Universitat de Barcelona
Departament de Matemàtica Aplicada i Anàlisi

Dealing with norms within the Packet World Multi-Agent Simulator

by
Patricio Emanuel Petruzzi

Interuniversitary Master in Artificial Intelligence
(UPC-UB-URV)

Master's Thesis

Advisors: Dr. Maite López Sánchez,
Dr. Marc Esteva and Dr. Alexander Artikis

September 2010

Contents

List of Figures

1 Introduction

Norms are expectations of an agent about the behavior of other agents in the society. The human society follows norms such as wait in queues and giving gifts at birthdays. Norms are of interest to researchers because they help to improve the predictability of the society.

Multi Agent Systems (MAS) are formed by a set of agents that interact in an environment to engage goals that can be individuals or collective. In these systems norms improve cooperation and coordination between agents and reduce the computation time required by agents to the decision-making process. Also limits the search space since agents shouldn't perform actions that violate the norms of the running environment.

Usually it is complicated to define norms due to the complexity of the systems and dynamicity of the environment. In Open MAS, agents (developed by different companies) compete against each other and it is hard to ensure that all them will behave as expected, this difficult the decision of which norms should be set. Therefore, tools to test and evaluate agent's behavior using norms are needed and a way of doing it is through simulated environments.

The purpose of this thesis is to extend the functionalities of the Packet World simulator by implementing a test bed where norms could be easily added and modified in order to evaluate the different outcomes of these norms. With the Normative Packet World developed, some norms where

proposed and evaluated.

The layout of this work follows the following structure; in section 2 we give a brief overview of the Packet Work architecture, in section 3 we explain the proposed extension of the system, in section 4 we present the laws already implemented in the Packet World and the institutional norms and norms added, in section 5 we detail the agent's normative behavior developed to test the norms, in section 6 the results of different experiments are explained and in section 7 the conclusions of this work are presented.

1.1 Situated Multi-Agents Systems.

A Situated MAS is a computing system composed of an environment populated with a set of localized agents that cooperate to solve a complex problem in a decentralized way. Situated agents are entities that encapsulate their own behavior and maintain their own state. They have local access to the environment, i.e. each agent is placed in a local context, which it can perceive and in which it can act and interact with other agents.

A situated agent does not use long-term planning to decide what action sequence should be executed, but instead it selects actions on the basis of its position, the state of the world it perceives and limited internal state. In other words, situated agents act in the present, "here" and "now". Intelligence in a situated MAS originates from the interactions between the agents, rather than from their individual capabilities.

1.2 Packet World

The Packet World is a simulator for situated agents. The environment is composed of a two dimensional grid with packets and destinations. In this

domain robots (agents) must move the packets to the correct destination. The goal of this application is to investigate under which conditions the robots will develop social conventions and how the robots can take advantage of the information communicated to each other.

1.2.1 Basic Setup

The basic setup of the Packet-World consists of a number of differently colored packets that are scattered over a rectangular grid. Agents in this virtual world have to collect these packets and bring them to the correspondingly colored destination. The task of the agents to deliver all packets in the world is called a *Job*.

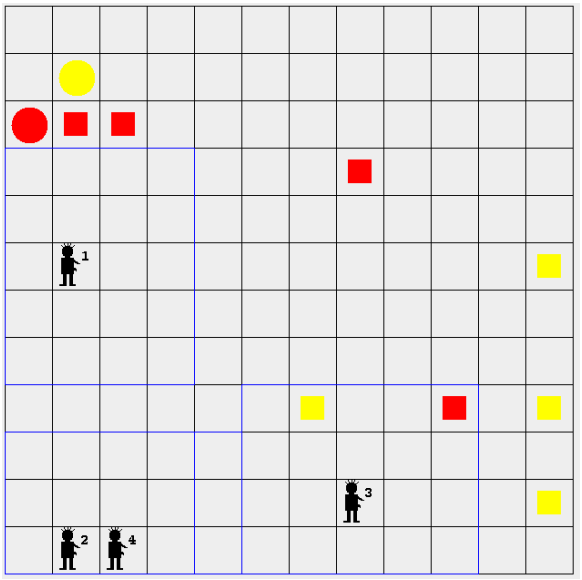


Figure 1: Example of Packet World

Figure 1 shows an example of a Packet-World of size 12x12 with 4 agents. Colored rectangles symbolize packets that can be manipulated by the agents and circles represent destinations.

In the Packet-World, agents can interact with the environment in several ways. Agents are allowed to make one step at a time to a free neighboring cell. If an agent is not carrying any packet, it can pick up a packet from one of its neighboring cells. An agent can put down a packet at one of the free neighboring cells or, of course, at that particular packet's destination point. If there is no sensible action for an agent to perform, it may wait for a while and do nothing.

Besides acting in the environment, agents can also send messages to each other. More concretely agents can ask each other to set up collaborations.

It is important to notice that each agent of the Packet-World has only a limited view on the world. The view-range of the world expresses how far, i.e. how many squares; an agent can perceive.

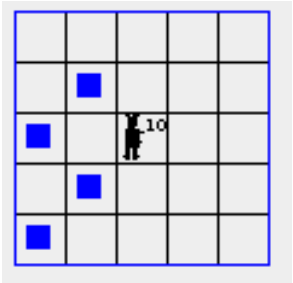


Figure 2: Local view of an agent

Figure 2 illustrates the limited view of agent 10; in this example the view-range is 2. The goal of the agents is to perform their job efficiently, i.e. with a minimum number of steps, packet manipulations and message exchanges.

The Packet-World can be monitored via two counters that measure the efficiency of the agents in performing their job.

The first counter measures the energy consumed by the agents. For instance, stepping with or without a packet, picking up a packet or putting

it down, even communicating messages, are actions with energy cost. As a default, when an agent makes a step without carrying any packet, it is consuming one unit of energy and stepping with a packet requires two units of energy. The energy required to pick up a packet or to put it down is also one unit. However, waiting and doing nothing is free of charge.

The second counter measures the number of messages sent. By default, this counter simply increments for each message that is transferred between two agents. The overall performance can thus be calculated as a weighted sum of all energy-consuming activities.

2 Architecture of the Packet World

In this chapter is presented the most relevant aspects of the current architecture of the Packet World which is based on D. Weyns', A. Helleboogh's and T. Holvoet's research on Situated Multi Agent Systems[1] and authors of the Packet World Simulator.

2.1 Overview

Packet World uses a model for action based on Ferber's theory of influences and reactions[2]. According to this theory, agents modify the environment through influences and the new state of the world is the result of the combination of all the agents' influences.

The architecture is formed by three primary abstractions: agents, ongoing activities and the environment (Figure 3).

