The Development of Normative Autonomous Agents: an Approach

Marx Viana[1], Paulo Alencar[2], Donald Cowan[2], Carlos J. P. de Lucena[1]

[1]PUC-Rio, Software Engineering Laboratory, LES - Rio de Janeiro – RJ – Brasil
{marxviana, lucena}@les.inf.puc-rio.br
[2]University Waterloo – Waterloo – ON – Canada
palencar@uwaterloo.ca

Technical Report CS-2015-05

Abstract. Open multi-agent systems (MASs) act as societies in which autonomous and heterogeneous agents can work towards similar or different goals. In order to cope with the heterogeneity, autonomy and diversity of interests among the different agents in the society, open MASs establish a set of behavioral norms that is used as a mechanism to ensure a state of cooperation among agents. Such norms regulate the behavior of the agents by defining obligations, permissions and prohibitions. Fulfillment of a norm may be encouraged through a reward while violation of a norm may be discouraged through punishment. Although norms are promising mechanisms to regulate an agent's behavior, we should note that each agent is an autonomous entity that is free to fulfill or violate each associated norm. Thus, agents can use different strategies when deciding to achieve their goals including whether to comply with their associated norms. Agents might choose to achieve their goals while ignoring their norms, thus overlooking the rewards or punishments they may receive. In contrast agents may choose to comply with all the norms although some of their goals may not be achieved. In this context, this paper proposes a framework for simulation of normative agents providing a basis for understanding the impacts of norms on agents.

1 Introduction

Open multi-agent systems (MASs) are societies in which autonomous, heterogeneous and independently designed entities work towards similar or different goals [9]. In order to deal with both the autonomy and diversity of interests among the different member agents, these complex systems can provide a set of norms as a mechanism to manage social outcomes. In this way, they provide a structure in which agents strive to meet both individual and societal goals [19].

Norms can be defined as mechanisms that regulate the behavior of agents by defining obligations (agents must accomplish a specific outcome), permissions (agents can act in a particular way) and prohibitions (agents must not act in a specific way) [13]. Norms are promising mechanisms to regulate the behavior of software agents as

agents are autonomous, and free to fulfill or violate each system norm. This type of agent reasoning is called a normative strategy [10].

Several approaches [4, 19] have been proposed for the specification and implementation of norms, while others have focused on the definition of parts of an infrastructure that can be used by belief-desire-intention (BDI) agents [14] to reason about norms [7, 11]. However, there is still a need to define an agent-oriented framework to support the implementation of goal-oriented normative agents; that is agents with the main purpose of achieving their goals and desires while attempting to conform to system norms. Although there are a number of existing agent-oriented platforms such as [1, 5], none provides support for normative agents.

In this context, we present a framework for Normative Agent Java Simulation (JSAN). This framework was defined to build and operate agents able to deal with goals, desires and norms and thus agents that can support normative reasoning. JSAN extends the JASON framework [1], which already provides support for the implementation of BDI agents and a set of hot-spots that enable the implementation of normative functions. By using these function extensions, it is possible to build BDI agents that can check if they should (i) adopt a norm, (ii) evaluate the effects on their desires with respect to the fulfillment or violation of a norm, (iii) detect and solve conflicts among norms, and (iv) select desires and plans based on the decision on whether to fulfill a norm. A preliminary overview of the framework is described in [21].

The paper is organized as follows. Section 2 focuses on the representation of norms. Section 3 presents the JASON Platform. Section 4 discusses related work. In Section 5 details of the JSAN framework are provided and Section 6 describes a case study by showing how agents deal with norms in real situations. Finally, Section 7 presents our conclusions and future work.

2 The Representation of Norms

Since norms [9] are designed to regulate the behavior of agents, a norm's definition must include the address of the agent being regulated, when the norm should be applied, the nature of the norm (permission, obligation or prohibition), and the consequences of fulfilling or violating the norm (reward or punishment). In this paper, we use the norm representation in [18]. The representation has an element *norm*, which is composed of several properties. Each of those properties is briefly described in Table 1. These properties include: *Addressee*, *Activation*, *Expiration*, *Rewards*, *Punishments*, *DeonticConcept* and *State*. The description of each property is provided in this table. For example, *Addressee* is used to specify the agents or roles responsible for fulfilling the norm.

To understand the definition of norms and their representation better, imagine that a Fireman Commander agent assumes the lead role in rescuing civilians who are in hazardous areas. This agent is responsible for regulating the behavior of all the other fireman agents and their use of available resources such as helicopters, vehicles and troops. We assume that such resources are limited. In addition, each firemen agent should attempt to perform a rescue according to specific norms. Eventually, a behav-

ioral norm is sent to each firemen agent that says: “protect lives of civilians in hazardous areas.” This norm has the following attributes: (i) the addressees are the firemen agents; (ii) the required deontic concept is obligation; and (iii) an agent agreeing to a norm means that agent will receive a reward which is either air or ground support for the agent’s mission. If a fireman agent violates a related norm in the environment, then the agent receives the punishment associated with that norm. For example, there are situations when a fireman agent requests aircraft support to accomplish a specific rescue operation that places him or her in a situation riskier than the one allowed by the norm. In this case, this agent will suffer the punishment associated with the norm, such as a warning or an order that he should be temporarily restricted to headquarters to assist other rescuers.

Note that the norm is activated if there is any person in a risky situation. In turn, the norm expires when all civilians are safe, and the state or element regulated by the norm is the action of using aircraft.

Table 1 – Norm Elements.

Property	**Description**
Addressee	It is the agent or role responsible for fulfilling the norm.
Activation	It is the activation condition for the norm to become active.
Expiration	It is the expiration condition for the norm to become inactive
Rewards	It represents the set of rewards to be given to the agent for fulfilling a norm.
Punishments	It is the set of punishments to be given to the agent for violating a norm
DeonticConcept	It indicates if the norm states an obligation, a permission or a prohibition.
State	It describes the set of states being regulated.

