A Distributed Architecture for Enforcing Norms in Open MAS

Natalia Criado[1], Estefania Argente[1], Pablo Noriega[2], and Vicent Botti[1]

[1] Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera s/n. 46022 Valencia, Spain
{ncriado,eargente,vbotti}@dsic.upv.es
[2] Institut d'Investigació en Intel.ligència Artificial
Consejo Superior de Investigaciones Científicas
Campus de la UAB, Bellaterra, Catalonia, Spain
pablo@iiia.csic.es

Abstract. Norms have been promoted as a coordination mechanism for controlling agent behaviours in open MAS. Thus, agent platforms must provide normative support, allowing both norm-aware and non norm-aware agents to take part in MAS controlled by norms. In this paper, the most relevant proposals on the definition of norm enforcement mechanisms have been analysed. These proposals present several drawbacks that make them unsuitable for open MAS. In response to these problems, this paper describes a new Norm-Enforcing Architecture aimed at controlling open MAS.

1 Introduction

One of the main applications of Multi-Agent Systems (MAS) is its usage for supporting large scale open distributed systems. These systems are characterized by the heterogeneity of their participants; their limited trust; a high uncertainty; and the existence of individual goals that might be in conflict [2]. In these scenarios, norms are conceived as an effective mechanism for achieving coordination and ensuring social order.

This paper points out the main deficiencies and drawbacks of current platforms and infrastructures when supporting norms. With the aim of enforcing norms in open MAS, in this paper a Norm-Enforcing Architecture is proposed. Specifically, our Norm-Enforcing Architecture has been integrated into the Magentix platform[1]. The Magentix platform allows the management of open MAS in a secure and optimized way. Its main objective is to bring agent technology to real domains: business, industry, e-commerce, among others. This goal entails the development of more robust and efficient mechanisms for enforcing norms that control these complex applications.

This paper is organized as follows: Section 2 contains the analysis of the main proposals on norm enforcement; Section 3 describes briefly the Magentix

[1] http://users.dsic.upv.es/grupos/ia/sma/tools/magentix2/

F. Dechesne et al. (Eds.): AAMAS 2011 Workshops, LNAI 7068, pp. 457–471, 2012.

platform; Section 4 describes the proposed Norm-Enforcing Architecture; and, finally, Section 5 contains a conclusion and future works.

2 Related Work

In general, norms represent an effective tool for regulating the actions of software agents and the interactions among them. Most of proposals on norms for controlling MAS tackle this issue from a theoretical perspective [5,25]. However, there are also works on norms from a computational point of view. These approaches are focused on giving a computational interpretation to norms in order to use them in the execution of MAS applications. Thus, they illustrate how MAS platforms and infrastructures can be extended to implement norms, given that the internal states of agents are not accessible. Therefore, norms cannot be imposed as agent's beliefs or goals, but they must be implemented in the platforms by means of *control* mechanisms [18].

These *control* mechanisms are classified into two categories [18]: *regimentation* and *enforcement* mechanisms. *Regimentation* mechanisms consist on making the violation of norms impossible by mediating the resources and the communication channel, as in case of the Electronic Institutions (EI) [13]. However, the regimentation of all actions can be not only difficult or impossible, but also sometimes it is preferable to allow agents to violate norms [7]. In response to this, the *enforcement* mechanisms are applied after the detection of the violation of a norm, reacting upon it.

Proposals on the enforcement of norms can be classified according to the entities that are in charge of observing norm compliance. There are proposals in which those agents involved by an interaction are responsible for monitoring norms. In these approaches, agents evaluate subjectively to their interaction partners. In accordance with this evaluation, agents may punish or reward their partners [4] or they may start a grievance procedure [9].

If there are agents not directly involved by an interaction that are capable of observing it, then they would be also capable of forming an own *image* about the interacting participants. Moreover, these evaluations or *reputations* may be exchanged. Thus, agents are persuaded to obey norms because their non-normative behaviour can be observed by others. In this case, the *society* as a whole acts as norm enforcer [24]. These non-compliant agents might be even excluded from the society [11]. The role of emotions in the social enforcement [12] is also interesting. For example, the work described in [14] models the emotion-based enforcement of norms in agent societies. In this approach, the whole society observes compliance of norms and generates social emotions such as contempt or disgust, in case of norm violation; and admiration or gratefulness, in case of norm compliant behaviour. In the same way, agents observe the expression of these emotions and are also able to generate emotions such as shame or satisfaction in response. The main drawback of these proposals is the fact that the infrastructure does not offer support for enforcing norms. Thus, the norm monitoring and reaction to violations must be implemented by agent programmers at user level. In this sense,

agent programmers are responsible for watching over norm compliance. Even if the infrastructure provides authority entities that act as *arbiters* or *judges* in grievance processes, agents must be endowed with capabilities for both detecting of norm violations and participating in these dispute resolution processes.

