Generation of virtual agents behavior

Development of a demonstration applet

Matthieu Macret

« Impact Project » - Information Technologies Specialization

Supervisors : Benoit Lacroix [2] lacroix.benoit@gmail.com
Philippe Mathieu [2] philippe.mathieu@lifl.fr
Didier Corbeel [1] didier.corbeel@ec-lille.fr

[1] Ecole Centrale de Lille
Cité Scientifique - bp 48
59655 Villeneuve D'Ascq Cedex

http ://www.ec-lille.fr

[2] Laboratoire d'Informatique Fondamentale de Lille
umr ustl/cnrs 8022 - B,t M3
Université des Sciences et Technologies de Lille
59655 Villeneuve D'Ascq Cedex

http ://www.lifl.fr/SMAC/

Table des matières

1 Introduction

1.1 Context

I made this « Impact project » during the third year training in the IT sector in « Centrale Lille ».

This work was realized in cooperation with the SMAC team (Multi-Agent Systems and Behavior team) of the Computing Science Laboratory of Lille (LIFL). The team studies Artificial Intelligence. The topics of their research are Artificial Intelligence, mainly its software engineering aspects, and specifically the modeling behavior between virtual entities..

The concept of agent and multi-agent systems has emerged in areas such as theoretical Game theory and artificial intelligence, but also in applied areas such as electronic commerce or simulation platform. The team specializes in modeling behavior, the implementation of these behaviors in environments that require autonomous entities, the evaluation of behavior between agents and the achievement of open-platform testing.

My « Impact Project » is about one of the most important issues studied by the team : the behavior of geographically located entities in the simulation platform.

Multi-agent systems, which belong to artificial intelligence, are applied in many areas. There are three ways of using them :

- simulation, for example in biology (simulation of insect behaviors), in medicine (modelling the development of cancer cells) or in finance (simulation of traders' behaviors),
- solving complex problems,
- design programs in distributed systems.

In this project, we will deal with simulations using multi agents systems. The realism of systems is of utmost importance in simulations because it has a lot of influence on the reliability of results. This realism depends on the diversity of the agents' behaviors. We will deal with one model of generation of agents that enables to increase the diversity of the population involved in the simulation.

1.2 Objectives :

The project aims at designing and programming an demonstation applet illustrating the possibilities of this model.

The main benefit of using an Java Applet is that it can be hosted on a web server (accessible by eveyone from everywhere). Another one is the deep complexity in programing allowed by such a technologie, which was needed for the development of a such model.

To illustrate as well as we can the diversity offered by this model, we chose to use it in a simulation. Here is the main scenario :

"Creatures of numerous species are generated, they move in an environment and have the possibility to interact with each others. The interaction stems from their characteristics and their species"

The institution will be the « primordial soup ». It contains all the possible attributes of a creature. (size, speed, agressiveness or number of childen) The norms will be the species. They contains one, some or all the institutional attributes. A behavior, linked to a species, will be the instantiation of the characteristics according to the norm associated to this species. The creatures will be the agents of the system.

Violating behaviors, that is to say behaviors which don't respect the whole norm, could also be generated.

With this simulation, we will be able to see the influence of the different parameters of the generation model on the results of the simulation, for example :

- Which species will survive ?
- What are the consequences of adding one or many violating creatures to the simulation for its species ?

As far as the technical architecture is concerned, it reproduces the model described before.

The aim of this project was to demonstrate the possibilities of the model. That is why the possible interactions and the applet graphic rendering were for us critical points of the project.

We spent a lot of time designing the HMI. We wanted it to be ergonomic, attractive, interactively rich and a lot of parameters to be defined by the generation model.

2 The model of the behavioral differentiation

2.1 Model presentation

The notion of norms, in the multi-agents systems, is often used to regulate the behavior of the agents. Norms can be considered as a way to specify the agent's behaviors, but also to control the respect of these specifications. Each agent is linked to a unique norm. The characteristics of each agent must respect this norm and they are initialized at the beginning of the simulation. If the parameters are dynamic, their values can evolve during the simulation : this evolution is controlled by the norm.

The generator of « violating » behaviors is planned in the model. A norm is assigned to each agent. One or several characteristics of the agent can be outside of the definition set specified by the norm. These violations can be quantified and so controled.

Enabling the violations leads to a higher level of realism. Indeed, in the reality, everything is not gradable and often the exceptions can have huge consequences on the system.

2.2 The model in depth

2.2.1 The institution

The institution is an entity which contains the list of all the parameters composing the behavior of an agent. It also contains the definition set in which these parameters must be instantiated.

These institutional parameters are defined by a name and a definition set which can be discrete or continous. The σ parameter describes the tendancy of the generation engine to generate "violating" behaviors.This parameter is tested before each generation of "violating" behaviors. (see Algorithm 1)

Definition 1 *An* institutional Parameter pi *is a quadrupled* $(l_{pi}, \mathcal{D}_{pi}, v_{d_{pi}}, f_{pi})$ *defined by :*
- l_{pi} *an unique label ;*
- $\mathcal{D}_{pi}$ *a finite definition set ;*
- $v_{d_{pi}} \in \mathcal{D}_{pi}$ *a default value ;*
- $f_{pi} : \mathcal{D}_{pi} \mapsto [0, 1]$ *a distance function.*

Definition 2 *An* Institution I *is a triple* $(l_i, \mathcal{P}_i, \sigma)$ *defined by :*
- l_{pi} *an unique label ;*
- $\mathcal{P}_i$ *a finite definition set of* institutional parameters *;*
- $\sigma \in [0, 1]$*, the tendancy of the generation engine to generate "violating" behaviors.*

2.2.2 The norm

A norm is a sub set of the institution. It can contain one or all the parameters of the institution. A norm parameter has a definition set which is included in the corresponding institutional parameter one. He also has a default value.

Definition 3 *A* Norm Parameter p *is a quintuplé* $(l_p, p_{\text{ref}}, \mathcal{D}_p, v_{d_p}, f_p)$ *defined by :*
- p_{ref} *a reference institutional parameter* $(l_{pi}, \mathcal{D}_{pi}, v_{d_{pi}}, f_{pi})$ *;*
- $l_p = l_{pi}$ *a label ;*
- $\mathcal{D}_p \subseteq \mathcal{D}_{pi}$ *a finite definition domain ;*
- $v_{d_p} \in \mathcal{D}_p$ *a default value ;*
- $f_p : \mathcal{D}_{p_{\text{ref}}} \mapsto [0,1]$ *A distance function.*

The norm also has a τ_N parameter which quantifies the tendancy of the generation engine to generate "violating" behaviors and a δ parameter which gives the admissible distance to the norm for a "violating" case. The calculation of the distance is given in the 7 definition.

