Concurrent and Parallel Programming project:

Relation

University of Padua
academic year: 2011/2012

Information about document

Document title	Relation
Date	2012/04/02
Author	Lorigiola Giacomo [592992]
Addressee	Prof. Mauro Conti
	Prof. Eyup S. Canlar

Summary

This document has the purpose to describe the architecture and design of the project.

Contents

List of Figures

1 Problem Description

The project has the purpose to simulate protocols for Wireless Sensor Networks.
Specifically the protocols implemented in the project are the LSM and RED one, that have the purpose to solve the clone detection problem.

Simulation is based on a *client* that reads configuration values from a file available on a http server and reachable throw a Url addres, then runs the simulations and finally elaborates specific statistics about simulation and sends them to a *RMI server*. The configuration values include a number of simulations, the number of nodes used, radius of comunication between nodes and other.

The simulation includes the use of n nodes distributed on an unity square area: each nodes have their unique identifier number and their position in the area, specified by two axes *(x,y)*.
For each simulation a selected node from the n one is cloned, this means that a new node is created whith the same identifier number of the original node selected, but with a differet position in the area. The applicated protocols have the target to find the clone.

2 Program Design

2.1 Class specifics

For a smart and extensible implementation of the project, the classes, divided into packages, are studied with some hierarchies, interfaces and main classes to manage the other ones.
The logic is also kept separate from the graphics and a connector class allows the communication between the parts.

2.1.1 Client classes

- Package client.logic
  - `Client`
    Implementation of client: class that starts the application and communicates with server at the end of the simulation to send the statistics.
  - `ActivatorNode` [**class** ActivateNode **extends** Thread]
    It's s *Singleton* class used to activate all nodes of a simulation; the class has a buffer of `Vector<Node>`, the class is waiting on this buffer, and when the buffer isn't empty, ActivatorNode wakes up and activates all nodes in the vectors.
  - `Connector`
    The class used to manage communications between different objects, in particular between logic and graphic objects, providing a total decoupling between the two parts. This decoupling makes a future extension or modification easier.
  - `Data`
    The class represents the information of each node at the end of a single simulation. This information is sent to ElaboratorData that calculates the statistics for the server.
  - `ElaboratorData` [**class** ElaboratorData **extends** Thread]
    It's a *Singleton* class that receives the results of the simulations and processes them producing the statistics needed to be sent to the server.
  - `Hypervisor` [**class** Hypervisor **extends** Thread]
    The main class that manages the entire cycle of simulation. Summarily it creates nodes, start simulations and listens to the terminations of simulations.
  - `Message` [**interface** Message]
    Common interface for message classes. Each type of message sent by a node is an implementation of this interface. The class declares the method `public int getIdSender()` , that return the identification number of the sender.

- `MessageSensorNetworks` [**class** MessageSensorNetworks **extends** Message]
  Common interface for messages used to implement "Wireless Sensor Networks". This interface adds the method `public Position getPosSender()` that return the position of sender.
- `MessageClaim` [**class** MessageClaim **implements** MessageSensorNetworks]
  The class represents the messages of *location claim*.
- `MessageControl` [**class** MessageControl **implements** MessageSensorNetworks]
  The class represents the messages for the control of the clone.
- `MessageDeath` [**class** MessageDeath **implements** Message]
  The class represents the message sent by a node, which has finished the energy.
- `Node` [**abstract class** Node **extends** Thread]
  The class represents generic nodes.
- `NodeLSM` [**class** NodeLSM **extends** Node]
  The class represents the nodes for LSM protocol.
- `NodeRED` [**class** NodeRED **extends** Node]
  The class represents the nodes for RED protocol.
- `Position`
  The class represents the position *(x,y)* of nodes.
- `ReadURL`
  The class used to read configuration file.
- `ReceiverStatInterface` [**public interface** ReceiverStatInterface **extends** Remote]
  Common interface between client and server of RMI object to print statistics.

* Package client.gui
  - `Gui` [**public class** Gui **extends** JFrame]
    The class implements the grafic interface for users.

