HW3 – Reliable Inter-process Communication

Due Date: *10/12/2012*
Estimated time: *20-28 hours*

Objectives

- Gain experience with creating reliable communication from underlying unreliable communications
- Master one typical architectural for reliable client/server style communications
- Become more familiar with unit testing techniques
- Become familiar with logging techniques and a logging tool

Overview

During this assignment, you will begin to implement your distributed water fight game, including components that support reliable communications between resource managers and users of those resources. To minimize your overall development costs, you will implement a reusable, extensible communications subsystem with the following classes of objects. Also, see Figures 1 and 2.

Communicator

This class is an abstraction for communicating with other processes via UDP-based messages. It should handle the basic send and receive operations for Message objects. Each process in the system should have one primary *Communicator*, so the initial messages of a conversation are always to/from a known port. Depending on your protocol definitions from HW2, subsequent messages in a conversation may occur through this primary communicator or on another communication channel specific to the conversation.

Listener

A *Listener* object is responsible for grabbing messages received by a *Communicator* object and placing them into a message queue for later processing by a *Doer* object. Note that the *Listener* object does not interpret or process the message.

Doer

A *Doer* object should oversee the processing of messages in the message queue in a way that is consistent with protocols defined in HW2. It should be responsible for taking a message from a message queue, but it can then delegate the processing of that message to one or more application-layer objects. The application-layer objects encapsulate and implementation the application's communication protocols. As they process they messages, they may send messages to others process via the *Communicator*.

BackgroundThread

This is a base class for *Listener* and *Doer*. It encapsulates all the attributes and behaviors that are common to both classes of objects.

Message Queue

A *MessageQueue* object is a queue of messages that have been received by the listener, but not yet processed by a doer. Since Listener and Doers will be running as separate threads, this object must guarantee correct queue behavior in the present of concurrent access.

Message Classes

These are the message classes that you built in HW2. They represent the data structures used in your communication protocols.

Using this communications subsystem and based or your specific requirements developed as part of HW2, you will begin to design and implement the necessary resource managers and players for your water-fight game.

Instructions

Naturally, the design of your system will be unique to your requirements and your thinking. However, regardless of your approach, your development process needs to include the following activities:

- Design, implement, and test communications components
- Design, implement, and test the resource managers
- Design, implement, and test the resource users

For HW3, you need to thoroughly test at least your Communicator, Listener, and MessageQueue classes. (See the Basic Grading Criteria.) However, it is recommended that you to test other major components, as well. (See the Advanced Grading Criteria.)

Submission Instructions

Zip up your document and your entire solution into an archive file called CS5200_hw3_<*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 85 points)	Max Points
A quality implementation of the classes that make up the communication subsystem.	35
Thorough unit testing of the Communicator, Listener, MessageQueue classes	15
A quality design and initial implementation of your resource managers and resource users. Note, it is not necessary to have all of your protocol s completely functional, but you should have at least 30% of them working and demonstrable.	35
Advanced Requirements (Worth up to 15 points)	**Max Points**
Thorough unit testing of at least three more classes, e.g. Doer, protocol handlers, etc.	15
An initial implementation for a meaningful user interface that allows a human user to control a player in the game	15
A game monitor that can be launched or shutdown independently of any other process that can show the approximate state of any player or resource manager.	30

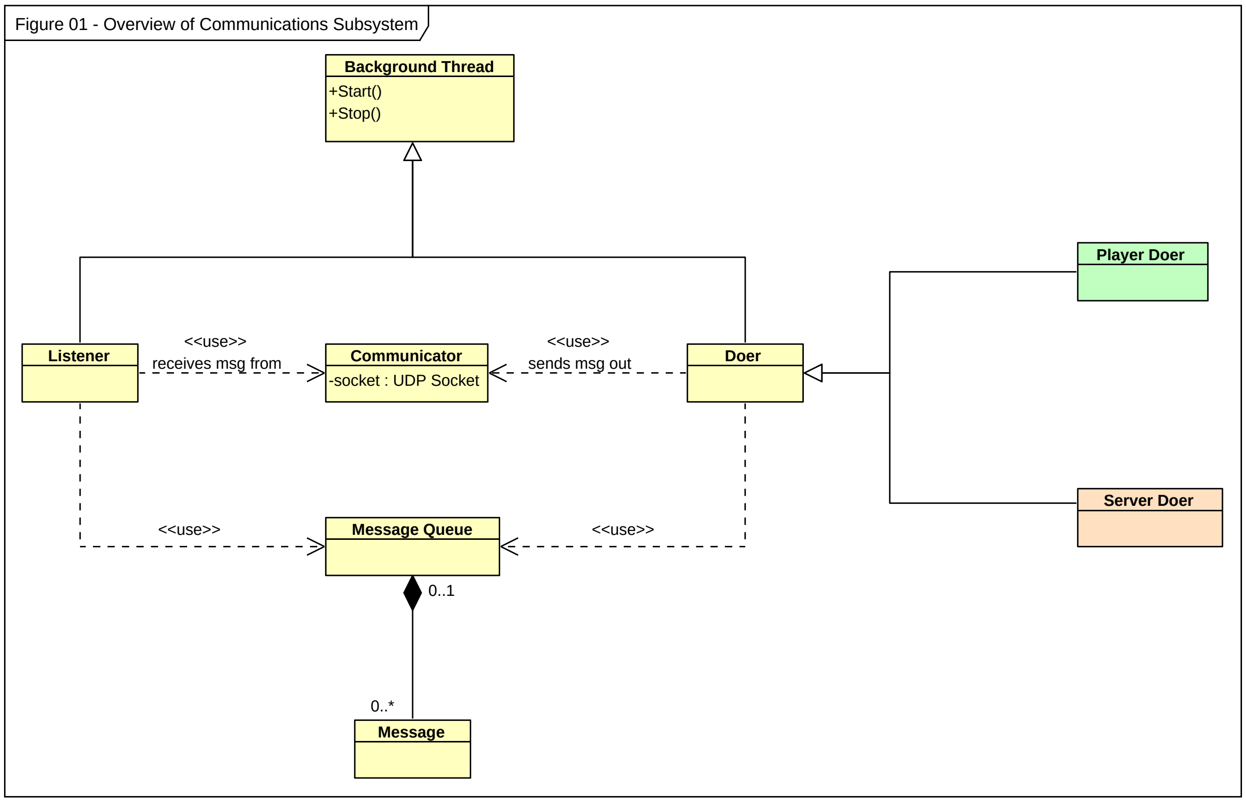
Figure 01 - Overview of Communications Subsystem