Definition 4 *A* Norm N *is a quadrupled* $(I, l_N, \mathcal{P}_N, \tau_N)$ *defined by :*
- I *the reference institution* $(l_i, \mathcal{P}_i, \sigma)$ *;*
- l_N *a unique label ;*
- $\mathcal{P}_N$ *a finite definition set of parameters p which verifies :*

$$\begin{cases} \forall p \in \mathcal{P}_N,\ p_{\text{ref}}(p) \neq 0 \\ \forall p_1 \in \mathcal{P}_N,\ \exists p_2 \in \mathcal{P}_{N_{\text{ref}}},\ p_{\text{ref}}(p_1) = p_2 \\ \forall p_1, p_2 \in \mathcal{P}_N,\ p_{\text{ref}}(p_1) = p_{\text{ref}}(p_2) \Rightarrow p_1 = p_2 \end{cases}$$

- $\tau_N \in [0,1]$*, the tendancy of the generation engine to generate "violating" behaviors ;*
- $\delta \in [0,1]$*, the admissible distance to the norm.*

2.2.3 The behavior

A behavior is linked to a norm, or directly to the institution. All the behavioral parameters are instanciated respecting the definition set of the linked norm, or of the linked institution.

Definition 5 *A* instantiated Parameter p *is a triple* $(p_{\text{ref}}, l_p, v_{d_p})$ *defined by :*
- $p_{\text{ref}} \in \mathcal{P}_I \cup \mathcal{P}_N$ *a reference norm parameter (N) or a reference institutional parameter (I)* $(l_{pref}, \mathcal{D}_{pref}, v_{d_{pref}}, f_{pref})$ *;*
- $l_p = l_{pref}$ *a label ;*
- $v_{s_c} \in \mathcal{D}_{pref}$ *a instantiated value.*

Definition 6 *A* behavior C *is a couple* $(N, \mathcal{P}_C)$ *defined by :*
- N *the reference norm* $(I\ (l_i, \mathcal{P}_i, \sigma), l_N, l_i, \mathcal{P}_i, \tau_N)$ *;*
- $\mathcal{P}_C$ *a finite set of instantiated parameters p, which verifies :*

$$\begin{cases} \forall p \in \mathcal{P}_C,\ p_{\text{ref}}(p) \neq 0 \\ \forall p_1 \in \mathcal{P}_C,\ \exists p_2 \in \mathcal{P}_i \cup \mathcal{P}_N,\ p_{\text{ref}}(p_1) = p_2 \\ \forall p_1, p_2 \in \mathcal{P}_C \times \mathcal{P}_N \ /\ p_{\text{ref}}(p_1) = p_2,\ \nexists p_3 \in \mathcal{P}_i,\ p_{\text{ref}}(p_1) = p_3 \end{cases}$$

Concerning the "violating" behaviors, a q_s constant is defined. It gives the admissible distance between the behavior and the norm. A behavior is admissible if $q_s < \delta$.

q_s is defined by :

Definition 7

$$q_s = \frac{\sum_{p \in \mathcal{P}_c} f_{p_{\text{ref}}}(v_{s_c})}{Card(\mathcal{P}_{N_s})}$$

Parameter	Definition Set
p1 = maximal speed	$dp1 = [0, +8]$
p2 = safety time	$dp2 = [0, +8]$
p3 = overtaking risk	$dp3 = [0, 1]$
p4 = speed limit risk	$dp4 = [0, +8]$
p5 = observe signs	$dp5 = true, false$
p6 = observe priority	$dp6 = true, false$

TABLE 1 – Institution

Parameter	Normal Behavior	Agressive Behavior
pn1 = maximal speed	dpn1 = [100, 140]	$dpa1 = [140, 160]$
pn2 = safety time	dpn2 = [0.8, 5.0]	$dpa2 = [0.1, 1.2]$
pn3 = overtaking risk	dpn3 = [-0.5, 0.5]	$dpa3 = [1.0, 2.0]$
pn4 = speed limit risk	dpn4 = [0.0, 1.0]	$dpa4 = [1.0, 10.0]$
pn5 = observe signs	dpn5 = true	$dpa5 = true, false$
pn= observe priority	dp= true	$dp = true, false$

TABLE 2 – Normes

2.3 The example of the automotive traffic

The automotive traffic can be modelised by a multi-agents system. The agents are the drivers. A diversified population is necessary to improve the realism of the simulation, that is to say it is necessary to have careful drivers, agressive drivers, normal drivers but also drivers that we can't classify in any of these categories.

The behavioral parameters which are used in the simulation can be the followings :
- the maximum speed of a driver,
- the minimum safety distance which a driver admits,
- the tendancy of a driver to overtake other driver,
- the tendancy of a driver to exceed the authorized speed limits,
- the respect of the roadsigns.

2.3.1 Implementation of the model

With these parameters, an institution can be defined. It is presented in the table 1.

Several norms can be defined with this institution. Two examples are presented in the table 2 : « normal driving » et « aggressive driving ». The « normal driving » norm uses all the parameters of the institution, the definition sets are sub set of the institution one. A driver generating with this norm don't take risks to overtake other drivers, don't exceed the speed limitations and he respects the roadsigns.

The « aggressive driving » norm also uses all the parameters of the institution. It allows the driver not to respect the speed limitations and the roadsigns. A driver generating with this norm will also take more risks to overtake, he will drive faster and his safety distances will be shorter.

These norms allow the generation of a large diversity of behaviors. In the table 3, two instantiations of the « agressive driving » norm are presented :
- the first one respects the norm : all the values belong to the definition set of the norm,
- the second one is a violating one : the maximum speed was instantiated outside the definition set of the norm. (210 km/h > 160 km/h).

q verifies $q < \delta$. So this violating behavior is admissible.

$q = \frac{210-160}{6} = 8,33 < 10\%$

Parameters	aggressive driver	violating aggressive driver
pb1 = maximal speed	vpb1 = 150 km/h	vpb1 = 210 km/h
pb2 = safety time	vpb2 = 0.2 s	vpb2 = 0.3 s
pb3 = overtaking risk	vpb3 = 2.0	vpb3 = 1.8
pb4 = speed limit risk	vpb4 = 1.6	vpb4 = 3.0
pb5 = observe signs	vpb5 = false	vpb5 = true
pb= observe priority	vp= false	vp= f alse

Table 3 – Examples of generation of behaviors

3 Specifications

3.1 Choice of the interactions and of the simulation parameters

We have chosen to limit the simulation to two interactions. Indeed, we prefer to give the priority to the implementation of the generation model and of the graphic interface.

These two interactions are :
- the reproduction : the encounter of two agents of the same species can lead to a reproduction,
- the fight : the encounter of two agents of different species can lead to a fight.

Concerning the choice of the institutional parameters, we have chosen to define four parameters.