First of all it is necessary to describe the agent architecture, the $Perception_i$ maps the local state of the environment onto a percept for the agent, with this percept the $KnowledgeIntegration_i$ updates the current agent's knowledge and the $Decision_i$ selects the action to perform. The $Communication_i$ process the incoming and produces outgoing messages according to the communication protocols.

Secondly, the ongoing activities take care of other processes that may produce activity in the system. They produce influences in the environ-

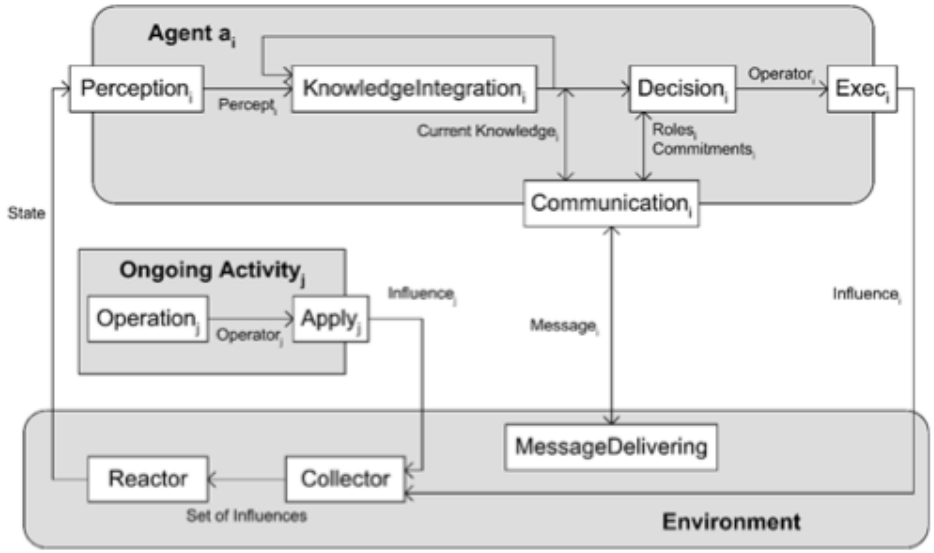


Figure 3: Overview of the Architecture

ment depending on the current state of world. Moving objects, evaporating pheromones or temperature are some examples of ongoing activities.

In the third abstraction, Environment, the *MessageDelivering* handles the transport of messages between agents. The *Collector* accumulates all the influences from the agents and the ongoing activities and passes them to the *Reactor* as a set. The *Reactor* generates the changes in the environment applying the received influence set according to a set of specific laws.

2.2 Environment

In the Packet World the environment has a grid structure and has been modeled as a collection of grid layers [3] (see figure 4) where each layer hosts a different kind of item, for example a packet-layer, an agent-layer, a flag-layer, etc. The state of the world can be seen as a combination of all layers and can be perceived by the agents. Constraints can be defined on each layer and also between layers, i. e. an agent cannot step where other

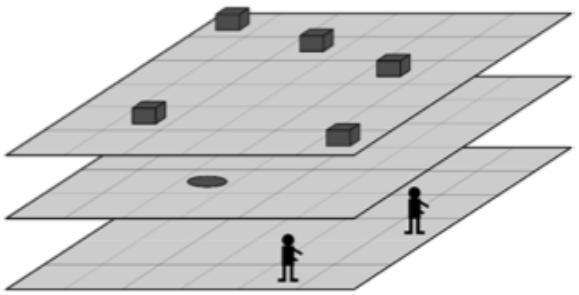


Figure 4: Environment structure decomposed in layers

agent is, neither step where a destination is located but it can step over a flag.

The advantages of the layered structure of the environment are the extensibility and re-usability to add new layers when needed and the simplification to generate the agent's perception and calculation of the effects of action. Agents are able to percept the environment selectively and only some layers are needed to generate the perception. Also when agents act the constraints defined by the laws can be defined for each particular layer.

2.3 Influences

Agents and ongoing activities modify the environment through influences and the new state of the world is the result of the combination of all this influences. They affect only one layer of the environment and it has to be specified at the influence.

```
public class InfStep extends Influence implements AnInfluence {

    public InfStep(Environment environment, int x, int y, int agent) {
        super(environment, x, y, agent, null);
    }


    public World getAreaOfEffect() {
        return getEnvironment().getWorld("AgentWorld");
    }

}
```

Figure 5: Influence InfStep

Figure 5 shows an example of the influence that agents pass to the environment to step, the *InfStep*. The environment, the X and Y coordinates of the step and the agent involved are defined. The layer at the environment "AgentWorld" is where this influence will be applied.

As it can be seen, the influence only communicates the intention, if the *Reactor* validates the influence according to a set of specific laws, it will notify the responsible layer at the environment to apply the action.

2.4 Agent's Decision Process

Agent's action selection in the Packet World is based on free-flow trees [6]. Existing free-flow trees are designed from the perspective of individual agents; they don't support explicit social behavior. Packet World uses an extended version of free-flow trees with concepts of a role and a situated commitment to enable an explicit social behavior [7]. Figure 6 shows a simplified action selection model for an agent.

A role is defined as a sub-tree in the hierarchy; the root node designates the top node of the role. All roles of the agent are constantly active and help to the decision making process. In the example, there are two roles

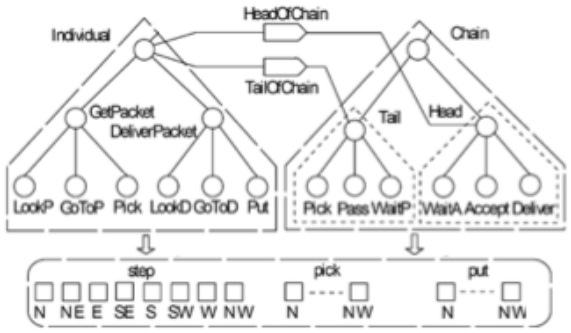


Figure 6: Action selection model for an agent

separated by dotted triangles. In the first role, Individual, the agent search for packets and carry them to the destination. The *Chain* role is composed by two different sub-roles, *Head* and *Tail*. Both roles are used when the agents collaborate in order to pass the packets in a chain to the destination.

Basic actions as *step*, *pick* or *put* a packet are defined independently of the roles and can be reutilized in different situations. For example, at the *Chain* sub-role *Tail* the agent can pick up a packet for passing it through the chain and other agent at the *GetPacket* sub-role of *Individual* can pick up a packet to deliver it by itself.

Situated commitments define relationships between roles. A *goal role* is the objective of the commitment and can be defined form a set of *source roles*.This enables agents to set up mutual commitments in collaboration. The situated commitment *HeadOfChain* in the example connects the single source role Individual with the goal role *Head*. One agent can be involved in different situated commitments at the same time. The activity received from different situated commitments and the regular activity received from the stimuli of the agent is combined to one result.

