LIVERPOOL JOHN MOORES UNIVERSITY

LJMU Research Online

Criado, N, Such, JM and Botti, VJ

Norm reasoning services

http://researchonline.ljmu.ac.uk/id/eprint/162/

Article

Citation (please note it is advisable to refer to the publisher's version if you intend to cite from this work)

Criado, N, Such, JM and Botti, VJ (2014) Norm reasoning services. Information Systems Frontiers, 16 (2). pp. 201-223. ISSN 1387-3326

LJMU has developed **LJMU Research Online** for users to access the research output of the University more effectively. Copyright © and Moral Rights for the papers on this site are retained by the individual authors and/or other copyright owners. Users may download and/or print one copy of any article(s) in LJMU Research Online to facilitate their private study or for non-commercial research. You may not engage in further distribution of the material or use it for any profit-making activities or any commercial gain.

The version presented here may differ from the published version or from the version of the record. Please see the repository URL above for details on accessing the published version and note that access may require a subscription.

For more information please contact researchonline@ljmu.ac.uk

http://researchonline.ljmu.ac.uk/

Noname manuscript No.
(will be inserted by the editor)

Norm Reasoning Services

N. Criado · J.M. Such · V. Botti

the date of receipt and acceptance should be inserted later

Abstract Norms are used in open Multi-Agent Systems as a formal specification of deontic statements aimed at regulating the actions of agents and the interactions among them. In this paper, we propose a set of services facilitating the development of both non-normative and normative agents for norm-governed MAS. Specifically, we propose to provide agents with norm reasoning services. These services will help agent designers/developers to programme agents that consider norm reasoning without having to implement the needed mechanisms to reason about norms by themselves. This article shows how these services perform as well as the results of the experiments that we conducted to evaluate their performance.

Keywords: norms, services, agents.

1 Introduction

The main feature of open Multi-Agent Systems (MAS) is that they are populated by heterogeneous agents, which can enter and leave the system dynamically. These heterogeneous agents may have been designed independently, according to different goals, and no assumption about their behaviours can be made [2]. To ensure social order and avoid potential conflicts, norms are used in open MAS as a formal specification of deontic statements aimed at regulating the actions of agents and the interactions among them [32].

A great amount of research has been conducted to support the use norms in open MAS [8]. For example, there are works aimed at: allowing system designers to

N. Criado
University of Bolton, UK,
E-mail: ncriado@bolton.ac.uk

V. Botti
DSIC, Universitat Politècnica de Valencia, Spain,
E-mail: vbotti@dsic.upv.es

J.M. Such
School of Computing and Communications, Lancaster University, UK,
E-mail: j.such@lancaster.ac.uk

define and represent the norms that regulate a particular application [30], controlling norms inside specific agent platforms [26], and proposing norm-autonomous agent architectures [4] that are endowed with norm reasoning capabilities. Norms have also been used in MAS to coordinate the activities of heterogeneous agents that participate in auctions modelled as Electronic Institutions [16]. Norms have also been applied into MAS used for social simulation purposes, e.g., in [21] Garrido et al. have applied norms in a MAS that simulates a water-right market, the basin, users, regulations and grievance situations. Moreover, norms have been applied to MAS to develop assistant agents to which humans can delegate tasks that are regulated by legal and social norms [12].

In the existing literature on norms and MAS, agents are classified as normative, when their behaviour is influenced by norms that are explicitly represented inside its mind; and as non-normative, when they are not endowed with capabilities for considering norms. There are different types of normative agents according to the norm reasoning capabilities that they have. Specifically, normative agents [7] are endowed with some of the following capabilities for: acquiring and recognising the norms that are in force (i.e., applicable) in their environment [1,7]; determining whether a norm concerns their case and it is relevant to them [23]; and making decisions about norm compliance [10]. In contrast, non-normative agents do not take norms into account. This does not imply that non-normative agents do not comply with norms. In fact, they may comply with norms unintentionally.

As previously mentioned, norms are needed in open MAS to control and coordinate the behaviour of heterogeneous agents. As the designers of open MAS, we cannot know a priori which are the capabilities for reasoning about norms of the agents that will join the MAS. Therefore, supporting different levels of norm reasoning capabilities is of crucial importance to ensure social order in open MAS. Agent platforms are the software that supports the development and execution of MAS. Thus, agent platforms should not only provide support for monitoring and controlling norms, but they should also facilitate non-normative as well as all kinds of normative agents to participate inside norm-governed open MAS.

In this paper, we propose a set of *norm reasoning services*. These services range from simple services that inform about the norms that are in force at a given moment, so that normative agents with capabilities for reasoning about norm relevance of norms and norm compliance can decide which norms are relevant to them and which ones they want to obey, to more elaborated services that allow agents to known which are the normative goals that they should pursue, allowing non-normative agents to behave accordingly to norms.

For example, a virtual assistant agent that imitates the behaviour of a human seller must be endowed with capabilities that allow it to participate in different and, even unknown, e-markets such as eBAY[1] or Amazon[2]. Each e-market has its own norms that define the protocols; e.g., the process by which purchase contracts are formalised, how transactions are fulfilled, the mechanism that must be used to sanction contract violations, etc. As a consequence, the virtual assistant agent requires capabilities for reasoning about norms and knowing the specific mechanisms and protocols by which the trading operations are performed in each market. Over

[1] `http://www.ebay.com`

[2] `http://www.amazon.com`

the course of this paper, we will use a running example of an e-market to illustrate and motivate the need for the norm reasoning services that we propose.

It is important to note that agents are not forced to use the norm reasoning services. Instead, they can decide (i.e., they are autonomous) whether they make use of the services or not. For instance, agents that want to follow the norms but do not know the norms that regulate their environment a priori may be interested in a service that provides information about the norms that are in force. Similarly, agents that want to follow the norms but are not endowed with mechanisms for interpreting and reasoning about them may be interested in using more elaborated services that suggest them how to behave. In contrast, agents that ignore the norms or agents that are fully capable of reasoning about the norms may not be interested in using our services. Clearly, this has also very important implications from the point of view of agent designers. In particular, agent designers can decide that their agents make use of our proposed services (i.e. outsourcing either part or the whole of their capabilities to reason about norms) or not (i.e., programming the norms in the agents' code, endowing agents with capabilities for reasoning about norms or even programming agents that ignore norms). Obviously, coordination among interacting agents is achieved when there is a common point of view of norms governing that interaction. To prevent agents from ignoring norms and persuade them to either reason about norms or use our norm reasoning services, a norm enforcing architecture, such as the one described in [11], can be put in place.

This paper is structured as follows: Section 2 contains a brief description of related works; Section 3 briefly describes Magentix2; Section 4 describes the norm reasoning services; Section 5 provides an evaluation of the norm reasoning services; and Section 6 contains a brief conclusion and future work.

2 Related Work

Traditionally, two different approaches have been considered for establishing norms in agent societies [33]. On the one hand, the *top-down approach*, where the system designer defines the normative system statically off-line as in Electronic Institutions [16], or norms are created dynamically on-line by some agent that acts as a leader or a norm recommender [25]. On the other hand, the *bottom-up approach*, which analyses how norms can emerge inside a group of agents [6]. In this approach norm has emerged when it is followed by a considerable portion of the society without being previously created. Therefore, agents recognise norms based on their observations[3].

Our proposal is based on the *top-down approach*. Thus, we assume that there is an explicit set of norms that specifies how agents should behave. In particular, norms may be created off-line by the system designer or on-line by an agent empowered to change the normative system.

The idea of providing agents with normative information or services in systems following the *top-down approach* is not new. In [17], Felicíssimo et al. propose a solution for continuously supporting agents with updated norm information. In the proposal of Felicíssimo et al. the scope of norms is defined using contexts. Therefore, agents are provided with information about the norms that are in force

[3] See [33] for a review of works on the recognition of norms.

in their current context. Similarly, in [28] Okuyama et al. propose the definition of *normative objects* that allow agents to be informed about the norms that regulate their context. However, these two solutions do not provide agents with information about norm dynamics (i.e., the activation and expiration of norms), norm compliance and norm judgement. Therefore, these functionalities must be implemented by agent programmers at the agent level.

Other solutions that provide agents with normative information are based on the use of normative artifacts. Artifacts are resources and tools that agents can create and use to perform their individual and social activities [29]. For example, the ORA4MAS [22] proposal defines artifacts as first-class entities to instrument multi-agent organisations to support agent activities within them. In the ORA4MAS the monitoring of norms has been implemented by means of artifacts, which detect norm violations; and by means of agents, which are informed about norm violations and carry out the evaluation and judgement of these situations. Therefore, the agent designers are responsible for programming agents endowed with capabilities for performing these tasks. Finally, in [31] Piunti et al. propose that normative artifacts provide a series of observable properties that can be inspected by agents to know the actual normative state of the organisation. Therefore, agents are able to know which norms are active at a given moment. Autonomous agents can use this information to reason about whether to follow or not the norms that are active.

Table 1 summarizes the performance of the proposals that provide agents with normative information with respect to the specific information that they provide. In particular, providing *norm information* consists of informing agents about the norms that are in force. Providing *norm relevance information* means informing agents about the instances that are relevant to the current situation. Providing *norm advice information* consists of advising agents to make decisions on whether or not to follow the instances that are relevant to them. Providing norm judgement information entails the detection of *norm violations* and *judging* these situations. As illustrated in this table, issues such as the provision of *norm advice information* and *norm judgement information* have not been properly addressed by the existing proposals. With the aim of meeting these pending requirements we propose in Section 4 a set of *norm reasoning services* for allowing heterogeneous agents to participate inside norm-governed open MAS. Specifically, the services proposed in this paper are based on the organization and interaction support offered by Magentix2. Next, the Magentix2 AP is briefly described.

3 The Magentix2 Agent Platform

Magentix2 is an agent platform for open MAS in which heterogeneous agents interact and organize themselves into Virtual Organizations (VOs) [20]. VOs are open systems formed by the grouping and collaboration among heterogeneous entities. In VOs there is a separation between form and function that requires defining how behaviour will take place [18]. VOs are social entities formed by agents that try to achieve the organizational goals. These agents are organized in groups that are controlled by norms.

Magentix2 is formed by different building blocks that provide support for VOs at three levels:

	Norm Information	Norm Relevance Information	Norm Advice Information	Violation Information	Judgement Information
Felicíssimo et al. [17]	✓				
Okuyama et al. [28]	✓				
ORA4MAS [22]				✓	
Piunti [31]	✓	✓			

Table 1 Summary of proposals on providing agents with normative information

– *Organization level.* Magentix2 provides access to the organizational infrastructure through the *Organization Management System* (OMS) [13], which is in charge of the management of VOs, taking control of their underlying structure, the roles played by agents, and the norms that govern the VO.
– *Interaction level.* Magentix2 provides support to: *agent communication*, supporting asynchronous reliable message exchanges and facilitating the interoperability between heterogeneous entities; *agent conversations* [19], which are automated Interaction Protocols; *tracing service support* [5], which allows agents in a MAS to share information in an indirect way by means of trace events; and, finally, Magentix2 incorporates a *security module* [34] that provides features regarding security, privacy, openness and interoperability.
– *Agent level.* Magentix2 provides native support for executing Jason agents [3] and conversational agents [19] that carry out simultaneous conversations.