This choice leads to a good diversity of behaviors while keeping an acceptable complexity of the interaction model for a demonstration applet.

These four parameters are the following :
- the agressiveness : : it quantifies the chances an agent has to interact with another agent when they meet : the higher it is, the more chances it has to interact. It is tested before the encounter of two agents if the fatigue test is passed (see Chapter 4.5) ;
- the number of children by reproduction : it gives the number of children which are produced during a reproduction (see Chapter 4.5) ;
- the moving speed : it gives the moving speed of the creature in the simulation. It is also used as a parameter in the simulation temporizing. (see Chapter 4.3).
- the size : it is the size of the agent in the simulation but also its strength during a fight with another creature. It is used in the test which gives the winner of the fight between two creatures of different species. (see Chapter 4.5) ;

Each creature also has a "fatigue" characteristic which is tested before each interaction.

3.2 Use scenario

An initial population of creatures belonging to each species is generated and is put randomly on the grid. This population respects the distribution specified by the user.

During the simulation, the individuals are moving. For each encounter between two creatures, two options are possible, according to their respective species :

The aggressiveness of the agent and the fatigue of the two creatures are tested :
- if the creatures belong to the same species : reproduction and generation of new agents belonging to this species.
- if they belong to different species : fight function of the aggressiveness and death of the weakest of the two individuals.

End outcome : users' stop or overpopulation.

3.3 Architecture

The architecture of the application is presented at the figure 1. It was designed to represent the generation model at best. However all the elements were not implemented, particularly the gestion of several probability distributions (we considerate only random draws following uniform distributions)

4 Implementation

4.1 Norm model

You'll find here a basic description of different Java classes used for implementing the generation model. All the methods which are presented in the UML diagram will not be described. For more information, you can see the corresponding JavaDoc.

4.1.1 Institution, norm, behavior

The architecture of the institution, of the norm and of the behavior respect the model which is presented above (Fig. 2, 4 and 6) You can see the part 7 : Annex to find all the UML models of the differents classes representing this elements of this model.

4.1.2 Generation engine

The input of the generation engine is a norm.

1. Firstly, a behavior is created.
2. With a uniform distribution, each parameter is instanciated respecting the constraints of the norm.
3. Then the behavior is completed adding and instantiating the institutional parameters which are not specified in the norm.

This generation is done by the "generateBehavior" method. It takes as argument the norm which is linked to the generated behavior.

4.1.3 Generation of violating behaviors

The input parameters of the generation engine of violating behaviors are a norm and δ (see Definition : 4), which specifies the admissible distance to the norm.

One or several characteristics of the violating behavior can be outside of the definition set specified by the norm.

A norm violation can be quantified on two levels :
– on the number of norm parameters which violate the norm ;
– on the distance to the definition sets.
The calculation of the distance allows to take into consideration these two levels. (see definition 7)

The generation of "violating" behaviors is done by the generateViolatingBehavior(Norm norm, double δ) method.

It uses the algorithm 1. Then the distance to the norm is calculated and compared to δ. If $q_s < \delta$, the behavior is admissible or else it is rejected.

An empty norm is used to generate behaviors directly from the institution. During the generation of behaviors, this norm is completed by all the institutional parameters of the institution. (1).

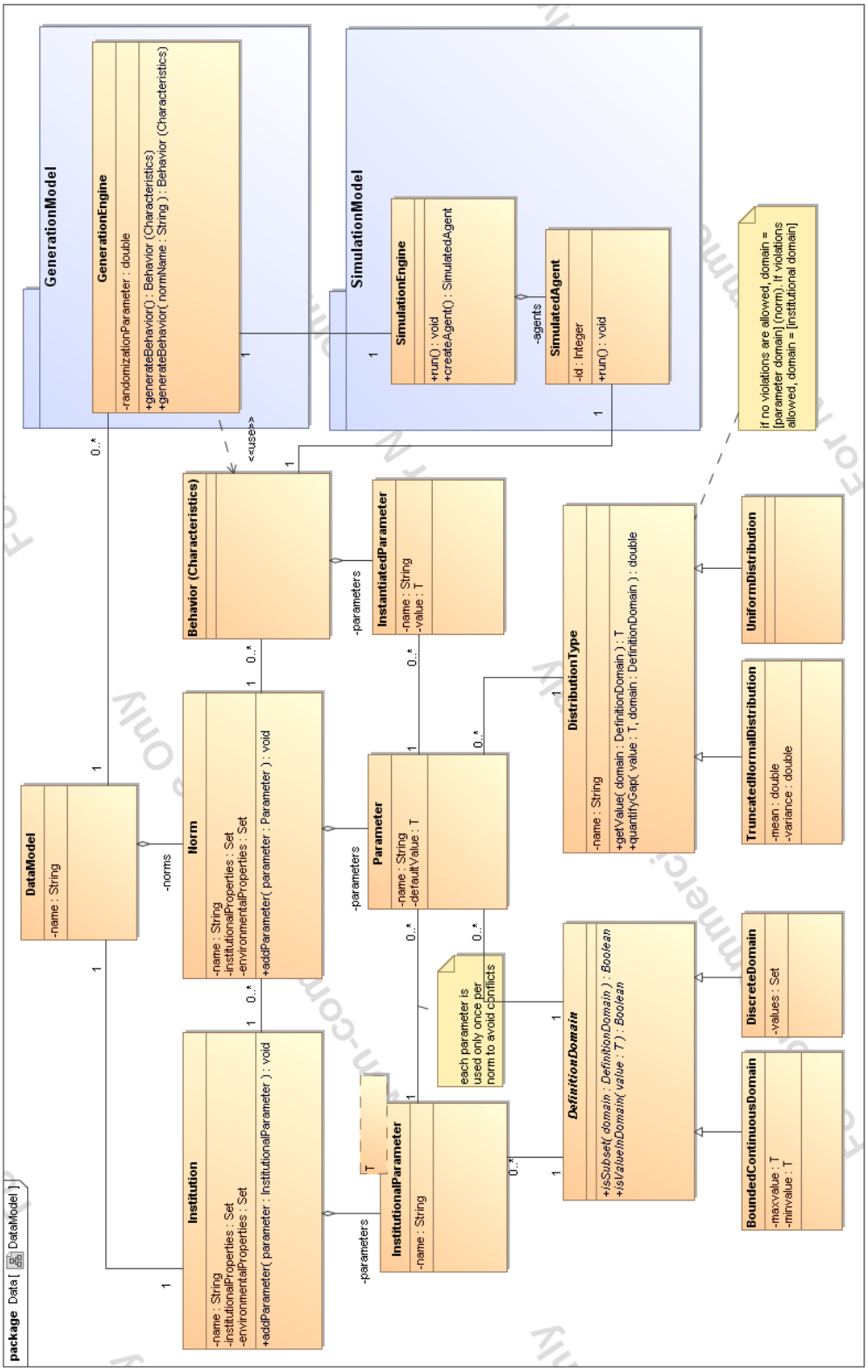

FIGURE 1 – Initial architecture

Algorithm 1 Algorithm of generation of a violating behavior

Require: a norm $N = \{I, l_N, \mathcal{P}_N, \tau_N\}$. σ the global determinism criterion of the institution.