Traditional approaches of commitment force agents to communicate each other explicitly when the conditions for the cooperation do not apply any-

more. For a situated commitment it is typically the local context (in which the involved agents are placed) that regulates the duration of the commitment. This approach fits the general principles of situatedness and robustness of situated multi-agent systems.

2.5 Protocol-Based Communication

Communication in multi-agent systems is traditionally based on speech act theory [8]. This theory treats communication as actions and these communicative acts are considered in isolation. In practice, however, speech acts are mostly part of series of logically related communicative acts. Communication protocols emphasize the relationship between the exchanged messages in communicative interactions.

A communication protocol has been defined as a set of protocol steps [9]. A protocol step is a tuple (conditions,effects), with conditions a set of boolean expressions that determine whether the protocol step is applicable. Conditions are based on received messages, the agent's available roles, and the agent's state (i.e. its current knowledge and the status of its situated commitments). Effects are the results of the application of the protocol step, i.e. the composition of a new message and/or the modification of the agent's state. A communicative interaction (conversation) is initiated by the initial step of a communication protocol. At each stage in the conversation there is a limited set of possible protocol steps. Terminal states determine the end of a conversation.

2.6 Stigmergy and Flags

The concept of stigmergy was first introduced by P. Grassé [10] to describe the indirect communication among individuals in social insect colonies. In the context of MAS, stigmergy is applied as various forms of indirect communication by means of markers in the environment. Stigmergy enables agents to influence each others behavior indirectly through manipulation of the state of the environment, while in learning approaches agents modify their own internal state based on feedback received from the environment[11].

Packet World supports different kinds of environmental markers but the one relevant to this work is flags. The use of flags was studied as a means to solve the "sparse world" problem, which arises when only a couple of packets are left. The behavior of the agents then becomes inefficient. Most agents keep searching aimlessly for packets.

When several agents detect one of the few packets remaining, all of them run towards it, while in the end only one of them is able to pick it up. To cope with the sparse world problem, agents can use flags to mark a part of the world in which no more packets are present. By placing flags, the agents divide the world in two zones: a marked and an unmarked zone. Agents avoid the marked zone, and only consider the unmarked one for further exploration. The agents' behavior for placing flags had to satisfy two requirements.

First, the destinations must always be part of the unmarked zone. Otherwise, the agents would forever try to search for the destination at the unmarked zone, even if all packets were collected.

Second, and for the same reason, there can only be one unmarked zone: all cells belonging to the unmarked zone must be connected.

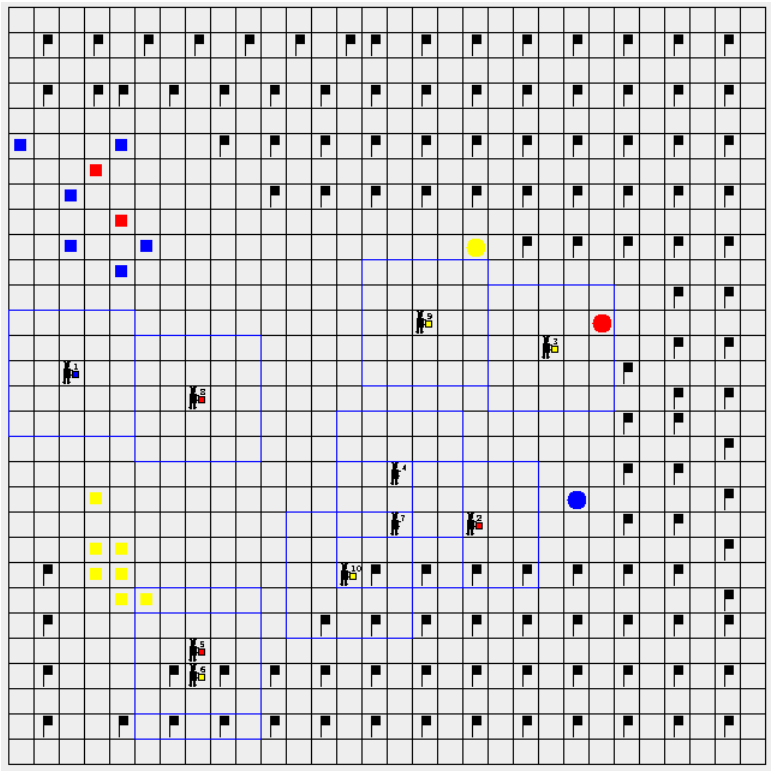


Figure 7: Simulation using Black Flags.

Figure 7 shows and example of the flag behavior using 10 agents. Zones without packets are marked with black flags; the destinations and the remaining packets are part of the unmarked zone.

3 Extension

To understand the objective of the Normative Packet-World is important to point what a Normative Multi-Agent System is. Boella[5] gives the following definition:

"A normative multi-agent system is a multi-agent system together with normative systems in which agents on the one hand can decide whether to follow the explicitly represented norms, and on the other the normative systems specify how and in which extent the agents can modify the norms."

In this work we will focus on the first part of this definition. Agents should be aware of the normative system and decide their actions based on the outcomes of following (or not) these norms and their interests.

To archive this a *Normative System* will be developed and integrated to the Packet-World core. It will be responsible of loading the norms and apply them when corresponding. Also agents should be permitted to ask for the current norms and evaluate the outcomes of their actions.

To test the normative system agents will be gifted with some initial points and through norms agents will get or loose points depending on their actions. This will stimulate agents to become selfish and they will collaborate only when they benefit. This benefit will be calculated using a utility function and their objective will be to get the maximum points

during the simulation.

3.1 Normative System

Based on the Packet World architecture explained in chapter 2 a new architecture is proposed (Fig 8). The *Normative System* will be located at the environment and will load all the norms to the system. The agent's *Decision* module will be able to query, using the *Ask* module, for a specified norm or the outcome of executing an action to the normative system. With this information an agent can decide which action to perform depending on the norms that apply in a particular context. Once the agent decides which action to perform the influence will follow the normal course until the *Reactor*. If the action is validated through the laws, the *Reactor* will communicate the influence to the *Normative System* and it will apply the norms that are applicable to that particular influence.