Usually, normative agent platforms provide entities that are in charge of both observing and enforcing norms. The work contained in [20] proposes a distributed enforcement mechanism in which each agent has an interface that sends legal messages and enforces obligations. One of the main drawbacks of this proposal is the fact norms can be only expressed in terms of messages sent or received by a single agent; i.e., this framework does not support the definition of norms that affect an agent as a consequence of a message independently sent by another agent. This problem is solved by Gaertner et al. in [17]. In this work, Gaertner et al. propose a distributed architecture for enforcing norms in EI. Specifically, dialogical actions performed by agents are taken into account by the normative level; i.e., a higher level in which norm reasoning and management processes are performed in a distributed manner. Norms only control the illocutions performed by agents, whereas non-illocutive actions and states of affairs cannot be controlled by this approach. Modgil et al. propose in [21] a general architecture for monitoring norm-governed systems. In particular, this architecture takes an *overhearing* approach; i.e., all messages exchanged among agents are observed and processed. Thus, it is a two layer architecture in which observers (i.e., the lower layer) are capable of reporting to monitors (i.e., the higher layer) on states of interest relevant to the activation, fulfilment, violation and expiration of norms. In this paper, we also propose a two layer approach to norm enforcement. However, in our approach the reasoning about norm enforcement is performed in the two layers whereas in the proposal of Modgil et al. the reasoning process is performed only by monitors. Moreover, our proposal takes as a reference a trace event system based on a publish/subscription procedure (this trace event system is explained in Section 3.1). It reduces appreciably the number of messages exchanged in the platform for controlling norms. Finally, the proposal of Modgil et al. does not give any detail of how the monitoring and observation levels can be dynamically distributed into a set of coordinated entities in response to a changing environment.

There are also works that use a mixed approach for controlling norms [10,19]. Thus, they propose the use of regimentation mechanisms for ensuring compliance with norms that preserve the integrity of the application. Moreover, institutional enforcement is also used for controlling norms that cannot be regimented due to the fact that they are not verifiable or their violation may be desirable. In these two proposals only the norms controlling access to the platform are controlled whereas other problem domain norms are not automatically controlled.

As being illustrated by this section, existing proposals that provide support to norm-enforcing present some drawbacks that make them unsuitable for controlling norms in open MAS. In summary, the most important requirements for norm-enforcing architectures are:

- **Automatic Enforcement.** It must provide support for the detection of norm violations and the application of remedial mechanisms. It implies that agents can trust the enforcement system that will sanction their partners if they behave dishonestly. Moreover, the enforcement architecture must provide normative information in order to allow norm-aware agents to realise that they or other agents have violated a norm. Thus, agents are persuaded to obey norms not only by a material system of sanctions but also since their non-normative behaviour can be observed by others, which may reject to interact with them in the future.
- **Control of general norms.** It must control complex and general norms. Thus, it must allow the definition and management of norms that control not only the messages exchanged among agents but also other actions carried out by agents. In addition, it must support the enforcement of norms that control states of affairs. Finally, it must bring the possibility of controlling norms that are defined in terms of actions and states of affairs that occur independently (e.g., actions that are performed by different agents).
- **Dynamic.** Dynamic situations may cause norms to lose their validity or to need to be adapted. Thus, norm-enforcing mechanisms should provide solutions to open MAS in which the set of norms evolves along time. Moreover, it must provide support for the enforcement of unforeseen norms that control activities and actions that are defined on-line.
- **Efficient, Distributed and Robust.** Finally, enforcement mechanisms must bring the possibility of performing this task in a distributed way. Therefore, they must be unlikely to fail. Thus, this distributed architecture must be capable of operating quickly and effectively in an organized way.

With the aim of meeting these requirements, we propose in Section 4 a Norm-Enforcing Architecture for controlling norms in the Magentix platform. Thus, the Norm-Enforcing Architecture takes as basis the organization and interaction support offered by Magentix. Next, the Magentix platform is briefly described.

3 The Magentix Platform

Magentix is an agent platform for open MAS in which heterogeneous agents interact and organize themselves into Virtual Organizations (VO) [16]. Thus, it provides support at two levels:

- *Organization level.* Magentix provides access to the organizational infrastructure [1] through a set of services included on two main components: the *Service Facilitator* [26], which is a service manager that registers services provided by entities and facilitates service discovering for potential clients; and the *Organization Management System* (OMS) [10], which is in charge of VO management, taking control of their underlying structure, the roles played by the agents and registering the norms that govern the system behaviour.