$\alpha \leftarrow$ uniform_random($[0, 1]$)
if $\alpha < \sigma$ **then**
    for all $p \in \mathcal{P}_N$ **do**
        $\beta \leftarrow$ uniform_random($[0, 1]$)
        if $\beta < \tau_N$ **then**
            execute generator of violating behaviors
        else
            execute generation of non-violating behaviors
        end if
    end for
else
    for all $p \in \mathcal{P}_N$ **do**
        execute generate of non-violating behaviors
    end for
end if

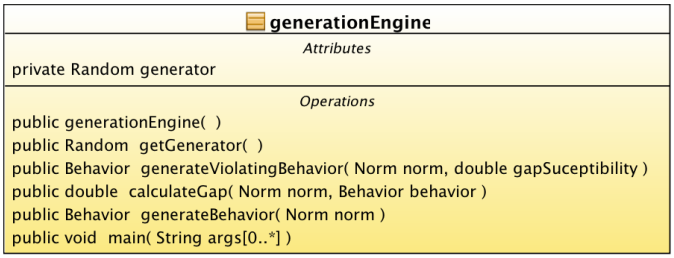

Figure 2 – UML Class : generationEngine

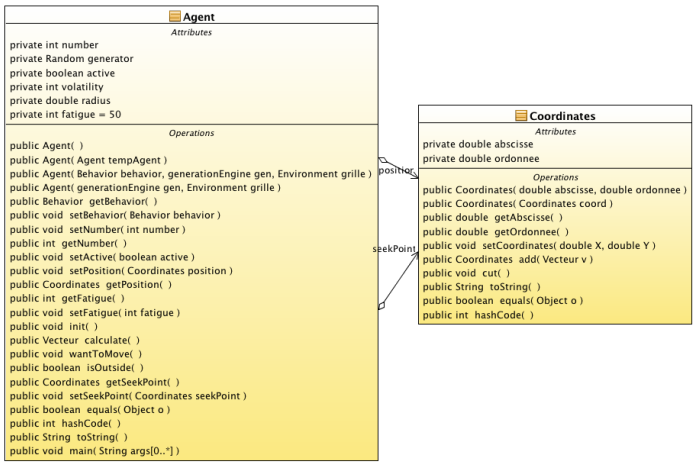

FIGURE 3 – UML Class : Agent

4.2 Simulation model

4.2.1 Agent

An agent has the following main characteristics :
- an unique id,
- a behavior (see Definition : 6) which contains :
  - a reference toward its generation norm (see Definition : 4)
  - the characteristics : size, number of children and moving,
  - a fatigue level,
  - its position,
  - its destination position.

A agent has the following main methods :
- init() which calculate randomly its first point of destination,
- Calculate() which calculate its next point of destination,
- WantToMove() which is the methode that is called after each step of time to "activate" the agent. It contains the algorithm of interaction and movement.

The model of the agent is presented on the figure 3.

4.2.2 Environment

The environment has the following main attributes :
- a hast table with coordinates (in px) as keys and the agents as elements.
- the width and the height of the simulation screen (in px).
- the maximum radius (Rmax) of an agent (in px).
- a simulation speed (in s).

The environment has the following principal methods :
- isFree(Coordinates coord) which checks if a coordinate is free around a radius equals to Rmax.
- GetFreePlace() which returns a free coordinate randomly.
- MoveAgent(Agent agent) which moves an agent and updates its characteristics (fatigue, position...),

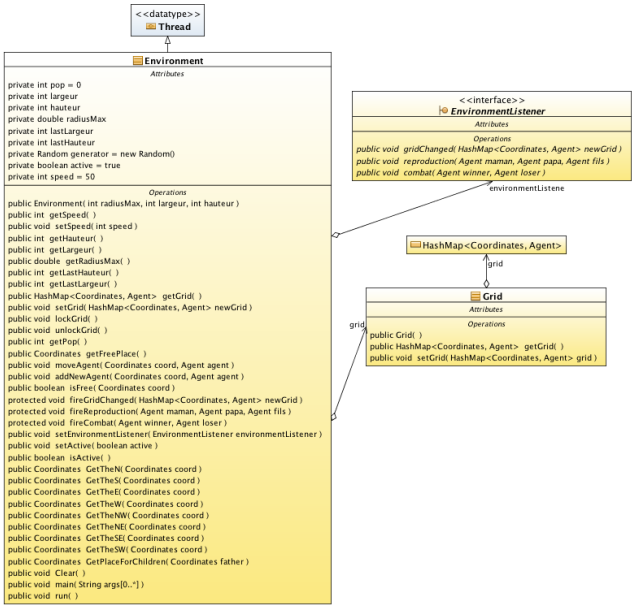

FIGURE 4 – UML Class : Environement

- AddNewAgent(Agent agent) which adds a new agent to the simulation and updates its characteristics.
- GetPlaceForChildren(Coordinate father) which gives free coordinates around the "father" agent where to put its children.
- Run() the thread routine which goes all over the hash table and activate the agents (calling the "WantToMove()" method). The temporizing is done at the end of the covering and its duration determines the simulation speed.

The model of the agent is presented on the figure 4.

4.3 Time management

For an agent, the time between two successive activations is proportional to the number of agents which are present in the environment. Indeed, the time is managed in this way in the simulation :

1. The hash table containing the agent is covered in the natural order and they are activated one by one.
2. The temporizing is done at the end of the hash table covering : the thread pauses with a sleep. The duration of a sleep is equals to the "Speed" variable of the "Environment" class that the user can specify. (see Figure 4).

For a better illustration of the time management, you can see the figure 5.

With this method of movement, the time step is not fixed. Indeed, the time step is equals to the temporizing (set by the "speed" variable) and the covering time of the hash table which is according to the total number of agents on the environment.

4.4 Moving model

We wanted a moving model which gives a realistic and fluid rendering. In this simulation, there are no other constraints or obstacles than the other agents and the window edge. Two different methods were tested :

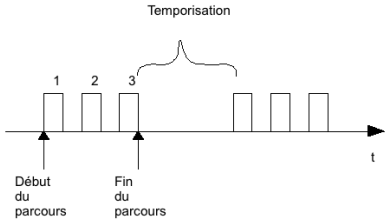

FIGURE 5 – Temporizing

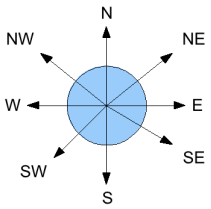

FIGURE 6 – Moving method 1

4.4.1 First method

The first method is the simpliest and natural method :

A step is defined.

