HW4 – Simple Webservices and Resource Management

Estimated time: 20-24 hours

Objectives

- Gain some experience with simple remote objects implemented as Webservices
- Become more familiar with inter-process communications and resource management concepts
- Become more familiar with unit testing techniques
- Become more familiar with logging techniques and a logging tool

Overview

In this assignment, you will continue to implement and refinement the inter-process communication protocols and resource management functionality for the *BSvsZP* game. Specifically, you will implement webservice communication for discovering the available games and their endpoint and you will need to complete about 60% of the functionality outlined in the requirements definition. Since the underlying middleware was implemented and test in HW3, most of the functionality that you will implement for this assignment will deal with resource management at the application layer.

Instructions

Step 1 Review the updated functional requirements. Specifically, consider the requirements for agents to be discover available games from a *Game Registry* via a webservice.

Step 2 If you haven't already done so, create a executable-program project for each type of agent that you need to build, i.e. Brilliant Student, Excuse Generator, and Whining Spinner.

Step 3 As part of each agent's startup behavior, have it get list of available games for the GameRegistry using a Webservice method invocation. Either have the agent automatically join an available game or let the user pick which available game to join.

Step 4 For each of these agents, do at least the following:

- Implement application-layer logic for holding and managing their state and the resources they are responsible for. For example, all agents will need to keep track of the game's end point, their local process id, and their status (strength, location, etc.) The *ExcuseGenerator* will need to manage a pool of *Excuses* that it has built but not yet been given out to any *BrilliantStudent*. Similarly, a *WhiningSpinner* will need to manage a pool of *WhiningTwine* that it has not yet been given out to any *BrilliantStudent*. A *BrilliantStudent* will need to manage a pool of *Bomb*

that it has not yet thrown. A BrilliantStudent will also want to keep track of information about other agents in the game. This is by no means an exhaustive list of state or resource information that each agent will need to keep track of. Your own specific design will dedicate much of what else is need.

- Implement as thread that is response for any "active" application-layer behavior. For example, independent of any messages coming into a Brilliant Student, it will need to make decisions about moving and throwing bombs. These active behaviors may initiate conversations with other components by sending request messages to the communicator.
- Continue implementation of execution strategies, which are in affect "reactive" behaviors for a type of agent, in response to request messages. As noted before, the execution of strategy is initiated by the Doer and should be on its own thread. However, the concrete implementations of the execution strategies are specific to each type of agent and should therefore be packaged with the agent projects, not the middleware.
- Note that you are shooting about 60% completion of the functionality in the requirements definition.

Step 5 Since there will be multiple threads in an agent process access its state and the resources it is managing, implement concurrency controls that ensure the integrity for those shared objects.

Step 6 Complete reasonably thorough unit testing code implemented in Steps 3 and 5.

Step 7 Write a README.TXT file that describes what you completed during this assignment. Specifically, describe what to did, if anything, for the advanced requirements.

Submission Instructions

Put your documentation and entire solution into an archive file called CS5200_hw4_<*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**
Properly setup of executable-program projects for BrilliantStudents, ExecuseGenerators, and WhiningGenerator.	10
A quality implementation of approximately 60% of active and reactive required for each agent.	30
Testing of the active and reactive behaviors	35
Advanced Requirements (Worth up to 25 points)	**Max Points**
A non-trivial graphical user-interface that display game and component status information	25
A referee agent	25