Norms define what is considered as permitted, forbidden or obligatory in an abstract way. However, norm compliance must be controlled considering the actions and messages exchanged among agents at the interaction level. Magentix2 fills the gap between the organizational level, at which norms are registered by the OMS; and the interaction level, at which actions and communications occur, through a *norm-enforcing architecture*, named MaNEA [11]. MaNEA is responsible for sanctioning agents that violate norms and rewarding norm compliance. Thus, it can be used in conjunction with our Norm Reasoning Services (NRSs) to persuade agents to comply with norms. However, not all agents in a MAS are endowed with norm reasoning capabilities. For agents with different norm reasoning capabilities we propose to extend Magentix2 with a set of NRSs. Specifically, we aim at filling the gap between the organizational level, at which norms are defined; and the agent level, at which norms must be considered before taking action. Next, the tracing service and the storage of norms, provided by the OMS, are described.

3.1 Tracing Service

In order to facilitate indirect communication (i.e., indirect ways of interaction and coordination), Magentix2 provides the Tracing Service Support [5]. This service is based on the publish/subscribe software pattern, which allows subscribers to filter events satisfying the values of some attributes (content-based filtering), so that agents only receive the information in which they are interested and only requested information is transmitted. In addition, security policies define which entities are authorized to receive which specific events. These tracing facilities are provided by a set of components named Trace Manager (TM). There can be three types of *tracing entities* (i.e., those elements of the system capable of generating and/or receiving events): agents, services or groups of agents.

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. *Application* events are domain dependent information.

Definition 1 (Event) An event e is defined as a tuple $e = \langle Type, Time, Origin, Data \rangle$, where:

- $Type$ is a constant that represents the nature of the information represented by the event;
- $Time$ is a numeric value that indicates the global time at which the event is generated;
- $Origin$ is a constant that identifies the tracing entity that generates the event;
- $Data = \psi_1 \wedge ... \wedge \psi_n$ is a conjunction of possibly negated first-order grounded atomic formulae that contains extra attached data required for interpreting the event.

Trace events can be processed or even combined to generate compound trace events, which can be used to represent more complex information.

Any tracing entity is provided with mail boxes for receiving or delivering events (E_{In} and E_{out}). Entities that want to receive certain trace events request the subscription to these events by sending to the TM a *subscription* event that contains the template of those events they are interested in.

Definition 2 (Template) A template t is a tuple $t = \langle Type, Origin, Data \rangle$ that contains the filtering specified criteria for events, where:

- $Type$ is a constant that represents the nature of the information represented by the event;
- $Origin$ is a constant that identifies the entity that generates the event;
- $Data = \psi_1 \wedge ... \wedge \psi_n$ is a conjunction of possibly negated first-order atomic formulae that may contain free variables.

Let us consider the standard notion of substitution as a finite and possibly empty set of pairs X/y where X is a variable and y is a grounded term. Let us also define the application of a substitution σ as:

1. $\sigma(c) = c$ if c is a constant.
2. $\sigma(X) = y$ if $X/y \in \sigma$; otherwise $\sigma(X) = X$.

3. $\sigma(\psi_1 \wedge ... \wedge \psi_n) = \sigma(\psi_1) \wedge ... \wedge \sigma(\psi_n)$.
4. $\sigma(\langle \rho_0, ..., \rho_n \rangle) = \langle \sigma(\rho_0), ..., \sigma(\rho_n) \rangle$

Therefore, the application of a substitution on a template is defined as follows:

$$\sigma(\langle Type, Origin, Data \rangle) = \langle Type, Origin, \sigma(Data) \rangle$$

since $Type$ and $Origin$ take constant values.

According to the definitions of events and templates the *matching* relationship between events and templates is defined as follows:

Definition 3 (Matching Function) Given an event $e = \langle Type, Time, Origin, Data \rangle$ and a template $t = \langle Type', Origin', Data' \rangle$, their *matching* is a boolean function defined as follows:

$$matching(e,t) = \begin{cases} true & \text{if } (Type = Type') \wedge \\ & ((Origin = Origin') \vee (Origin' \text{ is undefined})) \wedge \\ & (\forall \psi_i : Data' \vdash \psi_i \wedge Data \vdash \psi_i) \\ false & \text{otherwise} \end{cases}$$

Definition 4 (Unification Function) Given an event e and a template t, their $unification$ is a boolean function defined as follows:

$$unification(e,t) = \begin{cases} true & \text{if exists a substitution of variables } \sigma \text{ such that} \\ & matching(e, \sigma(t)) \text{ is } true \\ false & \text{otherwise} \end{cases}$$

3.2 Organization Management System (OMS)

The Organization Management System (OMS) [13] is responsible for the management of VOs and their constituent entities. The OMS provides a set of services: **structural services**, which comprise services for adding/deleting norms (*registerNorm* and *deregisterNorm* services allow entities to modify the norms that are in *force* or applicable within a VO), and for adding/deleting roles and groups; **informative services**, which 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 carries out any of these services successfully, then it generates an event for informing about the changes produced in the VO.

3.2.1 Virtual Organization Model

VOs have been employed as an abstraction for modelling open MAS. VOs include the integration of organizational and individual perspectives and also the dynamic adaptation of models to organizational and environmental changes [15]. In [9] the model of VO considered by the OMS is described in detail. It is based on the Human Organization Theory [14] and classifies the main aspects of a VO into four dimensions as follows:

- The *Structural* dimension, which describes components of the system and their relationships. Thus, it describes the roles and groups that form a VO. Roles allow dividing the VO functionalities in an abstract way. Groups represent the context in which these activities take place.
- The *Functional* dimension, which describes the system functionality in terms of the services provided by agents and groups.
- The *Normative* dimension, which describes the norms defined to control the society members. Norms are a coordination mechanism that attempt to: (i) promote behaviours that are satisfactory to the organization (i.e., actions that contribute to achievement of global goals); and (ii) avoid harmful actions (i.e., actions that prompt the system to be unsatisfactory or unstable).
- The *Environmental* dimension, which describes the environment in terms of its resources and how agents can perceive and act on them.

3.2.2 Norm Definition

According to the normative definitions provided in [30], in Magentix2 a distinction among *norms* and *instances* is made. *Norms* define patterns of behaviours by means of *deontic modalities*: *obligations*, which define which actions or states of affairs should be performed or satisfied by agents; and *prohibitions*, which define which actions or states of affairs should not be performed or achieved. Magentix2 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. Therefore, norms define a pattern of behaviour (or *norm condition* in our terminology) as obligatory or prohibited. This norm condition can be represented as actions to be performed or states of affairs to be achieved. In fact, we make no sharp distinction between actions and states of affairs, since what in one situation is best described as an action may be best described in another situation as a state of affairs [24]. Also inspired by the representation of [30], we define norms as conditional rules that are relevant to a set of agents under specific circumstances. Thus, the set of agents that is affected by a specific norm are the ones that are playing the *target* role of this norm. Thus, norms represent the responsibilities, rights and duties of roles. In general, norms are not applied at all times, but include the notions of activation and expiration conditions. Specifically, the *activation condition* defines when obligations and prohibitions must be instantiated and must be fulfilled by all agents playing the target role. These instances remain active, even if the activation condition ceases to hold. Specifically, the *expiration condition* defines the validity period or deadline of an instance. Finally, inspired by [24], norms also include information about the enforcement mechanisms: *sanctions*, to punish agents that do not obey the norm and *rewards*, for rewarding norm fulfilment.

Definition 5 (Norm) A *norm* n is defined as a tuple $n = \langle id, D, T, A, E, C, S, R \rangle$, 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 is the norm condition that 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.

Since Magentix2 builds on the event tracing approach to monitoring, the conditions A, E and C are expressed in terms of event templates.

As previously mentioned, the OMS provides agents with services for creating and deleting norms on-line. Once norms have been registered they are in force or applicable. Similarly, norms become deleted when they are unregistered. Figure 1 shows an overview of the norm life-cycle.

Fig. 1 Norm life-cycle

As mentioned in the introduction of this paper, we will use an e-market example which consists on an *auction house* that has been implemented as a VO in Magentix2. Heterogeneous agents can enter or leave the *auction house*. To control the system and avoid the potential excesses of malicious agents, this *auction house* is regulated by a set of norms that define which are the rights and responsibilities of each role in terms of obligations, prohibitions and permissions. For instance, let us suppose the existence of norm $n1$ that forbids bidding for an item once the auction corresponding to this item has been closed:

$$\langle n_1, \mathcal{F}, buyer, \langle auctionEnd, -, item(I)\rangle, \langle auctionStart, -, item(I)\rangle, \langle bid, -, item(I)\rangle, -, -\rangle$$

According to norm n_1 once the $\langle auctionEnd, -, item(I)\rangle$ event is sent, any agent that enacts the *buyer* role is forbidden to bid for the item I. This prohibition expires when the item I is auctioned again (i.e., when the $\langle auctionStart, -, item(I)\rangle$ event is sent).

3.2.3 Instance Definition

When the activation condition of a norm holds; i.e., the activation event is detected, then it becomes active and several norm *instances* (or instances for short) are created, according to the possible groundings of the activation condition.

Definition 6 (Instance) Given a norm $n = \langle id, D, T, A, E, C, S, R\rangle$ and a perceived event e, an instance i of n is the tuple $i = \langle id', D', T', E', C', S', R'\rangle$, where:

- $unification(e, A)$ is true, i.e., there is a substitution σ such that $matching(e, \sigma(A))$ is true (the norm is active);
- $C' = \sigma(C)$, $E' = \sigma(E)$, $S' = \sigma(S)$, and $R' = \sigma(R)$;

– $id' = id, D' = D$ and $T' = T$.

Figure 2 shows an overview of the instance life-cycle. Instances are active when their activation conditions hold. Then, if the instance is an obligation and the norm condition holds, then the obligation is fulfilled. On the contrary, if the instance is a prohibition and the norm condition holds, then it is violated. Finally, when the expiration condition of an instance is true or the norm that gives rise to the instance is deleted, then the instance expires.

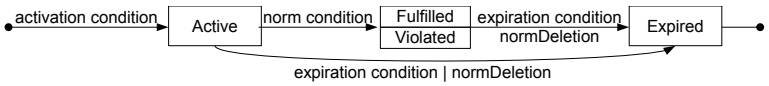


Fig. 2 Instance life-cycle

In our example, let us suppose that the event $\langle auctionEnd, -, item(car)\rangle$ is sent. Thus, norm n_1 will be instantiated as follows:

$$\langle n_1, \mathcal{F}, buyer, \langle auctionEnd, -, item(car)\rangle, \langle auctionStart, -, item(car)\rangle, \langle bid, -, item(car)\rangle, -, -\rangle$$

Definition 7 (Instantiation Function) Given an event $e = \langle Type, Time, Origin, Data\rangle$ and a norm $n = \langle id, D, T, A, E, C, S, R\rangle$, *instantiation* is a function that instantiates norm n as follows:

$$instantiation(e, n) = \langle id', D', T', E', C', S', R'\rangle$$

where: there is a substitution σ such that $matching(e, \sigma(A))$ is true; $C' = \sigma(C)$, $E' = \sigma(E)$, $S' = \sigma(S)$, and $R' = \sigma(R)$; $id' = id, D' = D$ and $T' = T$.

The operational semantics of norms and instances (i.e., how they are created, deleted, fulfilled and violated) is described in [11].

4 Norm Reasoning Services

Norm Reasoning Services (NRSs) have been designed with the aim of allowing both non-normative and normative agents (with different levels of norm reasoning capabilities) to interact within norm-governed VOs[4].

To provide their functionality, the NRSs require the existence of a Normative Monitor (NM) that keeps track of VOs. Figure 3 shows an overview our proposal. Specifically, the NM subscribes to those events that allow it to monitor the VOs. As a result, the NM is informed by the OMS about norms that have been registered and deregistered and the enactment of roles. With this information the NM observes the behaviour of agents and detects the activation and expiration of instances. The information maintained by the NM is consulted by the norm reasoning services when they answer agent requests. This picture also illustrates the use of MaNEA to persuade agents without norm reasoning capabilities to use the NRSs. Specifically, this picture shows an agent without norm reasoning capabilities that has been sanctioned.

