JSAN: A Framework to Implement Normative Agents

Marx Viana[1], Paulo Alencar[3], Everton Guimarães[2], Francisco Cunha[1], Donald Cowan[3], Carlos Lucena[1]
[1]Pontifical Catholic University - PUC-Rio – Rio de Janeiro, RJ - Brazil
mleles@inf.puc-rio.br, fplacido@inf.puc-rio.br, lucena@inf.puc-rio.br
[2]University of Fortaleza – Unifor – Fortaleza, CE - Brazil
eguimaraes@unifor.br
[3]University of Waterloo – Waterloo, Ontario - Canada
palencar@uwaterloo.ca, dcowan@uwaterloo.ca

Abstract— **Norms have become a promising mechanism to ensure that open multi-agent systems (MASs) produce a desirable social outcome. MASs can be defined as societies in which autonomous agents work to achieve both societal and individual goals. Norms regulate the behavior of agents by defining permissions, obligations and prohibitions, as well as encouraging and discouraging the fulfillment of norms through rewards and punishments mechanisms. Once the priority of software agent is the satisfaction of its own desires and goals, each agent must evaluate the effects associated to the fulfillment or violation of one or more norms before choosing which one should be complied. This paper introduces a framework for normative MASs simulation that provides mechanisms for understanding the impact of norms on an agent and the society to which an agent belongs.**

Keywords. Normative Agents; Multi-agent Systems, Simulation; Norms.

I. INTRODUCTION

Open multi-agent systems (MASs) are societies in which autonomous, heterogeneous and independently designed entities work towards specific goals [9]. In order to deal with autonomy and diversity of interests among the different members, such systems provide a set of norms that is used as a social control mechanism to ensure that a desirable social order in which agents can work is maintained [19]. For the best of our knowledge, norms can be defined as agent-oriented mechanisms for regulating the behavior of agents through the definition of 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]. Although norms are promising mechanisms to regulate the behavior of agents, one should take into account that they are autonomous and, therefore, free to decide to fulfill or violate each system norm. This type of agent reasoning refers to normative strategies [10].

Several approaches [4, 19] have been proposed for the specification and implementation of norms. According to Garcia Camino *et al.* [4], norms constitute a powerful coordination mechanism among heterogeneous agents. The authors propose means to specify and explicitly manage the normative constraints on agents (i.e., permissions, prohibitions and obligations), with which distinct deontic notions and their relationships can be captured. Silva presents a normative language to specify norms and proposes the implementation of such norms by using a rule-based system [19]. The implementation is achieved by automatically transforming the specification of each norm of the system into a set of rules used to govern the behavior of the agents according to the norm. In addition, these works have focused on the definition of parts of an infrastructure that can be used by Belief-Desire-Intention (BDI) agents, which consists of beliefs, desires and intentions as mental attitudes that deliberate human action [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. The main purpose of goal-oriented normative agents consists on achieving their goals and desires while satisfying system norms. Although there are a number of existing agent-oriented platforms such as [1, 5, 11, 14], there is a lack of support of a framework for normative agents.

In this context, we present a framework for Normative Agent Java Simulation (JSAN). This framework aims at providing support to build and operate agents able for dealing with goals, desires and norms – that is, agents that 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 new functions, it is possible to build BDI agents that can check whether 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 [20]. The reminder of this paper is organized as follows. Section 2 focuses on the representation of norms, while Section 3 presents the JASON Platform. Section 4 discusses related work. In Section 5 the JSAN framework is detailed 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.

II. REPRESENTATION OF NORMS

According to Lopez [9], norms are designed to regulate the behavior of agents, and therefore, a norm definition should include the address of the agent being regulated. When the norm need be applied, the nature of the norm (permission, obligation or prohibition), as well as the

(DOI reference number: 10.18293/SEKE2015-165)

consequences of either fulfilling or violating the norm (reward or punishment) should be described. In this work, we use the norm representation described in [18], which is composed the representation of an element *norm* – it contains many different properties. Those properties are briefly described in Table 1. For example, the property *Addressee* is used to specify the agents or roles responsible for fulfilling the norm.