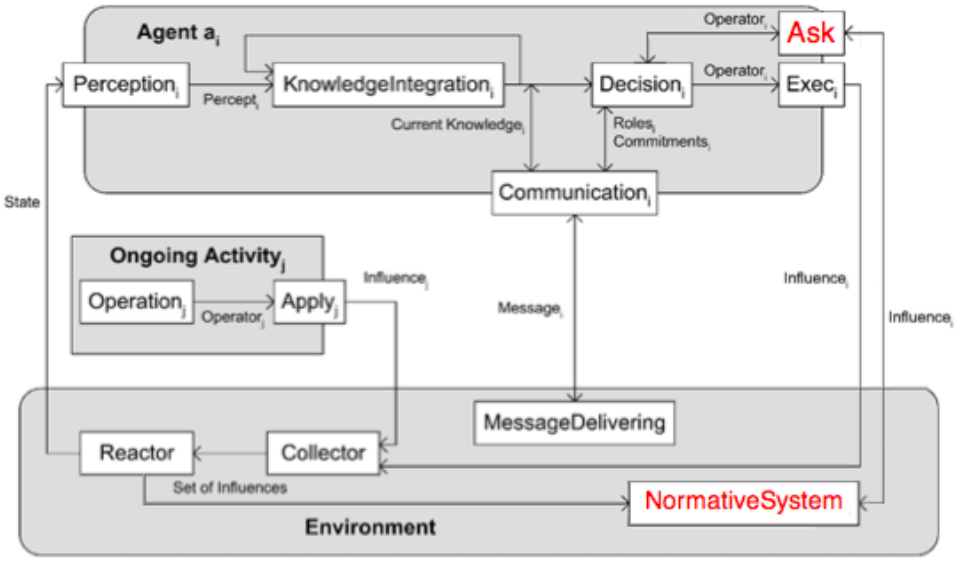


Figure 8: Overview of the Proposed Architecture.

3.2 Points Control

The Multi-Agent System must count the points that an agent has during the simulation. Agents will get or lose points depending of the actions they perform and it will condition future actions. This module will be also accessible for agents allowing them to know the number of points they have.

For example, an agent can decide to pass a packet instead of going to the destination and deliver it based on how many points it has left.

Because the simplicity of this functionality the environment will be extended to support this features.

3.3 Color flags

Currently agents in the Packet World use flags to mark a part of the world where no more packets are present. This helps to reduce the zone where agents search for packets, making them more efficient.

Flags, as an indirect method of communication, can be used to mark other things at the environment. In our Normative Packet World, we will add color flags to communicate the path until a destination. Agents will be able to put a flag when they see a destination or other flag at their perception's corner (see figure 9).

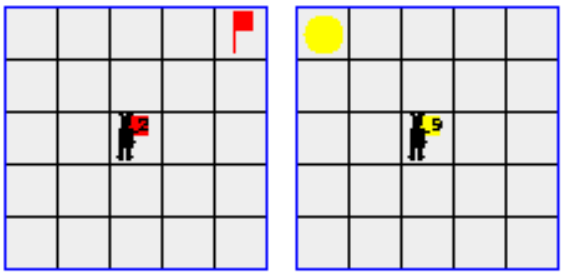


Figure 9: Color flags position.

Using color flags, agents can communicate a destination's path to other agents. When agents see a path made by color flags they must know in which direction they should go, otherwise instead approaching to the destination they will be moving away at every step. To solve this issue, a value at each flag will be added to tell the number of flags left to arrive to the destination. For example, a flag placed in a cell near enough to let the agent see the destination, will have a value of one, the following flag to the first flag will have a value of two, etc. This mechanism is easy to implement because a new flag will have the last flag's value plus one. Also agents will know the direction to the destination following the flags with lower value.

Adding norms to the system to give or take points for putting color flags will promote agent's selfishness or cooperation. They will be deciding to put them and collaborate based on their own interests and their benefit.

4 Laws, Institutional Norms and Norms

This chapter explains the differentiation between the regulation types of the Normative Packet World. This distinction will help researchers to compare their work in agent's societies and having these three different types of regulations will extend the possibilities of the framework.

Packet World previously supported Laws. These laws were used to limit what agents can do in the virtual environment and can be compared to physical laws, i.e. an agent can only step from one cell to a neighbor cell, this prevents an agent to "jump" to a virtually very distant cell doing only a step.

The Institutional Norms are norms set from the Packet World to the agents and allow to define the organizational aspects, i.e. an agent will receive a reward (points) for delivering a packet or will get a punishment for putting a flag in a wrong position.

Finally, Norms allow to auto-regulate the Multi Agent System and let the agents to regulate their interactions, i.e. if an agent passes a packet to other agent, it will receive some points from the agent that picks up the packet.

4.1 Laws

As mentioned before Packet World already had laws. These laws were implemented as Java classes using the interface *Law* defined with two methods.

First, *applicable* check if the law is relevant for a given influence and it is used to define what activates the law. For example, when an agent decides to step creates a step influence and passes it to the environment. At the Reactor (explained in chapter 2) the laws that are applicable to that influence will be activated.

Second, *apply* defines if the action enclosed by the influence complies with the law. Continuing with the previous example, if the cell where the agent wants to step is not free the law will prevent the action. Figure 10 show an example of an implemented law.

The laws are listed in a configuration file named *lawsoftheuniverse.prop*. To create a new law a Java class must be created implementing the interface *Law* and adding the name of this class to the configuration file. The structure of the file is *"<number>=<Class>"*, where the number is unique for each law. For example the first law should be added as *"1=environment.LawStep"*.

```
public class LawPutPacket implements Law {

    public boolean applicable(Influence inf) {
        if (inf instanceof InfPutPacket) {
            return true;
        }
        else {
            return false;
        }
    }

    public boolean apply(Influence inf) {
        return CollisionMatrix.PacketCanStandOn(env,
                                                inf.getX(),
                                                inf.getY());
    }
```

Figure 10: Law LawPutPacket.

The laws relevant to this work are:

- LawStep: Checks if the destination cell is a neighbor cell and if it is free.
- LawPickPacket: Checks if the packet that the agents tries to pick up exists.
- LawPutPacket: Checks if the cell where the agent wants to put the packet is free or a destination.
- LawPassPacket: Checks if the agent who will receive the packet exists and is not carrying a packet.
- LawPutFlag: Checks if the cell where the agent wants to put the flag is a free one.

And a new norm was added:

- LawStepPoints: Checks if the agent has points to move. In case it has no points, the agent will not be able to move.

4.2 Institutional Norms

The design of the Institutional Norms is based on the laws of the Packet World. An interface *Norm* had to be defined with the flexibility to allow different kinds of norms to be created. The interface proposed has three methods.

First, *applicable* checks if the norm is applicable for a given influence. This has the same functionally as *applicable* at laws.

Second, *ask* gives the outcome of applying an influence. Thanks to this functionality agents are able to query the result of an action and decide based on the outcome it produces.

