HW2 – Protocol Design and Message Implementation

Due Date: *9/21/2012*
Estimated time: *12-18 hours*

Objectives

- Practice principles related to protocol design
- Explore architectural design choices that will lead to good modularization, encapsulation, abstraction, coupling, and cohesion
- Become familiar with unit testing techniques

Overview

This assignment consists of four parts: game refinement, protocol design, message implement, and unit testing of the message implementation. For the first part, you will make the virtual water-game your own by refining and extending the analysis and requirements supplied by the instructor. For the second part, you will design the communications for your game. The protocols need to be well thought out, efficient, and accurately documented. For the third part, you will implement a class library that represents the messages of your protocols. Each message class must include a constructor for a sender to create a new instance of that type of message and a constructor for a receiver to create an instance of the message from a byte array was received on a socket. For the fourth part, you will create executable test cases that thoroughly test at least three of your message classes.

Instructions

Part 1 – Game Refinement

Using the provided use case diagram, class diagram, and requirement as start point, refine or extend the virtual water-fight game in ways that you think would be most interesting to potential users. For example, you choose to allow the player to have other water-related weapons, such as water guns or balloon launches. Feel free to make up new game rules and constraints, as long as you don't eliminate the following basic characteristics:

- The game has to remain a real-time multi-player game.
- The game must involve at least three different types of shared resources.
- Instances of at least one of the resource types need to be distributed.
- The game must allow new users to join at any time
- The game must anticipate substantial growth in its used (scalability)
- The game must anticipate additional features being added later on

Update the analysis diagrams and the requirements to capture and describe your version of the game.

Part 2 – Protocol Design

For this part, you need to design and document communication protocols that will support all of the features and rules described in your functional requirements. Each protocol defines how two or more processes interact to perform some action or in the accessing of shared resource.

Your design should minimize communication overhead, while ensuring that all requirements are beginning met. They should also minimize security threats and the effects of an occasional lost message. Your protocol documentation should describe the message syntax (message format and sequencing), timing, and semantics. May use several tools to create you protocol document, such as Visual Paradigm, Visio, Word, OpenOffice, etc. Regardless of the tools you used, please package up your protocol design document as a single Word, OpenOffice, or PDF file with the following sections:

1. Protocols

   List and explain all of the protocols that need to have in your system. To discover this, consider the games and features and requirements, and model typical conversations that need to occur between components of the system. UML communication diagrams are a good tool for analyzing and refining these typical conversations.

   In this section, provide overview of each project by a) giving it a meaningful name, b) describing it proposed, which components it involves, and why it will be used, when it will be used. High-level sequence diagrams and/or tables can document this information effectives. See sample the document.

2. Message Types and Their Definitions

   First, give an overview of your message classes. A UML Class Diagram would be a good way to describe a hierarchy of message classes and their syntax, with appropriate class attributes and generalization/specialization relations. Then, for each concrete message type (e.g., a leaf class in your class hierarchy), describe its meaning and intend use. In the description describe the message's meaning, uses (in the context of one or more protocol), and valid values for each attribute of the message. A table would be a good ways to capture these details.

3. Message Sequences and Timing

   Finally, describe message sequences with timing constraints and re-try expectations. Detailed UML sequence diagrams (with notes for time and re-try constraints) are relatively effective for describing this time of information.

Part 2 – Message Implementation

For this part, you need to implement a class library consisting of C# or Java classes that represent the types of messages used by your communication protocols. For example, the class for each message type needs have:

- A constructor that the send will use to create new message that it intends to send,
- A constructor that receiver can use re-instantiate a received message,
- A method package up the method into a byte array, and
- A method to un-package a byte array into the attributes of a message.

Use inheritance and composition to create effective classes with good abstractions, enforced encapsulation, loose coupling, and high cohesion.

Part 3 – Testing

For this part, you need to create executable Unit Tests to thoroughly test at least three of your message classes. You may using any testing framework that you would like and is well suited to your development environment and this application.

Submission Instructions

Zip up your analysis, requirements definition, your protocol design, and your entire software solution into an archive file called CS5200_hw2_<*fullname*>.zip, where *fullname* is your first and last names. Then, submit the zip file to the Canvas system.

Grading Criteria

Basic Criteria (worth up to 70 points)	**Max Points**
Appropriate and well-documented refinement/extension to the game	10
Thorough and efficient protocol designs	20
Implementation of message classes with good abstractions, encapsulation, low coupling, and high cohesion	25
Thorough unit testing of three classes	15
Advanced Requirements (Worth up to 30 points)	**Max Points**
Communication protocols that will enable the system to deal with duplicate messages or out-of-order messages	20
Thorough unit testing for six more message classes	30