Università degli Studi di Verona

DIPARTIMENTO DI INFORMATICA
Corso di Laurea in Informatica

Basi di Dati

Elaborato

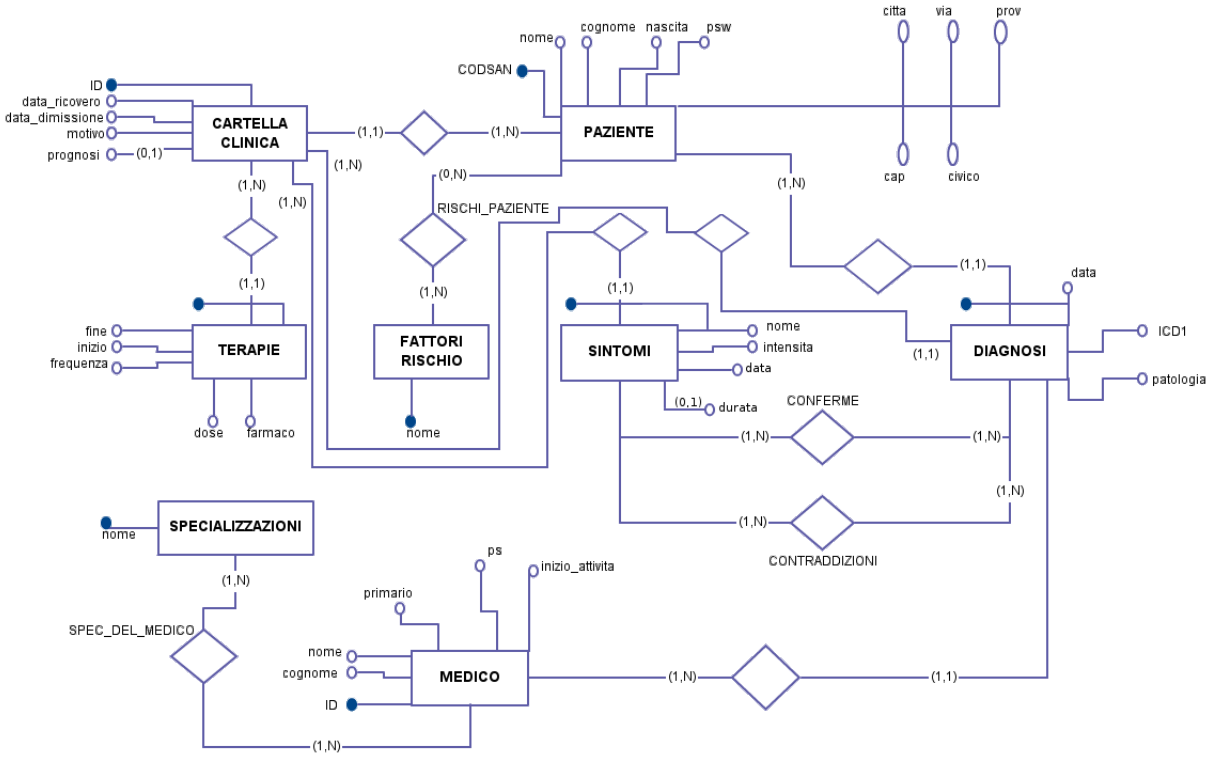
Candidati:
Enrico Giordano
Matricola VR359169

Cristian Pinna
Matricola VR361121

Anno Accademico 2013-2014

Indice

Parte I
Progettazione Concettuale



Considerazioni personali e strategie adottate durante lo sviluppo del progetto:

Parte II

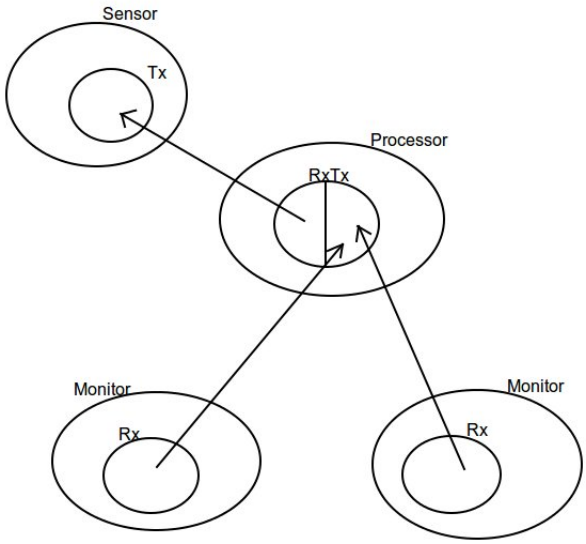
General pattern

Now we explain an example of system behaviour: when a car arrives or exits to the park, the node "car detector" sends a message to the node "process unit", that store the request (for future use) and send to the monitor the average number of car/hour and the number of free parking places.

