Software Distribution and Mobility Seminar 2011
weScribble Pro: Collaborative Drawing

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

Deadline 20 June 2011.

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**.

1 Assignment

The purpose of this exercise is to implement a collaborative drawing editor which extends the functionalities of the weScribble exercise done in the lab session 3. weScribble Pro allows multiple users to collaborate with each other for sharing and editing shapes on a common canvas. Users can draw several shapes (e.g. rectangle, square, line), move them in the canvas, and change their color. On top of that, users can select one or more shapes and group them in a *grouped shape*.

Users can join a drawing sessions by contacting any user which is already part of a drawing session. When a user joins a drawing session, he can add, remove or move shapes on the canvas of that session which is *virtually* shared between the different participants. This means that any change on the canvas of a user is propagated to the rest of the participants of the session which updates their canvas accordindly. We will call a participant in a drawing session a *drawer*.

2 Non-Functional Requirements

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

1. weScribble Pro should be designed in a peer-to-peer fashion:
   - You **cannot** assume a centralized server in your design to coordinate the drawing sessions or to discover new participants.
   - You cannot assume that there is *one* owner of a (grouped) shape during the drawing session. However, you may assume that a drawer takes *ownership* on a (grouped) shape when he starts changing or creating it. Once all the drawers in the session have been updated with his changes, the drawer releases the ownership.
2. You must assume that *every* computational unit in the network can fail at *any* point in time:
   - A drawer may enter and leave the network at any point in time. You may assume that participant disconnections are temporal.
   - Message sends to drawers can fail. Note that failures are unreliably detected using timeout as a heuristic.
3. Drawer disconnections should not hamper the drawing session:

- Drawers should not be excluded from the drawing session immediately when they leave the network. When a drawer disconnects from the drawing session, his shapes are hidden (or grayed out), but the remaining connected participants can continue drawing.
- If a drawer disconnects when he was holding the ownership of a (grouped) shape before sending any update on that (grouped) shape, the rest of drawers will hide (or gray out) his shapes and he loses his ownership. If a drawer gets ownership, sends some updates and then disconnects before releasing the ownership of the (grouped) shape, the drawers will accept the updates and he loses the ownership.
- On the other hand, when a drawer disconnects, he may continue making local changes. Upon reconnection, the local changes are synchronized with the common canvas once he comes online again, as long as his changes do not conflict. Otherwise, the drawer will just undo his local changes and update his canvas with changes of the common canvas.

4. You may use any of the language abstractions seen in the seminar or lab sessions for designing the application. For example, you can implement the application on top of the distributed hash table implementation of session 8 [1], you can use tuples to share the shapes amongst participants, model your drawing session using leases, etc.

5. As in the lab session, you can assume that each participant sees the entire canvas and that there is only one drawing session at a time.

Extra requirements (not compulsory but rewarded):

- If you don't like what your neighbour drew, just undo it! In order words, extend the application with a distributed undo functionality.
- Allow users to zoom in on a particular part of the canvas. weScribblePro will then only update the subset of the canvas the user can see.
- Weaken the application assumptions. For example, you can adapt the application to take into account permanent disconnections of drawers, allow multiple drawing sessions at a time, add support to solve conflicts on drawing actions based on a consensus amongst the drawers of a session, etc.

3 Project Profile

In order to implement the application, you will choose one of the following profiles:

Profile A: Academic weScribble Pro. The distributed design is the most important part of your project, and the hardware platform where the application will be deployed is not relevant. The emphasis is on the design of the application, and the correctness of the distributed interations. You will use the version of AmbientTalk for desktop machines. Note that you can use the Eclipse plugin as done in the lab sessions, or you can also download the AmbientTalk interpreter and use the TextMate bundle or Emacs mode.

Profile B: Applied weScribble Pro. The distributed design is an important part of your project, but the application will be deployed on the Android platform. The emphasis is on the correct working of the application on the Android platform and a usable user interface. You will use the version of AmbientTalk included in the android-core library from the lab session 3. Mobile device(s) will be provided for testing the project.

You should mail no later than **8 April 2011** which project profile you would like to do.

[1] In that case, you should assume that a shape is stored in a number of peers, so that if one peer dies the information does not get immediately lost.

4 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. In order to test the application, you will provide:

Profile A: testcases written in the AmbientTalk unit testing framework or a GUI in Java AWT as done in the lab sessions, or whatever other means you may find convenient.

Profile B: an Android GUI.

Keep in mind that these test scenarios should test distributed interactions (e.g. testing that connected peers receive a newly added shape) rather than local ones (e.g. testing that changing the color of a shape works locally in the GUI of the user).

You will write a small report about this project. This report must be **no longer than 7 pages** (excluding figures) and serves as a guide for evaluating your project. The report should include:

1. An overview of your application and how the implementation fulfills the requirements.
2. The important cases and design choices (w.r.t. distributed aspects) considered in the 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 IdeAT plugin version or AmbientTalk build you used.

5 Evaluation

The project counts for half of your final score for the SDM seminar. The project will be evaluated mostly on good distributed design. Each profile will be evaluated as follows:

Profile A: distributed design 80% stability and usability 20%

Profile B: distributed design 60% stability and usability 40%

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 the distributed point of view.
- Quality and structure of the code **is** important.
- The project must be implemented in AmbientTalk. As in the lab sessions, you can exploit the symbiosis between Java and AmbientTalk to make use of Java classes for e.g. your data structures such as a Hashmap. However, all distributed communication **must** be implemented in AmbientTalk. In other words, you **cannot** use other technologies (e.g. Java RMI) as your distributed computing framework.
- Make sure that Java code (if any) is compatible with **version 1.5**.
- This assignment is to be made **individually**. Cooperation is not allowed in any form.
- Problems with the software tools (e.g. Eclipse plugin, Android setup, etc..) and devices should be reported **no later than** two weeks before the deadline, i.e. **6 June 2011**.

Being creative and implementing additional requirements of your own, or any of the extra requirements to the project is appreciated. If you have any questions, do not hesitate to ask for help.

Good luck!