[4] Note that the NRSs are a set of related functionalities provided to agents. Thus, NRSs cannot be modelled as agents since they are not autonomous goal-driven entities.

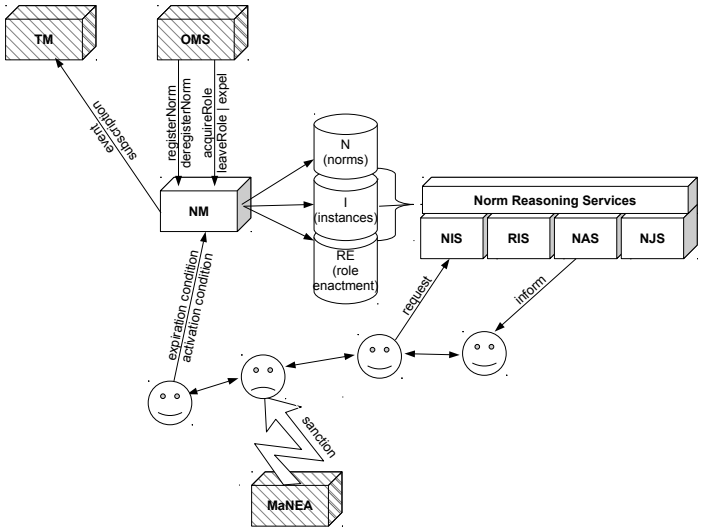


Fig. 3 NRSs architecture (ovals denote agents, cylinders denote lists or data stores, grey boxes denote those components of Magentix2 that have a close relationship the NRSs (i..e, the OMS, TM, and MaNEA). The white boxes denote the components we propose in this paper. Links represent the exchanges of information between components and the agents.

4.1 Normative Monitor (NM)

The NM is responsible for monitoring VOs and providing NRSs with the information that they require. Specifically, it maintains three lists that contain the set of norms (N), instances (I) and the roles that are enacted by agents (RE) at a given moment. Moreover, it records information that can be used to judge past actions. Specifically, it maintains two log files: the log named $LogI$ contains information about the activation and expiration of instances, and the $LogRE$ contains information about which roles agents are playing (or played) at a given moment [5]. To maintain these lists and logs, the NM subscribes to the events sent by the OMS related to the creation and deletion of norms (i.e., $registerNorm$ and $deregisterNorm$ events) and the enactment of roles (i.e., $acquireRole$, $leaveRole$ and $expel$ events). Algorithm 1 illustrates the pseudocode of the control loop performed by the NM. Each time the NM receives an event (e), it handles the event according to the event type. The NM carries out a process that can be divided into three differentiated tasks: norm management, instance management and role enactment management.

4.1.1 Norm Management

Algorithm 2 contains the portion of pseudocode corresponding to the norm management process. 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. When a norm is deregistered, then the NM removes

[5] The lists and log files may be implemented as blackboards, a database that can be accessed by the NRSs, or simply as files that are shared with the NRSs.

Algorithm 1 Normative Monitor Control Loop

Require: Norm list N
Require: Instance list I
Require: Instance log $LogI$
Require: Role Enactment list RE
Require: Role Enactment log $LogRE$
Add $\langle subscription, NM, \langle registerNorm, OMS, - \rangle\rangle$ to E_{Out}
//where NM stands for Norm Monitor
Add $\langle subscription, NM, \langle deregisterNorm, OMS, - \rangle\rangle$ to E_{Out}
Add $\langle subscription, NM, \langle acquireRole, OMS, - \rangle\rangle$ to E_{Out}
Add $\langle subscription, NM, \langle leaveRole, OMS, - \rangle\rangle$ to E_{Out}
Add $\langle subscription, NM, \langle expel, OMS, - \rangle\rangle$ to E_{Out}
while E_{In} is not empty **do**
  Retrieve e from E_{In} // $e = \langle Type, Time, Origin, Data \rangle$
  //...
  // Norm Management
  //...
  // Instance Management
  //...
  // Role Enactment Management
end while

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 registers the expiration of the instance in the corresponding log and unsubscribes from the expiration event.

Algorithm 2 Norm Management

if $Type = registerNorm$ **then** // $Data = \langle id, D, T, A, E, C, S, R \rangle$
  Add $Data$ to N
  Add $\langle subscription, NM, A \rangle$ to E_{Out}
end if
if $Type = deregisterNorm$ **and** $Data$ **in** N **then** // $Data = \langle id, D, T, A, E, C, S, R \rangle$
  Remove $Data$ from N
  Add $\langle unsubscription, NM, A \rangle$ to E_{Out}
  for all i **in** I **do** // $i = \langle id', D', T', E', C', S', R' \rangle$
    if $id' = id$ **then**
      Remove i from I
      for all (i', t_{In}, t_{Out}) **in** $LogI$ **do**
        if $i' = i$ **and** $t_{Out} = null$ **then**
          Remove (i', t_{In}, t_{Out}) from $LogI$
          Add $(i, t_{In}, Time)$ to $LogI$
        end if
      end for
      Add $\langle unsubscription, NM, E' \rangle$ to E_{Out}
    end if
  end for
end if

4.1.2 Instance Management

According to Algorithm 3, when the activation event of a norm is received, then the NM instantiates the norm and adds it to the instance list. At this moment, the

NM registers the creation of the instance in the corresponding log and subscribes to the expiration event. Similarly, when the NM receives the expiration event of any instance, then it removes it from the instance list, unsubscribes from the expiration event and registers the expiration of the instance in the log file.

Algorithm 3 Instance Management

for all n **in** N **do** // $n = \langle id, D, T, A, E, C, S, R \rangle$
  if $unification(e, A)$ **then** // *the norm is active*
    $i := instantiation(e, n)$ // $i = \langle id', D', T', E', C', S', R' \rangle$ *is an instance*
    if i **not in** I **then**
      Add i to I
      Add $(i, Time, null)$ to $ILog$
      Add $\langle subscription, NM, E' \rangle$ to E_{Out}
    end if
  end if
end for
for all i **in** I **do** // $i = \langle id', D', T', E', C', S', R' \rangle$
  if $unification(e, E')$ **then**
    Remove i from I
    for all (i', t_{In}, t_{Out}) **in** $LogI$ **do**
      if $i' = i$ **and** $t_{Out} = null$ **then**
        Remove (i', t_{In}, t_{Out}) from $LogI$
        Add $(i, t_{In}, Time)$ to $LogI$
      end if
    end for
    Add $\langle unsubscription, NM, E' \rangle$ to E_{Out}
  end if
end for

4.1.3 Role Enactment Management

Algorithm 4 illustrates the 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 NM updates the role enactment list and the log file. When the OMS informs that an agent is not longer playing a role, then both the role enactment list and log are updated.

Algorithm 4 Role Enactment Management

if $Type = acquireRole$ **then** // *Data is a pair* $(AgentID, RoleID)$
  Add $Data$ to RE
  Add $(Data, Time, null)$ to $LogRE$
end if
if $Type = leaveRole$ **or** $Type = expel$ **then** // *Data is a pair* $(AgentID, RoleID)$
  Remove $Data$ from RE
  for all $((agentID, roleID), t_{In}, t_{Out})$ **in** $LogRE$ **do**
    if $agentID = AgentID$ **and** $roleID = RoleID$ **and** $t_{Out} = null$ **then**
      Remove $((agentID, roleID), t_{In}, t_{Out})$ from $LogRE$
      Add $((agentID, roleID), t_{In}, Time)$ to $LogRE$
    end if
  end for
end if

4.2 Norm Reasoning Services

Magentix2 allows heterogeneous agents to interact via FIPA-ACL messages. Similarly, NRSs are provided with mail boxes for receiving or sending FIPA-ACL messages (M_{In} and M_{Out})[6]. For the purpose of this paper we will define a message as a tuple $\langle Type, Sender, Receiver, Content\rangle$; where $Type$ contains the message performative, $Sender$ contains the ID of the entity that has delivered the message, $Receiver$ contains the identifier of the entity to which the message is addressed, and $Content$ contains the content of the message. Agents access NRSs by sending a request message to the corresponding service. The result of the service is sent back to the agent through an inform message[7].

4.2.1 Norm Information Service (NIS)

The existence of a norm enforcing architecture, which persuades agents to fulfil norms by applying sanctions and rewards, does not ensure that all agents are considering the same norms. To solve this problem, we have defined a service that allows agents to know the norms that they should take into account. In our example, suppose that an *auditor* agent (a) analyses the performance of the *auction house* and that it aims at ascertaining the influence of the norms on the transactions that take place in the *auction house*. Thus, it needs to gather evidences from both the actions and the norms. Specifically, it needs to know not only which norms are in force, but also the instances that are active at a given moment.

The Norm Information Service (NIS) is in charge of proving information about the norms and the instances that have been created out of these norms. Requesting the NIS is equivalent to recognising and acquiring norm and instances. Algorithm 5 contains the pseudocode corresponding to this functionality.

Algorithm 5 Norm Information Service Control Loop

```
1: while M_In is not empty do
2:    Retrieve m from M_In // m = ⟨Type, Sender, Receiver, Content⟩
3:    if Type = request and Receiver = NIS and Content = norm then
4:       Add ⟨inform, NIS, Sender, norm(N)⟩ to M_out
5:    end if
6:    if Type = request and Receiver = NIS and Content = instance then
7:       Add ⟨inform, NIS, Sender, instance(I)⟩ to M_out
8:    end if
9: end while
```

In our example, if a wants to know the norms are in force, it could asks NIS about the norms that are in force by sending the following message:

$$\langle request, a, NIS, norm\rangle$$

Then the NIS checks the list of norms that are in force and sends the following message:

$$\langle inform, NIS, a, norm(\{\langle n_1, \mathcal{F}, buyer, \langle auctionEnd, -, item(I)\rangle, \langle auctionStart, -, item(I)\rangle, \langle bid, -, item(I)\rangle, -, -\rangle\})\rangle$$

[6] Do not confuse with the mail boxes E_{In} and E_{Out} for receiving/sanding events.

[7] Note that this is a simplification of the FIPA Request Interaction Protocol.

4.2.2 Relevance Information Service (RIS)

Even if agents know the same norms it does not imply that all agents apply (consider as active) the same set of instances; e.g, an agent with limited capabilities may not be aware that it is under the influence of an instance. To address this problem we have created a service that informs agents about the instances that are relevant to a particular agent. In our example, suppose that a *buyer* agent (b_1) enters the *auction house*. Agent b_1 wants to know which its expected behaviour is and which the expected behaviours of its interaction partners are.

The Relevance Information Service (RIS) is in charge of proving information about the instances that are relevant to a target agent, which is specified in the service request. Thus, requesting the RIS is equivalent to determining which norms concern and are relevant to agents. Algorithm 6 contains the pseudocode corresponding to this functionality. When the service is requested, then the RIS obtains the set of roles that are currently played by the target agent ($RoleList$). After this, the RIS searches the instance list for the instances that are addressed to the roles currently played by the target agent ($InstanceList$).