In order to understand the definition of norms and their representation better, imagine a Fireman Commander agent is leading the rescue of civilians who are in hazardous areas. This agent is responsible for regulating the behavior of all fireman agents and the usage of the resources available to them (e.g., helicopters, troops and land-based helicopters) – we are assuming the resources are limited. In addition, each fireman agent should perform a rescue according to specific norms. Eventually, a norm is sent to each fireman agent with the following state: "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; (iii) when an agent agrees to a norm that agent will receive a reward. In this case, the reward could be either air or ground transportation during the agent's mission. If the fireman agent does not follow a certain norm directed at him in the environment, after violating the norm, this agent receives the punishments associated with the norm. For example, there are situations when a fireman agent requests aircraft support to accomplish a specific rescue operation activity that places him or her in a degree of risk above the one allowed by the norm. In this case, a punishment (e.g. a warning or an order that he should be temporarily restricted to headquarters to assist other rescuers) associated with the norm will be applied for the agent. Note 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, because of the costs.

TABLE I. 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 are the 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.

III. THE JASON PLATFORM

The JASON platform enables the development and implementation of Belief, Desire and Intention (BDI) agents. In addition, the platform uses a language called AgentSpeak for implementing agents. Figure 1 illustrates how JASON interprets AgentSpeak programs [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 a literal to be added or deleted, and the intention structure required to change the belief. A single event is chosen by the *Event selection function* (SE) and the 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.

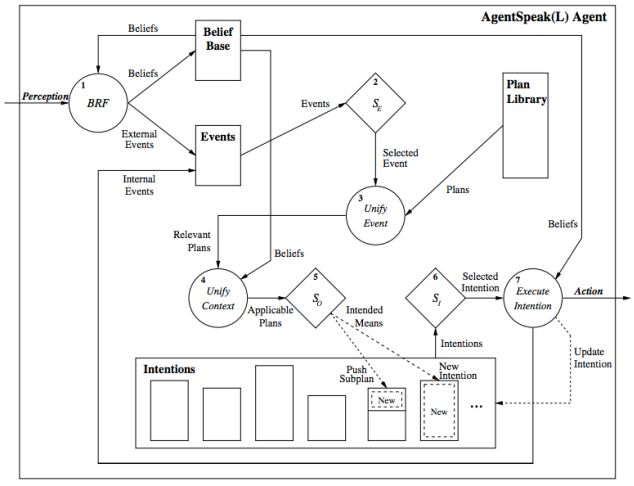

Figure 1. An Interpretation Cycle of an AgentSpeak Program [12].

IV. RELATED WORK

Some approaches [3, 8, and 10] have been proposed in the literature to develop agents that evaluate the effects of fulfilling or violating norms. For instance, the n-BDI architecture defined by Criado *et al.* [3] presents a model for designing agents capable of operating in environments governed by norms. Basically, the 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 governing a specific objective. However, it is not clear in this approach how the properties

of a norm can be evaluated. In addition, the approach does not support a strategy to deal with conflicts between norms.

In turn, Meneguzzi and Luck [10] proposed a formal model using the Z specification language, for modeling agents that achieve their objectives based on the systems' norms. For instance, an agent created from such a model should be 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 (i) identifying and resolving of conflicts between norms; (ii) checking fulfilled or violated norms; and (iii) showing the influence of norms on the plan selection process and intentions of the agents.

Finally, Lopes *et al.* [8] defined a set of strategies that can be adopted by agents to deal with norms as follows: *Pressured*, *Opportunistic* and *Selfish.* For instance, 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. Finally, the *Selfish* strategy is the combination of the *Pressured* and *Opportunistic* strategies. Although this work provides some mechanisms for agents that handle standards, the authors provide a framework that can be extended to create simulations of normative multi-agent systems by including new strategies. In addition, this work cannot extend mechanisms to collect information during the simulations or mechanisms for generating norms and goals of the agents.

