Developing Goal-Oriented Normative Agents: The NBDI Architecture

Baldoino F. dos Santos Neto[1], Viviane Torres da Silva[2], and Carlos J.P. de Lucena[1,⋆]

[1] Computer Science Department, PUC-Rio, Rio de Janeiro, Brazil
[2] Computer Science Department, Fluminense Federal University (UFF)
Rio de Janeiro, Brazil
{bneto,lucena}@inf.puc-rio.br,
viviane.silva@ic.uff.br

Abstract. In open multi-agent systems norms are mechanisms used to restrict the behaviour of agents by defining what they are obligated, permitted or prohibited to do and by stating stimulus to their fulfillment such as rewards and discouraging their violation by pointing out punishments. In this paper we propose the NBDI architecture to develop goal-oriented normative agents whose priority is the accomplishment of their own desires while evaluate the pros and cons associated with the fulfillment or violation of the norms. The BDI architecture is extended by including norms related functions to check the incoming perceptions (including norms), select the norms they intend to fulfill based on the benefits they provide to the achievement of the agent's desires and intentions, and decide to cope or not with the norms while dropping, retaining or adopting new intentions. The applicability of our approach is demonstrated through an non-combatant evacuation scenario implemented by using the Normative Jason platform.

Keywords: Norms and BDI agents.

1 Introduction

Normative regulation is a mechanism that aims to cope with the heterogeneity, autonomy and diversity of interests among the different members of an open multi-agent system establishing a set of norms that ensures a desirable social order [5].

Such norms regulate the behaviour of the agents by indicating that they are obligated to accomplish something in the world (obligations) [6], permitted to act in a particular way (permissions) and prohibited from acting in a particular way (prohibitions) [6]. Moreover, norms may define rewards to their fulfillment and may state punishments in order to discourage their violation[6].

In this paper we consider that agents are goal-oriented entities that have the main purpose of achieving their desires while trying to fulfill the system norms. In this context, the paper presents an abstract architecture to build agents able to deal with the norms

⋆ The present work has been partially funded by the Spanish project "Agreement Technologies" (CONSOLIDER CSD2007-0022,INGENIO 2010) and by the Brazilian research councils CNPq under grant 303531/2009-6 and FAPERJ under grant E-26/110.959/2009.

J. Filipe and A. Fred (Eds.): ICAART 2011, CCIS 271, pp. 176–191, 2013.

of a society in an autonomous way. The NBDI (Norm-Belief-Desire-Intention) architecture extends the BDI (Belief-Desire-Intention) architecture [8] by including norms related functions to support normative reasoning. The agents built according to the proposed architecture have: *(i)* a review function of norms and beliefs used to check the incoming perceptions (including norms), *(ii)* a norm selection function to select the norms they intend to fulfill based on the benefits they provide to the achievement of the agent's desires and intentions, and to identify and solve conflicts among the selected norms, and *(iii)* a norm filter where the agents decide to cope or not with the norms while dropping, retaining or adopting new intentions.

We demonstrate the applicability of the NBDI architecture through a non-combatant evacuation scenario where the tasks related to review, select and filter norms are implemented by using the Normative Jason platform [7] that already provides support to the implementation of BDI agents and a set of normative functions able to check if the agent should adopt or not a norm, evaluate the pros and cons associated with the fulfillment or violation of the norm, check and solve conflicts among norms, and choose desires and plans according to their decisions of fulfilling or not a norm.

The paper is structured as follows. In Section 2 we outline the background about norms that is necessary to follow the paper. In Section 3 we present the non-combatant evacuation scenario where norms are defined to regulate the behaviour of rescue agents. In 4 the proposed NBDI normative agent architecture is explained and exemplified by using the proposed scenario. Section 5 demonstrates the applicability of the NBDI architecture. Section 6 summarizes relevant related work and, finally, Section 7 concludes and presents some future work.