– *Interaction level*. Magentix provides support to: *agent communication*, supporting asynchronous reliable message exchanges and facilitating the interoperability between heterogeneous entities; *agent conversations* [15], which are automated Interaction Protocols; *tracing service support* [6], which allows agents in a MAS to share information in an indirect way by means of trace events; and, finally, Magentix incorporates a *security module* [3] that provides features regarding security, privacy, openness and interoperability.

Norms define what is considered as permitted, forbidden or obliged in an abstract way. However, norm compliance must be controlled considering the actions and messages exchanged among agents at the interaction level. The Norm-Enforcing Architecture proposed in this paper tries to fill the gap between the organizational level, at which norms are stored by the OMS; and the interaction level, at which actions and communications between agents can be traced. Next, the Tracing Service Support and the register of norms, provided by the OMS, are described in more detail.

3.1 Tracing Service Support

In order to facilitate indirect communication (i.e., indirect ways of interaction and coordination), Magentix provides Tracing Service Support [6]. This service is based on the publish/subscribe software pattern, which allows subscribers to filter events attending to some attributes (content-based filtering), so that agents only receive the information they are interested in and only requested information is transmitted. In addition, security policies define what entities are authorized to receive some specific events. These tracing facilities are provided by a set of components named Trace Manager (TM).

A trace event or *event* is a piece of data representing an action, message exchange or situation that has taken place during the execution of an agent or any other component of the MAS. *Generic* events, which represent application independent information, are *instrumented* within the code of the platform. In addition, *application* events are domain information generated by agents, artefacts or aggregations of agents. An event e is defined as a tuple $\langle Type, Time, Origin, Data \rangle$ where: $Type$ is a constant that represents the nature of the information represented by the trace event; $Time$ is a numeric constant that indicates the global time at which the event was generated; $Origin$ is a constant that identifies the entity that has generated the event; and $Data$ contains extra attached data required for interpreting the event. Trace events can be processed or even combined in order to generate compound trace events, which can be used to represent more complex information.

There can be three types of *tracing entities* (i.e., those elements of the system capable of generating and/or receiving events): agents, artefacts or aggregations of agents. Any tracing entity of the system is provided with a mail box for receiving or delivering events (E_{In} and E_{out}). In this sense, entities that want to receive certain trace events request the subscription to these events to the *TM* by adding an event template to their subscription template list (Sub). An event

template is a tuple $\langle Type, Origin, Data \rangle$ where: $Type$, $Origin$ and $Data$ are the filtering specified criteria.

3.2 Organization Management System (OMS)

The Organization Management System (OMS) [10] is responsible for the management of VO and their constituent entities. In order to allow this management, the OMS provides a set of services classified in: **structural services**, which comprise services for adding/deleting norms (*RegisterNorm* and *DeregisterNorm* services), adding/deleting roles and groups; **informative services**, that provide information of the current state of the organization; and **dynamic services**, which allow agents to enact/leave roles inside VOs (*AcquireRole* and *LeaveRole* services). Moreover, agents can be forced to leave a specific role (*Expulse* service). When the OMS provides successfully any of these services, then it generates an event for informing about the changes produced in the VO.

The $RegisterNorm/DeregisterNorm$ services allow entities to modify the norms that are in *force* (i.e., that are applicable) within the VO. In particular, a norm is defined as a conditional rule that defines under which conditions obligation, permission and prohibition instances should be created [23]. A *norm* is defined as a tuple $n = id : \langle D, T, A, E, C, S, R \rangle$ where: id is the norm identifier; $D \in \{\mathcal{F}, \mathcal{O}\}$ is the deontic modality of the norm, $\mathcal{F}$ represents prohibition and $\mathcal{O}$ represents obligation; T is the target of the norm, the role to which the norm is addressed; A is the norm activation condition, it defines under which circumstances the norm is active and must be instantiated; E is the norm expiration condition that determines when the norm expires and no longer affects agents; C represents the action or state of affairs that is forbidden or obliged; S and R describe the sanctioning and rewarding actions that will be carried out in case of norm violation or fulfilment, respectively. This work takes a closed world assumption where everything is considered as permitted by default. Therefore, permissions are not considered in this paper, since they can be defined as normative operators that invalidate the activation of an obligation or prohibition. As previously argued, our Norm-Enforcing Architecture builds on the event tracing approach to monitoring. Thus, all the norm conditions (i.e., A, E and C) are expressed in term of events.