According to observer pattern, the subject (Sensor) notify the observer (Process Unit) that a new message is pending, so the observer (Process Unit) receive istantly the message. After the computation, in the same the Process Unit send messages to the different displays. The observer pattern is used only in the comunication nodes.

So when the sender node has to send a message , it must call the receive method of the receiver node; in this way it's implemented an observer pattern, which is an optimization of a real system. In fact, in a real context, the receiver must poll on the receive path (or hardware) or must have an interrupt routine that manage the receiving process; in this system, the sender calls the "receive" method of the receiver.

In the picture, the arrows show how a class (or a method) call other class (a class is called to another class).



The "abstract" class Node is implemented in the following classes

- Detector: a sensor that controls car traffic (entering or exiting car);
- Processor: a processing unit that calculate average number of cars/hour and number of free parking places;
- Monitor: a display that shows the results of processing unit.

The channel comunication is implemented by "Channel" class, that create a link in a comunica-

tion grid. There are 2 types of Channel: WireChannel and WirelessChannel (in this project there is no difference).

All the operations in the Processor are implemented by other classes that implement basic operation (sum, min, ecc...). These classes extend NodeComputation interface. A complex computation is formed by a combination of basic operations, like an ALU in a real processor.

The NodeCommunication interface manages the exchange of messages between nodes and is implemented in the following classes:

- TxNode: must send a message (for Detector);
- RxNode: must receive a message (for Monitor);
- TxRxNode: must send and receive a message (for Processor);

Parte III

Simulation

The detector, the processing unit and 3 monitors are created in the main class. These monitors show a display for average number of car/hour, a display for number of free parking places and a (not required) display for car traffic.

The nodes are linked in the wireless grid so they can comunicate one another. A realtime clock in the processing unit simulates the time (for the average cars/hours) implemented with a thread that increase a counter every second.

GUI

In according to show a simulation of the system we have implemented a simple user interface. There are:

- a control button panel, which contains three buttons (start, stop, reset) that controll intuitively the system;
- a complex of simple monitor that show the evolution of the system.

The user can control the simulation with some control buttons: the “start” button starts the simulation, the “stop” button stops the simulation and the “restart” button restarts the simulation.

Parte IV
Questions

Which patterns are used within the scheme of Figure 2?

The first figure represents an interface class called "nodeCommunication" and three other classes (TxNode, RxNode and TxRxNode) that implement the nodeCommunication interface. The interface class is a type of class that presents many methods of a particular class; this methods are not implemented, but the class that implements this interface must implements them.

