# HW2 – Protocol Design and Message Implementation

*Due Date:* *1/31/2014*
*Estimated time:* *12-18 hours*

## Objectives

- Practice principles related to protocol design
- Become familiar with the mechanism of marshalling and unmarshalling objects
- 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 three parts: protocol design, message implement, and unit testing of your message implementation. For the first part, you will design the communication protocols for game. The protocols need to be well thought out, efficient, and accurately documented. For the second part, you will implement a class library that represents the messages used in 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. For the third part, you will create executable test cases that thoroughly test at least three of your message classes. For this homework, you will need to not implement any processes that send or receive the messages.

## Instructions

### Part 1 – Protocol Design

For this part, you need to design and document the communication protocols that will support all the features and rules described in game's functional requirements. Note, we will initially start with a conceptual description of the game and then work together as a class to define the functional requirements. You should have the complete functional requirements several by the end of the first week. You should be able to start you design activities from the conceptual description, proceed with partially requirements, and the complete your design activity as the functional requirements are completed.

In defining the communication protocols, you first take will to identify all of the different type of conversations that will need to take place in the system. Second, for each type of conversation, you will define a communication protocol. Your protocol designs should minimize communication overhead, and still meet all functional requirements.

Your document for each protocol definition must describe the protocol's

1. Syntax, i.e., possible message sequences, message format, validation
2. Semantics, i.e., the meaning of the messages and the data they contain
3. Process behavior for each process involved in the conversation, i.e., time outs, retries, handling of invalid messages, etc.

You can decide how to most efficient communicate this pieces of information. For example, if your communication protocols share the same kinds of messages, you may want to put all your message descriptions in a common place and just reference them the individual protocol definitions. If you protocols use common sequence patterns, like *Request-Reply*, then define that pattern is one place and just refer to it wherever needed. Be efficient in document. Remember the documentation is to "serve" you and your team, not the other way around.

You may use various tools to create you protocol documentation, such as Visual Paradigm, Visio, Word, OpenOffice, etc. Regardless of the tools you used, please package up your protocol design document into a single presentation, rendered as a PDF file.

## Part 2 – Message Implementation

For this part, you need to implement a **class library** that contains abstractions for the messages used by your communication protocols. Specifically, it should include a for each message type that has a

- A constructor that a sender can use to create new message of that type,
- A constructor that receiver can use re-instantiate a message from a byte array,
- A method for packaging up the method into a byte array, and
- A method for un-package a byte array into the attributes of the message.

Hints:

- User good object-oriented software design principles and implementation techniques: localization of design decisions, low coupling, high cohesion, aggregation, encapsulation, abstraction, inheritance, polymorphism, etc.
- Use inheritance and composition to create effective classes with good abstractions, enforced encapsulation, loose coupling, and high cohesion.
- You could also use one several different "factory" patterns to help manage the complexity of construction of messages in a message class hierarchy.

So you gain some experience in marshalling and unmarshalling message objects, please do not use any existing serialization, encoding, or marshalling libraries or tools (e.g., XML serialization, JSON, etc.)

## Part 3 – Testing

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

## 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 75 points)** | **Max Points** |
|---|---|
| Thorough and efficient protocol designs | 25 |
| Implementation of message classes with reasonable abstractions, encapsulation, low coupling, and high cohesion | 25 |
| Thorough unit testing of three classes | 25 |
| | |
| **Advanced Requirements (Worth up to 25 points)** *Note: You cannot earn points in this area if you have not addressed the basic criteria.* | **Max Points** |
| Thorough unit testing of at least three more messages classes | 15 |
| Extensibility, flexibility, and maintainability of the design & implementation | 10 |