V. JSAN – A Framework for Normative Agent Java Simulation

This section describes the main concepts required to understand the framework proposed in this paper. In addition, we provide an overview of JSAN framework and discuss the different its different components, including the kernel (frozen-spots) and flexible points (hot-spots) [21].

A. The Framework

The JSAN framework is implemented using software agents, as illustrated in Figure 2. The JSAN extends the JASON framework [1], which is an interpreter for an extended version of the AgentSpeak language [15]. For the best of our knowledge, the AgentSpeak language is an agent-oriented programming language that supports the creation of BDI agents. In addition, JSAN framework provides a platform for development of multi-agent systems (MASs). The framework comprises three main functions: (i) agents responsible for dealing with norms; (ii) visualizations for agent simulations involving norms; and (iii) creation of norms. In order to implement normative agents, it is necessary to instantiate the JSAN framework. For extending the framework, developers should implement the agents' goals, movement strategies (different agents' movement strategies) and normative strategies (to represent how the agents deal with norms). JSAN already supports a normative process composed of four activities, as detailed in Section IV.B.

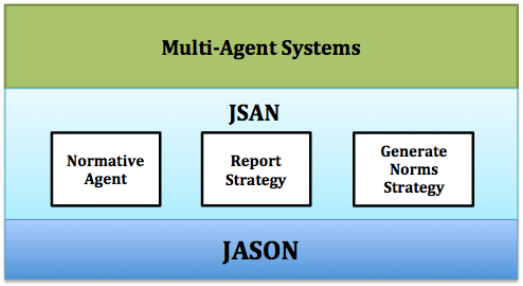

Figure 2. The JSAN Architecture.

B. The JSAN Architecture

The JSAN framework supports creating simulations that show the impact of norms on normative agents. Thus, the JSAN framework provides means for the implementation of Normative Agents (see Figure 3). For this purpose, there is a need to: (i) create the goals of an agent; (ii) create different movement strategies of an agent; and (iii) provide normative strategies to represent how the agents deal with norms.

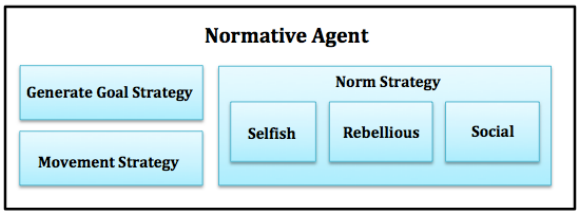

Figure 3. The Internal Structure of the Normative Agents provided by JSAN.

JSAN provides a normative application process to agents capable of reasoning about norms, represented by the *NormStrategy* class, which is composed of four activities (see Figure 4): *Norm Awareness* (Section IV.B.1), *Norm Adoption* (Section IV.B.2), *Norm Deliberation* (Section IV.B.3) and *Norm Impact* (Section IV.B.4).

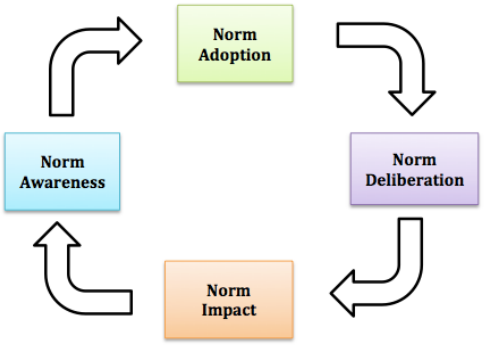

Figure 4. The Normative Strategy provided by the JSAN Framework.

The normative application process is based on the process proposed in [10]. Note that, although JSAN already

provides such process, it is possible to define alternative processes using the *NormStrategy* class. In addition, it is possible to implement different activities (or steps for the process) by extending the class *NormStrategy*. Once an active norm is defined in the environment for a specific software agent, some steps should be performed in the process in order to apply the norm. Each of those steps is described in the following.

1) Norm Awareness - the software agent identifies which norms are active in the environment, and hence, those norms are assigned to specific agentes.

2) Norm Adoption - the agents recognise their responsibilities towards other agents by internalising the norms where their responsibilities are specified.