3 The JASON Platform

The JASON platform enables the development and implementation of Belief, Desire and Intention (BDI) agents using a language called AgentSpeak. An overview of how JASON interprets AgentSpeak programs is shown in Figure 1 [12]. In this figure, sets of beliefs, events, plans and intentions are represented by rectangles. Diamonds represent the selection of an element of a set and circles represent some of the processes involved in the interpretation process.

Each interpretation cycle updates the list of events based on the agent’s perception of the environment, the messages the agent receives and the information coming from the agent’s own execution of a plan. The Belief Review Function (BRF) revises the Belief Base with both a literal to be added or deleted, and the intention structure that required the belief change. A single event is selected by the Event selection function

(SE) and this event is unified with the triggering events in the heads of plans by the Unify Event cycle that generates a set of all relevant plans. The context of such plans is verified according to the Belief Base by the Check Context cycle, which generates a set of options. The Option Select Function (SO) selects a single applicable option from the set of options, which becomes the intended means for handling the selected event. The option either pushes the plan on top of an existing intention (if the event was an internal one), or creates a new intention in the set of intentions (if the event was external, i.e., generated from perceptions of the environment). The Intention Select Function (SI) selects one of the agent's intentions and this intention is executed by the Execute Intention cycle. When all formulas in the body of a plan have been executed, the whole plan is removed from the intention list, and so is the achievement goal that generated the plan. This ends a cycle of execution, and the interpretation starts over again, checking the state of the environment after agents have acted upon it and generated the relevant events.

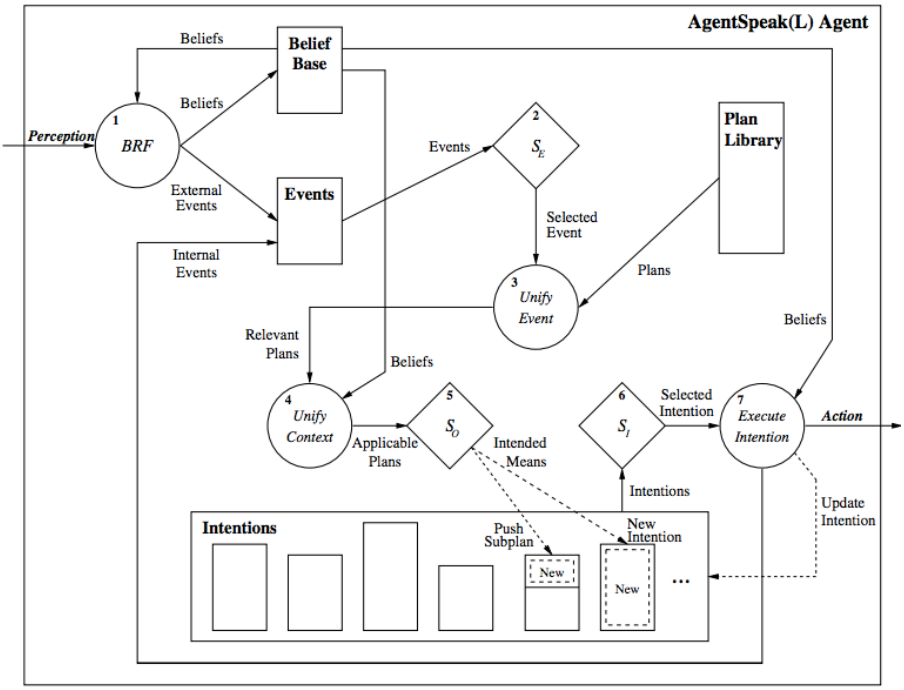

Figure 1. An Interpretation Cycle of an AgentSpeak Program [13].

4 Related Work

Some approaches have been proposed in the literature for developing agents that evaluate the effects of fulfilling or violating norms.

The n-BDI architecture [3] presents a model for designing agents capable of operating in environments governed by norms. This architecture selects objectives to be performed based on the priority associated with each objective. An objective's priority is determined by the priority of the norms that govern the objective. However, it is not clear in this approach how the components of a norm can be evaluated. In addition, the approach does not support a strategy to deal with conflicts between norms.

In [10], the authors propose a formal model, using the Z specification language, for modeling agents that achieve their objectives based on the norms of the system. According to [10] an agent created from such a model is able to: (i) check if it is the one responsible for fulfilling a norm; (ii) verify the activation and expiration of a norm based on the beliefs of the agent; (iii) evaluate and decide to fulfill or violate every norm of the system; and (iv) make the decision to fulfill or violate a norm while removing or adding agent goals. Besides not showing how the evaluation of a norm is performed, the authors do not focus on identifying and resolving conflicts between norms, checking fulfilled or violated norms, and showing the influence of norms on the plan selection process and intentions of the agents.

In [8], the authors present a set of strategies that can be adopted by agents to deal with norms. These strategies are: *Social*, *Pressured*, *Opportunistic*, and *Rebellious*. The *Social* strategy focuses on the agents complying with the rules without worrying about their individual goals. The *Pressured* strategy happens when agents fulfill the norms to achieve their individual goals considering only the punishments that will harm them. Another strategy is the *Opportunistic* strategy, in which agents consider only the effects of rewards on their individual goals, and seek to fulfill only the norms for which the rewards of the individual goals are more important than those of the social goals. The *Rebellious* strategy implies that the agents focus on achieving their individual goals, regardless of the punishments attached to the violation of the norms. Finally, the *Selfish* strategy is the combination of the *Pressured* and *Opportunistic* strategies.

5 JSAN - A Framework for Normative Agent Java Simulation

This section describes the main ideas behind the framework proposed in this paper. We provide an overview of JSAN and discuss some of its details, including its kernel (frozen-spots) and its flexible points (hot-spots) [22].

