# A Distributed Architecture for Norm Management in Multi-Agent Systems

A. García-Camino[1], J. A. Rodríguez-Aguilar[1], and W. Vasconcelos[2]

[1]IIIA, Artificial Intelligence Research Institute
CSIC, Spanish Research Council
Campus UAB, 08193 Bellaterra, Spain
{andres,jar}@iiia.csic.es

[2]Dept. of Computing Science
University of Aberdeen
Aberdeen AB24 3UE, UK
wvasconcelos@acm.org

**Abstract.** Norms, that is, obligations, prohibitions and permissions, are useful abstractions to facilitate coordination in open, heterogeneous multi-agent systems. We observe a lack of distributed architectures and non-centralised computational models for norms. We propose a model, *viz.*, normative structures, to regulate the behaviour of autonomous agents taking part in simultaneous and possibly related activities within a multi-agent system. This artifact allows the propagation of normative positions (that is, the obligations, prohibitions and permissions associated to individual agents) as a consequence of agents' actions. Within a normative structure, conflicts may arise – one same action can be simultaneously forbidden and obliged/permitted. This is due to the concurrent and dynamic nature of agents' interactions in a multi-agent system. However, ensuring conflict freedom of normative structures at design time is computationally intractable, and thus real-time conflict resolution is required: our architecture support the distributed management of normative positions, including conflict detection and resolution.

## 1 Introduction

An essential characteristic of open, heterogeneous multi-agent systems (MASs) is that agents' interactions are regulated to comply with the conventions of the system. Norms, that is, obligations, prohibitions and permissions, can be used to represent such conventions and hence as a means to regulate the observable behaviour of agents [3,18]. There are many contributions on the subject of norms from sociologists, philosophers and logicians (*e.g.*, [10,18]). Recently, proposals for computational realisations of normative models have been presented. Some of them operate in a centralised manner (*e.g.* [5,9,13]) which creates bottlenecks and single points-of-failure. Others (*e.g.* [3,12]), although distributed, aim at the regulation of communication between agents without taking into account that some of the normative positions (*i.e.*, their permissions, prohibitions and obligations) generated as a result of agent interaction may also affect other agents not involved in the communication.

The class of MASs we envisage consists of multiple, simultaneous and possibly related agent interactions, or *activities*. Each agent may simultaneously participate in several activities, and may change from one activity to another.

An agent's actions within one activity may have consequences – These are captured as normative positions that define, influence or constrain the agent's future behaviour. For instance, a buyer agent who ran out of credit may be forbidden from making further offers, or a seller agent is obliged to deliver the goods after closing a deal. Within a MAS normative conflicts may arise due to the dynamic nature of the MAS and simultaneous agents' actions. A normative conflict arises, for instance, when an action is simultaneously prohibited and obliged. Such conflicts ought to be identified and resolved. This analysis of conflicts can be carried out in each activity. However, ensuring conflict-freedom on a network of agent conversations (or activities) at design time is computationally intractable as shown in [7].

We propose means to handle conflicting normative positions in open and regulated MASs in a distributed manner. In realistic settings run-time conflict detection and resolution is required. Hence, we require a tractable algorithm for conflict resolution along the lines of the one presented in [7]. The only modification required for that algorithm is that it should return a list of updates (or *normative commands*), that is, the norms to be added and removed, instead of the resulting set of norms obtained from the updates.

We need an architecture to incorporate the previously mentioned algorithm. Among other features, we require our architecture to be distributed, regulated, open, and heterogeneous. These features are included in other architectures such as AMELI [3]. However, the propagation of normative positions to several agents or to an agent not directly involved in the interaction and the resolution of normative conflicts has not yet been addressed.

We thus propose an extension of the architecture presented in [3] fulfilling these features. We extend AMELI by including a new type of agent, *viz.*, the *normative managers*, also adding interaction protocols with this new type of agent, allowing for a novel conceptual differentiation of administrative (or "internal") agents. Thus, the main contribution of the paper is a distributed architecture to regulate the behaviour of autonomous agents and manage normative aspects of a MAS, including the propagation of normative positions to different conversations and the resolution of normative conflicts.

This paper is organised as follows. In Section 2 we present a scenario to illustrate and motivate our approach. Normative structures are introduced in Section 3. Section 4 presents our distributed architecture and, in Section 5, we comment on related work. Finally, we draw conclusions and report on future work in Section 6.