2 Norms

In this work, we adopt the representation for norms described in [7], as shown below:

norm (*Addressee*, *Activation*, *Expiration*, *Rewards*,*Punishments*, *DeonticConcept*, *State*)

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

3 Scenario: Rescue Operation

The applicability of the architecture proposed in this paper is demonstrated by using the simplified non-combatant evacuation scenario. In such scenario agents have the goals to plan the evacuation of members of a Non-Governmental Organisation (NGO) that are in hazardous location and, to do so, they can use different resources that help to evacuate the members, such as: *(i)* helicopters, *(ii)* troops and *(iii)* land-based helicopters. Considering that such resources are limited, we have a *Commander Agent* that is responsible

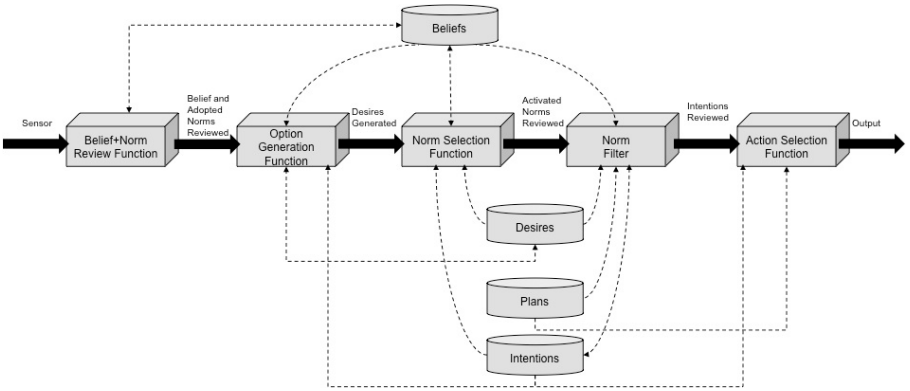

Fig. 1. NBDI Architecture

to control the use of the resources regulating the behaviour of the agents according to the norms 1, 2 and 3:

Norm 1

Addressee. *Rescue Entity*.
Activation. NGO workers are stranded in a hazardous location.
Expiration. NGO workers are stranded in a safe location.
DeonticConcept. Obligation.
State. To evacuate NGO workers.
Rewards. The *Commander Agent* gives more troops to *Rescue Entity*.
Rewards. The *Commander Agent* gives land-based helicopters to *Rescue Entity*.
Punishments. (obligation) *Rescue Entity* is obligated to return to the *Commander Agent* part of their troops.

Norm 2

Addressee. *Rescue Entity*.
Activation. The weather is bad.
Expiration. The weather is good.
DeonticConcept. Prohibition.
State. To evacuate NGO workers.
Punishments. (obligation) *Rescue Entity* is obligated to return to the *Commander Agent* part of their helicopters or land-based helicopters.

Norm 3

Addressee. *Rescue Entity*.
Activation. The weather is bad.
Expiration. The weather is good.
DeonticConcept. Prohibition.
State. To use helicopters.
Rewards. The *Commander Agent* gives more troops or land-based helicopters to *Rescue Entity*.
Punishments. (obligation) *Rescue Entity* is obligated to return to the *Commander Agent* part of their troops.

4 NBDI Architecture

The NBDI (Norm-Belief-Desire-Intention) architecture extends the BDI architecture to help agents on reasoning about the system norms. Norm is considered as a primary concept that influences the agent while reasoning about its beliefs, desires and intentions. The extensions we have made are represented in the NBDI architecture by the following components, as illustrated in Figure 4: *Belief+Norm Review Function*, *Norm Selection Function*, *Norm Filter* and *Plans* base[1] (that stores the plans of the agent).

In a nutshell, the NBDI architecture (Figure 4) works as follows. The agent perceives information about the world by using its sensors. The sensed information is the input of the *Belief+Norm Review Function*, an extension of *Belief Review Function* [8] defined in the BDI architecture that is responsible for reviewing the *Beliefs* base taking into account the current perception and ones already stored in the base.

In this work, we consider that norms are also stored in the *Beliefs* base, so, besides performing the original functionality of the *Belief Review Function*, the *Belief+Norm Review Function* is also responsible for: *(i)* in case the current perception is a norm, reviewing the sets of adopted norms by comparing the information loaded in the new norm with the norms and beliefs already stored in the base; and *(ii)* updating the sets of adopted and activated norms, considering that some may become active and others inactive due to the incoming perceptions.

Next, the *Option Generation Function* updates the agent's desires, and also their priorities. Such adaptation must consider both agent's current beliefs and intentions, and must be opportunistic, i.e., it should recognize when environmental circumstances change advantageously to offer the agent new ways of achieving intentions, or the possibility of achieving intentions that were otherwise unachievable [8]. Note that this function works exactly as the original function described in the BDI architecture. This function does not consider the norms stored in the *Beliefs* base while updating the agent desires because the agent must be able to generate new desires or adapt the existing ones without the influence of the norms. Our architecture considers that the agent is an *autonomous goal-oriented entity that fulfils the system norms if it decides to do so*.