5.1 The Framework

The JSAN framework is implemented using software agents, as illustrated in Figure 2. As previously mentioned, JSAN extends the JASON framework [1], which is an interpreter for an extended version of the AgentSpeak language proposed by Rao [16]. The AgentSpeak language is an agent-oriented programming language that supports the creation of BDI agents and provides a platform for the development of multi-agent systems (MASs).

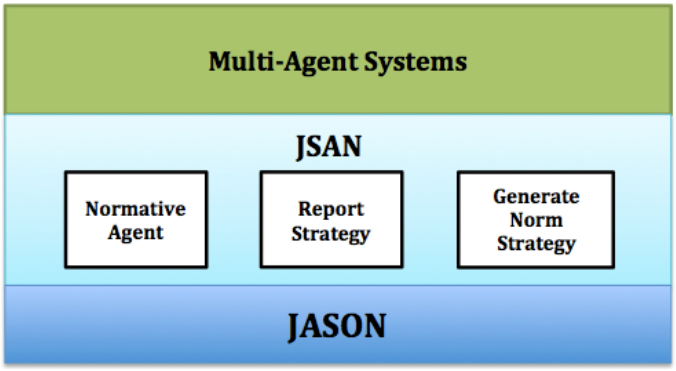

Figure 2. The JSAN Architecture.

JSAN is able to support three main functions: (i) representation of agents that can deal with norms; (ii) visualization for agent simulations involving norms; and (iii) creation of norms in the environment. In order to implement normative agents it is necessary to instantiate the JSAN framework by implementing the agents' goals, movement strategies in terms of efficiency (low cost) and normative strategies (how the agents deal with norms). JSAN already provides a normative process composed of four activities, as detailed in Section 5.2.

5.2 The JSAN Architecture

The JSAN framework allows the implementation of Normative Agents (see Figure 3) and thus supports the creation of simulations to understand the impact of norms on such agents. For this purpose there is a need to: (i) create the agent's goals; (ii) create different agent movement strategies; and (iii) represent how the agents will deal with norms, so-called normative strategies.

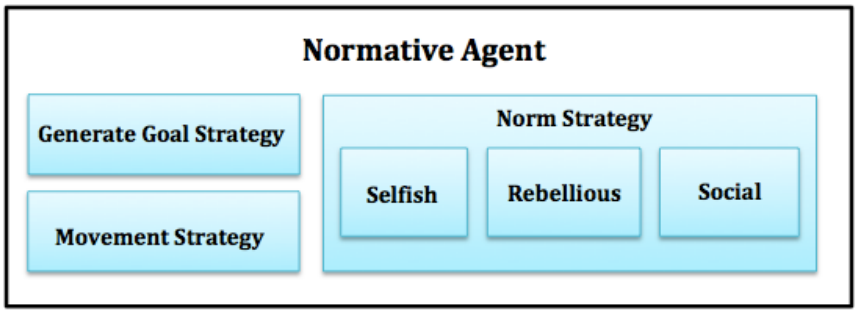

Figure 3. The Internal Structure of the Normative Agents provided by JSAN.

JSAN provides a normative application process represented by the *NormStrategy* class, which is composed of four activities (see Figure 4): *Norm Awareness* (Section 5.2.1), *Norm Adoption* (Section 5.2.2), *Norm D*eliberation (Section 5.2.3) and *Norm*

*Imp*act (Section 5.2.4). This normative application process was created based on the process in [11]. Note that, although JSAN already provides such a process, it is possible to define alternative processes using the *NormStrategy* class. It is also possible to implement different activities (or steps of the process) by extending the class *NormStrategy*.

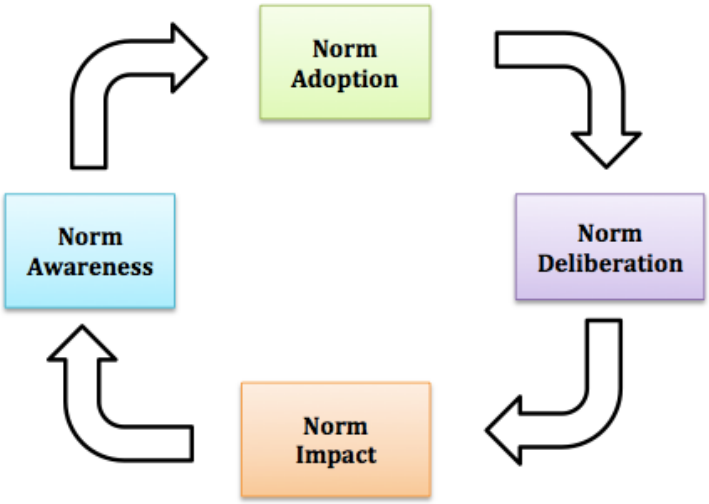

Figure 4. The Normative Strategy provided by the JSAN framework.

When we have an active norm in an agent's environment, the following steps take place when applying this norm [11].

1) Norm Awareness: Each agent identifies which norms are active in the environment and determines which norms apply to that agent's activities.

2) Norm Adoption: Each agent adopts the norms that are addressed to that agent.

3) Norm Deliberation: Each agent assesses the agent's goals that might be hindered by satisfying a norm, and the agent's goals that might benefit from the associated rewards and decides whether to comply with the norm and then acts accordingly.

4) Norm Impact: After an agent complies with a norm (step 3), the agent's goals will be updated. The cycle continues and the agent identifies other related norms.

It is important to note that these steps must take into account not only the goals of agents, but also the mechanisms the society of agnts uses to avoid the violation of norms, such as rewards and punishments. In other words, the agents must consider the "social pressure" of norms before making any decision regarding norms.