3) Norm Deliberation - in order to execute a specific norm, an agent should access to diferent information: (i) the goals that should be hindered by satisfying the normative goals; and (ii) the goals they might benefit from the associated rewards.

4) Norm Impact - after the agents execute the norm (step 3), the goals of the agents will be updated. After that, the cycle continues and the agent starts identifying other norms that should be addressed.

It is important to note the steps need to take into account not only the agents' goals, but also the mechanisms available to avoid the violation of norms (e.g. rewards and punishments). That is, agents should consider consider the so called social pressure (i.e., agents recognized their responsibilities before other agents), of norms before making any decision regarding norms.

C. Hot-Spots and Frozen-Spots

As previously mentioned, JSAN is an extension of JASON framework, and therefore, they share the same core and hot-spots. The process used for the communication between software agents, and the identifiers of agents are examples of hot-spots that JSAN inherited from JASON. In addition, JSAN define other specific hot-spots, which are:

Environment (*EnvironmentSimulation* class): The *ExecuteAction* method was extended from the Environment. Whenever an agent tries to perform an essential action, the agent's identification and its chosen actions are passed to this method. For this reason, the *ExecuteAction* method must verify that the action is valid and then perform the action. The action will possibly change an agent's perception. If this method returns *true* it means that the action was performed successfully.

Normative Agent (*NormativeAgent* class): By extending such a class and implementing the *execute* method it is possible to define different algorithms to execute the plans of an agent.

Created Norms (*GenerateNormsStrategy* 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 agents to deal with norms, and proceed to execute the activities of the normative application process. JSAN already 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 of 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

VI. Usage Scenario: Evacuating people from areas of risk

The need to build platforms to assist both experts on risk analysis as well as experts in planning the evacuation of civilians located in areas of risk is currently a critical problem, especially in large cities that have grown in an unplanned and often disorderly manner [2]. These cities experience many circumstances in which people need to be rescued from dangerous areas as a result of floods, landslides and other natural phenomena. Landslides, for example, are difficult to predict since they depend on many factors such as climate, soil properties, and humidity and the specific relationship between them. The annual number of landslides is estimated to be in the thousands, and the associated infrastructure damage resulting from them is worth billions of dollars [16]. Planning evacuations from areas of risk can be assisted by simulations using the JSAN framework. These simulations implement scenarios representing hazardous situations. For example, a simulation can be used as a way to examine different strategies that can be adopted by the firemen agents, who are regulated by norms, in order to rescue civilian personnel.

A. Overview

The implemented simulation is composed of firemen agents, civilian agents and norms, as illustrated in Figure 5. The goal of the firemen agents is to rescue people who are at risk. The civilian agents do not deal with norms. The structure of the norms is created to extend the JSAN *GenerateNormStrategy* class. This structure was proposed in [17]. These simulations are normative multi-agent systems that receive data containing information about (i) a hazardous area, such as weather conditions, (ii) civilian personnel, (iii) ways of saving civilians by removing them from these locations (with troops, land vehicles or aircraft), (iv) norms that firemen agents must follow during the rescue operation, and (v) rescue plans used in the simulation. Through the simulations, it is possible to find different solutions the firemen agents can follow in order to evacuate civilians from these hazardous areas.

To deal with these problems and understand the related norms, the fire-fighter agents have: (i) a set of objectives that

is connected directly to their individual satisfaction; (ii) an information base of facts collected by the simulation environment to help characterize risky locations; and (iii) a base of strategies used to deal 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 aims at analyzing 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 the agent will receive.