After reviewing the beliefs, desires, activated and adopted norms, the *Norm Selection Function* is executed in order to *(i)* evaluate the activated norms in order to select the ones that the agent has the intention to fulfil; and *(ii)* identify and solve the conflicts among these norms.

Next, the *Norm Filter*, an extension of *Filter* [8] defined in the BDI architecture, selects the desires that will become intentions taking into account the norms the agent wants to fulfil. The plans that will achieve the intentions are also selected by following the norms the agent wants to fulfil.

Finally, the *Action Selection Function* is responsible for performing the actions specified by the intention. The next subsections detail the components added to the original BDI architecture, the one that was extended and a set of algorithms that demonstrate how such components can be implemented.

[1] Plans are composed by actions and states that the agent has the desire to achieve.

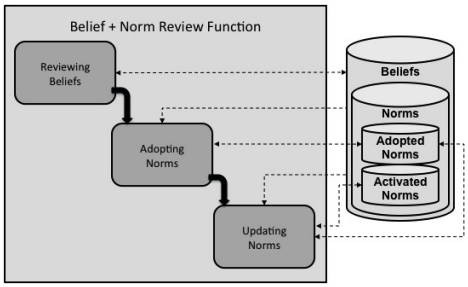

Fig. 2. Belief+Norm Review Function

Algorithm 1. Adopting Norms

Require: *Beliefs* base N: norms stored in the beliefs base
Require: agent: informations about the agent, such as: name and role
Require: *Beliefs* base NA: adopted norms stored in the beliefs base
Require: NN: new norms

```
1:  for all newNorm in NN do
2:      x = true
3:      for all n in N do
4:          if (n == newNorm) then
5:              x = false
6:          end if
7:      end for
8:      if x ∩ ((agent.Name == newNorm.Addresse) ∪ ((agent.Role == newNorm.Addresse)) then
9:          NA.add(newNorm)
10:     end if
11: end for
```

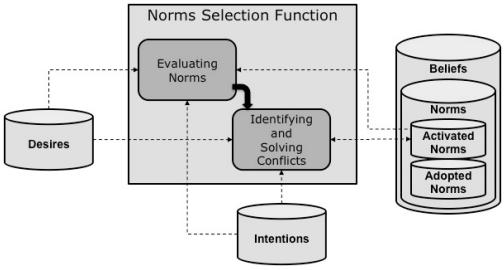

Fig. 3. Norm Selection Function

4.1 Belief+Norm Review Function

Besides performing the original functionality of the *Belief Review Function*, which is the revision of the beliefs (represented by *Reviewing Beliefs* task), the *Belief+Norm Review Function* (Figure 2), helps the agent on recognizing its responsibilities towards

Algorithm 2. Updating Norms

```
Require: Beliefs base NAD: adopted norms stored in the beliefs base
Require: Beliefs base NAC: activated norms stored in the beliefs base
Require: P: new perceptions
 1: for all n in (NAD ∪ NAC) do
 2:   for all p in P do
 3:     if (n.Activation.unify(p)) then
 4:       NAD.remove(n)
 5:       NAC.add(n)
 6:     else
 7:       if (n.Expiration.unify(p)) then
 8:         NAC.remove(n)
 9:         NAD.add(n)
10:       end if
11:     end if
12:   end for
13: end for
```

other agents by adopting new norms that specify such responsibilities (represented by *Adopting Norms* task) and updates the sets of activated and adopted norms (represented by *Updating Norms* task).

Adopting Norms (AN). This task recognizes from the set of receiving perceptions the ones that describe norms. After recognizing the norms, such function reviews the set of adopted norms applying the following verifications: (i) it checks if the new norm unifies with one of the norms already adopted, i.e., if the incoming norm already exists in the agent *Belief Base* (*Algorithm 1* from line 2 to 7), and (ii) it verifies if the agent is the addressee of the norm, i.e., if the field *Addressee* of the new norm unifies with the agent role or agent name, also stored as a belief in the *Belief Base* (*Algorithm 1* line 8). Finally, such function updates the set of adopted norms in the *Belief Base* if the new norm does not already exist and the agent is the addressee of the norm (*Algorithm 1* line 9).

With the aim to exemplify the use of this task, let's consider the scenario presented in Section 3 where two groups of agents are leaded by *Agent A* and *Agent B* playing the role *Rescue Entity*. When these entities receive information about the three system norms, the **AN** task is executed checking if the norms are not stored yet in the agent's belief base and comparing the addressee information with the role being played by the agents.

