) by default.
T2 MERTIS shall support 32-bit color palette (true color).
T3 MERTIS shall use OpenGL to render graphics.
T4 MERTIS shall use XML for saving and loading simulation data.
T5 MERTIS shall run in the Windows 2000 and Windows XP operating systems.
T6 MERTIS shall use a 2D graphics engine.
T7 MERTIS shall be able load BMP, TIFF, and PPM file formats for graphics.
T8 MERTIS shall use QT for Windowing.
T9 MERTIS shall use QT for XML parsing.
T10 MERTIS shall perform in real-time.
T11 MERTIS needs an input device.
T12 MERTIS shall be implemented in C++.
T13 MERTIS shall use Lua for event handling.
T14 MERTIS should be able to load JPEG, and GIF file formats for graphics.
T15 MERTIS shall load MIDI, WAV file formats for sound effects and music.
T16 MERTIS should load MP3 file formats for sound effects and music.

4. UML Specification

Based on the guidelines presented in [1], system boundaries, actors, and use cases have been elicited to further supplement the system requirements. Furthermore, use case scenarios have been defined for the more complex use cases in order to demonstrate a specific path through the use case.

4.1 Use Cases and Scenarios

Use cases are a way of expanding the functional requirements. They describe a relationship between actors and the system. In our project, we have two actors: the user and time. The Use Case Diagram depicting these interactions for MERTIS is shown in Figure 1.

Due to space limitations we present only two examples of use cases and two sample scenarios. The detailed use cases describe how either the user's interaction with the system or time's involvement with the system begins a certain flow of events. The scenarios are one path through the use case, and in these scenarios, are the primary flow of events in which no exceptions or errors occur.

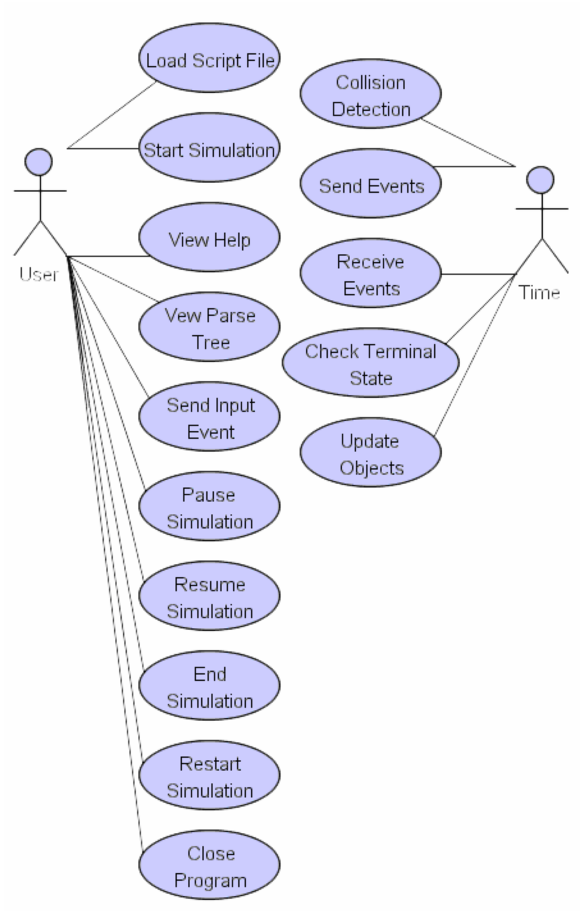

Figure 1: MERTIS Use Case Diagram

In Figure 2 we see what happens when the user chooses to load a script file. Figure 3 shows the flow of events of the system checking for collisions. Figure 4 presents the actions performed when updating the objects, and in Figure 5 we can observe what happens when the system determines there is a collision. These are just a sample of the many use cases we developed for MERTIS, and hope they have provided an overview for the remaining use cases.

4.2 Class Diagram

In Figure 6 we find an analysis-level class diagram showing a high-level representation of the system architecture. It shows the abstraction of the various parts of the system along with most of their essential attributes, operations, relationships, and multiplicity.