## 2 Scenario

We make use of a contract scenario in which companies come together at an online marketplace to negotiate and sign contracts in order to get certain tasks done. The overall transaction procedure may be organised as five distributed activities, represented as nodes in the diagram in Figure 1. The activities involve different participants whose behaviour is coordinated through protocols.

After registering at the marketplace, clients and suppliers get together in an activity where they negotiate the terms of their contract, *i.e.* actions to be performed, prices, deadlines and other details. The client will then participate in a *payment* activity, verifying his creditworthiness and instructing his bank to transfer the correct amount of money. The supplier in the meantime will delegate to specialised employees the actions to be performed in the *work* activity. Finally, agents can leave the marketplace conforming to a predetermined *exit* protocol. The marketplace accountant participates in most of the activities as a trusted provider of auditing tools.

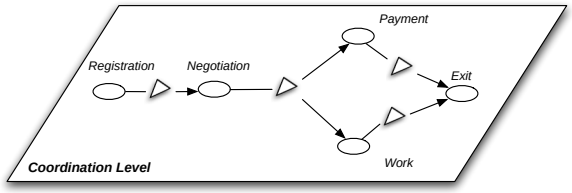


**Fig. 1:** Activity Structure of the Scenario

## 3 Normative Structure

We address a class of MASs in which interactions are carried out via illocutionary speech acts [14] exchanged among participating agents, along the lines of agent communication languages such as FIPA-ACL [6]. In these MASs, agents interact according to protocols which are naturally distributed. We observe that in some realistic scenarios, speech acts in a protocol may have an effect on other protocols. Certain actions bring about changes in the *normative positions* of agents – their "social burden": what each agent is permitted, obliged and forbidden to do. We use the term normative command to refer to the addition or removal of a normative position. Henceforth we shall refer to the application of a normative command as the addition or removal of a given normative position. Occurrences of normative positions in one protocol may also have consequences for other protocols.

We propose to extend the notion of MAS, regulated by protocols, with an extra layer called *normative structure* (NS). This layer consists of normative scenes, which represent the normative state, i.e. the set of illocutions uttered and normative positions, of the agents participating in a given activity, and normative transitions, which specifies by means of a rule the conditions under which some normative positions are to be generated or removed in the given normative scenes. The formal definition of normative structure is presented in [7], and here we informally discuss it.

Fig. 2 shows an example of how a normative structure relates with the coordination level. A normative transition is specified between the negotiation and payment activities denoting that there is a rule that may be activated with the state of negotiation activity and that may modify the state of the payment activity. In our example, the rule would be that whenever a client accepts an offer of a supplier, an obligation on the former to pay the latter is created in the payment activity. The rule connecting the payment and the work activity would

specify that whenever a client fulfils its payment obligation, an obligation on the worker to complete the contracted task is generated in the work activity.

We are concerned with the propagation and distribution of normative positions within a network of distributed, normative scenes as a consequence of agents' actions. In [7] the formal semantics of NSs was defined via a mapping to Coloured Petri Nets. Conflicts may arise after the addition of new formulae. Hence, if a new norm does not generate any conflict then it can be directly added. If a conflict arises, the algorithm presented in [11] is used to decide whether to ignore the new normative position or to remove the conflicting ones.

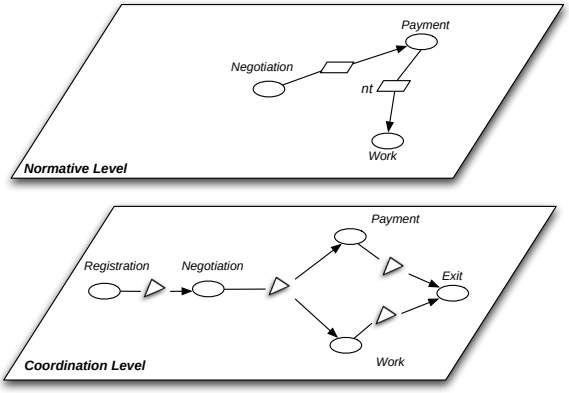


**Fig. 2:** Normative Structure and Coordination Level

## 4 Proposed Distributed Architecture

We propose an architecture to address the regulation of the behaviour of autonomous agents and the management of the normative state(s) of the MASs, including the propagation of normative positions and the resolution of normative conflicts. We assume the existence of a set of agents that interact in order to pursue their goals – we do not have control on these agents' internal functioning, nor can we anticipate it. We require the following features of our architecture:

**Regulated** The main goal of our architecture is to restrict the effects of agent behaviour in the specified conditions without hindering the autonomy of external agents.