Updating Norms (UN). UN task updates the set of activated and adopted norms checking if the fields *Activation* and *Expiration* of the norm unifies with the beliefs of the agent. If the activation conditions unify with the beliefs, the adopted norm is activated (*Algorithm* 2 from line 3 to 6). If the expiration conditions unify with the beliefs, the norm is deactivated and stored as an adopted norm (*Algorithm* 2 from line 7 to 10).

Following the example above, if the weather of the area operated by one of the two rescue entities is bad, both norms 2 and 3 are activated, since the activation condition of both norms is "The weather is bad". If the norms are activated, the rescue entity must not rescue NGO members and must not use helicopters. Both norms are deactivated when the expiration condition unifies with the information about a good weather stored in the agent's belief base.

4.2 Norm Selection Function

The main goal of the *Norm Selection Function* (Figure 3) is to select the norms that the agent has the intention to fulfil. In order to do this, such function performs two tasks: 1) *Evaluating Norms* and 2) *Identifying and Solving Conflicts*. The first task helps the agent on selecting, from the set of activated norms, the norms that it has the intention to fulfil and the ones it has the intention to violate. The function evaluates the benefits of fulfilling or violating the norms, i.e., it checks how close the agent gets of achieving its desires if it decides to fulfil or if it decides to violate the norms. The function groups the activated norms in two sub-sets: norms to be fulfilled and norms to be violated. Finally, the second task of this function identifies and solves the conflicts among the norms that the agent has the intention to fulfil and among the ones that the agent has the intention to violate.

Evaluating the Norms (EN). In order to evaluate the benefits of the fulfilment or violation of a norm according to the agent's desires and intentions, the steps below should be followed: **(Step 1)** In case of obligations, it checks if the state described in the norm is equal to one of the states that the agent has desire (or intention) to achieve. In affirmative cases, the contribution is positive and the function *g(n.DeonticConcept, n.State)* returns a value indicating the level of norm's contribution that is calculated according to the priority of the desire that is similar to the state described by norm. The function receives as parameters *n.DeonticConcept* representing the deontic concept type, i. e., obligation or prohibition, and *n.State* representing the state that is been regulated. In any other case, the contribution is zero since it does not disturb the achievement of the agent's desires or intentions. Such step is represented in *Algorithm* 3 from line 2 to 8. **(Step 2)** In case of prohibitions, it checks if the state described in the norm is equal to one of the states that the agent has desire (or intention) to achieve. In affirmative cases, the contribution is negative since it disturbs the achievement of the agent's desires or intentions and the function *g(n.DeonticConcept, n.State)* calculates the absolute value of the contribution. In any other case, the prohibition will contribute neutrally. Such step is represented in *Algorithm* 3 between lines 9 and 15. **(Step 3)** After analyzing the state being regulated, this step considers the influence that the rewards have to the achievement of the agent's desires. We consider that rewards can never influence the agent negatively but always positively or neutrally since they give permissions to achieve a set of states. Such step is represented in *Algorithm* 3 by line 16. Function *r(n.Rewards)* verifies the desires (or intentions) that are equal to the rewards and returns a value indicating the contribution that is the sum of the priorities of the agent's desires benefited by the rewards. **(Step 4)** Finally, the punishments are evaluated in order to check if they will

influence the achievement of the agent's desires and intentions negatively or positively. **(Step 4.1)**In case the punishment states a prohibition and the state being prohibited is one of the agent desires or intentions, the punishment will influence negatively since it will disturb the agent of achieving one of its desires. If it is not the case, the punishment will not influence. Such step is represented in *Algorithm* 4 from line 2 to 8. Function *g(n.punishments.DeonticConcept)* returns a absolute value indicating the contribution of the fulfilment of the prohibition to the achievement of the agent's desires and intentions. **(Step 4.2)** In case the punishment states an obligation and the state being obliged is one of the agent desires or intentions, the punishment will influence positively (or neutrally) since such state will already be achieved by the agent. If it is not the case, the punishment will not influence. Such step is represented in *Algorithm* 4 from line 9 to 15. Function *g(n.punishments.DeonticConcept)* returns a value indicating the contribution of the fulfilment of the obligation to the achievement of the agent's desires and intentions.