Each agent, during its activation, can move in one of the eight directions of the space : N,NE,NW,SE,SW,S,E,W. (see Figure 6) Before each movement, the agent pulls randomly a direction following a uniform distribution and move of one step in this direction.

However, this method gives a bad graphic rendering to the simulation : each agent seems to stay around its initial position. This can be explained by the use of an uniform distribution which gives the same importance to each direction. Moreover, the change of direction can be really brutal (brutal change from N to S). The moving step is imposed and fixed, what contribuate towards the bad rendering.

4.4.2 Second method

We had to find a moving model which would be more adapted to our needs. Therefore, we used the model which is called « Wander Steering Behaviors » taken from the works of Craig Reynolds, « Steering Behaviors For Autonomous Characters » . This method gives realistic agent movements in a simple way.

Here is its principle (see Figure 7) :

Choosing the V norm and the C radius (see Algorithm) allow to quantify the importance of the change of direction during each activation. It is also easy to manage the edge of the window taking the opposition of the direction vector when the agent goes on a edge. This creates the impression that the agent bounces.

A fluid movement is obtained with the new model. Moreover, all the drawbacks of the first model are eliminated : the movement seems more natural.

The moving step of each agent is specified by the norm. It is different between the agents. This value of "moving step" is also used to specify the radius of the C circle.

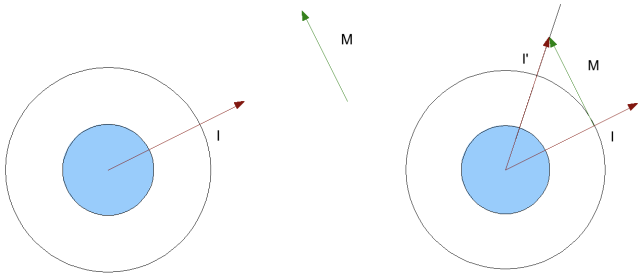

FIGURE 7 – Moving method 2

```
/**
     * This method calculates the next seekpoint of the agent and return
     * the direction vector.
     * Before the first mouvement, the seekpoint is equals to the position of the agent.
     * @return Direction vector.
     */
    public Vecteur calculate() {
        //A random vector is generated.
        Vecteur vect = new Vecteur(generator.nextInt(), generator.nextInt());

        //Add more random
        if (generator.nextInt() > 0.5) {
            vect.setX(vect.getX() * (-1));

        }
        if (generator.nextInt() > 0.5) {
            vect.setY(vect.getY() * (-1));

        //Scale the vector to respect the volatility parameter
        }
        if (vect.length() > volatility) {
            vect.normalize();
            vect.scale(volatility);
        }

        //Calculate the new direction vector.
        seekPoint = seekPoint.add(vect);
        Vecteur temp = new Vecteur(seekPoint.getAbscisse() - position.getAbscisse(), seekPo
        temp.normalize();
        temp.scale(radius);

        //Calculate the new seekPoint
        seekPoint = position.add(temp);

        //Verification if the seekPoint is inside the grid else the direction vector is
        //changed to the opposite.
        if (isOutside()) {

            temp.opposite();
            seekPoint = position.add(temp);
        }

        return temp;
    }
```

FIGURE 8 – Java method implementing the moving method 2

Algorithm 2 Algorithm for the movement : method 2

1. Concerning the first move of the agent :
   (a) A C circle of R radius is drawn around the agent.
   (b) A D vector which give the first direction of the agent is generated randomly.
   (c) The agent is moved in this direction one p step long.
2. Concerning the next moves :
   (a) The I interaction point between the C circle and the half straigth line of origin : O, the middle of the agent and D as director vector is spotted.
   (b) A M random vector of V norm is generated.
   (c) The I point is translated by the M vector : A new I' point is spotted.
   (d) The OI' vector become the new director vector.
   (e) The agent is moved in this direction one p step long.

In this way, an agent with a hight moving step more distance than an agent with a little moving step. This give the impression that the hight step agent is moving faster than the small step agent.

The variance of the direction is set to 10 px (see V Norm in the algo. 2).

The maximum radius of an agent is specified in the environement. It is set to 20 px.

You can see in what way this method of moving is implemented in Java on the figure 8.

4.4.3 Detection of other agents

In the environment, a R_{max} maximum radius of an agent is specified. This radius will be the reference for the detection of other agents during the deplacement.

An agent calculates its new destination point thanks to the algorithm (see Figure 7). Before moving to this point, it verifies if there are not other agents at a distance equals to R_{max} around this point

This is the way in which it is tested :

Thanks to the (x_0, y_0) coordinates of the destination point and the R_{max} radius, the equation of the corresponding disk is determined : $(x - x_0)^2 + (y - y_0)^2 \leq R_{max}^2$.

The grid is checked in the natural order. It is verified that no agent has coordinates which verifies this inequation. If this is true, the agent moves to its destination point else there is interaction.

Remarque : That is why the moving step must be superior to two times R_{max} else it could interact between itself.

4.4.4 The limits of this moving method

The problem of this moving method is that the agent can cross each other without interacting (see Figure 9). Indeed, if a moving step is enough hight (that is to say superior to the diameter of an agent), a agent can « jump » above an other agent without interacting with it.

4.5 Interactions

When an agent wanting to move finds other agents around its destination point, an interaction with the last agent which has been tested in the environment can happen :

– If this agent belongs to the same species, they try to reproduce.

– If the agent belongs to a different species, they try to fight.

When two agents of the same species are meeting, their level of fatigue are tested :

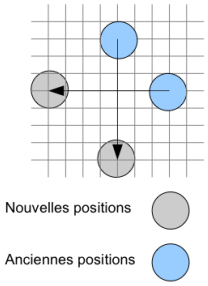

FIGURE 9 – Problem of this method

- If they are not nul, there is no interaction. The agent who wanted to move, go to the opposite direction than the calculate direction.
- If they are nul, the agent who wanted to move tests its agressiveness in this way :
  - A number between 0 and 1 is randomly drawn following a uniform distribution.
  - If this number is inferior to the agressiveness of the agent who wanted to move, interaction happens else the agent goes to the opposite direction without interacting.

4.5.1 Fight interaction

When two agents, which belong to the same species meet, if the fatigue test and agressiveness test are true, they are fighting each other :

- The taller agent wins the fight. Its fatigue is set to 20. The agent which lost is removed from the environment.
- If they have the same size, the agent which wanted to move wins.

4.5.2 Reproduction interaction

When two agents which belong to different species meet, if the fatigue test and agressiveness test are true, they reproduce :

- The number of children is given by the agent which initiated the movement. Its fatigue and its partner's one is set to 20.
- These children are generated thanks to the norm of the parents and are put around them.