Use case: Load Script File
ID: UC1
Actors: User
Brief Description: Allows user to select an XML file to open. Then opens the XML document for parsing and performs validation and verification processes.
Preconditions: 1. No simulation is currently running.
Flow of Events: 1. User chooses to open script from menu item or by directly pressing the open script button on the toolbar. 2. Open dialog box opens 3. User browses directories until desired file is found. 4. User chooses to open the file. 5. File is opened and sent to QT's XML parser. 6. Parser validates script file and either 6.1 Display error 6.2 Generate tree
Postconditions: 1. Based on the validity of the script file 1.1 An error dialog has been presented to the user 1.2 The structure of the script file has been loaded into a hierarchical tree

Figure 2: Use Case: Load Script File

5 Future Extensions

There are numerous possible extensions that could further enhance the usability and capabilities of MERTIS. Specifically, some of these enhancements include:

- The support of various types of sound clips (MP3, WAV, etc.) for different states of objects or different events. Events may include collision detection or user inputs.

Use case: Collision Detection
ID: UC11
Actors: Time
Brief Description: It performs an AND operation on collision masks to check for collisions.
Preconditions: 1. All objects have been updated
Flow of Events: 1. While (Objects haven't been checked) 1.1. Choose an object 1.2. Using Quadtree, select near by objects 1.3. For each nearby object 1.3.1. Get first object's collision mask and near-by object's collision mask 1.3.2. If masks overlap 1.3.2.1. Find sub region 1.3.2.2. Perform AND operation on each pixel in subregion 1.3.2.3. If a one is returned, then send "Collision event" to both objects 1.3.3. Mark objects having been checked
Postconditions: All objects that have collided have received a "Collision" event

Figure 3: Use Case: Collision Detection

- An optimized resource handler [2] for better management of objects in memory. As the number of objects and the complexity of the simulations increase so does the need for this optimization.
- An integrated development environment for creating and editing script files. This feature will provide the users with the ability to edit the script files while simultaneously viewing the dynamically updated object hierarchy tree.
- Visual simulation editor for drag and drop of objects into the environment. This would provide a means of creating the script file without manually typing it in.

Scenario for use case: Update Objects
ID: UC15
Actors: Time
Brief Description: Updates the internal states and attributes of each object.
Preconditions: 1. Objects have events that need to be processed
Primary Scenario: 1. Process all waiting events from previous turn 2. Process AI subroutines 3. Update objects variables 3.1. Change Location 3.2. Change Velocity and Acceleration 3.3. Change Image 3.4. Change other variables 4. Check object condition and rules 5. Send necessary events
Secondary Scenarios: Destroy Object Create Object
Postconditions: All objects reflect changes

Figure 4: Primary Scenario for Use Case: Update Objects

- Support for object inheritance. Providing the user with a library of objects that can be reused and extended, in effect, creates an API for the user.
- A library of Lua code for convenient reuse of common utilities and functions.
- Cross-platform compatibility so more users can create and run MERTIS simulations.

6 Conclusions

In this paper, we have presented the specification of MERTIS, a system intended to reduce the complexity of software involvement in writing games and simulations. It achieves this by removing

Scenario for use case: Collision Detection
ID: UC11
Actors: Time
Brief Description: It performs an AND operation on collision masks to check for collisions.
Preconditions: 1. All objects have been updated
Primary Scenario: 1. An object is chosen. 2. Nearby objects are chosen using the Quadtree. 3. The first nearby object's collision mask overlaps with the current object's collision mask. 4. An event is sent to the event handler, signifying a collision.
Postconditions: All objects that have collided have received a "Collision" event

Figure 5: Primary Scenario for Use Case: Collision Detection

the responsibilities of texture mapping, collision detection [6, 7] , physics, and other such complicated aspects of game development from the user. In order to ensure a well designed system we have adhered to the notations and techniques defined by the UML standard.

The analysis model of this system has been organized according to the Unified Process to ensure a rigorous development of the software. After the basic functionality of MERTIS is implemented, further improving the system according the extensions previously discussed will provide an invaluable tool that programmers of all experience will appreciate and find very useful.

Acknowledgement

We would like to sincerely thank Dr. Sergiu Dascalu for his exceptional support and encouragement. We would also like to acknowledge the University of Nevada, Reno and its Computer Science and Engineering Department for providing us with a fantastic place to learn and work.

References

[1] Arlow, J. and I. Neustadt, *UML and the Unified Process: Practical Object-Oriented Analysis & Design*, Addison-Wesley, 2002.

[2] Llopis, Noel. *C++ for Game Programmers*. Hingham, Massachusetts: Charles River Media, Inc., 2003.

[3] Matheson, Ash. *GameDev.net - An Introduction to Lua*. 30 Apr. 2003. Gamedev.net. 22 Feb. 2004 <http://www.gamedev.net/reference/articles/article1932.asp>.