Algorithm 3. Evaluating the fulfilment

```
Require: Desires base D
Require: Intentions base I
Require: Beliefs base N: norms stored in the beliefs base
 1: x = 0
 2: if n.DeonticConcept == Obligation then
 3:     for all d in (D ∪ I) do
 4:         if n.State == d then
 5:             x = x + g (n.DeonticConcept, n.State)
 6:         end if
 7:     end for
 8: end if
 9: if n.DeonticConcept == Prohibition then
10:     for all d in (D ∪ I) do
11:         if n.State == d then
12:             x = x - g (n.DeonticConcept, n.State)
13:         end if
14:     end for
15: end if
16: x = x + r (n.Rewards)
17: return x
```

Note that it is necessary to individually check the contribution of the fulfilment and violation of each norm to the achievement of the agent desires (or intentions). Such step is represented in *Algorithm* 5 in the lines 2 and 3.

After checking how each norm can contribute to the achievement of the agent's desires and intentions, the function helps the agent on deciding which are the norms that it should fulfil, i.e., the norms whose contribution coming from its fulfilment is greater than the contribution coming from its violation. Steps 1, 2 and 3 return the contribution of the norm if the agent chooses to fulfil it (*Algorithm* 3 line 17) and 4.1 and 4.2 return

Algorithm 4. Evaluating the violation

```
Require: Desires base D
Require: Intentions base I
Require: Beliefs base N: norms stored in the beliefs base
 1: x = 0
 2: if n.punishment == Prohibition then
 3:     for all d in (D ∪ I) do
 4:         if n.punishment.state == d then
 5:             x = x - g (n.punishments.DeonticConcept, n.punishments.State)
 6:         end if
 7:     end for
 8: end if
 9: if n.punishment == Obligation then
10:     for all d in (D ∪ I) do
11:         if n.punishment.state == d then
12:             x = x + g (n.punishments.DeonticConcept, n.punishments.State)
13:         end if
14:     end for
15: end if
16: return x
```

Algorithm 5. Reasoning about norms (Main)

```
Require: fulfilSet NF: norms stored in the fulfil set
Require: vilateSet NV: norms stored in the violate set
Require: Norms base N
1: for all Norm n in N do
2:     fulfil = Execute Algorithm 3 using n
3:     violate = Execute Algorithm 4 using n
4:     if fulfil >= violate then
5:         NF.add(n)
6:     else
7:         NV.add(n)
8:     end if
9: end for
```

its contribution if the agent chooses to violate the norm (*Algorithm* 4 line 16). Therefore, *Algorithm* 3 should be used to evaluate the contribution of the fulfilment of the norm to the agents' desires/intentions and *Algorithm* 4 should be used to calculate the contribution of the violation of the norm to the agents' desires/intentions.

If the contribution for fulfilling the norm is greater than or equal to the contribution for violating the norm, the norm is selected to be fulfilled and added to the sub-set *Fulfill* of the activated norms. Such step is represented in *Algorithm* 5 from line 4 to 6. Otherwise, it is selected to be violated and added to the sub-set *Violate* of the activated norms. Such step is represented in *Algorithm* 5 from line 6 to 8.

In order to exemplify the applicability of the EN task, let's consider the rescue operation scenario. The evaluation of the benefits of fulfilling and violating the three norms

Algorithm 6. Detecting Conflicts

Require: *fulfilSet* NF: norms stored in the fulfil set
Require: *vilateSet* NV: norms stored in the violate set
for all Norm n1 in NF **do**
  for all Norm n2 in NF **do**
    if n1.State == n2.State and (n1.DeonticConcept == Obligation and n2.DeonticConcept == Prohibition) or (n2.DeonticConcept == Obligation and n1.DeonticConcept == Prohibition) **then**
      Execute Algorithm 7 using n1 and n2
    end if
  end for
end for
for all Norm n1 in NV **do**
  for all Norm n2 in NV **do**
    if n1.State == n2.State and (n1.DeonticConcept == Obligation and n2.DeonticConcept == Prohibition) or (n2.DeonticConcept == Obligation and n1.DeonticConcept == Prohibition) **then**
      Execute Algorithm 7 using n1 and n2
    end if
  end for
end for

Algorithm 7. Solving Conflicts

Require: *fulfilSet* NF: norms stored in the fulfil set
Require: *vilateSet* NV: norms stored in the violate set
fulfiln1 = Execute Algorithm 3 using n1
violaten2 = Execute Algorithm 4 using n2
fulfiln2 = Execute Algorithm 3 using n2
violaten1 = Execute Algorithm 4 using n1
if fulfiln1 + violaten2 >= fulfiln2 + violaten1 **then**
  NF.**remove(n2)**
  NV.**remove(n1)**
else
  NF.**remove(n1)**
  NV.**remove(n2)**
end if

are shown in Tables 1, 3 and 2 that indicates the contribution of each norm element to the achievement of the agent goals. We consider that any norm element generates the same contribution that is 1.

