e-mail: egonzale@vub.ac.be
office: 10F704

1 Leader Election

Language available at `http://prog.vub.ac.be/amop/at/download`

1.1 The Paper-Scissor-Stone Game

The purpose of this exercise is to create a digital decentralized distributed version of the paper-scissor-stone game. This game works as follows: a number of clients join in a session and have to choose paper, scissor or stone. Depending on the choices one wins or loses the game:

- A choses scissor and B choses paper, then A wins. A gets one point and B loses one point.
- A choses scissor and B choses stone, then B wins. B gets one point and A loses one point.
- A choses paper and B choses stone, then A wins. A gets one point and B loses one point.
- in all other cases we have a draw (no winner). A and B's scores remain the same.

The game should also be playable with more than two people. For this the choices of all players are compared to one another one-by-one and using the decision rules described above.

After each round in the game the winner(s) of the game are displayed on each peer that participated in the game. After playing a game the scores remain (they are not reset). Once a game has started no other peers can enter the game until the outcome of the game is decided and a new game starts. When a participating peer fails while a game is in progress the game should continue but the choice of the failed peer should be disregarded.

1.2 Non-Functional Requirements

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

- It must be implemented in AmbientTalk. As in the lab sessions, you can use the symbiotic relationship 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.
- 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:
  - As a starting point you could implement the game in client/server style. In this style one actor plays the role of the coordinating server, which is responsible for collecting the answers and to decide the outcome of the game.
  - In order to make the whole fault-tolerant you should consider a peer-to-peer organization. As seen in the lab session, peer-to-peer architecture differs from a client/server approach in that every peer supports the same behavior. Hence, in the context of this game every peer should be able to act as the server.
  - Using a leader election protocol one peer can be selected to act as the server that coordinates the game. If that peer should fail another peer has to be selected to act as the coordinator and assume the role of the broken peer.
- Players may enter and leave the network at any point in time.
- Note that failures are unreliably detected using timeout as a heuristic.
- The scores (and thus the winners) of the game should remain consistent in server (i.e. the peer acting as the coordinator) irrespective of the peer who decides the outcome. In other words, if the peer who decided the outcome gets disconnected and another peer is elected as the coordinator, the later should keep properly the scores.
- You may assume that there is only one game that is being played at a time.
- Quality and structure of the code **is** important.

1.3 Testing and Report

A second part of the assignment is 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, a testcase (making use of the AmbientTalk unit testing framework) or whatever other means you may find convenient to test your application, but the distributed design should be the focus of your project.

You will write a small report about this project. This report must not be 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.

Note: this project assignment is to be made individually! Cooperation is not allowed in any form.

Deadline 20 June 2008.

Delivery Both documentation and code should be delivered in digital *and* paper format. On the day of the deadline a box in which you can deposit the paper version will be placed at 10F704. The digital version should be sent to egonzale@vub.ac.be including "Project DMS" in the subject line and indicating the AmbientTalk build used. Both paper and digital version of the project should be delivered on the day of the deadline before 16:00.

Veel succes!