5.3 Hot-Spots and Frozen-Spots

As JSAN is an extension of JASON, they share the same core functionality and hot-spots. For example, the process used by agents to communicate and identify other agents is an example of a hot-spot that JSAN inherited from JASON.
The hot-spots defined in JSAN are:

- Environment (EnvironmentSimulat*ion* class): The *ExecuteAction* method was extended from the *Environment*. Whenever an agent tries to perform an action, the agent's identification and its chosen actions are passed to this method. The code *ExecuteAction* method must verify that the action is valid and then perform the action. The action could possibly change the state of the agent. The method returns true if the action was performed successfully.
- Normative Agent (*NormativeAgent* class): By extending this class and implementing the execute method it is possible to define different algorithms to execute the plans of an agent.
- Create Norms (*GenerateNormStrategy* class): It is possible to define new strategies to create norms in the environment.
- Norm Strategies (*NormStrategy* class): It is possible to define new strategies (or plans) for the agents to interact with norms, and to execute the normative application process. JSAN provides a default process implemented in the classes *Selfish*, *Rebellious* and *Social*.
- Created Agent Goals (*GenerateGoalStrategy* class): It is possible to create new goals for agents.
- Movement Strategies (*MovementStrategy* class): It is possible to create new movement strategies for the agents in the environment.
- Simulation Environment Report (*ReportStrategy* class): It is possible to define reports as the output of the simulation. In order to use this mechanism it is necessary to extend the *ReportStrategy* class, where the following parameters should be provided: (i) the environment in which the simulation is being carried out; and (ii) an object of NormStrategy class, which contains the strategy used by the agent to handle the norms.

6 Application Scenario

6.1 Evacuating People from Areas of Risk

The need to build platforms to assist experts in both risk analysis and civilian evacuation planning is a critical problem as more of the population migrates to cities. [3]. Because of size and concentration of population, cities experience situations in which people need to be rescued as a result of floods, landslides or other natural phenomena often caused by climate change. Landslides, for example, are difficult to predict since they depend on many factors such as climate, soil properties and humidity and their specific relationship. The annual number of landslides globally is estimated to be in the thousands, and the associated infrastructure damage is billions of dollars [17]. Planning evacuations from these areas of risk can be assisted by simulations using the JSAN framework. For example, these simulations can be used to implement different scenarios in which fireman agents regulated by norms rescue civilians at risk.

6.1.1 Overview

The implementation of the system is composed of fireman agents, civilian agents and norms, as illustrated in Figure 5. The goal of the fireman agents is to rescue civilians who are at risk. The civilians are not governed by norms. The norms are structured to extend JSAN *GenerateNormStrategy* class as proposed in [18].

These simulations are normative multi-agent systems that receive data containing information about (i) a hazardous area, (ii) the existence of civilian personnel at risk, (iii) ways of removing civilians from these dangerous locations with troops, land vehicles or aircraft, (iv) norms that fireman agents must follow during the rescue operation, and (v) rescue plans to be used in the simulation. Using the simulations, it is possible to propose different solutions the fireman agents can follow in order to evacuate civilians from the areas of risk.

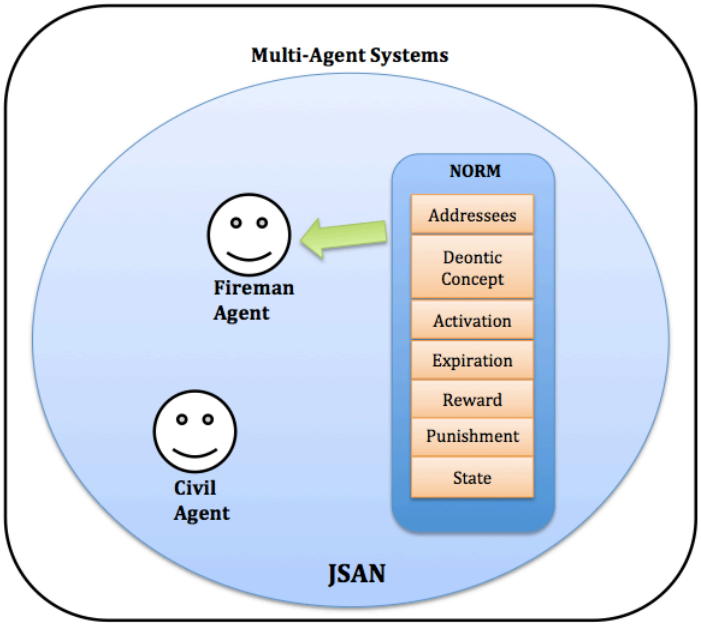

Figure 5. Conceptual model of the usage scenario.

The fireman agents in handling the information just described and in understanding their associated norms must have: (i) a set of objectives that meets the needs of each fireman agent; (ii) an information base to characterize the risky locations; and (iii) a base of strategies to be used when working with norms. The Selfish strategy of the firemen agents (See description in Section III) was implemented using the calculate method of the Selfish class (see Figure 3). This method analyses the situation where norm compliance will help the agent to meet its individual goals, without forgetting the social goals, since norm compliance is directly linked to the benefits received by the agent.

We used the *PlanGenerateNorms* class, an extension of the *GenerateGoalsStrategy* class (see Figure 2) that uses the *generateGoals* method to create the goals for the fireman agents based on their plans. The objectives generated for the simulation were: (i) to save civilians in areas of risk; (ii) not to put civilian lives at risk; and (iii) to receive aircraft support. The *generateNorms* method of the *GenerateNormStrategy* class was used to create norms in the environment aimed only at fireman agents, provided by the *PerceptGenerateNorms* class in the JSAN framework. The *MovementStrategy* class has been extended to implement the method to specify how to move the fireman agents. For this purpose, we used the *ReactiveMovement* class, which extends the *MovementStrategy* class. The movement strategy of the fireman agents checks if any civilian is at risk. If that is the case then one or more fireman agents must make the rescue, and put their lives at risk. Each fireman agent may ask for help in the form of land vehicles, aircraft, or even fire.