Algorithm 6 Relevance Information Service Control Loop

while M_{In} is not empty **do**
  Retrieve m from M_{In} // $m = \langle Type, Sender, Receiver, Content\rangle$
  if $Type = request$ **and** $Receiver = RIS$ **and** $Content = relevance(AgentID)$ **then**
    $InstanceList := \{\}$
    $RoleList := \{\}$
    for all $(agentID, roleID)$ **in** RE **do**
      if $AgentID = agentID$ **then**
        Add $roleID$ to $RoleList$
      end if
    end for
    for all i **in** I **do** // $i = \langle id', D', T', E', C', S', R'\rangle$
      if T' **in** $RoleList$ **then**
        Add i to $InstanceList$
      end if
    end for
    Add $\langle inform, RIS, Sender, relevance(InstanceList)\rangle$ to M_{out}
  end if
end while

For instance, if b_1 is not capable of reasoning about which of the instances are relevant to it, b_1 could ask RIS about its relevant instances by sending the following message:

$$\langle request, b_1, RIS, relevance(b_1)\rangle$$

Then the RIS checks the instances that are relevant to b_1. Since b_1 is affected by the instance that has been created out of n_1 (as detailed in Section 2.2), this instance is relevant to b_1, so that the RIS sends the following message:

$$\langle inform, RIS, b_1, relevance(\{\langle n_1, \mathcal{F}, buyer, \langle auctionEnd, -, item(car)\rangle, \\ \langle auctionStart, -, item(car)\rangle, \langle bid, -, item(car)\rangle, -, -\rangle\})\rangle$$

4.2.3 Norm Advice Service (NAS)

Now suppose that b_1 does not even have capabilities for reasoning about instances (e.g., b_1 does not understand the language used for representing instances). Thus, it could not consider them when it decides its next action. However, b_1 would like to respect the norms to avoid sanctions and maintain a good reputation.

The Norm Advice Service (NAS) determines which goals (*NormativeGoals*) must be pursued according to norms. Requesting the NAS is equivalent to accepting norms and making norm compliance decisions. Agents that request this service provide their intrinsic desires, which are a set of literals formed by event templates (or the negation of event templates). This set represents the actions or states of affairs pursued (or avoided) by the agent. With this information, the NAS calculates which goals must be pursued according to norms and to what extent these goals are advisable for the agent. Algorithm 7 contains the pseudocode corresponding to the NAS. Once the service receives a request, then the NAS calculates the set of instances that are relevant to the petitioner agent. For each relevant instance, the NAS computes how much the instance is advisable for the agent. This advisability degree is calculated by a function as follows:

Definition 8 (Advisability Function) Given an instance (i) and a set of goals ($Goals$) the advisability of this instance is calculated as follows:

$$advisability(i, Goals) = \frac{f_{Interest}(i, Goals) + f_{Expectations}(i, Goals)}{2}$$

The advisability function is defined as the average among the values calculated by two functions:

- $f_{Interest}$. This function considers the influence of norm compliance on the agent's goals:

$$f_{Interest}(\langle id', D', T', E', C', S', R' \rangle, Goals) = \begin{cases} 1 & \text{if } D' = \mathcal{O} \text{ and } C' \in Goals \\ 1 & \text{if } D' = \mathcal{F} \text{ and } \neg C' \in Goals \\ 0 & \text{otherwise} \end{cases}$$

- $f_{Expectations}$. This function considers the influence of the external enforcement on the agent's goals:

$$f_{Interest}(\langle id', D', T', E', C', S', R' \rangle, Goals) = \begin{cases} 1 & \text{if } R' \in Goals \text{ and } \neg S' \in Goals \\ 0.5 & \text{if } R' \in Goals \text{ or } \neg S' \in Goals \\ 0 & \text{otherwise} \end{cases}$$

The value calculated by the advisability function is used to annotate the goal that is added to the *NormativeGoals* set. Each normative goal is annotated with a real number that represents to what extent the agent is interested in complying with the instance that has inferred the normative goal. If the instance is an obligation, a new goal to pursue the obliged condition is added. On the contrary, if the instance is a prohibition, a new goal for avoiding the forbidden condition is added.

Let us assume that b_1 is only interested in buying cars. Therefore, its goal set only contains one proposition $\langle bid, -, item(car) \rangle$. Since it wants to obey norms, it asks NAS about its normative goals by sending the following message:

$$\langle request, b_1, NAS, \{\langle bid, -, item(car) \rangle\} \rangle$$

Algorithm 7 Norm Advice Service Control Loop

while M_{In} is not empty **do**
  Retrieve m from M_{In} // $m = \langle Type, Sender, Receiver, Content \rangle$
  if $Type = request$ **and** $Receiver = NAS$ **and** $Content = compliance(Goals)$ **then**
    $NormativeGoals := \{\}$
    $RoleList := \{\}$
    for all $(agentID, roleID)$ **in** RE **do**
      if $Sender = agentID$ **then**
        Add $roleID$ to $RoleList$
      end if
    end for
    for all i **in** I **do** // $i = \langle id', D', T', E', C', S', R' \rangle$
      if T' **in** $RoleList$ **then**
        if $D' = \mathcal{F}$ **then**
          $NormativeGoals := NormativeGoals \cup \{(\neg C', advisability(i, Goals))\}$
        end if
        if $D' = \mathcal{O}$ **then**
          $NormativeGoals := NormativeGoals \cup \{(C', advisability(i, Goals))\}$
        end if
      end if
    end for
    Add $\langle inform, NAS, Sender, compliance(NormativeGoals) \rangle$ to M_{out}
  end if
end while

The NAS checks the instances that are relevant to b_1 and calculates the advisability of complying with them. Only the instance that has been created out of n_1 is relevant to b_1. The advisability of complying with this instance is 0 and the NAS sends the following message:

$$\langle inform, NAS, b_1, compliance(\{(\neg\langle bid, -, item(car)\rangle, 0)\})\rangle$$

Since $\neg\langle bid, -, item(car)\rangle$ contradicts the main goal of agent b_1 and the advisability of complying with this norm is 0, then agent b_1 decides to violate the norm and it makes a bid.

In this paper we propose to determine the agent advisability of complying with a given instance by simply considering the effect of this instance on the agent goals. However, if the norm reasoning services are provided with domain information, then a more complex decision making procedure could be used. For example, the decision making mechanism proposed in [10] could be used if the norm reasoning services are informed about the importance of each norm and the situations that are predicted to occur when the norms are violated.

It should be noticed that the NAS does not ensures that the set of normative goals is consistent: i.e., it is possible that a proposition and its negation belong to the set of normative goals. It is the responsibility of agents to decide which of the normative goals will be pursued, resolving conflicts between normative goals and their intrinsic goals.

4.2.4 Norm Judgement Service (NJS)

Finally, suppose that a *seller* agent, identified by s_1, receives the bid made by b_1. s_1 may be unable to judge whether this bid is legal or not with respect to the normative system. Specifically, it may want to know whether b_1 has violated any

norm and its bid must be ignored, or whether b_1 has acted legally and b_1 wins the auction. In case the seller is not able to judge this bid, we propose it calls the NJS.

The Norm Judgement Service (NJS) allows agents to determine if an event that may have happened at some moment in the past is legal or not according to the normative system. Requesting the NJS is equivalent to accepting norms and making norm judgements on the basis of normative expectations. Therefore, the NJS judges the performance of this event with respect to the context (i.e., active instances and roles, and their interplay) when the event was performed. Algorithm 8 contains the source code corresponding to this functionality. When the NJS receives a request then it determines: which roles were played by the agent that performed the event ($Origin$), at the time the event was performed ($Time$). Then, the NJS determines which norms were relevant to that agent at that time. Agents can play several roles simultaneously and these roles may be affected by conflicting norms. For this reason, the NJS counts the number of prohibitions that were violated by the event ($FulfilmentCount$), and the number of obligations that were fulfilled by the event ($ViolationCount$). It is up to the agents how to use this information: e.g., to select the most suitable interaction partners, to exclude non-compliant agents, etc.

In our example, s_1 asks NJS about the bid made by b_1 by sending the following message:

$$\langle request, s_1, NJS, \langle bid, time, b_1, item(car)\rangle\rangle$$

The NJS answers with the following message:

$$\langle inform, NJS, s_1, normJudgement(1, 0)\rangle$$

Then s_1 is able to know that the bid is illegal, and thus, s_1 ignores.

4.3 Scalability

Initially, there is a single NM registered in the Magentix2 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 unsubscribe from the registerNorm event. Thus, the new NM is responsible for controlling the activation of the new norms. 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. Moreover, the data stores containing the norms, instances, and role enactment information can be distributed and/or replicated. Similarly, the NRSs been described assuming that there is a single service provider of each service. However, these services may be formed by a set of federated service providers that whose number can be dynamically adapted according to service demands by performing cloning and self-deletion operations.

These replication and distribution mechanisms are a simple example that illustrates how the NRSs infrastructure can dynamically adapt to changes in the scale MAS (i.e., situations in which the number of agents or norms to be controlled changes dramatically). However, the definition of more elaborated procedures for adapting dynamically to changing environments [27] is a complex issue that is out the scope of this paper.

Algorithm 8 Norm Judgement Service Control Loop

while M_{In} is not empty **do**
  Retrieve m from M_{In} // $m = \langle Type, Sender, Receiver, Content \rangle$
  if $Type = request$ **and** $Receiver = NJS$ **and** $Content = judgement(e)$ **then** // $e = \langle Type, Time, Origin, Data \rangle$
    $RoleList := \{\}$
    for all $((agentID, roleID), t_{In}, t_{Out})$ **in** $RELog$ **do**
      if $Origin = agentID$ **and** $t_{In} \leq Time$ **and** ($t_{Out} = null$ **or** $t_{Out} > Time$) **then**
        Add $roleID$ to $RoleList$
      end if
    end for
    $ViolationCount := 0$
    $FulfilmentCount := 0$
    for all i **in** $ILog$ **do** // $(\langle id', D', T', E', C', S', R' \rangle, t_{In}, t_{Out})$
      if T' **in** $RoleList$ **and** $t_{In} \leq Time$ **and** ($t_{Out} = null$ **or** $t_{Out} > Time$) **then**
        if $unification(e, C')$ **then**
          if $D' = \mathcal{O}$ **then**
            $FulfilmentCount := FulfilmentCount + 1$
          end if
          if $D' = \mathcal{F}$ **then**
            $ViolationCount := ViolationCount + 1$
          end if
        end if
      end if
    end for
    Add $\langle inform, NJS, Sender, normJudgement(ViolationCount, FulfilmentCount) \rangle$ to M_{out}
  end if
end while

5 Evaluation

This section provides an evaluation of the NRSs in terms of the behaviour exhibited by agents that make different uses of our NRSs in a general situation. We have performed two different type of experiments to analyse the performance of the NRSs, to determine their usefulness and to compare them to other similar proposals. In the first type of experiments, we seek to illustrate the performance and the usefulness of the NRSs. To this aim we have performed a set of random simulations to illustrate the performance of the NRSs and their effect on norm compliance (i.e., the number of norms that are complied by agents) in a wide range of situations. Moreover, we have analysed the effect of the different elements considered in the simulations on the performance of the NRSs. These experiments are described in Section 5.1. In the second type of experiments, we compare the performance of the NRSs with other similar proposals that provide agents with normative information. Specifically, we compare norm compliance when agents use the different proposals for being informed about the norms. This experiment is described in Section 5.2.

5.1 Performance Experiments

The experiments described in this section have two main goals: (i) to analyse the effect of the NRSs on norm compliance, and (ii) to evaluate the performance of

the NRSs in terms of service requests and the events that are sent. Specifically, in these experiments we demonstrate that the use of the NIS, the RIS and the NAS[8] aids agents to comply with norms. Moreover, we also demonstrate that the NRSs operate effectively in changing environments in which the number of agent, norms, etc., may change.