[4] OpenGL Documentation. OpenGL.org. 14 Sept. 2003 <http://www.opengl.org/documentation/index.html>.

[5] QT Reference Documentation. Trolltech Inc. 15 Feb. 2004 <http://doc.trolltech.com/3.3/index.html>.

[6] Roberts, Dave. SP95: *Collision Detection*. 1995. Dr. Dobb's Journal. 22 Feb. 2004 <http://www.ddj.com/documents/s=983/ddj9513a/>.

[7] TANSTAAFL, *GameDev.net - Collision Detection Algorithm*. 17 Sept. 1999. Gamedev.net. 22 Feb. 2004 <http://www.gamedev.net/reference/articles/article754.asp>.

[8] XML Programming. Dev Shed. 21 Jan. 2004 <http://www.devshed.com/c/b/XML/>.

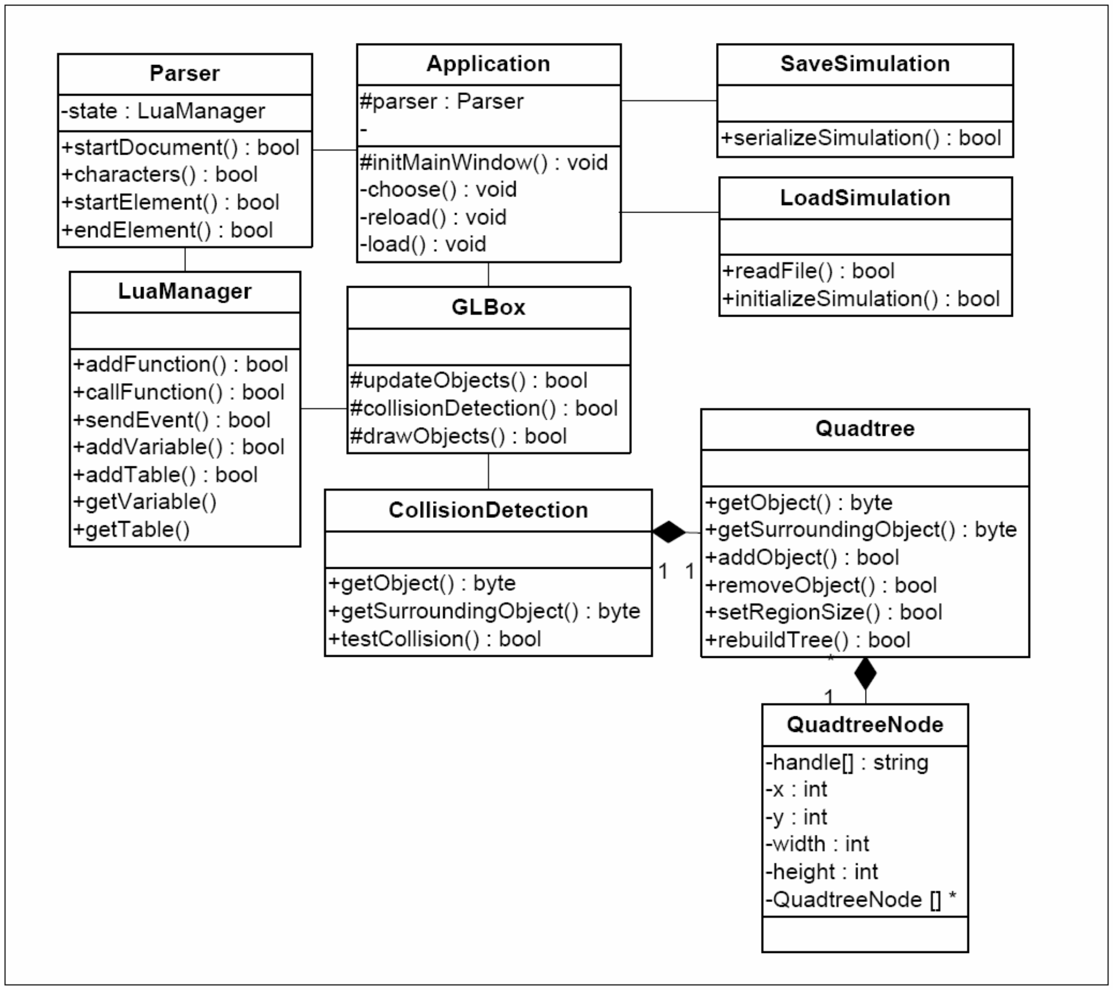

Figure 6: MERTIS Analysis Level Class Diagram