In the *Norm* class (see Table 1), the attributes were defined so that the sanctions, activation and expiration conditions can be created. These are elements that will be regulated by the norm, and the norm's deontic concept (see Section II). The *EnviromentSimulation* class is also responsible for managing a set of strategies for visualizing the simulation information represented by the *ReportStrategy* class. The social contribution strategy provided by the JSAN framework is used to check the social contribution of a norm in case the norm is fulfilled or violated by the fireman agents. The norm-related information namely (i) the rewards and punishments linked to norms; (ii) the norm's deontic concept; and (iii) the plans adopted to comply with the norm; is also tested.

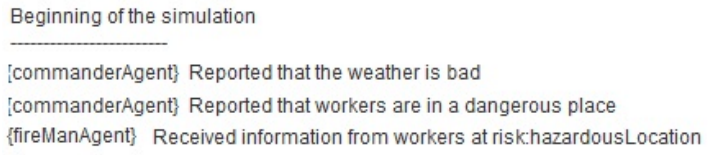

Figure 6. Flow of the start of the JSAN simulation.

In Figure 6, the Fireman Commander agent receives a message about bad weather and a second message indicating there are civilians in a hazardous location. After receiving the latter message, the Fireman Commander agent sends an alert to the fireman agents about the civilians at risk.

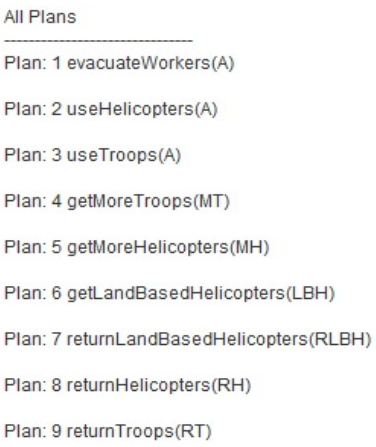

Figure 7. Emergency plans described in the simulation.

The plans for the rescue operation for this situation involve the use of aircraft, ground vehicles, and whether the fireman agents need reinforcements (see Figure 7). These plans are related to the goal: "protect the lives of civilians in areas of risk."

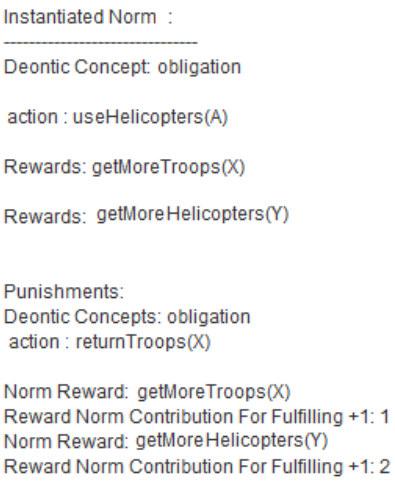

Figure 8. Norm created in the simulation.

The deontic concept embedded in the norm instantiation is an obligation. Specifically a reward is granted if the norm is met and the agent gets air or ground support, and if the norm is violated the agent will not get ground support for the rescue. Further the element that the norm regulates is the action of using aircraft and the norm is enabled if there is any person at risk and the norm is disabled when all civilians are safe. The norm Rewards in Figure 8 are the rewards associated with the norm and Reward Norm shows how the agent will get the reward if the agent complies with the norm.

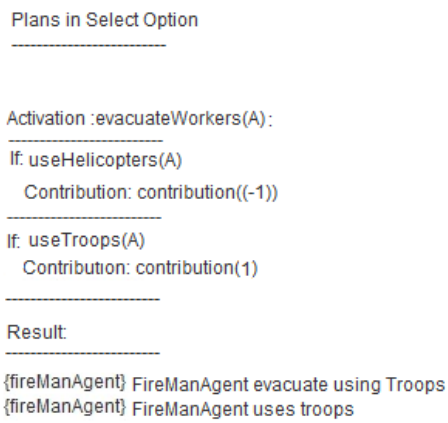

Figure 9. Fireman agents dealing with active norms in the usage scenario.

Figure 9 shows the specific plans the fireman agents decided to use after the norm has been activated because of the existence of civilians in areas of risk. If they choose to use aerial vehicles, this will contribute negatively, and if they choose to use land vehicles, their contribution will be positive. Because of the simulation fireman agents use the Selfish strategy to deal with the norms and they decide to evacuate civilians who were in this risky area using ground vehicles.

6.2 Crime Prevention

Crime prevention is a major area within criminology [2] with the challenges of prediction and prevention particularly in the emergence of new locations with a high crime rate. An approach used to find these new crime locations involves the use of social simulations combining research results in criminology and computer science.

In [2], the authors present a simulation to support crime prevention by analyzing the location of crime in a specific region. In this simulation, they explore different strategies for moving the police to protect civilians and prevent criminal assaults.

In this paper, we present an extension of the simulation for crime prevention in which the insertion of norms was implemented using the JSAN framework to support the police personnel. This framework provides the necessary mechanisms to create the norms in the simulation environment and to extend several strategies to deal with the norms related to police personnel. JSAN provides some pre-defined strategies already implemented in the framework.

The simulations to support crime prevention need information about the number of criminals, the number of police personnel and the number of civilians. Next, it is assumed that the city can be represented in terms of a number of separate areas or boroughs. It is important to know the density of each type of agent that is present in each city location. For example, we compute the density of criminals in a given location by dividing the number of crimes in that area by the total number of criminals in the city.