5.1.1 Simulation Description

We have considered a scenario in which there is a set of agents that belong to the same VO in Magentix2 (e.g., an e-market). Agents direct their activity towards achieving goals (e.g., participate in auctions for buying some items). To achieve these goals agents know a fixed and predefined set of plans. Moreover, a set of norms has been registered in the OMS for specifying the agents expected behaviour. Agents may perform different actions during their execution causing changes in the environment (e.g., sold items may be delivered) and in the normative state (i.e., a norm may become instantiated or expired). We have built a simulator of this scenario with the parameters that we sum up in Table 2.

Algorithm 9 contains the pseudo code of this simulator. According to this algorithm, in our simulator there is a VO in which a set of agents perform actions to pursue their goals. Specifically, each simulation is populated by agents with different norm-reasoning capabilities. Specifically four main agent types are considered in our simulation. Thus, 4 agents are created in each simulation (one of each type). These agents pursue goals that describe states of affairs. Each agent pursues a set of goals that are randomly selected form a set of 20 goals. A plan is a sequence of actions that allows agents to achieve one goal. Specifically, we generate randomly a set of 300 plans in each execution. We consider that agents are able to carry out 100 different actions. Thus, for each plan we generate randomly a sequence of actions (line 1 on Algorithm 9). Specifically, the length of plans (in terms of number of actions they involve) varies randomly within the $[1, 10]$ interval. All agents know the same plans for achieving goals.

Parameter	Fixed Value	Experimentation Interval
# of steps	100	$[10, 1000]$
# of actions	100	$[50, 200]$
# of plans	300	$[20, 1000]$
# of norms	20	$[1, 100]$
# of agents	4	$[4, 100]$

Table 2 Parameters used in the experiments

Norm Definition Agents enact one or more roles randomly in each simulation. Specifically, 4 different roles have been considered. In order to specify the desired behaviour of these roles, 20 norms are created in each simulation (line 2 on Algorithm 9). Each norm is randomly assigned to a single role, which is the target of

[8] Note that the effect of the NJS on norm compliance depends on the enforcement actions that agents carry out when they are informed by the NJS about the fulfilment or violation of norms. The definition of sanctioning and rewarding systems is out of the scope of this paper. For this reason, the NJS has not been considered in the experiments.

Algorithm 9 Pseudocode of algorithm executed by our simulator

Require: Action set Act
Require: Goal set G
1: $P := randomPlanCreation(G, Act)$// Plan Definition
2: $N := randomNormCreation(Act)$// Norm Definition
3: $E := randomEnvironmentCreation(N)$// Environment Definition
4: $NU := randomNormUnawareAgentCreation(G, P)$// Agent Definitions
5: $NI := randomNormIncapableAgentCreation(G, P)$
6: $NR := randomNormReasoningAgentCreation(G, P)$
7: $NA := randomNormAutonomousAgentCreation(G, P)$
8: $Fulfilment_{NU} := 0$// Obligation Fulfilment Count Initialization
9: $Fulfilment_{NI} := 0$
10: $Fulfilment_{NR} := 0$
11: $Fulfilment_{NA} := 0$
12: $Violation_{NU} := 0$// Prohibition Violation Count Initialization
13: $Violation_{NI} := 0$
14: $Violation_{NR} := 0$
15: $Violation_{NA} := 0$
16: $Event_{NA} := 0$// Event Reception Count Initialization
17: $Event_{NM} := 0$
18: $Request_{NAS} := 0$// Service Request Counts Initialization
19: $Request_{RIS} := 0$
20: $Request_{NIS} := 1$ // Norm-autonomous agents request NIS once
21: **while** the simulator has not been executed all the steps **do**
22: $Action_{NU} := NU.executeAction()$
23: $Fulfilment_{NU} := Fulfilment_{NU} + E.checkFulfilments(Action_{NU})$
24: $Violation_{NU} := Violation_{NU} + E.checkViolations(Action_{NU})$
25: **if** $NI.requestNRS()$ =**true then**
26: $Request_{NAS} := Request_{NAS} + 1$
27: **end if**
28: $Action_{NI} := NI.executeAction()$
29: $Fulfilment_{NI} := Fulfilment_{NI} + E.checkFulfilments(Action_{NI})$
30: $Violation_{NI} := Violation_{NI} + E.checkViolations(Action_{NI})$
31: **if** $NR.requestNRS()$ =**true then**
32: $Request_{RIS} := Request_{RIS} + 1$
33: **end if**
34: $Action_{NR} := NR.executeAction()$
35: $Fulfilment_{NR} := Fulfilment_{NR} + E.checkFulfilments(Action_{NR})$
36: $Violation_{NR} := Violation_{NR} + E.checkViolations(Action_{NR})$
37: $Action_{NA} := NA.executeAction()$
38: $Fulfilment_{NA} := Fulfilment_{NA} + E.checkFulfilments(Action_{NA})$
39: $Violation_{NA} := Violation_{NA} + E.checkViolations(Action_{NA})$
40: $Event_{NA} := Event_{NA}+NA.eventCounts(Action_{NU}, Action_{NI}, Action_{NR}, Action_{NA})$
41: $Event_{NM} := Event_{NM}+NM.eventCounts(Action_{NU}, Action_{NI}, Action_{NR}, Action_{NA})$
42: $E.instanceDynamics(Action_{NU}, Action_{NI}, Action_{NR}, Action_{NA})$
43: **end while**

the norm. As previously mentioned, norms are defined in terms of three conditions, which correspond to the activation, expiration and normative condition (i.e., A, E and C). We assume that these conditions are expressed in terms of event templates that inform about the actions that agents perform. Thus, in each simulation the actions that are included in the activation, expiration and normative condition of each norm are randomly selected from the set of 100 actions. Initially, we create randomly a set of instances out of the norms defined in the simulation (line 3 on Algorithm 9).

Agent Definition. As previously mentioned, agents have different capabilities for taking norms into account in their decisions. Specifically, we consider the following agent types:

Norm-unaware agents are the least sophisticated agents. They do not realise norms and try to achieve their goals regardless of norms. Thus, they start their execution by selecting randomly the goal to be pursued as one of their pending goals. Then, they select randomly one plan that achieves this goal. In each step they execute one action of the plan. They repeat this process until they achieve all their goals[9].

Norm-incapable agents are unable to make their own decisions according to norms and their goals. However, they want to fulfil norms and, as a consequence, they make use of the NAS proposed in this paper. Specifically, they request the NAS in order to know the best plan to be executed according to the instances of norms that are addressed to them and their goals. Then, they execute all actions indicated by the plan. Once they have completed the execution of this plan, they ask the NAS service again. They repeat the whole process until they achieve all their goals.

Norm-reasoning agents are able to make their own decisions according to norms and their goals. However, they need to be informed about the specific instances that are relevant to them at a given moment. Thus, they start their execution by asking the RIS about the specific instances that affect them. Then, they select the best plan to be executed according the instances that are addressed to them and their own goals. In each step they execute one action of this plan. Once they have completed the execution of this plan, they ask the RIS service again. They repeat the whole process until they achieve all their goals.

Norm-autonomous agents are also able to make their own decisions according to norms and their goals. Furthermore, these agents are able to keep track of the activation and expiration of instances and, as a consequence, they know which instances they must consider at any moment. When they join a VO they request the NIS in order to know the norms (not the instances) that are in force in this VO. As the NM does, these agents also subscribe to the events that make norms active and to the events that make instances expire (i.e., to the events that allow them to know which norms have been instantiated). However, in contrast to the NM they are only interested in receiving information about the specific norms that are addressed to them. Then, they select the best plan to be executed according the instances that are addressed to them and their goals. In each step, they execute one action of this plan. They repeat the whole process until they achieve all their goals.

Our simulations are populated by one agent of each type (lines 4-7 in Algorithm 9). Therefore, normative and non-normative agents are created. Specifically, *norm-unaware* and *norm-incapable* agents do not have norm reasoning capabilities and are non-normative. On the contrary, *norm-reasoning* and *norm-autonomous* are normative agents.

In each step of the simulations, we simulate the actions performed by each agent, the service request made by each agent to the corresponding NRS (NIS, RIS, and NAS) and the events that are sent to the NM and to agents (lines 21-42

[9] For simplicity we assume that there is a perfect execution of actions: i.e., agents never fail when they perform actions.

in Algorithm 9). With this information we calculate the percentage of obligations that are fulfilled and the percentage of prohibitions that are violated by each agent type, the number of service request that are made by each agent type and the number of events that are sent to the NM and to agents. We have repeated each experiment 10000 times to support the findings with statistically significant evidence.

We have performed 5 different experiments to illustrate the performance of the NRSs with respect to: the number of steps that the simulator is executed, the number of norms, the number of agents, the number of actions, and the number of plans. In the experiments, the values of the parameters range as indicated by the *Experimentation Interval* column in Table 2. The results of these experiments are described below.

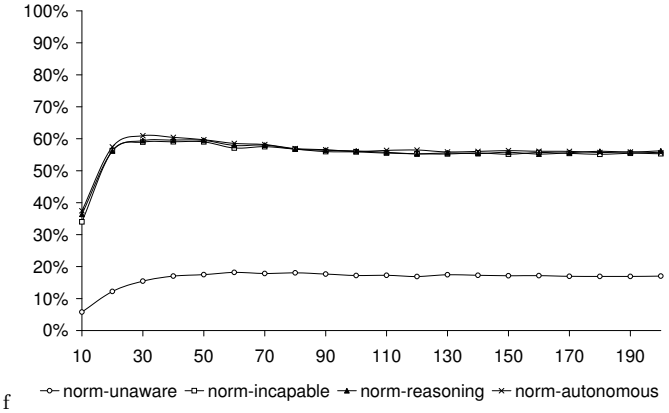
5.1.2 Results

Number of Steps. This experiment is aimed at determining the number of steps that the simulator must be executed to obtain significant results.

Figure 4 illustrates the percentage of obligations that are fulfilled on average per agent type with respect to the number of steps that the simulation is executed. All agents that use the NRSs (i.e., *norm-incapable*, *norm-reasoning* and *norm-autonomous* agents) comply with a similar percentage of obligations. This means that with the information provided by the NRSs, agents with limited reasoning capabilities (e.g., *norm-incapable* and *norm-reasoning* agents) are able to comply with as many obligations as agents with full reasoning capabilities (e.g., *norm-autonomous*). When the number of steps is very low agents have lower possibilities to comply with obligations and the percentage of obligations that are fulfilled is low. As the number of steps increases, agents execute more actions and it is more possible that they fulfil obligations. We can observe that the percentage of fulfilled obligations peaks around 30 steps and then decreases slightly over time. This is explained by the fact that all normative agents consider norms to select the next plan to be executed among the ones that achieve one of their goals. When simulations are executed less than 30 steps, agents have achieved few of their goals. Thus, any time normative agents use the norms to select the next plan to be executed, they can choose among several plans. When simulations are executed more than 30 steps, agents have achieved most of their goals. Thus, at some point normative can choose among a few plans. In this situation, the probability of finding a plan that achieves one remaining goal while complying with the norms is low. Finally, when agents have achieved all their goals they stop executing actions and, as a consequence, they cannot fulfil their pending obligations. Note that the obligation fulfilment percentage of all normative agents in our simulation converges to 60%. This is due to the fact that normative agents are not always able to comply with their pending obligations; e.g., it is possible that an agent selects a plan that contains one action that fulfils one of its pending obligations but that when this action is executed the obligation has already expired, so this would count a as violation.