**Open** Instead of reprogramming the MAS for each set of external agents, we advocate persistent, longer-lasting MASs where agents can join and leave them. However, agents' movements may be restricted in certain circumstances.

**Heterogeneous** We leave to each agent programmer the decision of which agent architecture include in each external agent. We make no assumption concerning how agents are implemented.

**Mediatory** As we do not control external agents internal functioning, in order to avoid undesired or unanticipated interactions, our architecture should work as a "filter" of messages between agents.

**Distributed** To provide the means for implementing large regulated MAS, we require our architecture to be distributed in a network and therefore spreading and alleviating the workload and the message traffic.

**Norm propagative** Although being distributed, agent interactions are not isolated and agent behaviour may have effects, in the form of addition or removal of normative positions, in later interactions possibly involving different agents.

**Conflict Resolutive** Some conflicts may arise due to normative positions being generated as result of agent's behaviour. Since ensuring a conflict-free MAS at design time is computationally intractable, we require that resolution of normative conflicts would be applied by the MAS. This approach promotes consistency since there is a unique, valid normative state established by the system instead of a lot of different state versions due to a conflict resolution at agent's level.

To accomplish these requirements, we extend AMELI, the architecture presented in [3]. That architecture is divided in three layers:

**Autonomous agent layer** It is formed by the set of external agents taking part in the MAS.

**Social layer** An infrastructure that mediates and facilitates agents' interactions while enforcing MAS rules.

**Communication layer** In charge of providing a reliable and orderly transport service.

External agents intending to communicate with other external agents need to redirect their messages through the social layer which is in charge of forwarding the messages (attempts of communication) to the communication layer. In specified conditions, erroneous or illicit messages may be ignored by the social layer in order to prevent them from arriving at their addressees.

The social layer presented in [3] is a multi-agent system itself and the agents belonging to it are called *internal agents*. We propose to extend this architecture by including a new type of agent , the normative manager ($NM_1$ to $NM_p$ in fig. 3), and by adding protocols to accommodate this kind of agent. We call AMELI$^+$ the resulting architecture.

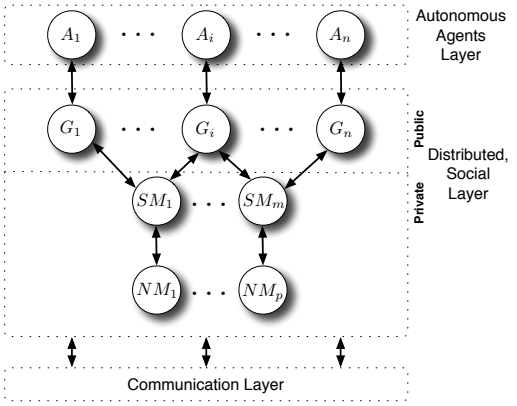


**Fig. 3:** AMELI$^+$architecture

In AMELI$^+$, internal (administrative) agents are of one of the following types:

**Governor (G)** Internal agent representing an external agent, that is, maintaining and informing about its social state, deciding or forwarding whether an attempt from its external agent is valid. One per external agent.

**Scene Manager (SM)** Internal agent maintaining the state of the activity[1], deciding whether an attempt to communicate is valid, notifying any changes to normative managers and resolving conflicts.

**Normative Manager (NM)** This new type of internal agent receives normative commands and may fire one or more normative transition rules.

In principle, only one NM is needed if it manages all the normative transition rules. However, in order to build large MAS and avoid bottlenecks, we propose the distribution of rules into several NMs.

To choose the granularity of the normative layer, i.e. to choose from one single NM to one NM per normative transition, is an important design decision that we leave for the MAS designers. After choosing the granularity, the NMs are assigned to handle a possibly unary set of normative transitions. Recall that each normative transition includes a rule. The SMs involved in the firing of the rules are given a reference to the NM that manages the rule, i.e. its address or identifier depending on the communication layer. External agents may join and leave activities, always following the conventions of the activities. In these cases, its governor registers (or deregisters) with the SM of that scene.