After analysing the contribution of the norms shown in Tables 1, 3 and 2, Norm 1 is included in the set of norms to be fulfilled since the contribution for fulfilling it is equal to "+3" and greater than the contribution for violating it that is equal to "-1". Norm 2 is also included in the fulfil set since the contribution for fulfilling it is equal to "-1" and greater than the contribution for violating it that is equal to "-2". And, finally, Norm 3 is included in the fulfil set since the contribution for fulfilling it is equal to "+1" and

Table 1. Evaluating norm 1

Norm	*Contribution*		
	positive	**neutral**	**negative**
Obligation	1	0	0
Reward	1	0	0
Reward	1	0	0
Punishments			
Obligation	0	0	1

Table 2. Evaluating norm 3

Norm	*Contribution*		
	positive	**neutral**	**negative**
Prohibition	0	0	1
Reward	1	0	0
Reward	1	0	0
Punishments			
Obligation	0	0	1

Table 3. Evaluating norm 2

Norm	*Contribution*		
	positive	**neutral**	**negative**
Prohibition	0	0	1
Punishments			
Obligation	0	0	1
Obligation	0	0	1

greater than the contribution for violating it that is equal to "-1". It indicates that the agent has the intention to fulfil the three norms.

Detecting and Solving Conflicts (DSC). If two different norms (one being an obligation and the other one a prohibition) specify the same state, it is important to check their status, i.e., to check if they are in the set of norms that will be violated or fulfilled since they may be in conflict. If the agent intends to fulfil the obligation but does not intend to fulfil the prohibition, these norms are not in conflict. The same can be said if the agent intends to fulfil the prohibition and to violate the obligation. On the other hand, if the agent intends to fulfil both norms or to violate both norms, they are in conflict and it must be solved. Such step is represented in *Algorithm* 6.

For instance, in case of conflicts between two norms that the agent intends to fulfil or violate, the one with highest contribution to the achievement of the agent's desires (and intentions) can be selected. If the contributions have equal values we can choose anyone. That is, if the contribution coming from the fulfilment of the first norm (Steps 1, 2 and 3) plus the contribution coming from the violation of the second norm (Steps 4.1 and 4.2) is greater to or equal than the contribution coming from the fulfilment of the second norm plus the contribution coming from the violation of the first norm, the first norm is selected to be fulfilled and the second one to be violated. It is represented in *Algorithm* 7 from line 5 to 8. In case the opposite happens, the second norm is selected to be fulfilled and the first to be violated, as described in *Algorithm* 7 from line 8 to 11.

Considering the norms evaluated in the *EN* function, a conflict between Norm 1 and 2 is detected and should be solved. The conflict is solved by selecting Norm 1 to be fulfilled and Norm 2 to be violated since the contribution coming from the fulfilment of the first norm (+3) plus the contribution coming from the violation of the second norm

(-2) is greater than the contribution coming from the fulfilment of the second norm (-1) plus the contribution coming from the violation of the first norm (-1).

4.3 Norm Filter

The *Norm Filter* (Figure 4) is executed in order to drop any intention that does not bring benefits to the agent, retains intentions that are still expected to have a positive overall benefit and adopt new intentions, either to achieve existing intentions, or exploit new opportunities. In order to accomplish theses tasks and besides performing the original functionality of the *Filter function*, such function performs two additional steps.

Selecting Desires (SD). In this step the filter selects the desires that will become intentions taking into account the norms the agent wants to fulfil. The desires are selected according to their priorities and the norms may increase or decrease such priorities. If the agent has a desire to achieve a state and there is a norm that obliges the agent of achieving such state, the desire priority is increased according to the importance of the norm (represented in *Algorithm* 8 from line 3 to 5). If the agent has a desire to achieve a state and there is a norm that prohibits the agent of achieving such state, the desire priority is decreased according to the importance of the norm (represented in *Algorithm* 8 from line 5 to 9). If there is not any norm related to the desires, its priority is not modified. Finally, the function *getDesireHighestPriority()* (*Algorithm* 8 line 12) returns the desire with highest priority.

By applying this function to our example, the goal "Evacuating the members of a NGO to a safe location" is selected because such goal has highest priority since it receives a positive influence of Norm 1.