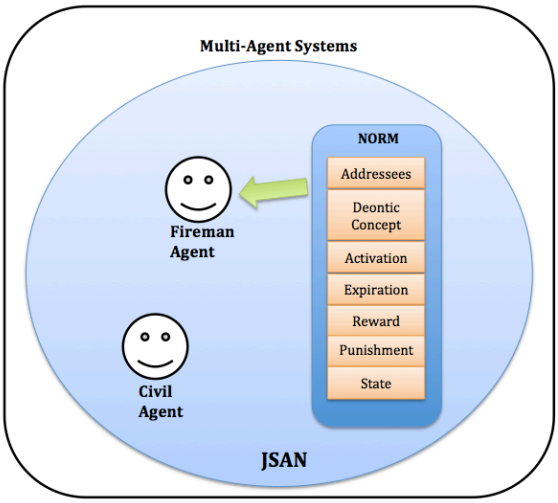

Figure 5. Conceptual model of the usage scenario.

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 firemen agents based on their plans. The objectives generated for the simulation were: (i) to save civilians in hazardous areas; (ii) not to put civilian lives at risk; and (iii) to receive aircraft support. The *generateNorms* method of the *GenerateNormsStrategy* class was used to create norms in the environment aiming only at firemen agents, namely a specific set of agents provided by the *PerceptGenerateNorms* class in the JSAN framework. With the ability to specify how to move the firemen agents, the *MovementStrategy* class has been extended to implement the method to specify the specific form of the firemen agents' movement. For this purpose, we used the *ReactiveMovement* class, which extends the *MovementStrategy* class. The movement strategy of the firemen agents checks whether there is any civilian at risk. If there is, they must make the rescue, and they place their lives at risk. To manage that risk a fireman agent can ask for help that will be sent in the form of land vehicles, aircraft, or even fire.

In the *Norm* class (see Table 1), the attributes were defined so that they can create the sanctions, activation and expiration conditions, which are elements that will be regulated by the norm, and also 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 was used to check the social contribution of a norm in case the norm is fulfilled or violated by a firemen agent. The norm-related information is also tested, that is, (i) the rewards and punishments linked to norms; (ii) the deontic concept associated with norms; and (iii) the plans adopted to comply with the norm.

```
Beginning of the simulation
-----------------------
[commanderAgent] Reported that the weather is bad
[commanderAgent] Reported that workers are in a dangerous place
{fireManAgent} Received information from workers at risk:hazardousLocation
```

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

In Figure 6, the Fireman Commander agent receives a message about bad weather and then receives a second message about some civilians in a hazardous location. After receiving the latter message, the Fireman Commander agent sends an alert to the firemen agents saying that information about the existence of civilians in a hazardous area was received.

```
All Plans
-----------------------------
Plan: 1 evacuateWorkers(A)

Plan: 2 useHelicopters(A)

Plan: 3 useTroops(A)

Plan: 4 getMoreTroops(MT)

Plan: 5 getMoreHelicopters(MH)

Plan: 6 getLandBasedHelicopters(LBH)

Plan: 7 returnLandBasedHelicopters(RLBH)

Plan: 8 returnHelicopters(RH)

Plan: 9 returnTroops(RT)
```

Figure 7. Emergency plans described in the simulation.

In this case, the rescue operation plans involve the use of aircraft, ground vehicles, and these plans include asking if the firemen agents need the air or ground support (see Figure 7). These plans are related to the goal: "protect the lives of civilians in hazardous areas."

```
Instantiated Norm :
-----------------------------
Deontic Concept: obligation

action : useHelicopters(A)

Rewards: getMoreTroops(X)

Rewards: getMoreHelicopters(Y)


Punishments:
Deontic Concepts: obligation
 action : returnTroops(X)

Norm Reward: getMoreTroops(X)
Reward Norm Contribution For Fulfilling +1: 1
Norm Reward: getMoreHelicopters(Y)
Reward Norm Contribution For Fulfilling +1: 2
```

Figure 8. Norm created in the simulation.

As part of the norm instantiation, (i) the deontic concept in this case is an obligation; (ii) 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; (iii) the norm is enabled if there is any person at risk and the norm is disabled when all civilians are safe; and (iv) the element that the norm regulates is the action of using aircraft. The norm Rewards in Figure 8 are the rewards associated with norm and Reward Norm shows how the agent will get the reward if the agent complies with the norm.

Finally Figure 9 illustrates the specific plans the firemen agents decide to use after the norm has been activated because of the existence of civilians in hazardous areas risk.