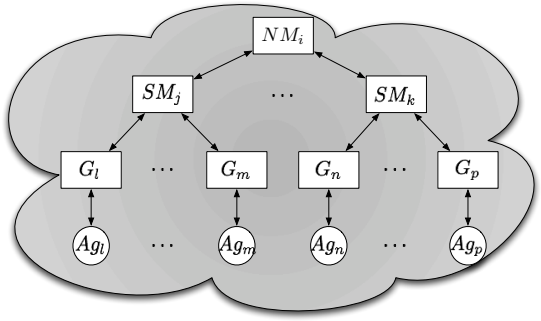


**Fig. 4:** Channels involved in the activation of a rule

### 4.1 Social Layer Protocols

Fig. 4 shows the communication within the social layer – it only occurs along the following types of channels:

**Agent / Governor** This type of channel is used by the external agents sending messages to their respective governors to request information or to request a message to be delivered to another external agent (following the norms of the MAS). Governors use this type of channel to inform their agents about new normative positions generated.

**Governor / Scene Manager** Governors use this type of channel to propagate unresolved attempts to communicate or normative commands generated as a result of such attempts. SMs use this type of channel to inform governors in their scenes about new normative commands generated as a result of attempts to communicate or conflict resolution.

[1] Hereafter, activities are also referred to as scenes following the nomenclature of AMELI.

**Scene Manager / Normative Manager** This type of channel is used by SMs to propagate normative commands that NMs may need to receive and the ones resulting from conflict resolution. NMs use this channel to send normative commands generated by the application of normative transition rules.

Fig. 5 shows an enactment of a MAS in our architecture. Agents send attempts to governors (messages 1, 4 and 7) who, after finding out the normative commands attempts generate, propagate the new normative commands to $SM_{s1}$ and $SM_{s2}$ (messages 2, 5 and 8) who, in turn, propagate them to the NM (messages 3, 6 and 9). As a normative transition rule is fired in the NM, a normative command is sent to $SM_{s3}$ (message 10). After resolving any conflicts, $SM_{s3}$ sends the new normative commands to all the involved governors (messages 11 and 11′) who, in turn, send them to their represented agents (messages 12 and 12′).

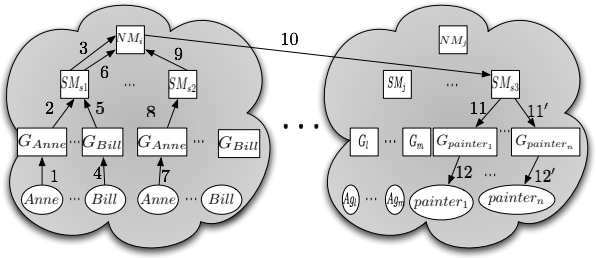


**Fig. 5:** Enactment of a normative transition rule

As the figure of the previous example shows, our architecture propagates attempts to communicate (and their effects) from agents (shown on the bottom of Fig 5) to the NMs (shown at the top of the figure). NMs receive events from several SMs whose managed state may be arbitrarily large. Since NMs only need the normative commands that may cause any of its rules to fire, NMs subscribe only to the type of normative commands they are supposed to monitor. For instance, if a rule needs to check whether there exists a prohibition to paint in a scene $work1$ and whether there exists the obligation of informing about the completion of the painting job, then the NM will subscribe to all the normative commands adding or removing prohibitions to paint in scene work1 as well as all normative commands managing obligations to inform about the completion of the painting job.

In the following algorithms, $\Delta$ refers to essential information for the execution of the MAS, i.e. a portion of the state of affairs of the MAS that each internal agent is managing. As introduced above, depending on the type of the internal agent, it manages a different portion of the state of affairs of the MAS, e.g. a governor keeps the social state of the agent, and a scene manager keeps the state of a given scene. These algorithms define the behaviour of internal agents and are applied whenever a message $msg$ is sent by an agent ($ag_i$), a governor ($g_i$), a SM ($sm_i$) or a NM ($nm_i$) respectively.

When an external agent sends to its governor an attempt to communicate (messages 1, 4 and 7 in Fig. 5), the governor follows the algorithm of Fig. 6(a). This algorithm checks whether the attempt to communicate generates normative

```
algorithm G_process_att(ag_i, msg)
input ag_i, msg
output ∅
begin
01 new_cmmds := get_norm_cmmds(msg, Δ)
02 foreach c ∈ new_cmmds do
03    Δ := apply(c, Δ)
04    sm := scene_manager(c)
05    send(c, ag_i)
06    send(c, sm)
07 endforeach
08 if new_cmmds = ∅ then
09    sm := scene_manager(msg)
10    send(msg, sm)
11 endif
end
```