Once the activation condition of a norm holds; i.e., the activation event is detected, then it becomes active and norm *instances* (or instances for short), according to the possible groundings of the activation condition, must be created. Given a perceived event e, a norm $n = id : \langle D, T, A, E, C, S, R \rangle$ is instantiated into an instance $i = id : \langle D, T, E', C', S', R' \rangle$ where: there is a substitution σ such as $e = \sigma(A)$; $C' = \sigma(C)$; $E' = \sigma(E)$; $S' = \sigma(S)$; and $R' = \sigma(R)$.

From that moment on, a new instance is created and all agents playing the target role are under its influence. Thus, a normative *power* (or power for short) represents the control over a concrete agent that is playing the target role of an instance. Thus, a power is defined as a tuple $p = id : \langle D, T, AgentID, C, S, R, W \rangle$ where: id, D, T, C, S, R are defined as in case of instances; $AgentID$ is a constant

that identifies the agent affected by the power; and W is a boolean constant that expresses if C has been detected or not (i.e., if the event C has been received).

The next section describes the Norm-Enforcing Architecture proposed in this paper. It is a two layer architecture formed by: a higher level responsible of detecting the instantiation of norms; and a lower level in charge of enforcing powers on agents. Thus, the operational semantics of norms, instances and powers (i.e., how they are created, deleted, fulfilled and violated) is explained below.

4 Norm-Enforcing Architecture

The main purpose of the architecture described in this section is to endow the Magentix platform with a norm enforcing framework that is capable of controlling norms in open applications in which unforeseen scenarios may occur. For this reason, this Norm-Enforcing Architecture has been distributed into two layers. In particular, the higher layer is formed by *Norm Manager* (NM) entities that control all processes related with the creation and elimination of both norms and instances. The lower layer is formed by *Norm Enforcing* (NE) entities that are responsible for controlling the agents' behaviours. Next, the NM and NE entities are described in detail.

4.1 Norm Manager

The Norm Manager (NM) is responsible for determining what norms are active (i.e., have been instantiated) in a given moment. Algorithm 1 illustrates pseudocode of the control loop performed by the NM. Each time the NM receives an event (e), then it handles the event according to the event type. Mainly, the NM carries out a process that can be divided into two differentiated tasks: norm management and instance management. Thus, the NM maintains a list (N) that contains all norms that have been registered in Magentix and a list (I) that contains all instances that remain active at a given moment.

Norm Management. In order to maintain the norm list, the NM subscribes to those events sent by the OMS related to the creation and deletion of norms (i.e., $RegisterNorm$ and $DeregisterNorm$ events). Thus, any time the NM receives an event informing about the creation of a new norm, then it adds this norm into its norm list and subscribes to the event that activates the norm (i.e., it adds the event template $\langle A, -, - \rangle$ to its subscription list Sub).

When a norm is deregistered, then the NM removes it from its norm list. Moreover, it removes all instances that have been created out of this norm. For each one of these deleted instances, the NM unsubscribes from the expiration event (i.e., it removes the template $\langle E', -, - \rangle$ from Sub) and generates an event for informing about the deletion of the instance (i.e., a $NormDeletion$ event is sent through the event sending box).

Instance Management. Once the activation event of a norm is received (i.e., $matches(e, A)$), then the NM instantiates the norm (i.e., $instantiation(e, n)$) and adds it to the instance list. At this moment, the NM subscribes to the expiration event and informs about the activation of the norm (i.e., the $InstanceActivation$ event is sent by the NM).

Similarly, when the NM receives the expiration event of any instance (i.e., $matches(e, E)$), then it removes it from the instance list, unsubscribes from the expiration event and informs about the expiration of this instance (i.e., the $InstanceExpiration$ is sent by the NM).

Initially, there is a single NM registered in the Magentix platform. However, the NM is capable of simple adaptation behaviours (i.e., replication and death) in response to changing situations. For example, before the NM collapses (i.e., its event reception box is full), it might replicate itself and remove its subscription to the $RegisterNorm$ event. Thus, the new NM would be responsible for controlling the activation of the new norms. Similarly, if the NM reaches a state in which it has no norm to control and it is not the last NM subscribed to the $RegisterNorm$ event, then it removes itself. These replication and death mechanisms are a simple example that illustrates how the higher layer of the Norm-Enforcing Architecture can be dynamically distributed into several NMs. However, the definition of more elaborated procedures for adapting dynamically to changing environments [22] is a complex issue beyond the scope of this paper.