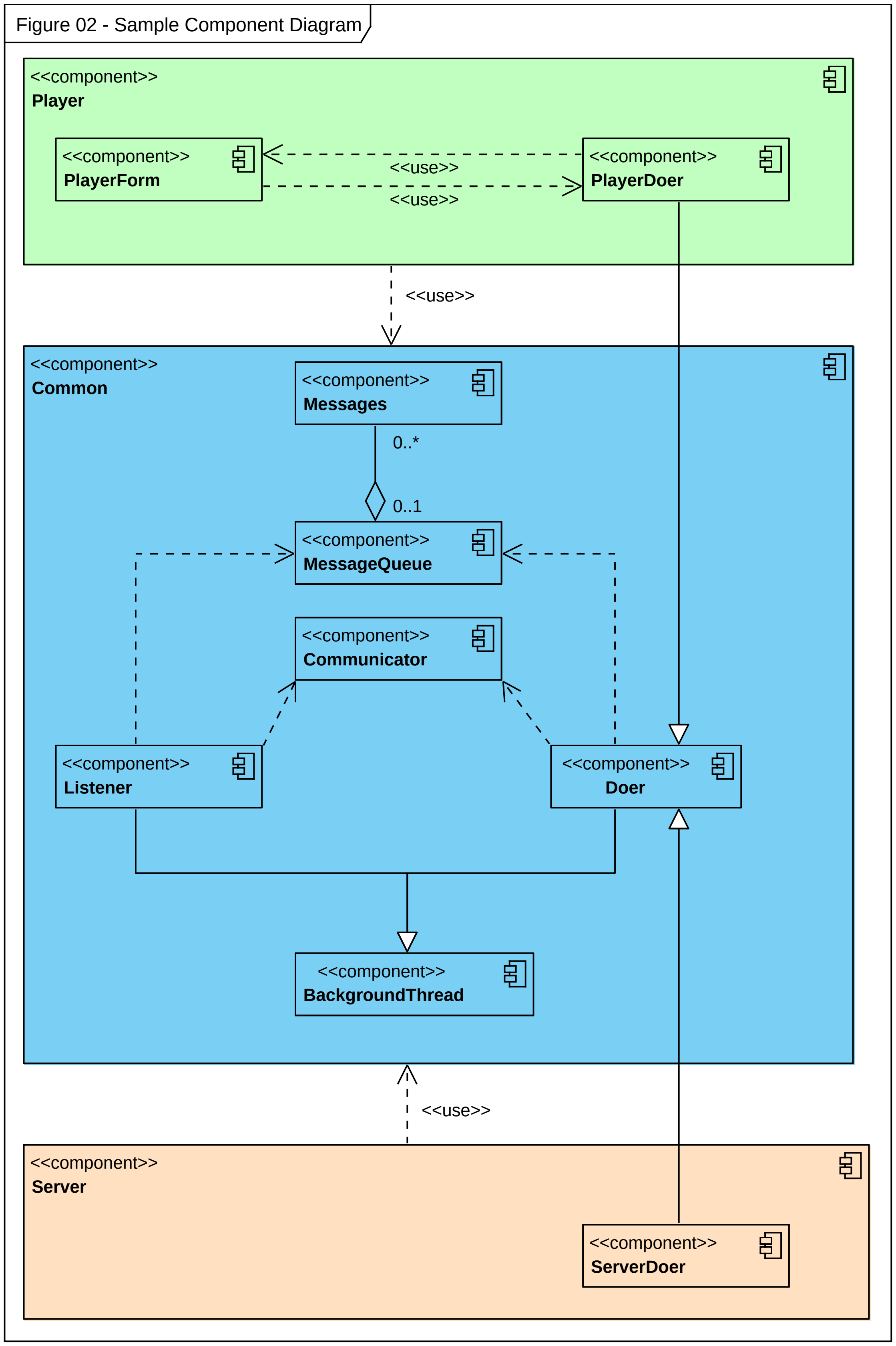
Figure 02 - Sample Component Diagram
<<component>>
Player
<<component>>
PlayerForm
<<use>>
<<use>>
<<component>>
PlayerDoer
<<use>>
<<component>>
Common
<<component>>
Messages
0..*
0..1
<<component>>
MessageQueue
<<component>>
Communicator
<<component>>
Listener
<<component>>
Doer
<<component>>
BackgroundThread
<<use>>
<<component>>
Server
<<component>>
ServerDoer