Selecting Plans (SP). After selecting the desires with highest priorities, i.e., after generating the agent intentions, the agent needs to select the plans that will achieve such intentions. Like in the selection of desires, the selection of plans will also be influenced by the norms.

While selecting a plan it is important to make sure that such plan will achieve the state described in the obligation norm and that will not achieve an state being prohibited. Therefore, the plans that achieve a given intention are ordered according to their priorities. If the state described by an obligation norm is equal to one of the states included in the plan, the norm increases the priority of such plan (represented in *Algorithm* 9 from line 4 to 6). Otherwise, if the state described by an prohibition norm is equal to one of the states included in the plan, the norm decreases the priority of such plan (represented in *Algorithm* 9 from line 6 to 10).

Let's consider that the desires of the agents in our example with highest priority is "Evacuating the members of a NGO to a safe location", that the agent has the intention to fulfil Norm 3, and that the priority of plans that uses helicopters has decrease. When SP step is executed it selects the plan with highest priority that tries to rescue the NGO workers and that will not use helicopters to do so.

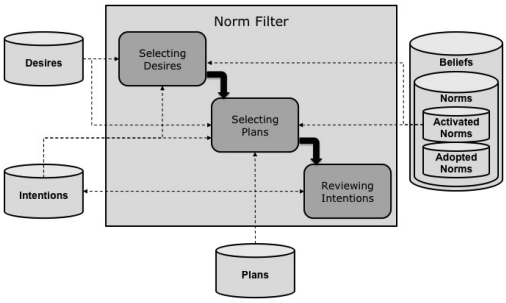

Fig. 4. Norm Filter

Algorithm 8. Selecting Desires

Require: *fulfilSet* NF: norms stored in the fulfil set
Require: Desires base D
Require: Intentions base I
for all Norm n in *NF* **do**
  for all d in $(D \cup I)$ **do**
    if $(n.State == d) \cap (n.DeonticConcept == Obligation)$ **then**
      d.annotatePriority(+1)
    else
      if $(n.State == d) \cap (n.DeonticConcept == Prohibition)$ **then**
        d.annotatePriority(-1)
      end if
    end if
  end for
end for
return getDesireHighestPriority()

5 Implementing the NBDI Architecture

The NBDI architecture proposed in section 4 was implemented by translating the functions related to review, select and filter norms proposed in such architecture to the Normative Jason platform, described in [7]. The platform provides a set of normative functions, as follows: (i) the Norm Review function helps the agent on recognizing its responsibilities towards other agents by incorporating the norms that specify such responsibilities. Such function implements the *Algorithm* 1 described in the **(AN)** step; (ii) the main task of the Updating Norm function is to update the set of activated and adopted norms. It implements the algorithm *Algorithm* 2 described in the **(UN)** step; (iii) the Evaluating Norm function helps the agent on selecting the norms that it has the intention to fulfil and the ones it has the intention to violate by executing the *Algorithm* 5 and, consequently, *Algorithms* 3 and 4 described in the **(EN)** step; (iv) the Detecting and Solving Conflicts function checks and solves the conflicts among the norms. It executes *Algorithms* 6 and 7 described in the **(ISC)** step; (v) the Selecting Desires function selects the desires that will become intentions taking into account their priorities and

executes *Algorithm* 8 described in the **(SD)** step; and, finally, (vi) the Selecting Plans function chooses a single applicable plan from the set of options based on their priorities and executes *Algorithm* 9 described in the **(SP)** step. The applicability of the NBDI architecture and its implementation was demonstrated by the developing of the non-combatant evacuation scenario, presented in section 3.

Algorithm 9. Selecting Plan

Require: *fulfilSet* NF: norms stored in the fulfil set
Require: P: all plans that achieve the selected desire
1: **for all** Norm n in *NF* **do**
2: **for all** p in P **do**
3: **for all** state in p **do**
4: **if** $(n.State == state) \cap (n.DeonticConcept == Obligation)$ **then**
5: p.annotatePriority(+1)
6: **else**
7: **if** $(n.State == state) \cap (n.DeonticConcept == Prohibition)$ **then**
8: p.annotatePriority(-1)
9: **end if**
10: **end if**
11: **end for**
12: **end for**
13: **end for**
14: **return** getPlanHighestPriority()