In addition, in order to describe the migration of the different types of local agents to other locations, information about the reputation (or attractiveness) of each location is required. This reputation factor is different for each type of agent. For example, police personnel may not normally be attracted to places where civilians are safe, that is, places where some policemen are already present and where there is little criminal activity. Police personnel are certainly attracted to areas with a significant number of criminals, where there are many civilians and no policemen. The interaction among the concepts described in the previous paragraph is shown in Figure 10. This Figure depicts the influence among different groups in one location, where circles refer to concepts and arrows indicate the influence of the concepts (e.g., the influence related to attractiveness is shown by a dotted arrow). The norms included in the simulation are only directed to policemen, who use the *Selfish* strategy for dealing with the norms (see Section 4). For example, a norm created for this simulation by *RandomGenerateNorms* class was "not put civilian lives at risk." The norm has the following structure: (i) the norm's deontic concept is an obligation; (ii) if the norm is fulfilled the agent is rewarded by being allowed to drive faster on the way to fulfilling the norm and punished by being obligated to drive slower and thus losing the ability to make arrests in other situations; (iii) the norm is activated if the policeman is an active civil servant and civilians are at risk in a certain location and deactivated when that location is safe for civilians; and (v) the element that the norm regulates is the action of arresting criminals.

When a policeman receives the previously mentioned norm, being *Selfish*, their strategy will be to compare this norm with their individual goals and assess whether they will suffer sanctions that may affect their individual goals if they violate the norm, and whether they will receive rewards to reach their individual goals if they satisfy the norm. After comparing benefits (in terms of rewards) from satisfying the norm against losses (in terms of punishments) if they violate the norm, the agents make their decisions based on their self-interests.

The policemen receive information from various areas of the city, and they can use this information to fight crime using different movement strategies. In this case, they choose their movement according to the environmental situation. For example, if there are several burglaries in a particular site, they will act reactively by getting to the location immediately after the robberies occur [2]. Alternatively, it is possible to act proactively. If a large number of civilians are gathered in a specific location for a demonstration and that area has a high crime rate, the policemen can be instructed (according to the chosen strategy) to get to the area of the location before robberies happen. In this way the policemen try to anticipate the movement of criminals to ensure people are safe [2]. There is still the possibility of adopting a hybrid scheme, which can be a combination of a reactive and a proactive approach, in order to be more successful in the fight against crime.

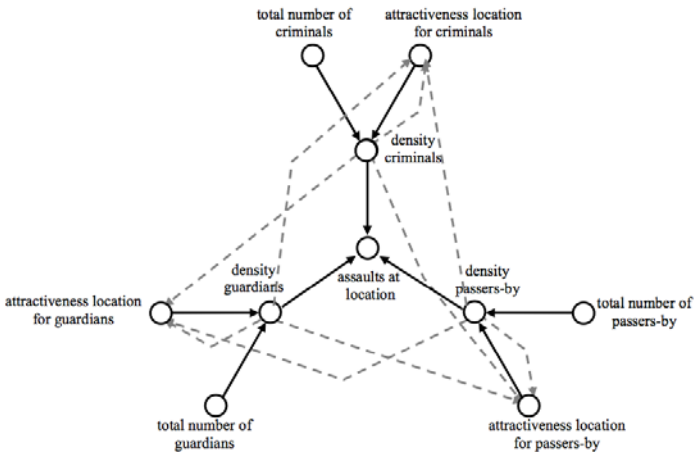

Figure10. Interaction between criminals, police personnel and passers-by [Bosse and Gerritsen 2010].

In the simulation (i) each agent has an identifier to indicate whether the agent is a criminal or civil guard; (ii) the city is divided into different zones and it is possible to know the location of each agent; and (iii) the rate of crime is calculated during every time unit of the simulation. In addition, the agents have: (i) objectives that are connected directly to their individual objective; (ii) information collected in the simulation environment to assist them in capturing criminals; and (iii) strategies to handle the norms.

In order to adopt the *Selfish* strategy to deal with the norms the police personnel used the *Selfish* class (See *Selfish* class in Figure 3), which is one of the extensions of *NormStrategy* class. In the *Selfish* class provided by the JSAN framework a method called *calculate* was implemented, which receives the identity of the agents as a parameter. This method helps the agents to analyze the situation and assess whether compliance with the existing norms can help them to achieve their individual goals.

The *EnvironmentSimulation* class is responsible for managing a set of strategies for the creation of the individual goals of the agents, and the choice of the goals is made by the *GenerateGoalStrategy* abstract class. We used the class *RandomGenerateGoals* and the *generateGoals* method and a set of previously known objectives, to choose certain goals for the policemen randomly. The goals generated for the simulation were: (i) to increase the travel speed of the policemen in the arrest of criminals; (ii) not to put civilian lives at risk; and (iii) to win a medal for relevant services rendered to society. The *EnvironmentSimulation* class is responsible for managing a set of strategies for generating norms represented by the *GenerateNormStrategy* abstract class, which has the abstract method *generateNorms*. This method will generate norms for the simulation given a set of norms previously known by the *RandomGenerateNorms* class provided by the JSAN framework. An example of norms that can be generated has been previously discussed.

To support crime prevention, different strategies regarding the movement of the policeman agents are implemented through extensions of the *MovementStrategy* class,

an extension point provided by JSAN. The *Reactive* and *Anticipatory* classes were used to implement the *Execute* method to define how the strategy will be used. The reactive strategy implemented in the *Reactive* class assumes that the number of agents that need to move to a new location is proportional to the percentage of crimes that recently happened at that place. The *Execute* method of the *Anticipatory* class assumes that the number of policemen that move to a new place of duty is proportional to the predicted density of civilians that are passing through the area. Finally, a new class was created, the *Hybrid* class, in which the *Execute* method sets the number of policeman agents who move to a new location to the average sum of the agents determined by two strategies.