4.2 Norm Enforcer

The Norm Enforcer (NE) is responsible for controlling agent behaviours. Thus, it detects violations and fulfilments of norms, and reacts upon it by sanctioning or rewarding agents. Algorithm 2 illustrates the control loop executed by the NE. As illustrated by this algorithm, the NE maintains a list (I) with the instances that hold in a given moment. Thus, it subscribes to the events sent by the NM that inform about the activation and expiration of instances, and deletion of norms. Besides that, the NE is also in charge of controlling agents affected by the instances. Thus, it maintains a list P that contains all powers that have been created out of the instances. In order to determine what agents are controlled by these instances, it also maintains a list (RE) containing information about role enactment (i.e., the set of roles that each agent is playing at a given moment). In order to update this list, the NE subscribes to the events sent by the OMS that inform about the fact that an agent has acquired or left a role ($AcquireRole$ and $LeaveRole$ events). In addition, the NE also subscribes to the $Expel$ event, which informs about the fact that a particular agent has been forced to leave a role as a disciplinary measure.

As in case of the NM, the NE starts by retrieving an event from its event reception box. Then, different operations are performed according to the type of the event that has been received. In concrete, the NE carries out a process that can be divided into three different activities: role enactment management, instance management and observation of behaviours.

Algorithm 1. Norm Manager Control Loop

Require: Event reception box E_{In}
Require: Event sending box E_{Out}
Require: Subscription list Sub
Require: $\langle RegisterNorm, OMS, - \rangle$ **in** Sub
Require: $\langle DeregisterNorm, OMS, - \rangle$ **in** Sub
Require: Norm list N
Require: Instance list I

while E_{In} is not empty **do**
  Retrieve e from head of E_{In}// $e = \langle Type, Time, Origin, Data \rangle$
  //Norm Management
  if $Type = RegisterNorm$ **then** // $Data = id : \langle D, T, A, E, C, S, R \rangle$
    Add $Data$ to N
    Add $\langle A, -, - \rangle$ to Sub
  end if
  if $Type = DeregisterNorm$ **and** $Data$ **in** N **then** // $Data = id : \langle D, T, A, E, C, S, R \rangle$
    Remove $Data$ from N
    Remove $\langle A, -, - \rangle$ from Sub
    for all i **in** I **do** // $i = id' : \langle D', T', E', C', S', R' \rangle$
      if $id' = id$ **then**
        Remove i from I
        Remove $\langle E', -, - \rangle$ from Sub
        Add $\langle NormDeletion, NM, id : \langle D', T', E', C', S', R' \rangle \rangle$ to E_{Out}
      end if
    end for
  end if
  //Instance Management
  for all n **in** N **do** // $n = id : \langle D, T, A, E, C, S, R \rangle$
    if $matches(e, A)$ **then** // *the norm is active*
      $i = instantiation(e, n)$ // *i is an instance*
      if i **not in** I **then**
        Add i to I
        Add $\langle InstanceActivation, NM, i \rangle$ to E_{Out}
        Add $\langle A, -, - \rangle$ to Sub
      end if
    end if
  end for
  for all i **in** I **do** // $i = id : \langle D, T, E, C, S, R \rangle$
    if $matches(e, E)$ **then**
      Remove i from I
      Remove $\langle E, -, - \rangle$ from Sub
      Add $\langle InstanceExpiration, NM, i \rangle$ to E_{Out}
    end if
  end for
end while

Algorithm 2. Norm Enforcer Control Loop

Require: Event reception box E_{In}
Require: Event sending box E_{Out}
Require: Subscription list Sub
Require: $\langle NormDeletion, NM, - \rangle$ **in** Sub
Require: $\langle InstanceActivation, NM, - \rangle$ **in** Sub
Require: $\langle InstanceExpiration, NM, - \rangle$ **in** Sub
Require: $\langle AcquireRole, OMS, - \rangle$ **in** Sub
Require: $\langle LeaveRole, OMS, - \rangle$ **in** Sub
Require: $\langle Expel, OMS, - \rangle$ **in** Sub
Require: Instance list I
Require: Power list P
Require: Role enactment list RE
1: **while** E_{In} is not empty **do**
2: Retrieve e from E_{In}// $e = \langle Type, Time, Origin, Data \rangle$
... // *Role enactment management*
... // *Instance management*
... // *Observation of Behaviour*
66: **end while**

Role Enactment Management. Algorithm 3 illustrates pseudocode corresponding to the role enactment management process. Specifically, if the OMS informs that an agent ($AgentID$) has acquired a new role ($RoleID$), then the NE updates the role enactment list. Moreover, the list of instances is also checked for determining what of the instances affect the $RoleID$. For each one of these instances, the NE creates a new power addressed to the agent identified by $AgentID$. In addition, the NE subscribes to the event expressed in the norm condition in order to be aware of the fulfilment or violation of this norm (i.e., it adds the event template $\langle C, AgentID, - \rangle$ to the subscription list Sub).