And third, *apply* executes the norm producing changes in the environment and/or agents. The difference between norms and laws is that applying a norm produces changes and applying laws only validates influences. Also a norm can produce different changes depending on the context that are executed. For example, when a norm applicable for a step influence is applied, the agent that created the influence loses one point but if the agent is carrying a packet loses two points.

```
public class InstNormPointForPacket implements NormPoints {

    public boolean applicable(Influence inf) {
        if (inf instanceof InfPutPacket) {

            if(env.getWorld("DestinationWorld").getItem(inf.getX(),inf.getY())
                    instanceof Destination){
                return true;
            }
        }
        return false;
    }

    public NormOutcome ask(Influence inf) {
        if(applicable(inf))
            return new NormOutcomePoint(points);
        else
            return null;
    }

    public boolean apply(Influence inf) {
        env.addPoints(inf.getID(), points);
        return true;
    }

    public void setRatio(int ratio) {
        points = PointsControl.getInstance().getPointsForRatio(ratio);
    }
```

Figure 11: Institutional Norm InstNormPointForPacket.

To enable the norms to handle points an extension of the *Norm* interface was also developed. The *NormPoints* interface adds the *setRatio* method, which is used to calculate how many points the norm gives or takes from the agent. The points are determined by the number of cells of the environment divided by the ratio set in this norm. This helps the norm to adapt to different environments. Figure 11 show the implementation of the Institutional Norm InstNormPointForPacket.

A file name *institutionalNorms.prop* was created to list all the institutional norms of the system.

The structure of the file is *"<number>=<class>[ratio]"*, where the number is unique for each norm, the class is the name of the Java class with the implementation and the ratio can be defined here to simplify the modification of points of a norm (only for *NormPoints*).

The list of Institutional Norms added to the normative system is:

- InstNormPointForPacket: Sets the number of points that an agent gets when it delivers a packet.

- InstNormPointForColorFlag: Sets the number of points that an agent loses when it puts a color flag. This number differs from putting the flag in a right or a wrong position.

- InstNormPointForBlackFlag: Sets the number of points that an agent gets when it puts a black flag. If the flag is not correct, the agent will lose points.

- InstNormPointForStep: Sets the number of points that an agent loses when it does a step.

- InstNormPointForPickPacket: Sets the number of points that an agent loses when it picks up a packet.

- InstNormPointForPutPacket: Sets the number of points that an agent loses when it puts down a packet.

4.3 Norms

The design of the norms uses the same structure and interface than the Institutional Norms; the difference between them relies on the implementation. These norms only affect the interaction between agents. For example, when an agent passes a packet to other agent gets some points from the first agent, this only affect them because from an institutional point of view no points were earned or consumed.

The file that contains the list of norms is named *norms.prop* whith the same structure of institutional norms' file.

The norms added to the normative system are:

- NormPointForPassPacket: Sets the number of points that an agent must pay to another agent for receiving a packet.
- NormPointForUsingColorFlag: Sets the number of points that an agent must pay to another agent for stepping in a cell with a color flag that he previously put.

5 Normative Behavior

In order to test the normative system, a normative behavior has been developed. The purpose of this behavior is to expand the agents' decision process with the ability to decide their actions depending on the current norms. In this behavior agents search for packets and bring them to the destination, they can put black flags to mark zones where no packets are left and color flags to mark the path to the destination. Also agents can collaborate passing packets between them.

5.1 Behavior Detailed

Figure 12 shows the action selection model for the behavior. There are four main roles that are dependant from the other ones. Using another words, these four roles will be executed first than the other roles.

5.1.1 Main Roles

Update Perception

It's responsability is to update the belief base with the current agent's perception. This belief base is populated with all the packets, destinations and flags that the agent sees while it is moving around the environment. The belief base persists only for ten time steps, beliefs older that these steps are

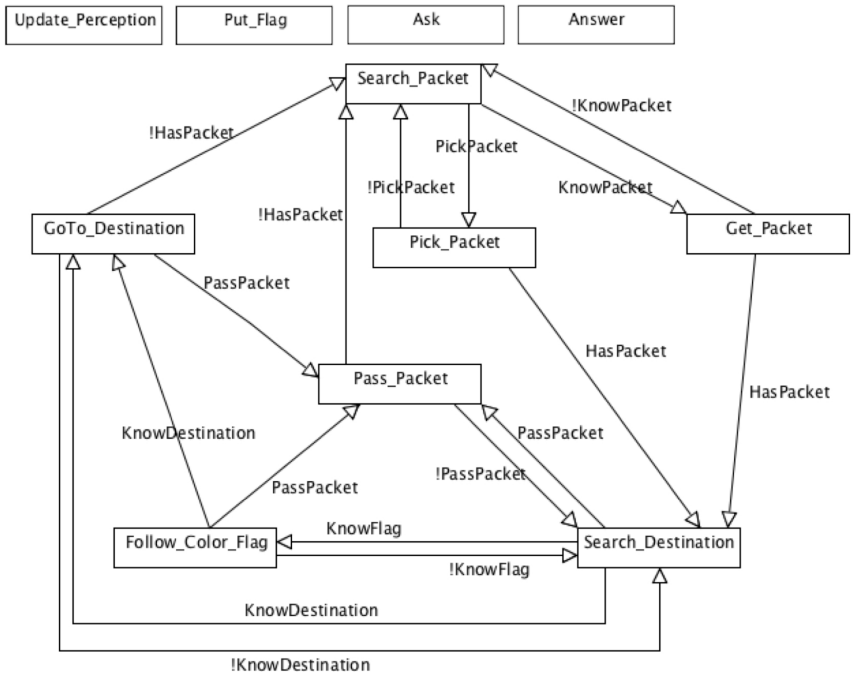


Figure 12: Agent's normative behavior.

removed. For example, if an agent sees the red destination after ten steps it will forget it's location.

Put Flag

This role handles if the agent should put a color or black flag in its current location. The role will check if it is correct to put the flag and the outcome of putting it asking to the normative system. If this outcome benefits the agent it will put the flag. The decision process will be explained at the end of this chapter.

Ask Packet

This role does not act but instead communicates with other agents. In case that the agent is not carrying any packet, it does not know where is possible to pick up one and, then it sees another agent (in its perception range with

a packet), it will ask the other agent to pass its packet. The protocol for this communication process is explained in section 5.2.

Answer

This role handles the incoming messages and the answers. When an agent is carrying a packet, it can receive requests to pass the packet to other agents and must confirm if it is interested or not. Furthermore, if an agent is asked for a packet, it will go to receive it only when it receives a confirmation message. This role starts the commitment to pick or pass the packets between agents.

5.1.2 Other Roles

Search Packet

The agent moves around the environment searching for a packet, and it does it avoiding empty zones (which are marked with black flags).