In the *Norm* class attributes were defined so that some elements can be created namely: (i) the sanctions, activation and expiration conditions; (ii) the agent that will be regulated by the norm; and (iii) the deontic concept that is associated with the norm. For example, the norm: "Do not put civilian lives at risk" has the following attributes: (i) the addressees are the policeman agents; (ii) the required deontic concept is obligation; and (iii) a reward for agreeing to a norm is either more guns or support in the mission to capture criminals. If a policeman agent violates a related norm, then the agent receives the associated punishment. For example, there are situations when a policeman agent requests more guns to accomplish a specific capture operation in a dangerous location and that places the agent in a situation riskier than the one allowed by the norm. In this case, this agent is punished according to the norm, by receiving a warning or an order to be temporarily restricted to headquarters to assist other policemen in administrative work.

The scenario in this article consists of four locations L1, L2, L3 and L4 [2], which undergo changes in their attraction factor for local civilians over time. In the example scenario number two (2), all locations start with the same basic attraction value for the calendar where L1 = L2 = L3 = L4 = 0.25. In addition the movement strategy shown in the graphs is anticipatory (see Figure 10). After reaching time 39 in phase two, the base attractiveness of the location L1 and L2 are increased where L1 and L2 = 0.7 and L3 = L4 = 0.25. Such a change in attraction can be caused by an event such as a rock concert or circus coming to town. Later, in phase three, the event is over and moves to another city at time 90, and the attraction value for all locations becomes equal again (0.25). The other parameters were chosen as follows (for all scenarios). The total population is made up of 800 offenders, 400 policemen agents and 4,000 civilians. Initially, these agents are distributed equally over the four locations (that is, each area contains 200 criminals, 100 policemen and 1000 civilians). The speed of the agents is set to 0.5, and the total simulation time is 100 units of time.

To illustrate the development of the crime prevention support application using the JSAN, we present the graphs generated by the extension of the *ReportStrategy* class in Figures 11 through 14 (See section 5.3).

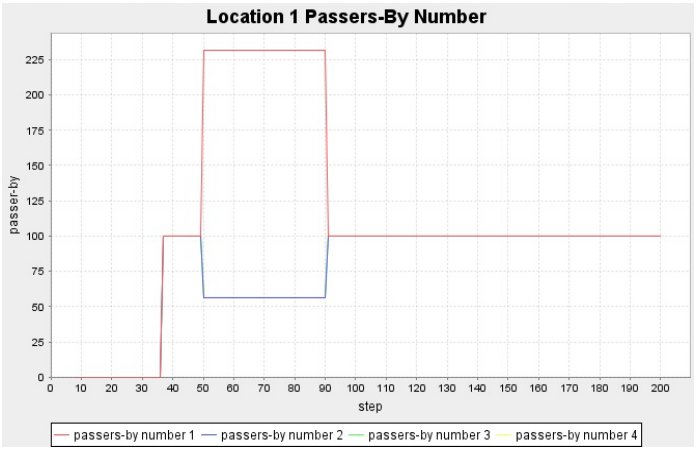

Figure 11. Details about the numbers of civilians.

Figure 11 shows the number of civilians in different locations. In all the figures (11, 12, 13 and 14), the graphs are plotted in different colors: the red line shows the number of civilians in the locality L1, the blue line shows the number in L2, the green line shows the number in L3, and the yellow line shows the number in L4. In the case of Figures 12, 13 and 14, the numbers of civilians in the locations L2, L3 and L4 are the same as the number of civilians in the location L1 (the blue line).

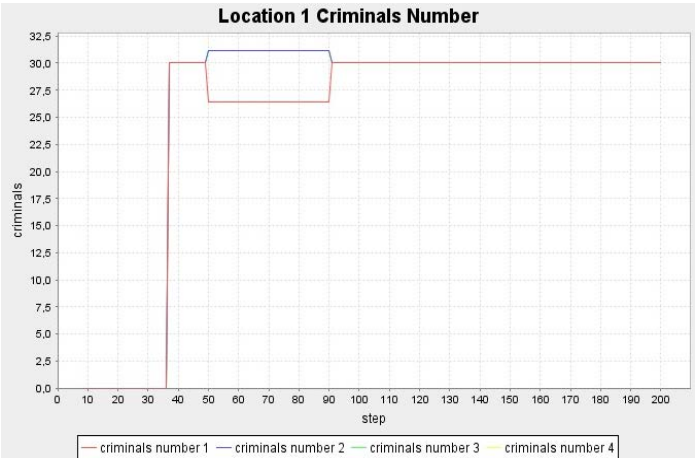

Figure 12. Details about the numbers of criminals.

Figure 12 shows the number of criminals who have moved to the locality L1, because of the arrival of a circus in town, and shows that there is a decrease in the number of criminals in the time period between 39 and 90 because the police agents began to act according to norms and start to move to that location, as seen in Figure 13.

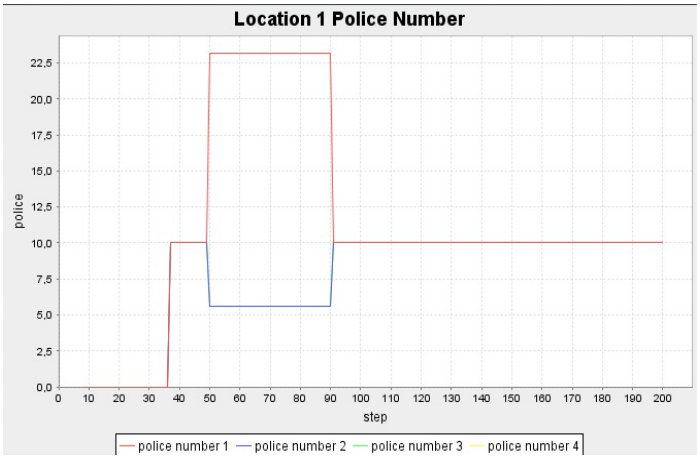

Figure 13. Details about the numbers of policeman agents.

Figure 14 shows the rate of assaults that happened during the simulation time. The crime rate increases in almost all locations until the time instant 39.

