Distributed and Mobile Programming Paradigms
Academic Year 2012-2013
Exam Requirements for the WPO Part of the Course
weScrabble

Deadline 10 June 2013.

Delivery Both the documentation and the code should be delivered via the drop box of PointCarre in the form of a <name>-project.zip file on the day of the deadline **before 16:00**.

Material The project material consists of the assignment and a supporting AmbientTalk file. They are available at PointCarre, and `http://soft.vub.ac.be/amop/teaching/dmpp`.

1 Assignment

The purpose of this project 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 weScrabble 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 that the team has to consume to win.

2 Non-Functional Requirements

The weScrabble game is a multi-player mobile application. Each participant of the game has a (mobile) device running the application. There are some requirements w.r.t. the distributed design of the game.

Peer-to-peer design. The game should be designed in a peer-to-peer fashion. You can assume that there is a WiFi network available where players meet and play together (e.g. a cafeteria area). However, you **cannot** assume a centralized server in your design to coordinate the

game, e.g to discover new players or to keep track of the game's state (e.g. the consumed letters of each team).

Fault-tolerant design. You must assume that *every* computational unit in the network can fail at *any* point in time. In particular:

- 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.
- 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 may be unreliably detected using timeout as a heuristic.

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

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

Extra requirements (not obligatory):

- 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.
- 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.
- Weaken the game assumptions. For example, adapt the application to take into account permanent disconnections of players.

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

2.1 Programming Platform

The game should be implemented in AmbientTalk and will be deployed on Android devices. You can use any of the language abstractions seen in class or during the lab sessions for designing the application. You can also exploit the symbiosis between Java and AmbientTalk to make use of Java classes for e.g. your data structures such as a Hashmap as done in class.

Nevertheless, all **distributed** communication **must** be implemented in AmbientTalk. As such, you **cannot** use other technologies than AmbientTalk (e.g. Java RMI) as your distributed computing framework. The front-end of your application will be implemented in the Android platform.

Those students who do not own an Android device, should mail no later than *21 April 2013* so that we can distributed devices if necessary.

3 Testing and Report

You should define a number of interesting scenarios to test your implementation. These test scenarios show how your application behaves w.r.t. the different distributed operation modi. In order to test your applications in the face of network disconnections, you can use the various AmbientTalk constructs used during the lab sessions.

You will write a small report about the project. This report must be **no longer than 8 pages** (excluding figures) and serves as a guide for evaluating your project. The report should include:

1. Overview of your application and how the implementation fulfils the requirements.
2. Description of the important cases and design choices (w.r.t. distributed aspects) to consider for this project.
3. Description of test scenarios and behaviour of your application.
4. A small "manual" about how to run and test your application.
5. Which Android version you used.

4 Evaluation

The project will be evaluated mostly on good distributed design, but the application should be effectively deployed on the Android platform. In particular, the distributed design counts up to 70% and the stability and usability of the application on Android devices 30%.

What is **important**:

- You should aim to fully exploit a peer-to-peer architecture and a fault-tolerant design.
- The testing should be focused on trying the application from a distributed point of view.
- Quality and structure of the code is **very** important.
- All distributed code should be written in AmbientTalk. The AmbientTalk files are the basis for evaluating of your distributed design, not the Java ones.
- This assignment is to be made **individually**. Cooperation is not allowed in any form.
- Problems with the software tools (e.g. Eclipse, Android setup, etc.) and devices should be reported **no later than** two weeks before the deadline, i.e. **27 May 2013**.

Please do not hesitate to contact me for further questions.
e-mail: egonzale@vub.ac.be, office: 10F736

Good luck!