* Package client.exception
  - `ExcEndEnergy` [**public class** ExcEndEnergy **extends** Exception]
    Exception throwned when a node ends its energy.
  - `ExcFindClone` [**public class** ExcFindClone **extends** Exception]
    Exception throwned when a node finds the clone.
  - `ExcNoNeighbors` [**public class** ExcNoNeighbors **extends** Exception]
    Exception throwned when a node has no more neighbors.
  - `ExcReadFile` [**public class** ExcReadFile **extends** Exception]
    Exception throwned if configuration file isn't correct.

2.1.2 Server classes

* Package server
  - `ReceiverStatImp` [**class** ReceiverStatImp **extends** UnicastRemoteObject **implements** ReceiverStatInterface]
    Implementation of RMI object to receive and print statistics.
  - `ReceiverStatInterface` [**public interface** ReceiverStatInterface **extends** Remote]
    Common interface between client and server of RMI object to print statistics.
  - `ServerRMI`
    The class represents the server.

2.1.3 Common classes

* Package common
  - `Stat` [**public class** Stat **implements** Serializable]
    The class represents the statistics for server.

2.2 Class diagram

Below are represented Client and Server classes diagrams.

2.2.1 Client class

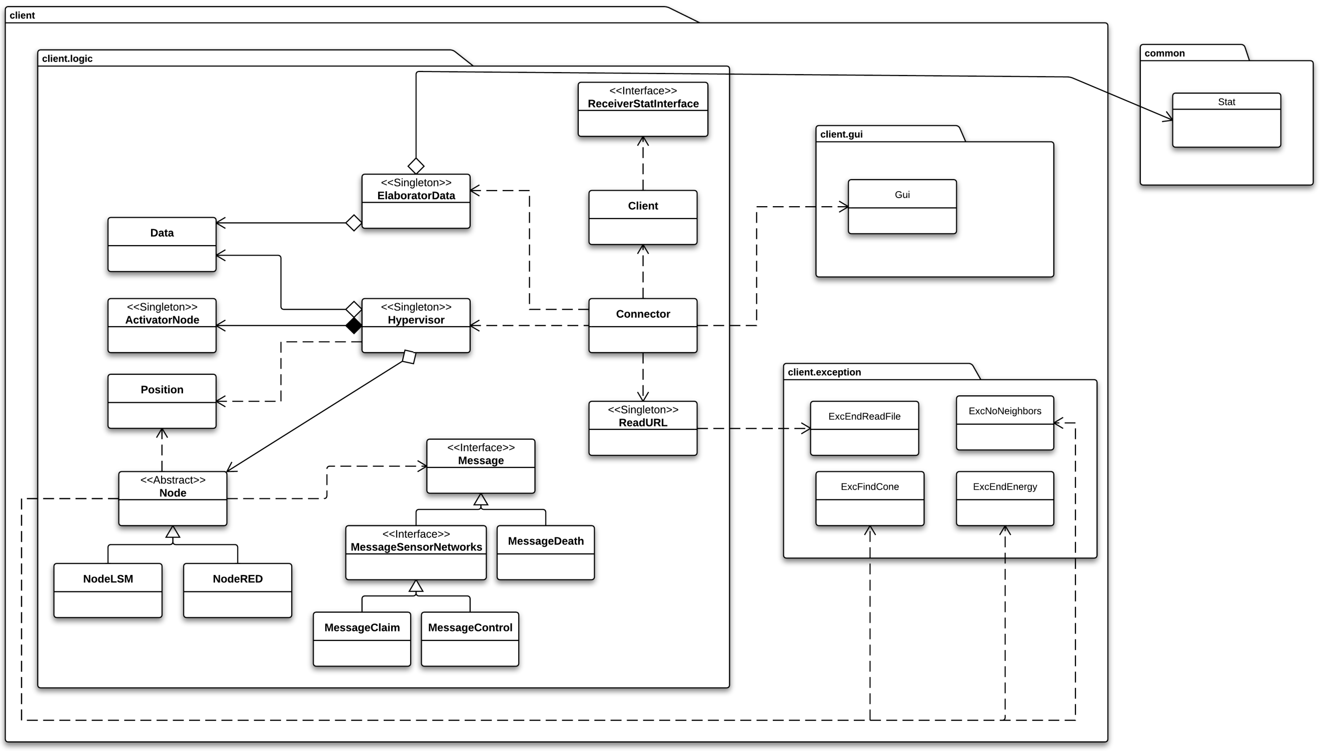


Figure 1: Client class diagram

2.2.2 Server class

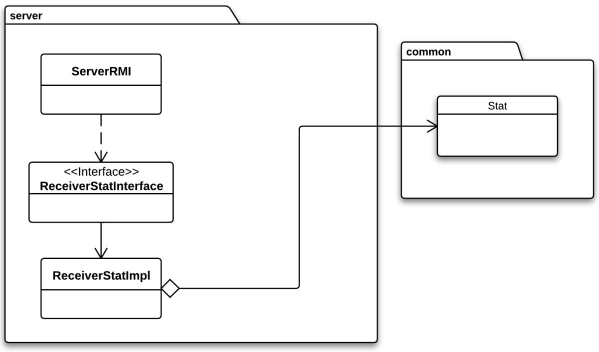


Figure 2: Server class diagram

3 Motivation of your design choices

I chose to make the `Hypervisor` *Singleton* class because the main class that menages all simulations must have only one instance. It's very important in order to avoid mistakes in programming or design if the application will be extended or modified.

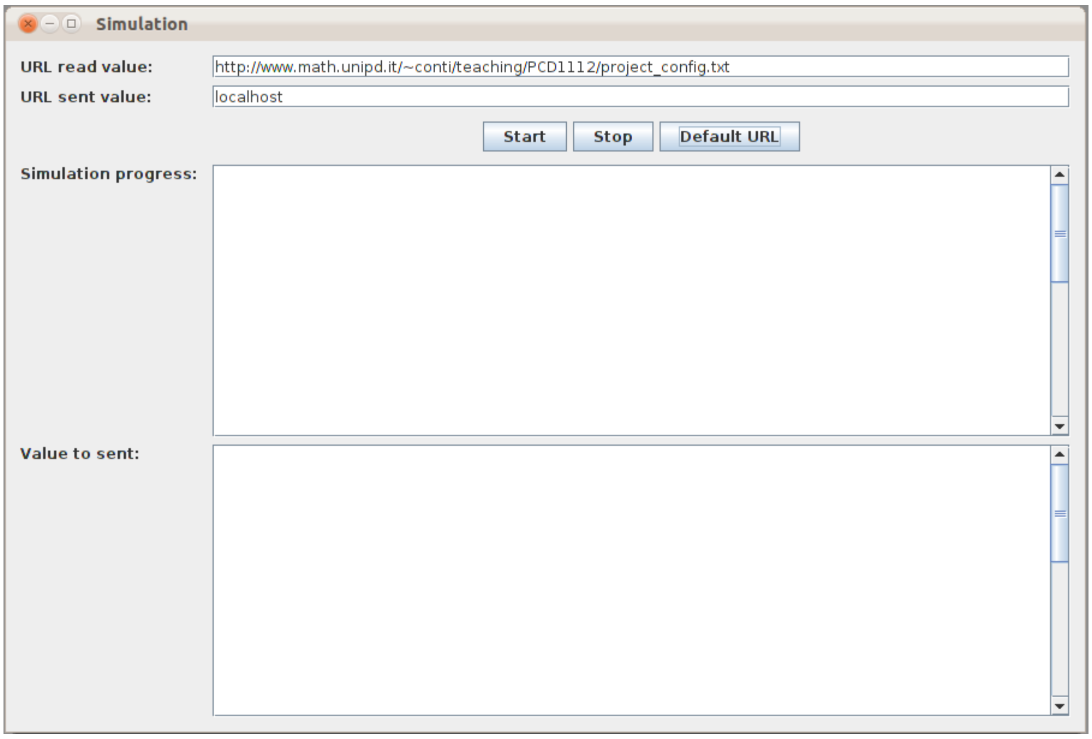
`ActivatorNode`, `ElaboratorData` and `ReadURL` classes are also *Singleton* because they are used only on demand of Hypervisor and a single instance is necessary for simulation.