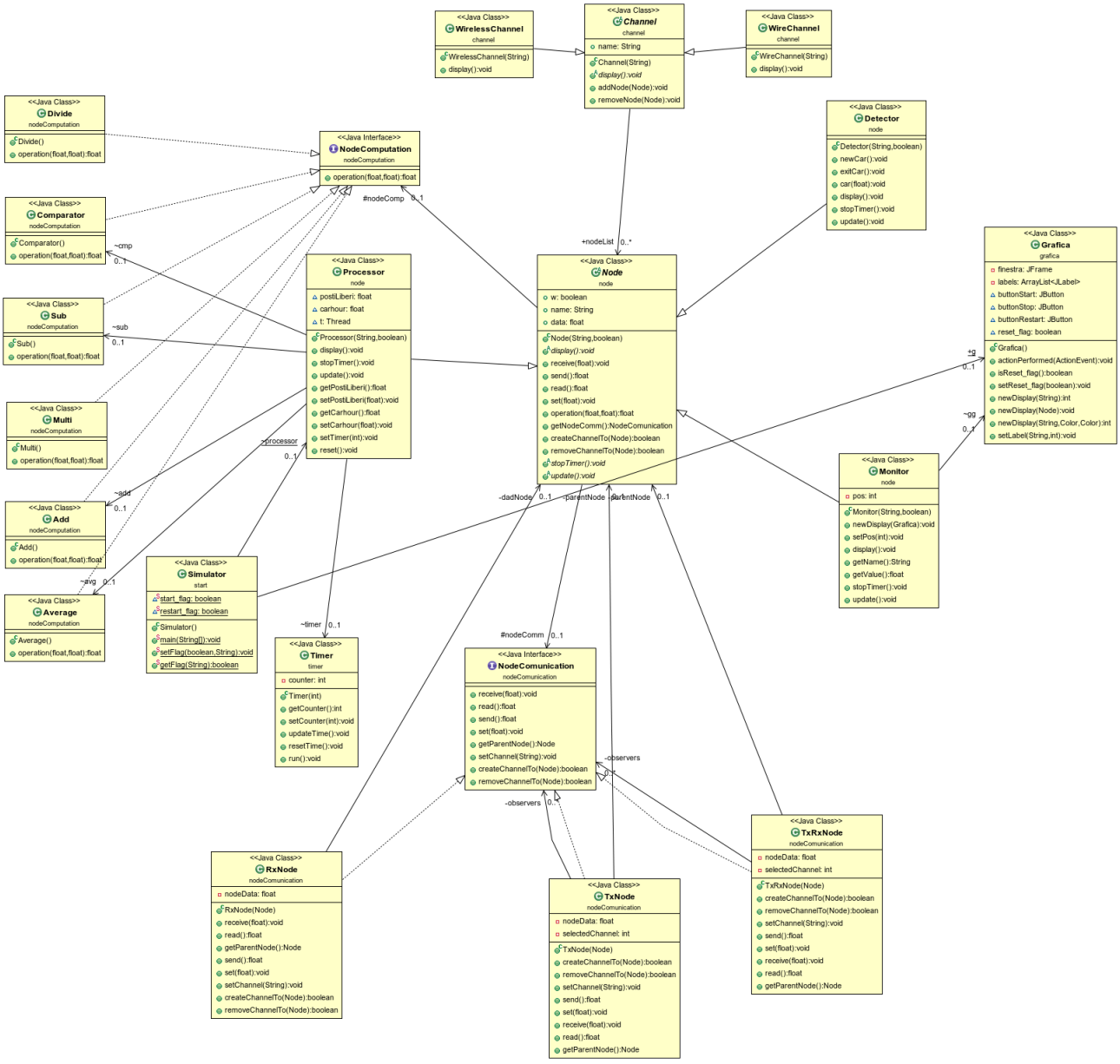
The second figure represents an another interface called "nodeComputation" and some other classes (Add, Sub, Average, Divide, Multi, Comparator) that implement nodeComputation interface. Every nodeComputation class must have a method that must be implemented by the programmer. Interfaces cannot be instantiated, but rather are implemented. A class that implements an interface must implement all of the methods described in the interface, or be an abstract class. They simulate multiple inheritance.

The third figure represents an abstract class called "Node" and three other classes (Detector, Processor, Monitor) that inherited the superclass methods ("Node"). The abstract class must have some implemented methods and the son classes have the same method and the same attribute. If a son classes have a different implementation of a method, it must override it.

An abstract type may provide no implementation, or an incomplete implementation. Abstract types will have one or more implementations provided separately, like in this case.

So this is a Strategy Pattern. The strategy pattern is a software design pattern that enables an algorithm's behavior to be selected at runtime. For instance, a class that performs validation on incoming data may use a strategy pattern to select a validation algorithm based on the type of data, the source of the data, user choice, or other discriminating factors. These factors are not known for each case until run-time, and may require radically different validation to be performed. The validation strategies, encapsulated separately from the validating object, may be used by other validating objects in different areas of the system (or even different systems) without code duplication.

Explanation of used patterns



The first class, Simulator, can instantiate the other class of the system. Every node is chained to the other by a wireless grid and can comunicate thanks to the observer pattern, using a "subnode" called Tx (for trasmission), Rx (for receiving) and TxRx (both). The "actors" of the system (Detector, Monitor, Detector, Processor) are an implementation of Node class.

For node implementation, it was used a Strategy pattern, because there is a superclass called Node and other node classes are an implementation of Node class.