(a) G response to an agent attempt

```
algorithm NM_process_cmmd(sm_i, msg)
input sm_i, msg
output ∅
begin
01 foreach cmmd ∈ msg do
02    Δ := apply(cmmd, Δ)
03    ncs := get_RHS_from_fired_rules(Δ)
04    foreach c ∈ ncs do
05       sm := scene_manager(c)
06       send(c, sm)
07    endforeach
08 foreach
end
```

(b) NM response to a command

```
algorithm SM_process_att(g_i, msg)
input g_i, msg
output ∅
begin
01 new_cmmds := get_norm_cmmds(msg, Δ)
02 foreach c ∈ new_cmmds do
03    Δ := apply(c, Δ)
04    send(c, g_i)
05    foreach ⟨nm, ev⟩ ∈ subscriptions do
06       if unify(c, ev, σ) then
07       send(c, nm)
08       endif
09    endforeach
10 endforeach
11 if new_cmmds = ∅ then
12    s := scene(msg)
13    c := content(msg)
14    send(rejected(s, c), g_i)
15 endif
end
```

(c) SM response to a forwarded attempt

```
algorithm SM_process_cmmd(nm_i, msg)
input nm_i, msg
output ∅
begin
01 Δ' := apply(msg, Δ)
02 if inconsistent(Δ') then
03    msg := resolve_conflicts(Δ, msg)
04 endif
05 foreach cmmd ∈ msg do
06    Δ := apply(cmmd, Δ)
07    foreach ⟨nm, ev⟩ ∈ subscriptions do
08       if unify(c, ev, σ) then
09          send(c, nm)
10       endif
11    endforeach
12    foreach g ∈ governors(cmmd) do
13       send(cmmd, g)
14    endforeach
15 endforeach
end
```

(d) SM response to a command

**Fig. 6.** Internal Agents Algorithms

commands (line 1), i.e. it is accepted[2]. This check may vary depending on the type of specification and implementation of the scenes: e.g. using Finite State Machines (FSM), as in [3], or executing a set of rules, as in [9].

If the attempt generates normative commands (line 2), they are applied to the portion of the state of affairs the governor is currently managing creating a new partial state (line 3). These normative commands are sent to the external agent (line 5) and to the scene manager (messages 2, 5 and 8 in Fig. 5) in charge of the scene where the normative command should be applied (line 6). Otherwise, the attempt is forwarded to the SM of the scene the attempt was generated in (line 10).

If the governor accepts the attempt (after the check of line 1), it sends the SM a notification.The SM then applies the normative command received and forwards it to the NMs subscribed to that event (messages 3, 6 and 9 in Fig. 5).

However, if the governor does not take a decision, i.e. normative commands are not generated, the governor sends the attempt to the SM who should decide whether it is valid or not by following the algorithm of Fig. 6(c). This algorithm,

[2] In our approach, an ignored attempt would not generate any normative command.

like the one in Fig. 6(a), checks whether the received attempt generates normative commands in the current scene state, i.e. the portion of the state of affairs referring to that scene (line 1). If this is the case (line 2), they are applied to the current state of the scene (line 3) and forwarded to the governor that sent the attempt (line 4) and to the NMs subscribed to that normative commands (line 7). Otherwise (line 11), a message informing that the attempt has been rejected is sent to the governor mentioned (line 14).

In both cases, if the attempt is accepted then the normative manager is notified and it follows the algorithm of Fig. 6(b) in order to decide if it is necessary to send new normative commands to other scene managers. This algorithm processes each normative command received (line 1) by applying it to the state of the NM (line 2) and checking which normative transition rules are fired and obtaining the normative commands generated (line 3). Each of them are propagated to the SM of the scene appearing in the normative command (line 6, message 10 in Fig. 5).

If normative commands are generated, SMs receive them from the normative manager in order to resolve possible conflicts and propagate them to the appropriate governors. In this case, the SMs execute the algorithm of Fig. 6(d). This algorithm applies the normative command received on the scene state creating a temporary state for conflict checking (line 1), then checks if the new normative command would raise an inconsistency (line 2). If this is the case, it applies the conflict resolution algorithm presented in [7], returning the set of normative commands needed to resolve the conflict (line 3). Each normative command caused by the message sent by the NM or by conflict resolution, is applied to the scene state (line 6) and it is sent to the subscribed NMs (lines 7-11) and to the governors (messages 11 and 11' in Fig. 5) of the agents appearing in the normative command (lines 12-14).