On the contrary, if the OMS informs that an agent is not longer playing a role, then the role enactment list is updated . Similarly, all powers that affect the $AgentID$ as a consequence of being playing $RoleID$ are removed. Therefore, the NE does not have to observe the norm condition and unsubscribes from this event. Finally, if any agent leaves a role voluntarily (i.e., the *LeaveRole* event is received) before fulfilling its pending obligations, then it will be sanctioned (i.e., the NE would perform the sanctioning action S). Moreover, the NE would inform about the fact that an agent has been sanctioned for non-compliance with an obligation (i.e., the *Sanction* event is sent through the E_{Out} box).

Instance Management. This process is contained in Algorithm 4. If the NE is informed by the NM the creation of a new instance (i.e., the $InstanceActivation$ event is received), then the NE updates its instance list and creates new powers for controlling all agents that are playing the target role at this moment. The watch condition (W) of the new powers is initially set to false. Thus, for each one of the new powers the NE starts to observe indirectly norm compliance by subscribing to the event C.

Algorithm 3. Role Enactment Management

```
3: if Type = AcquireRole then // Data is a pair (AgentID, RoleID)
4:    Add Data to RE
5:    for all i in I do // i = id : ⟨D, T, E, C, S, R⟩
6:       if T = RoleID then
7:          Add id : ⟨D, T, AgentID, C, S, R, false⟩ to P
8:          Add ⟨C, AgentID, −⟩ to Sub
9:       end if
10:   end for
11: end if
12: if Type = LeaveRole or Type = Expel then // Data is a pair
    (AgentID, RoleID)
13:   Remove Data from RE
14:   for all p in P do // p = id : ⟨D, T, A, C, S, R, W⟩
15:      if T = RoleID and A = AgentID then
16:         Remove p from P
17:         Remove ⟨C, AgentID, −⟩ from Sub
18:         if D = O and W = False and Type = LeaveRole then
19:            Perform S// against AgentID
20:            Add ⟨Sanction, NE, Violated(id, AgentID)⟩ to E_Out
21:         end if
22:      end if
23:   end for
24: end if
```

If an instance has no longer effect (i.e., the *InstanceExpiration* or *NormDeletion* events is perceived), then the NE updates the instance list and removes all powers created out of this instance. An instance becomes ineffective whenever the expiration condition hold or the norm that has given rise to it has been abolished. In the first case (i.e., the *InstanceExpiration* event is received), the agent is responsible for fulfilling the norm. Thus, if the instance obliges an agent to reach some state of affairs (e.g., the agent is obliged to perform an action) and this state has not been observed yet (i.e., the watch condition W is false), then the agent will be sanctioned. On the contrary, if the agent is prohibited to reach some situation and the forbidden state has not been observed (i.e., W is false) then the agent will be rewarded. Finally, if an instance becomes ineffective due to the deletion of a norm, then the agent is not responsible for the fulfilment of the norm and enforcement actions are not performed.

Observation of Behaviours. This functionality is implemented by Algorithm 5. The NE checks for each one of the powers if the C event has been detected (i.e., $matches(e, C)$). If it is the case, then the power is updated. The watch condition is registered as true indicating that the norm condition has been perceived. Next, enforcement actions are performed according to the deontic modality of the power. For example, if the power is an obligation then the obligation is considered as fulfilled (i.e., the power is deleted from P) and the agent is rewarded. Similarly, if it is a prohibition then the agent will be sanctioned.

Algorithm 4. Instance Management

25: **if** $Type = InstanceActivation$ **then** // $Data = id : \langle D, T, E, C, S, R \rangle$
26: Add $Data$ to I
27: **for all** $(AgentID, RoleID)$ **in** RE **do**
28: **if** $RoleID = T$ **then**
29: Add $id : \langle D, T, AgentID, C, S, R, \textbf{false} \rangle$ to P
30: Add $\langle C, AgentID, - \rangle$ to Sub
31: **end if**
32: **end for**
33: **end if**
34: **if** ($Type = InstanceExpiration$ **or** $Type = NormDeletion$) **and** $Data$ *in* I **then** // $Data = id : \langle D, T, E, C, S, R \rangle$
35: Delete $Data$ from I
36: **for all** p **in** P **do** // $p = id' : \langle D, T, AgentID, C, S, R, W \rangle$
37: **if** $id = id'$ **then**
38: Remove p from P
39: Remove $\langle C, AgentID, - \rangle$ from Sub
40: **if** $Type = InstanceExpiration$ **then** // *The agent is responsible for norm fulfilment*
41: **if** $W =$ **false and** $D = O$ **then** // *The obligation had not been fulfilled before it has expired*
42: Perform S// *against* $AgentID$
43: Add $\langle Sanction, NE, Violated(id, AgentID) \rangle$ to E_{Out}
44: **end if**
45: **if** $W =$ **false and** $D = F$ **then** // *The prohibition had been observed*
46: Perform R// *in favour of* $AgentID$
47: Add $\langle Reward, NE, Fulfilled(id, AgentID) \rangle$ to E_{Out}
48: **end if**
49: **end if**
50: **end if**
51: **end for**
52: **end if**