Class `Node` is *Abstract* because in each protocol, nodes have different behavior in implementing sensor networks, therefore Node is the base class and its extensions are the implementation of node for each protocol: metods that are equal for each protocol are defined in class Node, other ones are abstract and they are defined in each extension class of Node.

Messages are also implemented with a hierarchy of classes: `Message` are the common interface for all type of message and declare the method `public int getIdSender()` because all message have sender identifier. Then there is the interface `MessageSensorNetwork` that extends Message and it is the base class of all messages used to implement sensor network: MessageClaim and MessageControl. This class has the common method `public Position getPosSender()` that returns the position of the sender.
When a node finishes its energy, it send to all its neighbors a `MessageDeath`; it is assumed that the sending of these messages does not request energy, and these messages are put in first position in the buffer of messages received from the receiving node, so that it is tried as first.
If a node is sending a message to a neighber that is dying, the message is sent successfully and the recipient that has died will not process the message.
If a node has no more neighbors to interact with, it terminates its life cycle.

4 Abstract description of the main working of the system

When launching application, GUI appears as shows in Figure 3.



http://www.math.unipd.it/~conti/teaching/PCD1112/project_config.txt
localhost

Figure 3: GUI application

In the field **URL read value**, the URL of the configuration file that contains the values for the simulation is indicated.

In the field **URL sent value**, the URL of the server to send statistics is indicated.
At the beginning in these fields are proposed default values, however user can modify them and then restore default falues by clicking the **Default URL** button.
Then clicking the **Start** button, simulation start.
If some problem occurres, such as connection problem, the URL of the file or server isn't correct, or the contents of the file is incorrect, an error text appears in the **Simulation progress** area and indicates the error type.
If everything is right, in the **Simulation progress** area, the configuration values of the simulation and then the progress of simulation for all cicle is printed.
At the end of each cycle, in the **Value to send** area, are printed the statistics to send to the server.
Finally, at end of the simulation in the **Simulation progress** area, the correct termination of simulation and the confirmation of the correct sending of the statistics to the server, are printed.
At any time the simulation can be stopped by clicking the **Stop** button (or the red button of closure): the simulation is stopped and all its threads are terminated and the application is closed.

5 Testing and Evaluation of your programs

Below are presented some statistics generated by the application with different parameters.
Standard parameters for simulation:
NSIM=100 **p**=0.1 **g**=2 **n**=100 **r**=0.1
E=32000 **E_send**=20 **E_receive**=10 **E_signature**=50

In red are represent the parameters modified from standard parameters.

1. Simulation: standard parameters have been used.

   - Statistics with **LSM** protocol (whitout simulation parameters)

```
- 0 14 4.73 3.23   0 8 1.79 1.98   0 8 1.51 1.88   0 280 89.8 63.48    0 8 1.51 1.88   0
- 0 13 4.1 3.16    0 9 1.5 1.98    0 9 1.24 1.95   0 290 77.52 62.65   0 9 1.24 1.95   0
- 0 13 4.33 2.93   0 8 1.42 1.67   0 8 1.16 1.62   0 240 78.42 51.6    0 8 1.16 1.62   0
- 0 9 3.51 2.17    0 7 1.08 1.39   0 6 0.91 1.34   0 210 64.46 43.68   0 6 0.91 1.34   0
- 0 10 3.62 2.5    0 7 1.21 1.54   0 6 1.07 1.46   0 230 68.42 50.32   0 6 1.07 1.46   0
...
```

   - Statistics with **RED** protocol (whitout simulation parameters)

```
- 0 14 3.45 2.66   0 12 0.97 1.77   0 12 0.44 1.52   0 300 58.12 45.94   0 12 0.44 1.52   0
- 0 15 4.63 3.6    0 10 1.63 2.83   0 10 0.46 1.49   0 270 77.13 63.49   0 10 0.46 1.49   0
- 0 28 4.75 4.69   0 26 2.05 4.29   0 24 0.65 2.68   0 570 83.86 93.09   0 24 0.65 2.68   0
- 0 14 3.78 2.67   0 11 1.23 1.93   0 12 0.55 1.91   0 280 65.25 49.12   0 12 0.55 1.91   0
- 0 19 4.62 4.2    0 17 1.88 3.18   0 17 0.65 2.52   0 390 80.89 77.05   0 17 0.65 2.52   0
...
```