Figure 5 illustrates the percentage of prohibitions that are violated on average per agent type with respect to the number of steps that the simulation is executed. Again, agents that use the NRSs (i.e., *norm-incapable*, *norm-reasoning* and *norm-autonomous* agents) violate a similar percentage of prohibitions. Again,

when the number of steps is very low, agents execute few actions and there is a low probability of violating norms. When the number of steps increases agents execute more actions and it is more possible that they violate prohibitions. In light of the results shown by these two figures, we can conclude that the information provided by the NRSs aids agents comply with norms noticeably; i.e., the percentage of fulfilled obligations is higher in agents that make use of NRSs and the percentage of violated prohibitions is lower in agents that make use of NRSs. Since prohibitions and obligations are considered by agents similarly (i.e., obligations and prohibitions are used to select the next plan to be executed), we can derive the same conclusion from analysing the percentage of fulfilled obligations and the percentage of violated prohibitions. Thus, for the rest of the experiments condicted we only show the percentage of fulfilled obligations.



f

Fig. 4 Percentage of obligations that are fulfilled on average per agent type with respect to the number of steps that the simulation is executed.

Figure 6 illustrates the number of service requests that are made on average per each agent type. Obviously, *norm-unaware* agents never request any of the NRSs. *Norm-incapable* and *norm-reasoning* agents are the ones that send more service requests on average. Specifically, *norm-incapable* agents make a request to the NAS any time they select the next plan to be executed. Similarly, *norm-reasoning* agents make a request to the RIS any time they select the next plan to be executed. Finally, *norm-autonomous* agents make only one request to the NIS at the beginning of the simulation to know the norms that are in force. As the number of steps increases, the number of service requests made by *norm-incapable* and *norm-reasoning* agents also increases.

Figure 7 illustrates the number of events that are received on average per each agent type and the NM. Only *norm-autonomous* agents and the NM receive events. The number of events that are received by the NM is higher than the number of events that are received by *norm-autonomous* agents since the NM must receive events to detect the instantiation of all of the norms, whereas *norm-autonomous* agents only receive the events that allow them to determine the instantiation of

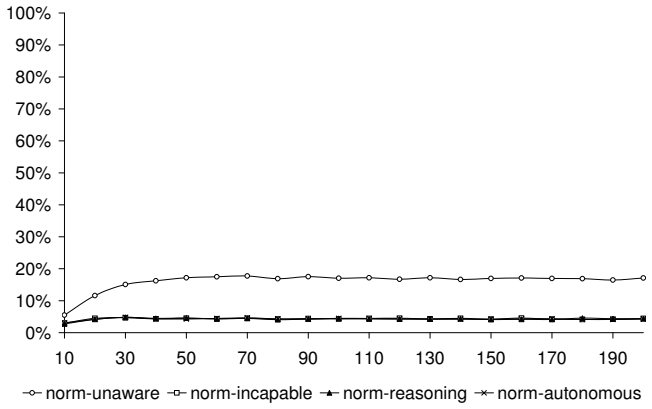


Fig. 5 Percentage of prohibitions that are violated on average per agent type with respect to the number of steps that the simulation is executed.

the norms that affect them[10]. Again, the number of events increases as the number of steps increases. However, when the number of steps is equal or higher to 100, then the number of events remains quite stable. This is explained by the fact that when the number of steps is equal or higher to 100, all agents have achieved all their goals and no more actions are executed. If no more actions are executed, no more events are sent for reporting those actions.

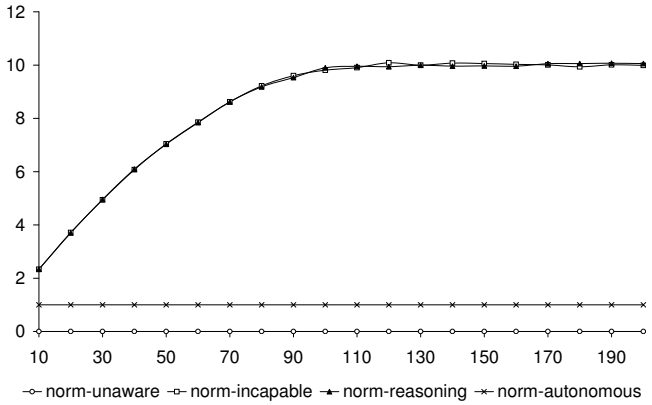


Fig. 6 Number of service requests made on average per agent type with respect to the number of steps that the simulation is executed.

[10] Note that in this experiment there is one *norm-autonomous* agent and it is only affected by a subset of the norms. However, when there is more than one *norm-autonomous* agent, then the number of events sent to these agents is higher than the number of events that is sent to the NM. This can be observed in the following experiments.

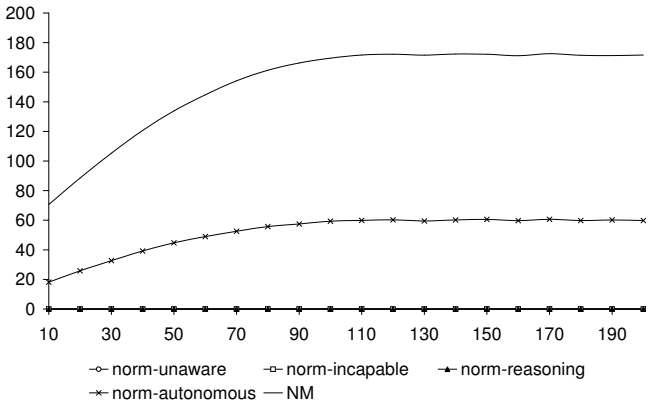


Fig. 7 Number of events received on average per agent type and the Normative Monitor (NM) with respect to the number of steps that the simulation is executed.

As shown in the previous figures, in all experiments the results obtained remain quite stable from 100 steps, so that we have fixed the number of steps to 100 in the following experiments.

Number of Norms. This experiment illustrates the performance of the NRSs when the number of norms that regulate the VO changes. It demonstrates that the NRSs are scalable with an increasing number of norms.

Figure 8 illustrates the percentage of obligations that are fulfilled on average per agent type with respect to the number of norms that are created in each simulation. As the number of norms increases, agents are affected by more obligations and, thus, they are able to comply with a lower percentage of them. However, we can observe that agents that use the NRSs comply with more obligations than *norm-unaware* agents; i.e., *norm-incapable*, *norm-reasoning*, and *norm-autonomous* agents comply with a similar number of obligations regardless their reasoning capabilities.

Regarding the number of service requests (see Figure 9), this number is not affected by the number of norms and, as we expected, the results remain stable regardless of the number of norms.

Finally, Figure 10 illustrates the number of events that are received on average by *norm-autonomous* agents, which are the only agents that receive events for determining which norms are relevant (the other ones obtain this information when they actively contact any of the NRSs), and the NM, which receives the events and updates the information needed by the NRSs accordingly. As we expected, the number of events received by the NM and *norm-autonomous* agents increases linearly with the number of norms; i.e., if there are more norms, more events must be sent to keep norm-instantiation information up to date.

Number of Agents. This experiment shows the performance of the NRSs when the number of agents that populate a VO changes. It shows which of the NRSs are more suitable to handle an increasing number of agents.

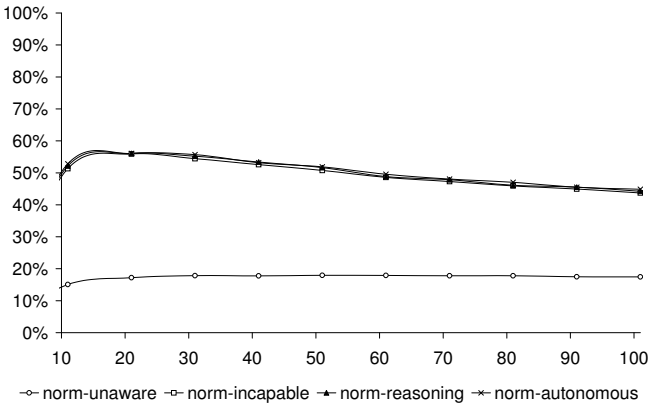


Fig. 8 Percentage of obligations that are fulfilled on average per agent type with respect to the number of norms.

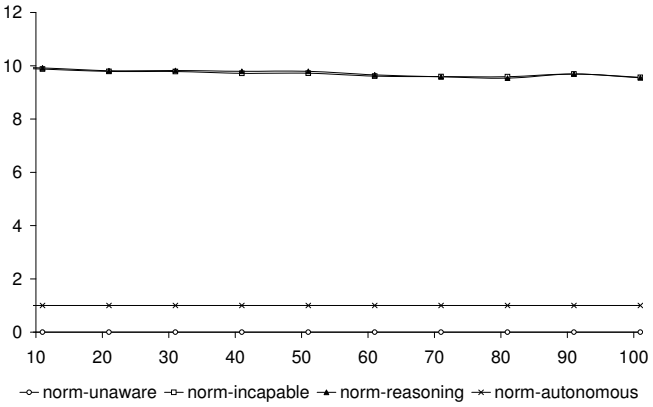


Fig. 9 Number of service requests made on average per agent type with respect to the number of norms.

Figure 11 illustrates the percentage of obligations that are fulfilled on average per agent type with respect to the number of agents that are created in the simulations. In each simulation, we create the same number of agents per agent type. When the number of agents is low, few actions are executed in each step and obligations are active during several steps. As a consequence, agents comply with a higher percentage of obligations since they have more opportunities for complying with them. As the number of agents increases, more actions are executed in each step and obligations are active during fewer steps. For this reason, the percentage of complied obligations decreases in all agents.

Obviously, if there number of agents increases, the number of service requests made by these agents increases as well (see Figure 12). Specifically, the number of service requests increases linearly with the number of agents in case of agents that use the NRSs.

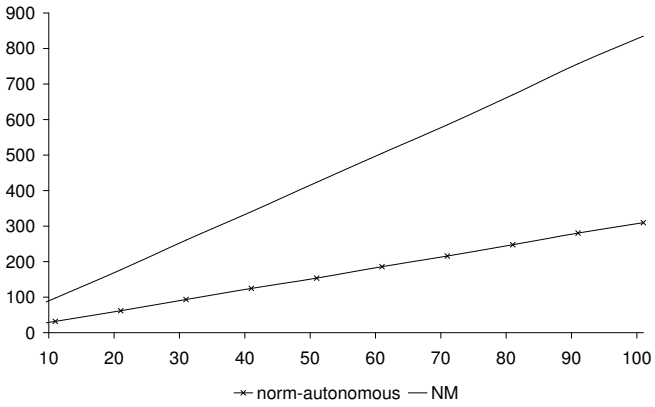


Fig. 10 Number of events received on average per agent type and the Normative Monitor (NM) with respect to the number of norms.

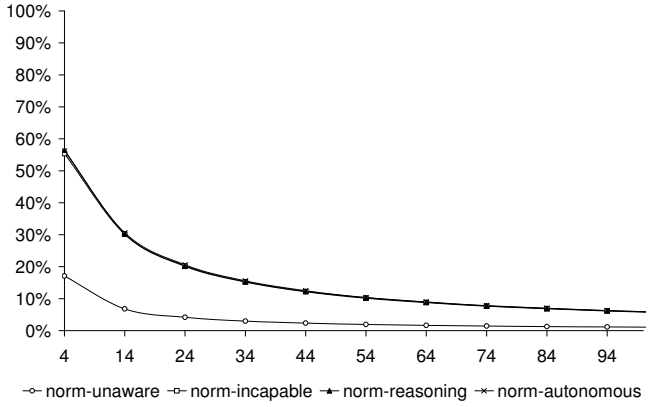


Fig. 11 Percentage of obligations that are fulfilled on average per agent type with respect to the number of agents.

Finally, Figure 13 illustrates the number of events that are received on average by *norm-autonomous* agents and the NM. In this picture we can observe how the number of events sent to the NM increases linearly with the number of agents. This is explained by the fact that if there are more agents then more actions are executed and, thus, more events reporting the execution of actions are sent. However, the number of events that are sent to *norm-autonomous* agents increases exponentially with the number of agents. This is explained by the fact that several agents play the same roles and are affected by the same norms. Thus, all of these agents receive the same events for detecting the instantiation of norms, so that replicated events are sent to several agents. Therefore, the use of the RIS and the NAS, that externalise the detection of norm activations and expirations (so that this task is carried out by the NM), reduces the overload of events that are sent in the system.

