Software Distribution and Mobility Seminar 2009 Project

e-mail: egonzale@vub.ac.be
office: 10F731

Deadline 22 June 2009.

Cooperation This project assignment is to be made individually. Cooperation **is not** allowed in any form.

Delivery Both documentation and code should be delivered in digital format. A mail should be sent to egonzale@vub.ac.be including "Project SDMS" in the subject line on the day of the deadline **before 16:00**. You should **also** indicate the AmbientTalk build used.

1 Assignment: The AmbiScrabble Game

The purpose of this exercise is to create a digital decentralized distributed version of a word game similar to Scrabble. This game works as follows: a number of players work collaboratively to create words out of "virtually lettered tiles". Players are organized in teams and each player has a rack of letters. The goal of the game is to consume all the letters of the team by forming words. In other words, the team that first consumes all its letters wins.

When two players of the same team are in communication range, they can see each other's letters. A player can then request particular letters of another team member to form a word. Once a player forms a word, half of the letters forming the word get consumed for his/her team. The player gets the opportunity to throw the other half of the letters to nearby opposite team members. Those thrown letters get also consumed for the team when the opposite team members acknowledge the letter(s) reception. If a player does not form a word for a while, a new letter gets added to his/her rack.

When a player starts the AmbiScrabble application, he/she chooses a team. The application automatically generates a rack of (random) letters for that player. When two or more players of the same team are in range, they can start forming words. Once a game has started other peers can join the game, incrementing the number of letters of that the team has to consume to win.

2 Non-Functional Requirements

There are some requirements w.r.t. the distributed design of the game.

1. It must be implemented in AmbientTalk. As in the lab sessions, you can employ the symbiosis between Java and AmbientTalk to make use of Java classes for e.g. your data structures (e.g. Vector, Hashmap, etc..). However, all distributed communication must be implemented in AmbientTalk. In other words, you cannot use Java RMI as your distributed computing framework. Make sure that your Java code (if any) is at least compatible with version 1.5.

2. The game should be designed in a peer-to-peer fashion. You **cannot** assume a centralized server in your design to coordinate the game, e.g to discover new players or to keep track of game state (e.g. the consumed letters of each team).

3. The game should be fault-tolerant such that failing computational units do not hamper the game from being played. You must assume that *every* computational unit in the network can fail at *any* point in time.

   Hints:

   - Player disconnections should not hamper the game progress. For example, if a player disconnects, the other team members can continue forming words.
   - Message sends to remote peers can fail. Note that failures are unreliably detected using timeout as a heuristic.

4. Players may enter and leave the network at any point in time. However, you may assume that player disconnections are temporal. If a player disconnects, other players may have outdated game information, for example, about the state of his/her rack of letters. The game information should get properly updated once players come online again.

5. Words are validated by means of a dictionary. You may use the provided `dictionary.at` module.

6. You may assume that there is only one game that is being played at a time.

Extra requirements (not obligatory):

- Explore team constraints to make the game more fair, challenging, competitive, etc.. For example, the application could adapt time period rates for the appearance of letters so that small teams get less letters, or the time to form words so that bigger teams get less time.
- Validate the words by achieving consensus amongst nearby players. For example, the player needing to validate a word may start a poll with all nearby players.
- Weaken the game assumptions. For example, adapt the application to take into account permanent disconnections of players.

3 Testing and Report

Besides implementing the application, you also have to create interesting scenarios to test your implementation. These test scenarios show how your application behaves w.r.t. the different operation modi. Note that you can implement them using a GUI in Java, a testcase in the AmbientTalk unit testing framework, or whatever other means you may find convenient to test your application. Keep in mind that the distributed design should be the focus of your project! You don't have to have a fancy GUI to test your application, i.e. GUI is of **no** importance.

You will write a small report about this project. This report must be **no longer than 10 pages** and serves as a guide for evaluating the implementation of your project. The report should include following items:

1. What were the important cases and choices (w.r.t. distributed aspects) to consider for this project?

2. An overview of your implementation and how the implementation fulfills the requirements.

3. A small "manual" about how to run and test your application and explanation of the test scenarios.

4 Evaluation

The project is taken into account for half of your final score for the SDM seminar. It will be evaluated mostly on good distributed object-oriented design. What is **important**:

- You should aim to fully exploit a peer-to-peer organization and a fault-tolerant design.
- Quality and structure of the code **is** important.
- In your testing, you should focus on trying the application from the distributed point of view.

Being creative and adding additional requirements of your own or any of the extra requirements to the project is appreciated.

Good luck!