To find where to place the children, we use a algorithm similar to the first moving method which was described above : All the points situated at 2*Rmax in the 8 space directions are checked to find if they are free. If all are not free, we look for a new free place around the last checked point. (et ainsi de suite)

When an agent is born, its fatigue is equals to 50.

4.5.3 Fatigue management

When an agent is add to the environment, its initial fatigue is equals to 50. Then, when it moves or interacts, its fatigue is decreased by one point.

5 Results

5.1 Presentation of the HIM

The HMI is composed of three tabs :

1. The Simulation Panel (See Fig. 10 and Chap. 5.1.1)

2. The Species Description tab (See Fig. 11 and Chap. 5.1.2)
3. The Creature Edition tab(See Fig. 12 and Chap. 5.1.3)

After the launching of the applet, three norms are generated by default (species1, species2, species3) and five agents belonging to each norm are put on the simulation panel. In this way, a user can test immedialtelely the application without configuring species.

5.1.1 Simulation Panel Tab

It is the main tab of the application (see 10) where the user can see the agents move and interact.

The « Simulation control » subset is used to launch the simulation (Start button), to pause it (Pause button) or to remove all the agents of the simulation (Clear Button). The user also can ajust the simulation speed.(see Chap.4.3)

The « Generate Population » subset is used to replace the present population of agents or to add a new population of X agents (X can be specified by the field : « Total number agent » . The repartition between the norms (Proportion of each species) and the violating rate (Percentage of violations) for each population also can be chosen.

The « Manually add an agent » subset is used to add manually an agent. The user selects a generation method : or . he also schooses if he wants that the generated agent has a violating behavior or not. Pressing the « Generate » button, an agent is generated. Before adding it to the simulation with the « Add agent in the simulation » button, it is possible to modify manually its characteristics, which are restrained either by the institution or by the chosen norm.

The « Statistics » subset gives in real time the number of agents of each norm and the number of violating agents.

The textbox under the simulation panel indicates the differents interactions between the agents during the simulation.

5.1.2 Species Description Tab

This tab is used to configure the norms, to add one or to remove an other.(see Figure 11)

To modify a norm, the user selects the norm he wants to in the combo box then he modifies the name or the bounds of the different norm parameters. To add a norm, press the « Add species » button. A new norm of the name : « NewSpecies » is added to the combo box. you can modify it as specified above. To remove a norm, select it in the combo box, then press the « Remove species » button.

All the modifications, adding or removing norms lead to the removing of all the agents on the panel simulation.

5.1.3 Creature Edition tab

This tab is used to show the characteristics of each agent which are on the simulation panel.(see Figure 12)

The « Population » subset displays a tree which organizes the agents by species. Clicking on an agent, its characteristics appears in the « Individual characteristics » subset.

They can be modified individualy by the user.