6 Related Work

Our work was influenced by the architecture proposed in [2]. Such architecture to build normative agents also contemplates functions to deal with the adoption of norms and the influence of norms on the selection of desires and plans. However, our work presents details about the verifications that must be satisfied in order to agents adopt norms, the evaluations that must be made to select the norms the agents intend to fulfil and violate and a guidelines to help agents on selecting plans according to the norms they want to fulfil and violate. The BOID (Belief-Obligation-Intention-Desire) architecture proposed in [1] is an extension of the BDI architecture that considers the influence of beliefs, obligations, intentions and desires on the generation of the agent desires. The BOID architecture applies the notion of agent types to help on the generation of the desires. Thus, their approach could have been used in the **(SD)** function being proposed in our paper since this function is the one responsible to select the desires. Instead of basing the selection of desires on the agent type, we have used the norm contribution and the priority of the desires (and intentions) to provide a quantifiable solution to the selection of the agent desire. The approach described in [4] proposes an architecture to build norm-driven agents whose main purpose is the fulfilment of norms and not the achievement of their goals. In contrast, our agents are desire-driven that take into account the norms but are not driven by them. In [6] the authors provide a technique to extend BDI agent languages by enabling them to enact behaviour modification at runtime in response to newly accepted norms, i.e., it consists of creating new plans to

comply with obligations and suppressing the execution of existing plans that violate prohibitions. However, they have not considered the desires and plans priorities. In our work we consider that obligations and prohibitions may increase or decrease the priority of a desire or a plan, and that the selection of desires and plans are based on their priorities. The agents built according to the architecture presented in [5] are able to evaluate the effects of norms on their desires helping then on deciding to comply or not with the norms. This architecture is based on the BDI architecture whose properties have been expanded to include normative reasoning. In the *Norm Review* process being proposed, we extend some of the verifications defined in [5], such as: our architecture checks *(i)* if the norm was not adopted already and *(ii)* if the agent is the addressee of the norm. Besides, in the *Norm selection* process, although the approach proposed in [5] evaluates the positive and negative effects of norms on the agent desires, it does not consider the influence of rewards in such evaluation. The authors in [3] present concepts, and their relations, that are used for modelling autonomous agents in an environment that is governed by some (social) norms. Although such approach considers that the selection of desires and plans should be based on their priorities and that such priorities can be influenced by norms, it does not present a complete strategy with a set of verification in the norm review process, and strategies to evaluate, identify and solve conflicts between norms such as our work does.

7 Conclusions

This paper proposes an extension to the BDI architecture called NBDI to build goal-oriented agents able to: (i) check if the agent should adopt or not a norm, (ii) evaluate the pros and cons associated with the fulfilment or violation of the norm, (iii) check and solve conflicts among norms, and (iv) choose desires and plans according to their decisions of fulfilling or not a norm.

By implementing the algorithms from 1 to 9 and using the Normative Jason platform, the applicability of NBDI architecture could be verified in the example presented in Section 3. Such agents are responsible to plan the evacuation of people that are in hazardous location, check the incoming perceptions (including norms), select the norms they intend to fulfil based on the benefits they provide to the achievement of the agent's desires and intentions, identify and solve conflicts among the selected norms, and decide to cope or not with the norms while dropping, retaining or adopting new intentions. We are investigating the need for extenting the AgentSpeak language with new predicates that better represent the norms.

References

1. Beavers, G., Hexmoor, H.: Obligations in a bdi agent architecture. In: IC-AI (2002)
2. Castelfranchi, C., Dignum, F., Jonker, C., Treur, J.: Deliberative normative agents: Principles and architecture. In: Proc. of the 6th Int. Workshop on Agent Theories, Architectures, and Languages (1999)
3. Dignum, F.: Autonomous agents and social norms. In: Proc. of the Workshop on Norms, Obligations and Conventions, pp. 56–71 (1996)

4. Kollingbaum, M.: Norm-Governed Practical Reasoning Agents. PhD thesis, University of Aberdeen (2005)
5. Lopez-Lopez, F., Marquez, A.: An architecture for autonomous normative agents. In: Proc. of the 5th Int. Conf. in Computer Science (2004)
6. Meneguzzi, F., Luck, M.: Norm-based behaviour modification in bdi agents. In: Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (2009)
7. Neto, B.F.S., da Silva, V.T., de Lucena, C.J.P.: Using jason to develop normative agents. In: Proc. of the XX Brazilian Symposium on Artificial Intelligence (2010)
8. Weiss, G.: Multiagent Systems: A Modern Approach to Distributed Modern Approach to Artificial Intelligence. Massachusetts Institute of Technology (1999)