Get Packet

The agent knows the location of a packet and goes to pick it up. If knows more than one location, it goes to the nearest one.

Search Destination

The agent has a packet and it's moving around searching for the destination. It does it avoiding empty zones (which are marked with black flags).

Follow color flag

When the agent has a packet and knows where is a flag of the same color than the packet that it is carrying, it will follow the flags until the destination.

Go to destination

The agent knows where is the destination and goes there to deliver the packet.

Pick Packet

The agent committed to pick the packet to other agent at some location. If it is next to the other agent picks the packet, otherwise goes to that location.

Pass Packet

The agent committed to pass the packet to other agent at some location. If it is next to the other agent pass the packet, otherwise goes to that location.

5.2 Pass Packet Protocol

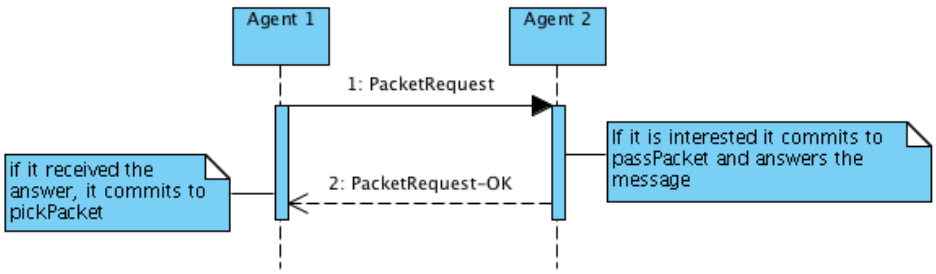


Figure 13: Pass Packet protocol.

Different protocols where tested in order to develop the communication protocol to request a packet to other agent. The result of these tests was that agents move while they communicate and if two agents that were agreeing to pass a packet were moving to opposite directions, at every step they move away from each other. When they commit to pass the packet they

had to move back, wasting steps and points. The best option was to use a simple protocol to create an agreement as soon as possible if they were interested. The result is the protocol shown in figure 13, where an agent asks only to one other agent at each step if it will pass the packet, sending a PacketRequest message. If the other agent is interested, commits to pass the packet to the first agent, fixes the location where it will happen and sends a PacketRequest-OK message. At next step the agent that receives the PacketRequest-OK message commits to pick the packet from the other agent and goes to the designated location. If no PacketRequest-OK is received the agent will send a PacketRequest to other agent if is available.

5.3 Normative Decisions

The objective of this behavior was to let the agents decide their actions based on the norms of the system. To do this, two key aspects where decided to evaluate: putting flags and passing packets.

5.3.1 Put Color Flag Decision

To decide to put or not a color flag agents consult the norms InstNormPointForColorFlag and NormPointForUsingColorFlag. These norms were connected using following formula:

*PointForUsingColorFlag * (Max FlagValue – Current FlagValue) * (1 + Known Packets) > PointForColorFlag*

The Max FlagValue is the number of flags that can be put in the environment from one corner to the other so, it depends on the environment size and the agent's perception. The Current FlagValue is the value that corresponds to the flag in case of putting it in the actual agent position.

This value represents the number of flags remaining to the destination. If the formula is verified the agent puts the corresponding color flag in its current position. This helps to put flags near to the destination and near to the packets, where there are more possibilities that other agents will use them.

5.3.2 Pass Packet Decision

Agents will accept a passPacket request only in the following scenarios:

- The agent does not know the path to the destination and knows where is placed another packet.
- The agent has no points.
- The agent knows the path to the destination and it has not enough points to reach the destination.
- The agent knows the path to the destination and calculates the points it will earn delivering the packet, it also counts the points it will spend carrying the packet and it will pass the packet if it gets more points than delivering the packet.

6 Experiments

Each run of the Normative Packet World counts different factors. These factors are useful to analyze the results of different simulations. The factors are:

- Packets Passed: counts the number of packets passed between agents.
- Agents without points: counts the number of agents that at some point of the simulation had no points left.
- Points Average: does the average of points of all the agents.
- Points Min: tracks the minimum amount of points that an agent had during the simulation.
- Points Max: tracks the maximum amount of points that an agent had during the simulation.
- Points STD Deviation: calculates the Standard deviation between the points of all the agents.
- Points Spent: counts the number of points spent by all the agents while they move, pick up or put down a packet.
- Packets Delivered: counts how many packets have been delivered.
- Color Flags: counts how many color flags have been put in the environment.

- Black Flags: counts how many color flags have been put in the environment.

- Cycles: counts the number of cycles spent to deliver all packets.

The Graphical User Interface has been modified to include these factors. They were added at the bottom of the screen over the buttons that control the simulation (see figure 14). This will help to follow the progress during the simulation.

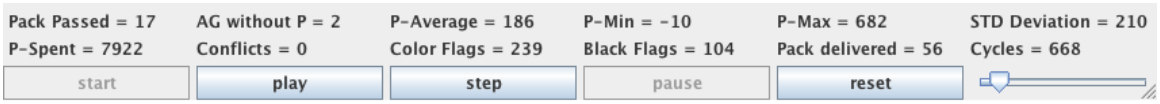


Figure 14: Changes at the Graphical User Interface.

The environment chosen for the tests is shown in figure 15. It consists in a grid of 30 by 30 cells with 56 packets ramdomly distributed in three different colors: blue, yellow and red. Agents have the perception range of 2 cells.