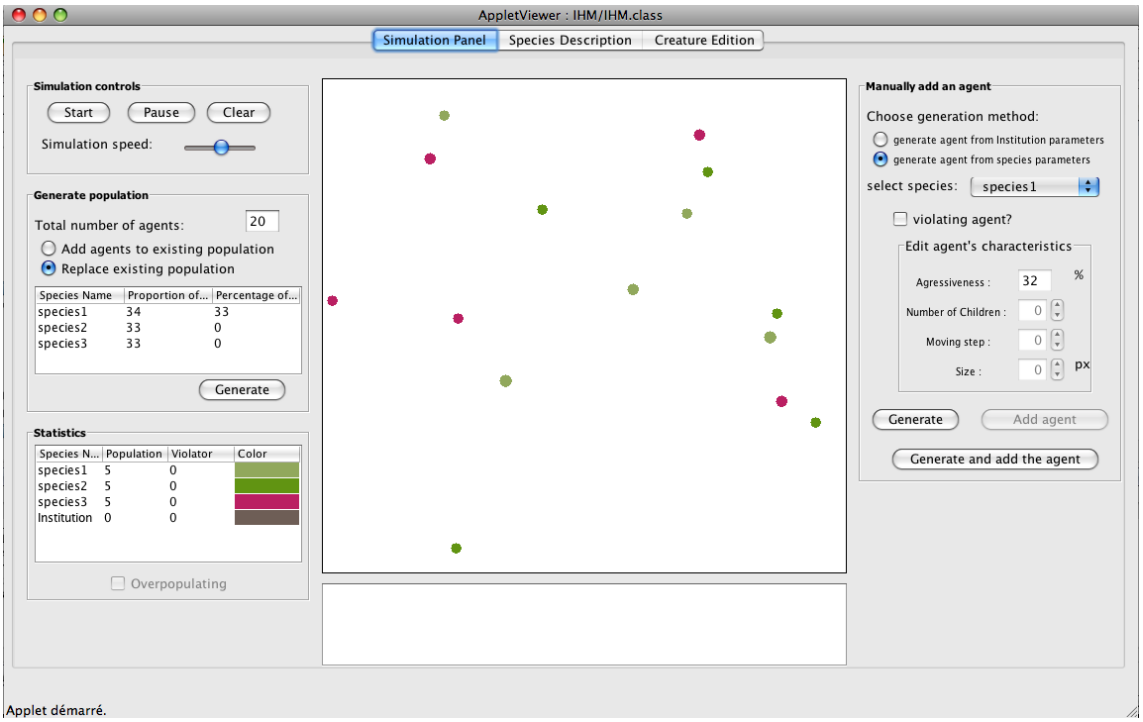

FIGURE 10 – Simulation Panel

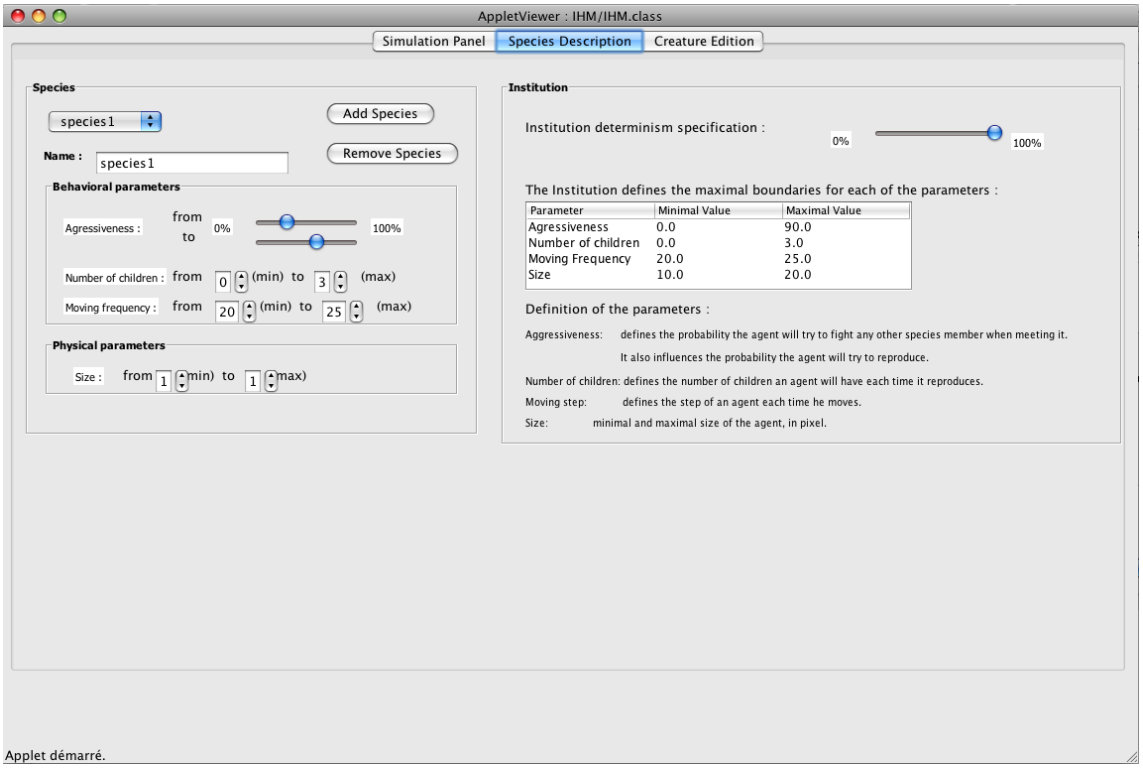

FIGURE 11 – Species Description

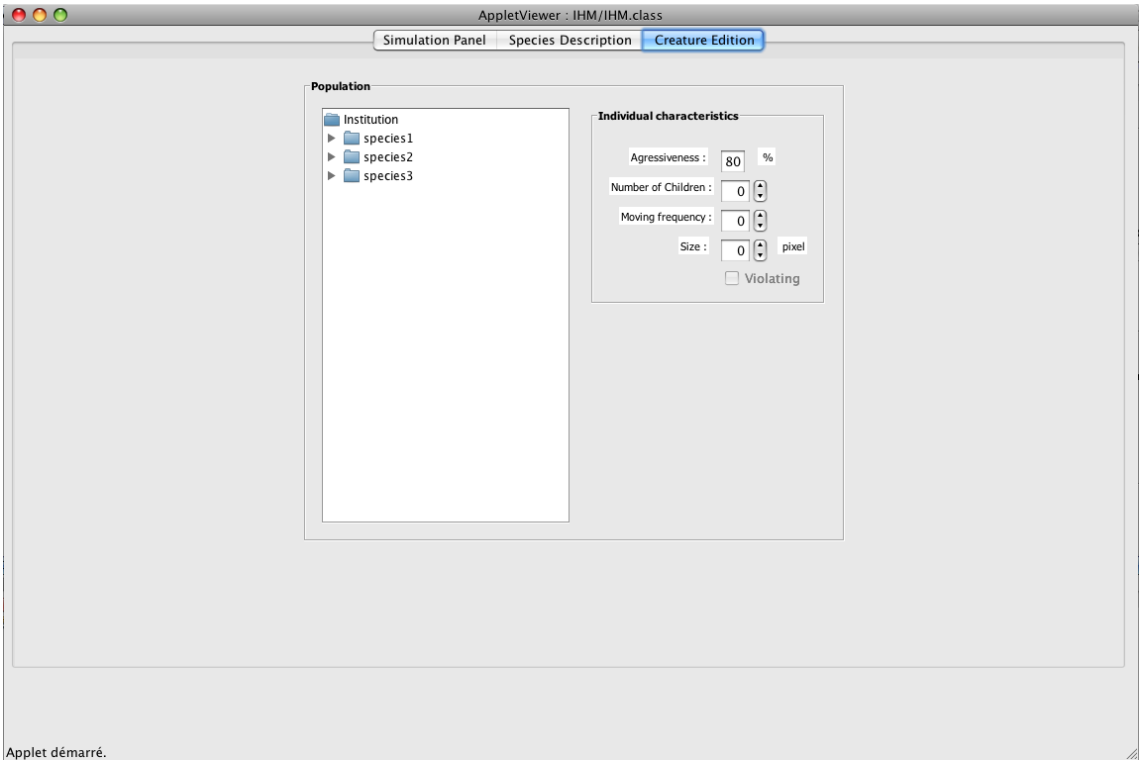

FIGURE 12 – Creature Edition

5.2 Use

5.2.1 Launch a simulation

1. Launch the applet : The « Simulation Panel » appears.
2. Press the « Start » button : The agents begins to move and to interact. You can see the detail of these interactions in the text box under the simulation panel and the evolution of the population in the « Statistics » subset.
3. Press the « Pause » button : the simulation pauses.

5.2.2 Generate/add a population

1. In the « Generate Population » subset , set a number of agents inferior to 100 (else overpopulation) in the « Total number agents » field : then modify the « Proportion of each species » field (the total must be 100) and the « Percentage of violations » field.
2. Select « Replace existing population ».
3. Press the « Generate » button : a new population which respects the proportions that you have set is added to the simulation panel.
4. Press the « Start » button : the simulation resumes.
5. Press the « Pause » button.
6. Select « add agent to existing population » and go to the step 4 again.
7. Press the « Generate » button : a new population which respects the proportions that you have set is added to the existing population.

5.2.3 Check the population

1. Go into the « Creature Edition » tab. You can see nodes with the names of the species of the simulation in the tree.

2. Expand one of these nodes : The list of the agent belonging to this species appears.
3. Select one of these agents : in the "Individual characteristics" subset , its characteristics are shown.
4. Modify one or many of these characteristics : The agent and its characteristics are automatically updated into the simulation.

5.2.4 Add manualy an agent in the simulation

1. Go back to the « Simulation Panel ».
2. In the « Add manually an agent » subset , select « generate agent from species parameters » and a species in the combo box.
3. Press the « Generate » button. In the « Edit agent's characteristics » subset : characteristics respecting the norm are generated.
4. You can modify individually each of these characteristics which are restrained by the norm.
5. Press « Add agent in the simulation » : an agent is added to the simulation panel.
6. Go back to the « Creature edition » tab : the agent you have just added is present in the tree under the node linked to the selected norm.

5.2.5 Modify/Add a norm

1. Go to the « Species Description » tab.
2. Select a species in the combo box : the bounds of the definition sets of the different norm parameters are updated.
3. Modify the name of the species and the different bounds.
4. Press the « Add species » button : a norm is added to the combo box ans is usable in other tabs.
5. Modify this norm.
6. Select an other norm and press the button « remove norm » : the norm is removed form the combo box and is not accessible by other tabs.
7. Go back to the « Simulation Panel ».