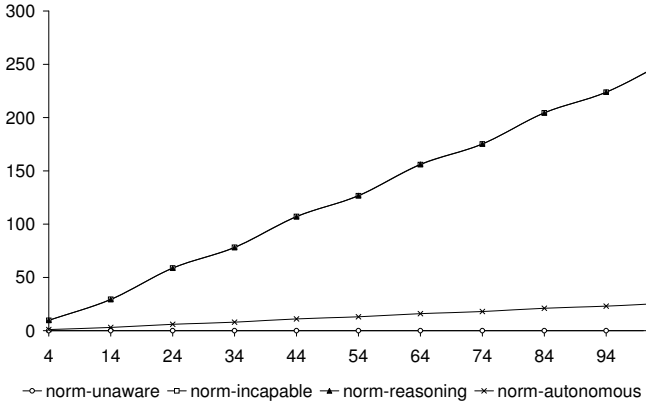


Fig. 12 Number of service requests made on average per agent type with respect to the number of agents.

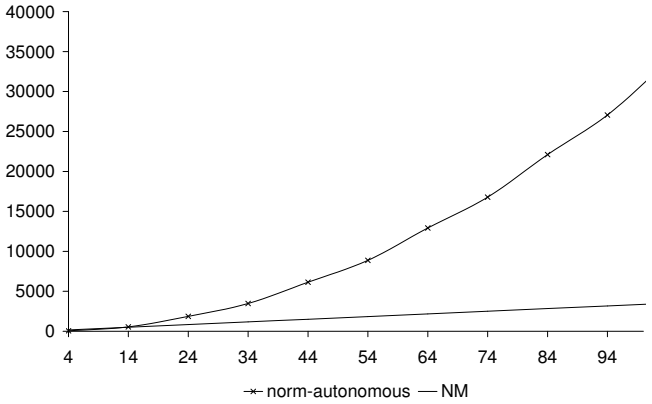


Fig. 13 Number of events received on average per agent type and the Normative Monitor (NM) with respect to the number of agents.

Number of Actions. This experiment illustrates the performance of the NRSs when the number of actions that can be performed by agents changes. It demonstrates that the NRSs remain effective and efficient with an increasing number of actions.

Figure 14 illustrates the percentage of obligations that are fulfilled on average per agent type with respect to the number of actions that agents can execute. When the number of actions is low, then almost all actions have an effect on the state of norms; i.e., they make norms become activated or expired. Therefore, obligations are active for few steps and agents have little opportunities for complying with them. As the number of actions increases, obligations are active for more steps and agents have more opportunities to comply with norms. For this reason, the percentage of complied obligations increases in all agents that use NRSs. However, we can observe that in agents that do not use the NRSs the percentage of complied obligations decreases lightly as there are more actions. This is explained by the fact that if there are more actions, there are fewer probabilities that the obliged

actions are executed. Recall that *norm-unaware* agents randomly pick the plans that fulfil their goals.

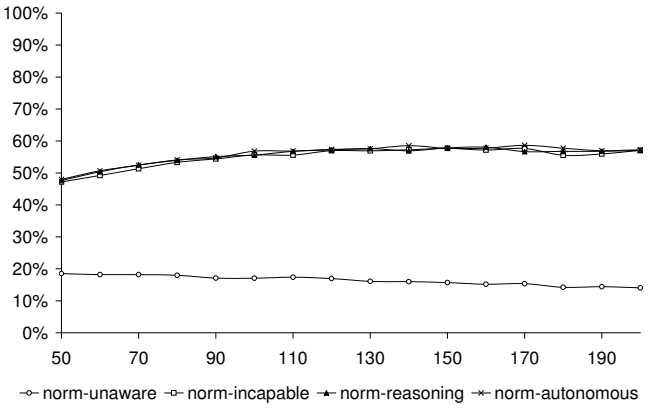


Fig. 14 Percentage of obligations that are fulfilled on average per agent type with respect to the number of actions.

Regarding the number of service requests, this number is not affected by the number of actions and, as we expected, the results remain stable regardless of the number of actions (see Figure 15).

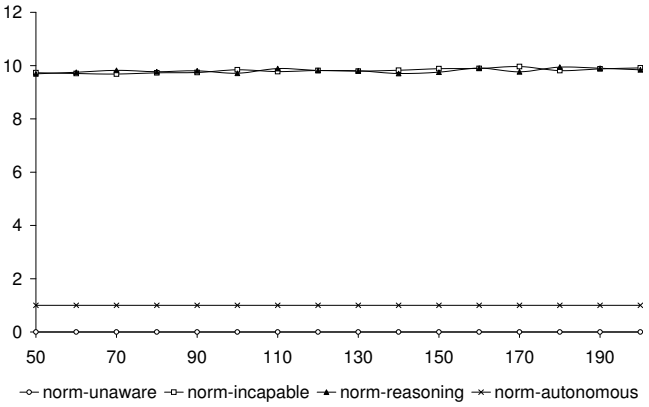


Fig. 15 Number of service requests made on average per agent type with respect to the number of action.

Finally, Figure 16 illustrates the number of events that are received on average by *norm-autonomous* agents (the rest of agents do not receive events) and the NM. As aforementioned, when the number of actions is low almost all actions have an effect on the state of norms and more events reporting those changes are sent. As

the number of actions increases, fewer actions have an effect on the state of norms and fewer events are sent[11].

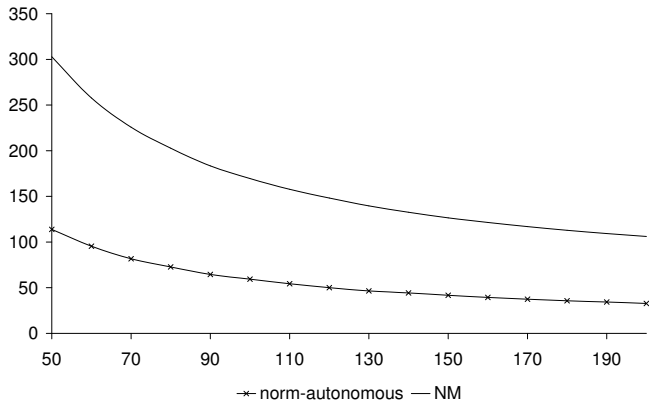


Fig. 16 Number of events received on average per agent type and the Normative Monitor (NM) with respect to the number of actions.

Number of Plans. This experiment shows the performance of the NRSs when the number of plans known by the agents changes.

Figure 17 illustrates the percentage of obligations that are fulfilled on average per agent type with respect to the number of plans that can be executed by agents. When the number of plans is low, then agents know fewer plans that achieve their goals. Thus, agents that make use of the NRSs have few possibilities for complying with norms. As the number of plans increases, these agents know more options to achieve their goals and it is possible to select one that ensures compliance with norms. For this reason, the percentage of complied obligations increases as the number of plans increases in all agents that use NRSs. *Norm-unaware* agents select plans to achieve their goals randomly and, as a consequence, their behaviour is not affected by the number of plans.

Agents made the same service request regardless of the number of plans that they know for achieving their goals. Similarly, the number of plans does not affect the execution of actions that have a normative effect. Therefore, the results in this case have been obtained since they remain stable regardless of the number of plans.

5.1.3 Discussion of Results

In light of all the results of all the experiments described in this section, we can conclude that NRSs allow heterogeneous agents (i.e., agents with different norm

[11] Notice that there is only one *norm-autonomus* agent which is affected by a subset of norms and, as a consequence, only the events that determine the activation and expiration of this subset of norms is sent to it. In contrast, the NM receives events that determine the activation and expiration of all norms.

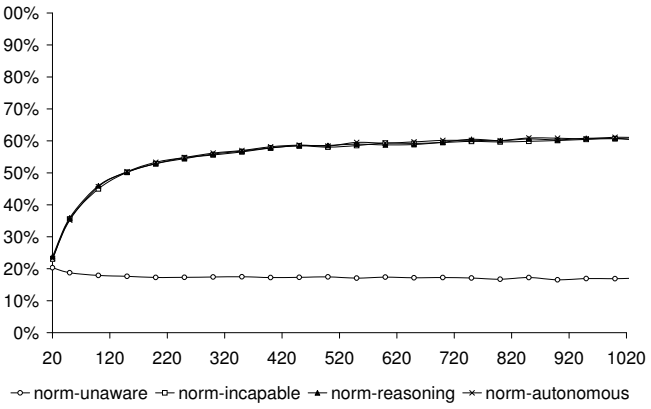


Fig. 17 Percentage of obligations that are fulfilled on average per agent type with respect to the number of plans.

reasoning capabilities) to participate in VO regulated by norms; i.e., to fulfil their goals while complying with norms. Specifically, all agents that use the NRSs comply with a similar percentage of norms, which means that the information provided by the NRSs allow agents with limited reasoning capabilities to be able to behave (i.e., complying with norms) as agents with full reasoning capabilities.

Another interesting result is that the use of the RIS and NAS reduces the number of events that are sent with respect to the use of the NIS; i.e., having a NM that controls the instantiation of all norms is more efficient than informing all agents about changes in the VO. Specifically, the experimental results show that the number of events sent to the NM increases linearly with the number of agents, whereas the number of events sent to agents increases exponentially with the number of agents. Obviously, if agents externalise the reasoning about norms (i.e., if they make use of the RIS and NAS instead of using the NIS), they need to request the NRSs more frequently. However, the experimental results show that the number of requests made to any service increases linearly with the number of agents and remains stable in the rest of experiments. Thus, the use of the RIS and NAS reduces the overload of the system since it reduces the number of events that are generated.

For simplicity, the NRSs have been described assuming that there is a single entity that acts as service provider. Although the number of service requests increases linearly with the number of agents, it could become a bottle-neck in ultra large-scale systems (with thousands of agents making thousands of service requests). However, it is also possible to dynamically adapt the amount of providers of the NRSs by performing cloning and self-deletion operations. Similarly, 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. Thus, the new NM is responsible for controlling the instantiation 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, then it removes itself.The definition of these elaborated procedures for adapting dynamically to changing environments [27] is a complex issue that is out the scope of this paper.

5.2 Comparison Experiment

This section contains the results that have been achieved in the experiments that compare our proposal against similar proposes. Specifically, we have compared to what extent the use of the NRSs services increases compliance with norms with respect to: (i) proposals that allow agents to be informed about the norms but that do not provide agents with information about norm dynamics (i.e., the proposals made by Felicíssimo et al. in [17] and Okuyama et al. in [28]); and (ii) works that allow agents to know information about instances (i.e., the proposal made by Piunti et al. in [31]). Note that the ORA4MAS [22] proposal does not provides agents with information that allows them to take norms into account in their actions, but it provides information about norm violations that can be used by agents to select the most suitable interaction partners.