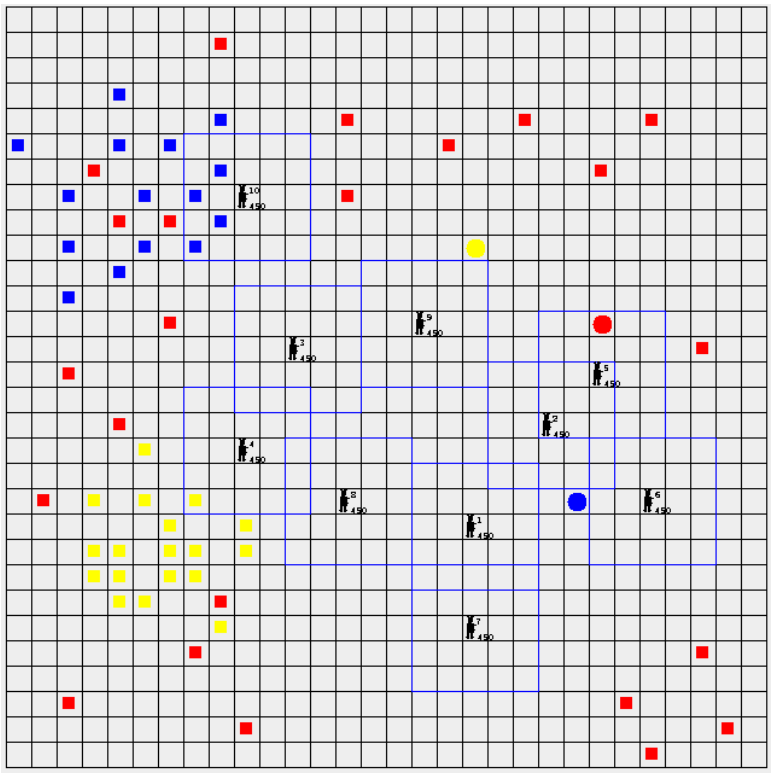
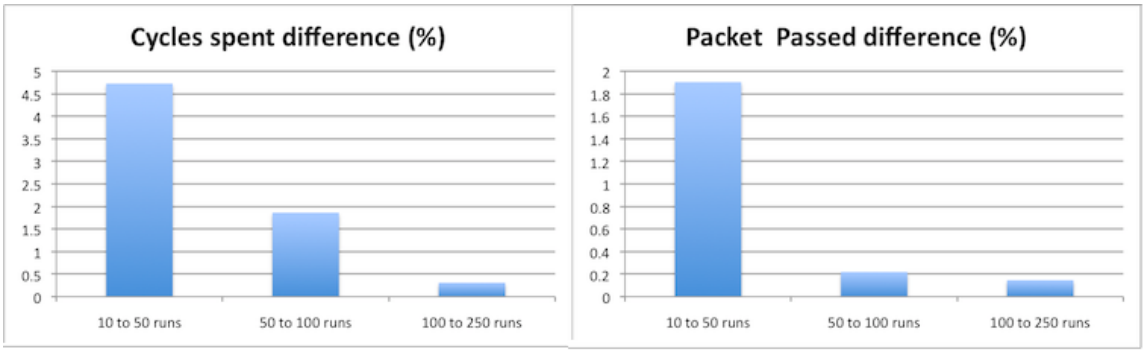


Figure 15: Environment choosed for the experiments.

6.1 Number of runs

The randomness of the steps when an agent searches for a packet or destination makes impossible to rely on the results of only one simulation. To solve this a number of runs for simulation had to be defined.



Runs	Cycles Spent	Packet Passed
10	625	13,4
50	656	13,66
100	644	13,69
250	646	13,71

Figure 16: Differences between number of runs.

Figure 16 shows the difference between the number of cycles and the number of packets passed for 10, 50, 100 and 250 simulations. Only these two factors are shown for simplicity, the results on the other factors were similar. As it can be seen in the chart, the difference from one hundred runs to two hundred and fifty runs per simulations is less than 0,50 percent. The number of runs per simulation chosen was one hundred.

From now on, in all the references to experiments the results will be an average after running one hundred simulations.

6.2 Starting configuration

The starting configuration of norms has been defined to guarantee that the agents deliver all the packets and accumulate numerous points to finish the simulation. The norms configuration is the following:

- Agents start with 450 points.

- InstNormPointForPacket: Agents receive 100 points for delivering a packet.

- InstNormPointForColorFlag: Agents must pay 9 points to put a color flag.

- InstNormPointForBlackFlag Agents get 20 points if they put a black flag.

- InstNormPointForStep: Agents spend 1 point to step without carry and 2 with carry.

- InstNormPointForPickPacket: Agents spend 1 point to pick up a packet.

- InstNormPointForPutPacket Agents spend 1 point to put down a packet.

- NormPointForPassPacket: The agent who receives a packet gives 30 points to the agent who had the packet.

- NormPointForUsingColorFlag: When an agent step in a cell with a color flag gives 4 points to the agent that put that flag.

The results of this configuration will be used for comparison with further executions of different norms configurations. The results are:

Packets Passed	14,47
Agents without Points	1,71
Points Average	256,67
Points Min	6
Points Max	627,84
Points STD Deviation	159,45
Points Spent	7519,08
Packets Delivered	56
Color Flags	219,42
Black Flags	98,26
Cycles	643,61

Figure 17: Results of the starting configuration.

6.3 Experiment 1: InstNormPointForPacket

In order to analyze how the agents behave depending on how many points they receive for each packet, the Institutional Norm InstNormPointForPacket has to be modified. Two simulations were done changing the Institutional Norm, one to 128 points and the other one 150 points.

Points per Packet Delivered	**100**	**128**	**150**
Packets Passed	14,47	8,96	8,49
Agents without Points	1,71	0,76	0,61
Points Average	256,67	409,18	545,71
Points Min	6	29,82	45,44
Points Max	627,84	788,88	955,64
Points STD Deviation	159,45	201,24	240,88
Points Spent	7519,08	7528,29	7466,58
Packets Delivered	56	56	56
Color Flags	219,42	216,91	217,9
Black Flags	98,26	95,43	99,45
Cycles	643,61	621,21	611,93

Figure 18: Experiment 1 results.

Comparing the three simulation's results we see that the agent's points (average, minimum and maximum) increase as it was expected because agents are receiving more points for each packet delivered, and consequently the number of agents without points decreases. It is also interesting that fewer packets are passed, agents become selfish and they prefer to earn

points by delivering packets than passing them to other agents. The number of put flags does not change because the decision process to put a flag is independent of the norm changed.

Finally, the simulation performs better when fewer packets are passed because agents spend less points moving through the environment and require less cycles to deliver all the packets.

6.4 Experiment 2: NormPointForPassPacket

In the previous test was demostrated how increasing the number of points that an agent receives for each packet that delivers, at the same time the number of packets that an agent will agree to pass decreases, making agents to become more selfish. Now we will increase the number of points an agent will pay to another agent to pass the packet, this is set at the Norm NormPointForPassPacket. The stating configuration set to 30 points this norm and now will be tested with 45 and 60 points.

Points per Packet Passed	**30**	**45**	**60**
Packets Passed	14,47	20,17	31,8
Agents without Points	1,71	1,87	2,82
Points Average	256,67	252,74	222,71
Points Min	6	-2,02	-26,22
Points Max	627,84	634,34	638,29
Points STD Deviation	159,45	161,38	161,36
Points Spent	7519,08	7593,6	7971,08
Packets Delivered	56	55,99	55,93
Color Flags	219,42	216,75	216,56
Black Flags	98,26	98,84	102,93
Cycles	643,61	647,62	691,4

Figure 19: Experiment 2 results.

In Figure 19 we can see how the number of packets passed increases. As happened in test 1, the increase of packet passed decreases the overall performance of the simulation, the number of points spent and the number

of cycles needed to deliver all the packet increase. The agent's average points only decreases at the third simulation; it is caused because the simulation takes more cycles to finish and agents spend more points to move around the environment. Here also the number of flags put stays stable.