2. Simulation: **r**= 0.3 **E**= 320

   - Statistics with **LSM** protocol (whitout simulation parameters)

```
- 10 30 23.8 5.17    0 6 2.16 1.49   0 5 0.93 1.06   130 320 284.65 52.64   0 5 0.93 1.06   0
- 6 30 22.53 4.49    0 5 2.76 1.27   0 5 1.21 1.23   70 320 280.1 53.78     0 5 1.21 1.23   0
- 7 30 23.76 6.17    0 6 1.92 1.51   0 4 0.84 1.0    90 320 281.29 61.32    0 4 0.84 1.0    0
- 6 30 23.71 5.57    0 6 1.95 1.51   0 4 0.81 1.13   120 320 280.4 56.58    0 4 0.81 1.13   0
- 10 30 24.29 5.93   0 6 1.77 1.46   0 5 0.65 1.12   130 320 283.17 57.92   0 5 0.65 1.12   0
...
```

   - Statistics with **RED** protocol (whitout simulation parameters)

```
- 2 30 23.68 5.92    0 7 1.88 1.49    0 2 0.04 0.28    30 320 270.59 65.47    0 2 0.04 0.28    0
- 11 30 23.68 5.7    0 8 1.7 1.55     0 7 0.14 0.82    120 320 270.59 63.56   0 7 0.14 0.82    0
- 5 30 23.41 6.2     0 8 1.61 1.54    0 7 0.07 0.69    60 320 265.94 68.04    0 7 0.07 0.69    0
- 8 30 23.86 4.73    0 10 2.13 1.75   0 7 0.09 0.72    90 320 275.45 55.87    0 7 0.09 0.72    0
- 5 30 22.89 6.05    0 9 1.89 1.8     0 5 0.07 0.51    60 320 262.77 67.87    0 5 0.07 0.51    0
...
```

3. Simulation **g**= 4 **r**= 0.6

   - Statistics with **LSM** protocol (whitout simulation parameters)

```
- 1 190 83.32 31.31    0 85 31.19 17.99     0 75 26.31 17.09     20 3510 1418.12 605.39
  0 75 26.29 17.07     0
- 2 166 83.36 39.61    0 86 27.67 17.85     0 76 22.86 16.46     30 3070 1348.91 659.53
  0 76 22.84 16.43     1
- 8 214 86.21 32.12    0 139 31.3 21.52     0 128 25.98 20.36    90 4820 1444.85 668.51
  0 128 25.96 20.35    0
- 23 198 93.6 29.84    0 121 36.08 21.02    0 113 30.31 20.11    240 4290 1609.9 625.41
  0 112 30.27 20.02    0
- 26 179 91.94 30.65   0 109 36.57 23.98    0 104 30.89 23.45    270 3780 1604.06 707.77
  0 104 30.87 23.42    0
...
```

   - Statistics with **RED** protocol (whitout simulation parameters)

```
- 47 337 99.63 50.19    2 1571 36.41 160.5     0 1580 18.09 157.46    540 32000 1551.39 3211.07
  0 1580 18.09 157.46   0
- 0 291 88.04 42.14     0 830 25.59 91.6       0 836 10.49 85.16      10 17110 1251.19 1860.44
  0 835 10.47 85.04     0
- 38 409 98.99 44.81    1 1567 35.15 162.44    0 1575 20.23 158.57    400 32000 1553.76 3250.95
  0 1575 20.23 158.57   0
- 36 272 100.95 43.34   0 1565 34.78 157.31    0 1575 17.87 156.47    370 32000 1546.14 3153.06
  0 1575 17.87 156.47   1
- 44 297 103.22 41.98   2 1567 35.7 158.05     0 1582 18.43 157.67    480 32000 1583.56 3158.78
  0 1582 18.43 157.67   0
...
```