As in case of the NM, the lower level of the Norm-Enforcing Architecture has been described as it was formed by a single entity. However, this layer may be formed by a set of specialized NEs. For example, the set of instances can be distributed among NEs according to the target role. Thus, each NE is responsible for controlling actions in which a specific set of roles is involved. It is also possible to specialize NE for controlling a specific group of agents independently of the roles that they play. Finally, it is also possible the dynamic adaptation of the amount of NEs by performing cloning and self-deletion operations.

Algorithm 5. Observation of Behaviours

53: **for all** p **in** P **do** // $p = id : \langle D, T, AgentID, C, S, R, W \rangle$
54: **if** $matches(e, C)$ **then**
55: Remove p from P
56: **if** $D = F$ **then** // *The prohibition has been violated*
57: Add $id : \langle D, T, AgentID, C, S, R, \mathbf{true} \rangle$ to P
58: Perform S// *against* $AgentID$
59: Add $\langle Sanction, NE, Violated(id, AgentID) \rangle$ to E_{Out}
60: **else** // *The obligation has been fulfilled and it expires*
61: Perform R// *in favour of* $AgentID$
62: Add $\langle Reward, NE, Fulfilled(id, AgentID) \rangle$ to E_{Out}
63: Remove $\langle C, AgentID, - \rangle$ from Sub
64: **end if**
65: **end if**
66: **end for**

5 Conclusions and Future Works

In this paper, we have described a Norm-Enforcing Architecture considering the facilities provided by the Magentix platform. The main aim of this Norm-Enforcing Architecture is to overcome problems of existing proposals on norm enforcement. Thus, the requirements taken into account by our proposal are:

- **Automatic Enforcement.** Our proposal enforces norms providing support to those agents that are not endowed with normative reasoning capabilities. In addition, the generation of events for informing about sanctions and rewards allows norm-aware agents to use this information for selecting the most suitable interaction partners.
- **Control of general norms.** Our definition of norm is based on notion of event. An *event* represents an action, message exchange or situation that has taken place during the execution of any tracing entity (i.e., an agent, aggregation of agents or an artefact). Thus, norms are defined in terms of events that can be generated independently by different tracing entities.

- **Dynamic.** Magentix allows the dynamic modification of norms. Moreover, new event types can be dynamically defined at run time. Accordingly, our proposal has been designed taking into account the possibility that norms and events can be created or deleted on-line.
- **Efficient, Distributed and Robust.** Finally, our Norm-Enforcing Architecture is build upon a trace event system, which provides support for indirect communication in a more efficient way than overhearing approaches. Besides that, we have provided a preliminary solution to the adaptation of the architecture in response to situations in which the number of agents or norms to be controlled dramatically changes.

As future work, we plan to deal with complex scenarios in which there are norms whose violation cannot be directly observed, since they regulate situations that take place out of the institution boundaries. Or even more, norms can be interpreted ambiguously. This entails the development of intelligent and proactive norm-enforcing entities (i.e., agents) [8] endowed with capabilities for applying techniques such as negotiation or conflict resolution procedures.

Acknowledgments. This paper was partially funded by the Spanish government under grants CONSOLIDER-INGENIO 2010 CSD2007-00022, TIN2009-13839-C03-01, TIN2008-06701-C03-03,TIN2008-04446 and by the FPU grant AP-2007-01256 awarded to N. Criado. This research has also been partially funded by the Generalitat de Catalunya under the grant 2009-SGR-1434 and Valencian Prometeo project 2008/051.

References

1. Argente, E., Botti, V., Carrascosa, C., Giret, A., Julian, V., Rebollo, M.: An Abstract Architecture for Virtual Organizations: The THOMAS approach. Knowledge and Information Systems, 1–35 (2011)
2. Artikis, A., Pitt, J.: A formal model of open agent societies. In: Proc. of AAMAS, pp. 192–193 (2001)
3. Bellver, J., Such, J.M., Espinosa, A., García-Fornes, A.: Developing Secure Agent Infrastructures with Open Standards and Open-Source Technologies. In: Proc. of PAAMS (in Press, 2011)
4. Boella, G., van der Torre, L.: Norm governed multiagent systems: The delegation of control to autonomous agents. In: Proc. of IAT, pp. 329–335. IEEE (2003)
5. Boella, G., Van Der Torre, L.: Regulative and constitutive norms in normative multiagent systems. In: Proc. of KR, pp. 255–265 (2004)
6. Burdalo, L., Terrasa, A., Julian, V., Garcia-Fornes, A.: TRAMMAS: A Tracing Model for Multiagent systems. In: Proc. of ITMAS, pp. 42–49 (2010)
7. Castelfranchi, C.: Formalising the informal? Dynamic social order, bottom-up social control, and spontaneous normative relations. Journal of Applied Logic 1(1-2), 47–92 (2003)
8. Criado, N., Argente, E., Botti, V.: Towards Norm Enforcer Agents. In: Proc. of PAAMS (in Press, 2011)