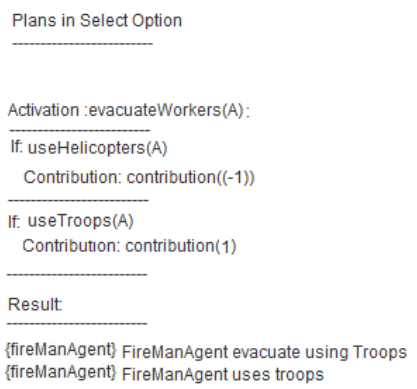

Figure 9. Firemen agents dealing with active norms in the usage scenario.

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 in the simulation firemen agents use the Selfish strategy to deal with the norms, they decided to evacuate civilians who were in this hazardous area using ground vehicles.

VII. Conclusion 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 the norm; (ii) to evaluate the effects of the fulfillment or violation of the norm on their desires and intentions; and (iv) to select desires and plans according to their choices of fulfilling or violating a norm. The applicability of such an implementation can be verified by using the scenario presented in Section VI, where agents are responsible for planning the evacuation of people that are in hazardous locations [2]. The agents were able to reason about the norms they would like to fulfill, and to select plans meeting an agent's intention of fulfilling or violating the norms.

For future work we are defining an experimental study in order complete the evaluation of our approach. It is also our aim to study other BDI architectures and platforms in order 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 levels of restriction and communities can influence the satisfaction of a norm application [8, 17]. 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 to achieve better results in promoting a desirable social order and, as a result, agents can work in function of the common goals of the society in which they are inserted.

References

[1] Bordini, R. H.; Hübner, J. F.; Wooldridge, M. Programming Multi-Agent Systemns in AgentSpeak using Jason. [S.l.]: [s.n.], 2007.

[2] Cerqueira, S. L. R. et al. Plataforma GeoRisc Engenharia da Computação Aplicada à Análise de Riscos Geo-ambientais. PUC-RIO. Rio de Janeiro, 2009.

[3] Criado, N., Argente, E., Noriega, P., and Botti, V. Towards a Normative BDI Architecture for Norm Compliance. COIN@ MALLOW2010, pages 65–81, 2010.

[4] Garcia-Camino, A., Rodriguez-Aguilar, J., Sierra, C., Vasconcelos, W.: Norm-oriented programming of electronic institutions. In: AAMAS, 2006.

[5] Jadexhomepage,http://jade.tilab.com/.

[6] Jennings, N.; Wooldridge, M. Software Agents, IEEE Review,. p. 17-20, 1996.

[7] Kollingbaum, M.: Norm-Governed Practical Reasoning Agents. PhD thesis, University of Aberdeen, 2005.

[8] Lopez, F. L.; Luck, M.; D'Inverno, M. Constraining Autonomy through Norms, AAMAS, 2002.

[9] Lopez, Fabiola López. Social Power and Norms. Diss. University of Southampton, 2003.

[10] Lopez, L. F.; Marquez, A. A. An Architeture for Autonomous Normative Agents, IEEE, Puebla, México, 2004.

[11] Meneguzzi, F., Luck, M.: Norm-based behaviour modification in bdi agents. In: Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems, 2009.

[12] Machado, R. Bordini, R. H. "Running AgentSpeak(L) agents on SIM AGENT", August 1–3, 2002.

[13] Oren, N., Luck, M., Norman, T.: Argumentation for normative reasoning. In: Proc. Symp. Behaviour Regulation in Multi-Agent Systems, pp. 55–60, 2008.

[14] 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.

[15] 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.

[16] 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.

[17] Santos Neto, A deontic Approach for the Development of Autonomous Agents Normative. Pontifical Catholic University - PUC-Rio – Rio de Janeiro, RJ – Brazil, 2012.

[18] Silva, V. T. D.; Lucena, C. J. P. D. Modeling Multi-agente Systems, Communica-tions of ACM, p. 103-108, 2007.

[19] 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.

[20] 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).

[21] 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, 1998.