5.2.6 Add an agent with a behavior which violates the norm.

1. Select a norm in the combo box.
2. Tick « violating agent ».
3. Press the « Generate » button : in the "Edit agent's characteristics" subset, characteristics which can violate the norm are generated.
4. Modify these characteristics : they are not restrained by the norm but by the institution.
5. Go to the « Creature Edition » tab : the agent you have just added is present in the chosen species and is shown as « violating ».
6. Go back to the « Simulation Panel ».

5.2.7 Add an agent from the institution

1. Select « generate agent from institution parameters » from the « Manually add an agent » subset.
2. Press the « Generate » button then press the « Add agent in the simulation » one : an agent which doesn't belong to any norm is added to the simulation.
3. Go to the « Creature Edition » tab : this last agent appears under the « Institution » node.

You can simulate a huge diversity of species and agents thanks to all these functionalities.

5.3 Outlook

The aims of the project has been completed.

However, improvements can be considered, for example :
– Make the time management independant of the number of the agent of the simulation.
– Improve the moving method to avoid that the agent can cross each other without interacting.

The main scenario stays very simple. Designing a more complex scenario could be a good thing to illustrate the generation model in a better way. We can imagine adding new interactions between agents or a new reproduction system which takes into consideration the characteristics of the two parents. Some variables of the program as direction change speed (see chap. 4.4) or the initial fatigue of each agent can't be set by the user. It could be interesting to make them customizable to improve the quality of the simulation.

6 Conclusions

So, the aim of the project was to develop an applet to illustrate an innovative generation model of behavior for virtual agents.

With this project, I could discover the field of multi-agents systems, a field I find very interesting because of its transversal side and its high technical nature.

To achieve the goals of the project, I had to conduct an applet development project from the specifications to the implementations. I learnt a lot in term of programming and multi-agents systems topics.

7 Appends

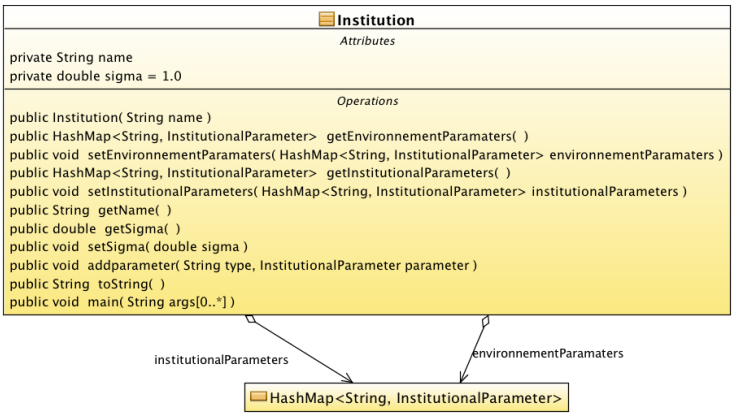

FIGURE 13 – UML Class : Institution

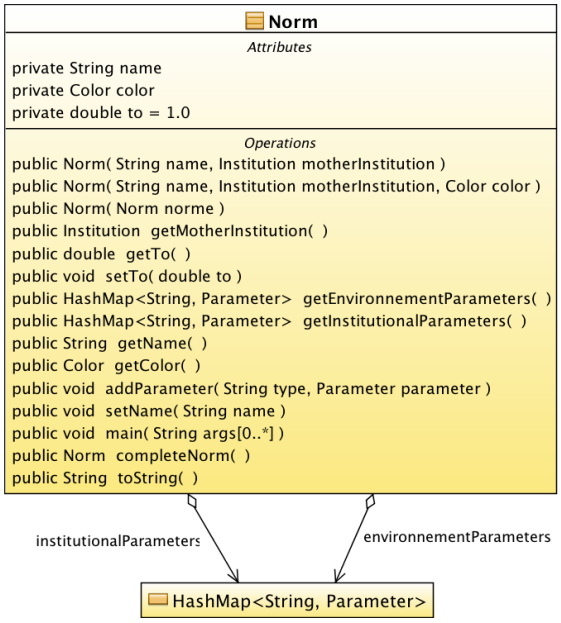

FIGURE 14 – UML Class : Norme

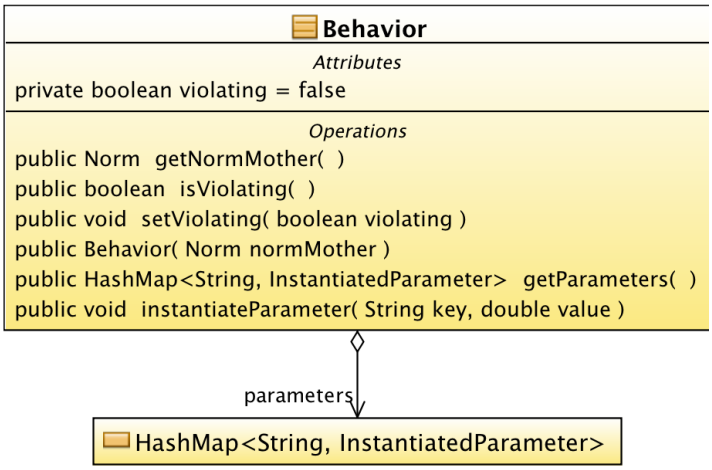

FIGURE 15 – UML Class : Behavior

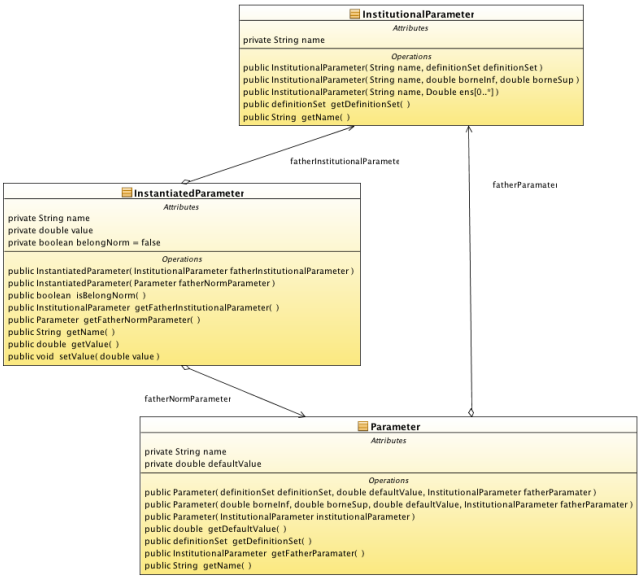

FIGURE 16 – UML Class : InstitutionalParameter, Parameter, InstantiatedParameter