5.2.1 Simulation Description

Again, we considered a scenario with the same parameters that we sum up in Table 2. We have also implemented agents that can be *norm-unaware*, *norm-incapable*, *norm-reasoning* and *norm-autonomous*. The main difference is that these agents can make use of three different set of services: NRSs, the Felicíssimo-Okuyama services (i.e., a set of services that implement the functionality described in the proposals made by Felicíssimo et al. [17] and Okuyama et al. [28]) and the Piunti services (i.e., a set of services that implement the functionality described in the proposal made by Piunti et al. [31]). Each agent type makes use of the different set of services as follows:

- *Norm-unaware* agents. They do not care about norms and they do not make use of any normative service.
- *Norm-incapable* agents. If they make use of the NRSs, they request the NAS in order to know the best plan to be executed. Felicíssimo-Okuyama services and the Piunti services do not provide this functionality. As a consequence, if *norm-incapable* agents make use of any one of these two set of services they cannot ask for any service and *norm-incapable* agents behave as *norm-unaware* agents by selecting randomly a plan to be executed.
- *Norm-reasoning* agents. When they make use of the NRSs, they ask the RIS to know the specific instances that affect them. Felicíssimo-Okuyama services only inform about the norms that have been registered. Thus, when *norm-reasoning* agents make use of the Felicíssimo-Okuyama services they ask for the norms that have been registered and they consider that norms are always relevant[12]. Piunti services inform about the instances that are relevant to agents. Thus, when *norm-reasoning* agents make use of the Piunti services, they are able to know the specific instances that affect them.
- *Norm-autonomous* agents. When they make use of the NRSs, they request the NIS in order to know the norms that have been registered. This functionality is also provided by the Felicíssimo-Okuyama services and the Piunti services. As a consequence, *norm-autonomous* agents are always able to know the norms that are in force.

[12] *Norm-reasoning* agents could also consider all norms as not active. However, we have not chosen this alternative since they would behave again as *norm-unaware*

As in the previous experiments, we simulate the actions performed by each agent in each step[13]. With this information we calculate the number of obligations that are fulfilled by each agent type and the number of prohibitions that are violated by each agent type. We have repeated each experiment 10000 times to support the findings with statistically significant evidence.

We have performed 3 different experiments to illustrate the performance of the each agent type with respect to the set of services that they use: the NRSs, the Felicíssimo-Okuyama services and the Piunti services. The results of these experiments are described below.

5.2.2 Results

Figure 18 illustrates the percentage of obligations that are fulfilled on average per agent type with respect to the set of services that they use. As previously mentioned, *norm-unaware* agents do not make use of any service and the percentage of fulfilled obligations is the same regardless of the set of services that is used. When *norm-incapable* agents make use of the Felicíssimo-Okuyama services or the Piunti services they behave as *norm-unaware*. Only when *norm-incapable* agents use the NRSs, they comply with a higher percentage of obligations because the NAS provides them with the goals they should pursue to comply with norms. When *norm-reasoning* agents make use of the Felicíssimo-Okuyama services they consider that norms are always relevant and they comply with less norms. This is explained by the fact that *norm-reasoning* are not able to determine which norms are relevant to them and they focus on complying with all the norms (e.g., agents may select a plan to comply with norms that are not even active). On the contrary, when they make use of the NRSs or the Piunti services, they comply with a higher number of obligations. Finally, *norm-autonomous* agents comply with a similar percentage of obligations regardless of set of services that are used. Similar results could be observed if we analyse the percentage of violated prohibitions.

5.2.3 Discussion of Results

From these results we can determine that the use of the NRS in VO that are populated by heterogeneous agents endowed with different norm-reasoning capabilities increases norm compliance with respect to existing approaches. In particular, we have demonstrated that the existence of a more complete set of services increases compliance with norms noticeably.

6 Conclusion

In this paper, we described a set of Norm Reasoning Services (NRSs) that were developed considering the facilities provided by the Magentix2 platform. The main aim of the NRSs is to provide agents with normative information that can be of

[13] Note that implementations of proposals made by Felicíssimo et al., Okuyama et al. and Piunti et al. are not available and we cannot evaluate the performance of these implementations. Thus, we have implemented a simulator that computes the number of norms that are complied by agents when they receive the normative information that is described by the algorithms specified by Felicíssimo et al., Okuyama et al. and Piunti et al.

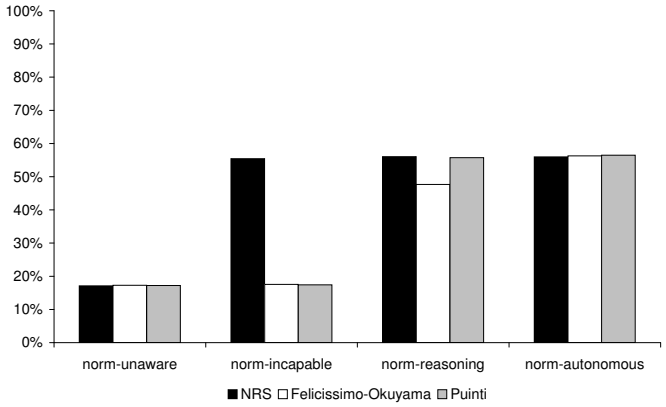


Fig. 18 Percentage of obligations that are fulfilled on average per agent type with respect to the set of services that they use.

interest to heterogeneous agents (i.e., agents that are endowed with different capabilities for reasoning about norms) that participate in norm-regulated VO. Our proposal extends previous works since the NRSs range from simple services that inform agents about norms and instances, to more elaborated services that can help agents to make decisions about norm compliance and to judge the behaviour exhibited by them or other agents. We have also demonstrated experimentally that the normative information provided by the NRSs aids agents (with different capabilities to reason about norms) to fulfil their goals while complying with norms. In particular, we have shown that if these agents use our proposed services, this allow them to equally comply with norms regardless of their specific capabilities.

As future work, we plan to improve the norm judgement service to deal with norm conflicts. Currently, the norm judgement service only considers how many norms are violated and fulfilled by an event without considering norm salience. In the future, we plan to annotate norms with their salience. The salience of norms determines the hierarchy of norms. With this information the judgement process will be able to determine not only the number of norms that are violated and fulfilled by an event, but also to determine if an event may be considered as an offence in case of norm conflict.

7 Acknowledgments

This work has also been partially funded by grants CONSOLIDER-INGENIO 2010 CSD2007-00022, TIN2009-13839-C03-01. This research has also been partially funded by Valencian Prometeo project 2008/051.

References

1. G. Andrighetto, M. Campenni, R. Conte, and F. Cecconi. Conformity in multiple contexts: imitation vs. norm recognition. In *World Congress on Social Simulation*, pages 14–17, 2008.

2. A. Artikis and J. Pitt. A formal model of open agent societies. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 192–193, 2001.
3. R. Bordini, J. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. Wiley-Interscience, 2008.
4. J. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. van der Torre. The boid architecture: conflicts between beliefs, obligations, intentions and desires. In *Proceedings of the fifth international conference on Autonomous agents*, pages 9–16. ACM, 2001.
5. L. Burdalo, A. Garcia-Fornes, V. Julian, and A. Terrasa. TRAMMAS: A tracing model for multiagent systems. *Engineering Applications of Artificial Intelligence*, 24(7):1110–1119, 2011.
6. R. Conte, G. Andrighetto, M. Campennì, and M. Paolucci. Emergent and immergent effects in complex social systems. In *Proc. of the AAAI Symposium, Social and Organizational Aspects of Intelligence*, pages 42–47, 2007.
7. R. Conte, C. Castelfranchi, and F. Dignum. Autonomous norm acceptance. *Intelligent Agents V: Agents Theories, Architectures, and Languages*, pages 99–112, 1999.
8. N. Criado, E. Argente, and V. Botti. Open Issues for Normative Multi-Agent Systems. *AI Communications*, 24(3):233–264, 2011.
9. N. Criado, E. Argente, and V. Botti. Thomas: An agent platform for supporting normative multi-agent systems. *Journal of Logic and Computation*, 2011.
10. N. Criado, E. Argente, P. Noriega, and V. Botti. Determining the Willingness to Comply With Norms (Extended Abstract). In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.
11. N. Criado, E. Argente, P. Noriega, and V. Botti. MaNEA: A Distributed Architecture for Enforcing Norms in Open MAS. *Engineering Applications of Artificial Intelligence*, 26(1):76–95, 2012.
12. N. Criado, E. Argente, P. Noriega, and V. Botti. Human-inspired model for norm compliance decision making. *Information Sciences*, 245:218–239, 2013.
13. N. Criado, V. Julián, V. Botti, and E. Argente. A norm-based organization management system. *Coordination, Organizations, Institutions and Norms in Agent Systems V*, pages 19–35, 2010.
14. R. Daft. *Organization Theory and Design*. South-Western College, 2003.
15. V. Dignum and F. Dignum. Towards an agent-based infrastructure to support virtual organisations. In *Proc. of the Working Conference on Infrastructures for Virtual Enterprises: Collaborative Business Ecosystems and Virtual Enterprises*, pages 363–370, 2002.
16. M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In *Proc. of the first international joint conference on Autonomous agents and multiagent systems (AAMAS)*, volume 3, pages 1045–1052. ACM, 2002.
17. C. Felicíssimo, C. Chopinaud, J. Briot, A. Seghrouchni, and C. Lucena. Contextualizing normative open multi-agent systems. In *Proc. of the ACM symposium on Applied computing*, pages 52–59. ACM, 2008.
18. J. Ferber, F. Michel, and J. Baez. Agre: Integrating environments with organizations. *Environments for Multi-agent Systems*, pages 48–56, 2005.
19. R. L. Fogues, J. M. Alberola, J. M. Such, A. Espinosa, and A. Garcia-Fornes. Towards Dynamic Agent Interaction Support in Open Multiagent Systems. In *Proc. of the International Conference of the Catalan Association for Artificial Intelligence (CCIA)*, volume 220, pages 89–98. IOS Press, 2010.
20. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200, 2001.
21. A. Garrido, A. Giret, V. Botti, and P. Noriega. mWater, a Case Study for Modeling Virtual Markets. In *New Perspectives on Agreement Technologies*, volume Law, Gover, pages 563–579. Springer, 2013.
22. J. Hubner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Journal of Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 2010.
23. M. Kollingbaum. *Norm-governed practical reasoning agents*. PhD thesis, University of Aberdeen, 2005.
24. F. López, M. Luck, and M. dInverno. A normative framework for agent-based systems. *Computational & Mathematical Organization Theory*, 12(2):227–250, 2006.

25. F. López y López and M. Luck. A model of normative multi-agent systems and dynamic relationships. In *Proc. of the Workshop on Regulated Agent-Based Social Systems*, volume 2934 of *LNCS*, pages 259–280. Springer, 2002.
26. S. Modgil, N. Faci, F. Meneguzzi, N. Oren, S. Miles, and M. Luck. A framework for monitoring agent-based normative systems. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 153–160, 2009.
27. T. Nakano and T. Suda. Self-organizing network services with evolutionary adaptation. *IEEE Transactions on Neural Networks*, 16(5):1269–1278, 2005.
28. F. Okuyama, R. Bordini, and A. da Rocha Costa. A distributed normative infrastructure for situated multi-agent organisations. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1501–1504, 2008.
29. A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.
30. N. Oren, S. Panagiotidi, J. Vázquez-Salceda, S. Modgil, M. Luck, and S. Miles. Towards a formalisation of electronic contracting environments. *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, pages 156–171, 2009.
31. M. Piunti, A. Ricci, O. Boissier, and J. Hubner. Embodying organisations in multi-agent work environments. In *Proc. of IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies (WI-IAT)*, volume 2, pages 511–518, 2009.
32. R. Rubino and G. Sartor. Preface. *Artificial Intelligence and Law*, 16:1–5, 2008.
33. B. T. R. Savarimuthu and S. Cranefield. A categorization of simulation works on norms. In *Normative Multi-Agent Systems*, number 09121 in Dagstuhl Seminar Proceedings, 2009.
34. J. M. Such, A. Espinosa, A. García-Fornes, and V. Botti. Partial Identities as a Foundation for Trust and Reputation. *Engineering Applications of Artificial Intelligence*, 24(7):1128–1136, 2011.