9. Criado, N., Argente, E., Garrido, A., Gimeno, J.A., Igual, F., Botti, V., Noriega, P., Giret, A.: Norm Enforceability in Electronic Institutions? In: De Vos, M., Fornara, N., Pitt, J.V., Vouros, G. (eds.) COIN 2010. LNCS, vol. 6541, pp. 250–267. Springer, Heidelberg (2011)
10. Criado, N., Julián, V., Botti, V., Argente, E.: A Norm-Based Organization Management System. In: Padget, J., Artikis, A., Vasconcelos, W., Stathis, K., da Silva, V.T., Matson, E., Polleres, A. (eds.) COIN@AAMAS 2009. LNCS, vol. 6069, pp. 19–35. Springer, Heidelberg (2010)
11. de Pinninck, A., Sierra, C., Schorlemmer, M.: Friends no more: norm enforcement in multiagent systems. In: Proc. of AAMAS, p. 92. ACM (2007)
12. Elster, J.: Rationality and the Emotions. The Economic Journal 106(438), 1386–1397 (1996)
13. Esteva, M., Rosell, B., Rodríguez-Aguilar, J.A., Arcos, J.L.: Ameli: An agent-based middleware for electronic institutions. In: Proc. of AAMAS, pp. 236–243 (2004)
14. Fix, J., von Scheve, C., Moldt, D.: Emotion-based norm enforcement and maintenance in multi-agent systems: foundations and petri net modeling. In: Proc. of AAMAS, pp. 105–107. ACM (2006)
15. Fogués, R.L., Alberola, J.M., Such, J.M., Espinosa, A., García-Fornes, A.: Towards Dynamic Agent Interaction Support in Open Multiagent Systems. In: Proc. of CCIA, vol. 220, pp. 89–98. IOS Press (2010)
16. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications 15(3), 200 (2001)
17. Gaertner, D., Garcia-Camino, A., Noriega, P., Rodriguez-Aguilar, J., Vasconcelos, W.: Distributed norm management in regulated multiagent systems. In: Proc. of AAMAS, p. 90. ACM (2007)
18. Grossi, D., Aldewereld, H., Dignum, F.: *Ubi Lex, Ibi Poena*: Designing Norm Enforcement in E-Institutions. In: Noriega, P., Vázquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., Matson, E. (eds.) COIN 2006. LNCS (LNAI), vol. 4386, pp. 101–114. Springer, Heidelberg (2007)
19. Hubner, J., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents. Journal of Autonomous Agents and Multi-Agent Systems 20(3), 369–400 (2010)
20. Minsky, N., Ungureanu, V.: Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. ACM Transactions on Software Engineering and Methodology (TOSEM) 9(3), 273–305 (2000)
21. Modgil, S., Faci, N., Meneguzzi, F., Oren, N., Miles, S., Luck, M.: A framework for monitoring agent-based normative systems. In: Proc. of AAMAS, International Foundation for Autonomous Agents and Multiagent Systems, pp. 153–160 (2009)
22. Nakano, T., Suda, T.: Self-organizing network services with evolutionary adaptation. IEEE Transactions on Neural Networks 16(5), 1269–1278 (2005)
23. Oren, N., Panagiotidi, S., Vázquez-Salceda, J., Modgil, S., Luck, M., Miles, S.: Towards a Formalisation of Electronic Contracting Environments. In: Hübner, J.F., Matson, E., Boissier, O., Dignum, V. (eds.) COIN@AAMAS 2008. LNCS, vol. 5428, pp. 156–171. Springer, Heidelberg (2009)
24. Sen, S., Airiau, S.: Emergence of norms through social learning. In: Proc. of IJCAI, pp. 1507–1512 (2007)
25. Sergot, M.: Normative positions. Norms, Logics and Information Systems, 289–308 (1998)
26. Val, E.D., Criado, N., Rebollo, M., Argente, E., Julian, V.: Service-Oriented Framework for Virtual Organizations. In: Proc. of ICAI, vol. 1, pp. 108–114 (2009)