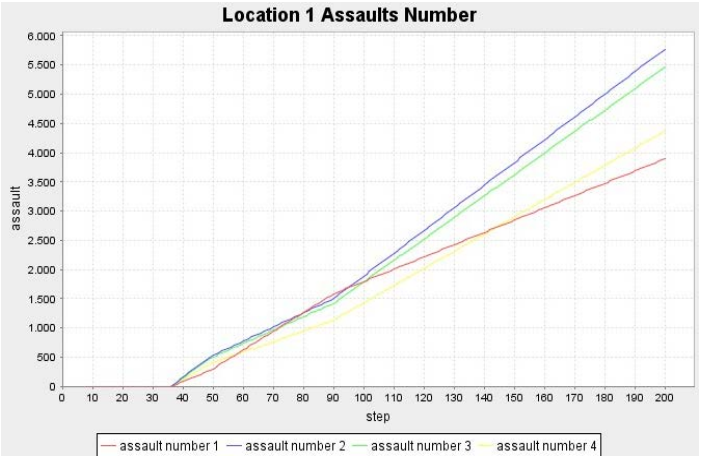

Figure 14. Details about the numbers of crimes.

After the police agents start to move to a specific location based on the norms and begin to help prevent crimes in that location we note that the slope of the curves decrease.

7 Conclusions and Future Work

This paper proposes the JSAN framework, a framework for Normative Agent Java Simulation, to build goal-oriented agents that can reason about norms. The implementation helps agents (i) to check if they should adopt or violate their related norm; (ii)

to evaluate the effects of the fulfillment or violation of the norm on their desires and intentions; and (iii) to select desires and plans based on their choices (i.e., fulfilling or violating a norm).
The applicability of such implementation can be verified by using the scenario presented in Section 6, where in the first scenario, agents are responsible for planning the evacuation of people in hazardous locations [3]. In the second scenario, agents are responsible for arresting criminals [2]. The agents, built according to the proposed implementation, were able to reason about the norms they would like to fulfill, and to select plans related to their intention of fulfilling or violating the norms.

As future work we are in the process of defining an experimental study in order to complete the evaluation of our approach. It is also our aim to study other BDI architectures and platforms to investigate the possibility of extending them to support the development of BDI normative agents. We also plan to implement new mechanisms to deal with different levels of agent autonomy and show how different restriction levels and communities can influence the satisfaction of a norm application [9], [18]. In the current version of the framework the autonomy-related restriction levels were not taken into account. However, the framework can be enhanced with different levels of restrictions, thus offering the possibility of achieving better results in terms of promoting a desirable social order. As a result, agents can work toward the common goals of the society in which they are inserted.

8 References

1. Bordini, R. H.; Hübner, J. F.; Wooldridge, M. Programming Multi-Agent Systemns in AgentSpeak using Jason. [S.l.]: [s.n.], 2007.
2. Bosse, T.; Gerritsen, C. An Agent-Based framework to Support Crime Prevention, AAMAS, Toronto, 2010. 525-532.
3. Cerqueira, S. L. R. et al. Plataforma GeoRisc Engenharia da Computação Aplicada à Análise de Riscos Geo-ambientais. PUC-RIO. Rio de Janeiro, 2009.
4. Criado, N., Argente, E., Noriega, P., and Botti, V. Towards a Normative BDI Architecture for Norm Compliance. COIN@ MALLOW2010, pages 65–81, 2010.
5. Garcia-Camino, A., Rodriguez-Aguilar, J., Sierra, C., Vasconcelos, W.: Norm-oriented programming of electronic institutions. In: AAMAS, 2006.
6. Jadexhomepage,http://jade.tilab.com/.
7. Jennings, N.; Wooldridge, M. Software Agents, IEEE Review,. p. 17-20, 1996.
8. Kollingbaum, M.: Norm-Governed Practical Reasoning Agents. PhD thesis, University of Aberdeen, 2005.
9. Lopez, F. L.; Luck, M.; D'Inverno, M. Constraining Autonomy through Norms, AAMAS, 2002.
10. Lopez, Fabiola López. Social Power and Norms. Diss. University of Southampton, 2003.
11. Lopez, L. F.; Marquez, A. A. An Architeture for Autonomous Normative Agents, IEEE, Puebla, México, 2004.
12. Meneguzzi, F., Luck, M.: Norm-based behaviour modification in bdi agents. In: Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems, 2009.
13. Machado, R. Bordini, R. H. “Running AgentSpeak(L) agents on SIM AGENT”, August 1–3, 2002.

14. Oren, N., Luck, M., Norman, T.: Argumentation for normative reasoning. In: Proc. Symp. Behaviour Regulation in Multi-Agent Systems, pp. 55–60, 2008.
15. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a bdi-architecture. In: Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning, 1991.
16. Rao, A.S.:Agentspeak(l): BDI agents speak out in a logical computable language. In:Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038. Springer, Heidelberg, 1996.
17. Santos Neto, B. F. D.; Lucena, C. J. P. D. JAFF: implementando agentes auto-adaptativos orientados a serviços, Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2010.
18. Santos Neto, A deontic Approach for the Development of Autonomous Agents Normative. Pontifical Catholic University - PUC-Rio – Rio de Janeiro, RJ – Brazil, 2012.
19. Silva, V. T. D.; Lucena, C. J. P. D. Modeling Multi-agente Systems, Communica-tions of ACM, p. 103-108, 2007.
20. Silva, V.: From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. In: JAAMAS, pp. 113–155, 2008.
21. Viana, M. L., Cunha, F. P., Santos Neto, B., Alencar, P., Lucena, C. J. P. A Framework for Supporting Simulation with Normative Agents. WESAAC, 2015. (To Appear).
22. Wooldridge, M. and Jennings, “N. R. Pitfalls of agent-oriented development,” Proceedings of the Second International Conference on Autonomous Agents (Agents'98), ACM Press, pp. 385-391