6.5 Experiment 3: InstNormPointForColorFlag

With this test agent's decision process to put flags will be evaluated. The number of points that color flags cost influences the agent's decision to put flags. Two simulations were proposed changing the Institutional Norm InstNormPointForColorFlag. The starting configuration sets the number of points for this norm to 9, and simulations with 8 and 7 points were run.

Points for a color flag	**9**	**8**	**7**
Packets Passed	14,47	14,2	13,74
Agents without Points	1,71	1,44	1,32
Points Average	256,67	277,36	289,44
Points Min	6	7,51	7,95
Points Max	627,84	649,56	660,72
Points STD Deviation	159,45	169,63	176,76
Points Spent	7519,08	7545,81	7511,58
Packets Delivered	56	56	56
Color Flags	219,42	218	227,11
Black Flags	98,26	98,39	95,03
Cycles	643,61	632,93	631,53

Figure 20: Experiment 3 results.

The simulation slightly performs better when the flags costs are lower. The agent's average points increase and the number of cycles to finish the simulation decrease. The number of flags put in the environment does not change; this is because agents put flags when they are near to the destination or near the packets and at the end they cover the entire environment. The difference is that flags which are closer to the destination are put first because agents don't need to check if there are packets around. With cheaper

flags the radius of the flags around the destination is bigger and more flags are put faster. This means that agents are more collaborative, whit cheaper flags their expectations to recover the points of the flag and earn points are bigger (they always do it based in their interests) and they put more flags at the beginning of the simulation.

6.6 Experiment 4: NormPointForUsingColorFlag

Previously we analyzed how reducing the costs of a flag affects the results of the simulation. In this test we will evaluate how it will affect to the results a change in the number of points that an agent must pay to other agent when it steps in a cell where a color flag is located. The norm responsible for this is NormPointForUsingColorFlag and is set to 4 points at the starting configuration. Two simulations were run; one using 5 points and other using 6 points.

Points for step over flag	**4**	**5**	**6**
Packets Passed	14,47	18,82	22,53
Agents without Points	1,71	2,36	2,68
Points Average	256,67	233,96	232,56
Points Min	6	-4,13	-10,83
Points Max	627,84	642,3	666,01
Points STD Deviation	159,45	170,31	182,46
Points Spent	7519,08	7711	7750,63
Packets Delivered	56	55,99	56
Color Flags	219,42	227,22	227,48
Black Flags	98,26	100,05	101,42
Cycles	643,61	676,45	692,55

Figure 21: Experiment 4 results.

In these simulations the fact that moving through flags is more expensive forces agents to pass more packets in order to get more points. The agent's selfishness is counterproductive because the agent that is carrying the packet knows how to deliver it, but instead of doing it, it spends points agreeing

to pass the packet to earn more points. The number of color flags does not change, the black flags increase because they take longer time to finish the simulation and the empty zones are marked. The points spent by the agents and the cycles done increase, this means that the simulation perform worse than the starting configuration.

7 Conclusions and Future work

In this thesis we have shown how an agent-based framework capable of supporting norms was developed to use in the Packet World Simulator.

Chapter one gave an introduction to the underlaying concepts employed in this work. Chapter two presented the Packet World architecture. Chapter three explained the proposed extension of the system. In chapter 4 the Laws were described and Institutional Norms and Norms were defined. Chapter 5 explained the behavior developed to test the normative system and in chapter 6 a set of experiments were exposed and analyzed.

This extended version of Packet World, the Normative Packet World, provides a mechanism for agents to stop being cooperative and become selfish. Agents decide to collaborate based in their own interests and will collaborate only when they benefit.

In summary, our framework enables researchers to evaluate and compare their work in agent's societies using Laws, that model the physical environment with its restrictions, Institutional Norms that allows setting the organizational aspects (rules of the game) and Norms between agents which enables Multi Agent System to auto-regulate (what can also allow norm emergence if agents with the necessary skills were defined).

Concluding it can be stated that the Packet World has been improved and become more realistic adding points that encourages agents to work (earning points) and rationalize the coordination (exchanging points).

7.1 Future work

As future work more tests could be done evaluating the benefits of not following the norms. A behavior where agents' benefit by misplacing flags or hiding packets can be worth of experiment.

Currently the agents' global behavior is the same for all of them and the possibility to have groups of agents behaving differently will give more flexibility to the system. Competitions between these groups could be done encouraging agents to collaborate with their groups and to compete with the others.

Finally, an utility to create, edit, activate and deactivate norms using a graphical user interface will help to promote the Normative Packet World and perform experiments easier and faster.

Bibliography

[1] D. Weyns, E. Steegmans and T. Holvoet, *The Packet-World: A Test Bed for Investigating Situated Multi-Agent Systems*. Whitestein Series in Software Agent Technology (2005).

[2] J. Ferber, Multi-Agent Systems, *An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, Great Britain (1999).

[3] B. Vandeweerdt, *A Model for the Environment in Reactive Multi-Agent Systems*. Master thesis Katholieke Universiteit Leuven (2003), only available in Dutch.

[4] D. Weyns, E. Steegmans and T. Holvoet, *Towards Active Percption in Situated Multi- Agent Systems*. Journal on Applied Artificial Intelligence 18(8-9) (2004), 867–883.

[5] Guido Boella, Leendert van der Torre, and Harko Verhagen. *Introduction to normative multiagent systems*. Dagstuhl, Germany (2007).

[6] K. Rosenblatt and D. Payton. *A fine grained alternative to the subsumbtion architecture for mobile robot control*. IEEE Joint Conference on Neural Networks (1989).

[7] D. Weyns, E. Steegmans and T. Holvoet. *Combining Adaptive Behavior and Role Modeling with Statecharts*. 3th International Workshop

on Software Engineering for Large-Scale Multi-Agent Systems, ICSE, Scotland (2004).

[8] J. L. Austin. *How To Do Things With Words*. Oxford University Press, UK (1962).

[9] D. Weyns, E. Steegmans and T. Holvoet. *Protocol Based Communication for Situated Multi-Agent Systems*. 3th International Joint Conference on Autonomous Agents and Multi-Agent Systems, New York (2004), 118–127.

[10] P. Grassé. *La theorie de la Stigmergie: Essai d'interpretation du Comportement des Termites Constructeurs*. Insectes Sociaux, Vol. 6 (1959).

[11] H.R. Berenji and D. Vengerov. *Cooperation and Coordination Between Fuzzy Reinforcement Learning Agents*. 8th IEEE Conference on Fuzzy Systems, Korea (1999).