NMs are notified about the resolution of possible conflicts in order to check if the new normative commands fire normative transition rules. If NMs receive this notification, they follow again the algorithm of Fig. 6(b) as explained above. When governors are notified by a SM about new normative commands, they apply the normative command received to the normative state of the agent and notify to its agent about the new normative command (messages 12 and 12' in Fig. 5).

In our approach, conflict resolution is applied at the SM level requiring all normative commands generated by a NM to pass through a SM who resolves conflicts and routes them. This feature is justified because SMs are the only agents who have a full representation of a scene and know the agents are participating in it and which role they are enacting. For example, if a prohibition for all painters to paint arrives at the work activity, a SM will forward this prohibition to the governors of the agents participating in that activity with the painter role and to the governors of all the new painters that join that activity while the prohibition is active. An alternative approach is to apply conflict resolution at the level of governor agents, curtailing some of the normative positions of its associated external agent. However, this type of conflict resolution is more limited

since a governor only maintains the normative state of an agent. For example, a case that cannot be resolved with this approach is when all agents enacting a role are simultaneously prohibited and obliged to do something, i.e. when more than one agent is involved in the conflict.

Another approach would be if governors became the only managers of normative positions; in this case they would need to be aware of all normative positions that may affect its agent in the future, i.e. they would have to maintain all the normative positions affecting any of the roles that its agent may enact in every existing scene. For instance, a governor of an agent that is not yet enacting a painter role would also need to receive the normative positions that now applies to that role even if the agent is not in that scene or is enacting that role yet. This approach does not help with scalability since a large MAS with various scenes may generate a very large quantity of normative positions affecting agents in the future by the mere fact of their entering the MAS.

## 5 Related Work

The subject of norms has been studied widely in the literature (*e.g.*, [18,16,15]), and, more recently, much attention is being paid to more pragmatic and implementational aspects of norms, that is, how norms can be given a computational interpretation and how norms can be factored in the design and execution of MASs (e.g. [1,2,5,9,8]).

However, not much work has addressed the management of norms and reasoning about them in a distributed manner. Despite the fact that in [4,12] two languages are presented for the distributed enforcement of norms in MAS, in both works each agent has a local message interface that forwards legal messages according to a set of norms. Since these interfaces are local to each agent, norms can only be expressed in terms of actions of that agent. This is a serious disadvantage, *e.g.* when one needs to activate an obligation to one agent due to a certain message of another agent.

In [17] the authors propose a multi-agent architecture for policy monitoring, compliance checking and enforcement in virtual organisations (VOs). Their approach also uses a notion of hierarchical enforcement, i.e. the parent assimilates summarised event streams from multiple agents and may initiate further action on the subordinate agents. Depending on its policies, a parent can override the functioning of its children by changing their policies. Instead of considering any notion similar to our scene (multi-agent protocol where the number of participants may vary) and assigning an agent exclusively dedicated to the management of one scene, they assign another participant in the VO as parent of a set of agents. Although the parent would receive only the events it needs to monitor, it may receive them from *all* the interactions their children are engaging in. This can be a disadvantage when the number of interactions is large converting the parents in bottlenecks. Although they mention that conflict resolution may be accomplished with their architecture, they leave this feature to the VO agent thus centralising the conflict resolution in each VO. This can also be a disadvan-

tage when the number of interactions is large since the VO agent has to resolve all the possible conflicts. This would require either all the events flowing through the VO agent or the VO agent monitoring the state of the whole VO in order to detect and resolve conflicts. The main theoretical restriction in their approach is that all the agents involved in a change in a policy must share a common parent in the hierarchy of the VO. In an e-commerce example, when a buyer accepts a deal an obligation to supply the purchased item should be added to the seller. However, as they are different parties, their only common parent is the VO agent converting the latter in a bottleneck in large e-commerce scenarios.

## 6 Conclusions and Future Work

We base the architecture presented in this paper in our proposal of normative structure and conflict resolution of [7]. The notion of normative structure is useful because it allows the separation of normative and procedural concerns. We notice that the algorithm presented in that paper is also amenable to the resolution of normative conflicts in a distributed manner.

The main contribution of this paper is an architecture for the management of norms in a distributed manner. As a result of the partial enactment of protocols in diverse scenes, normative positions generated in different scenes can be used to regulate the behaviour of agents not directly involved in previous interactions. Furthermore, conflict resolution is applied at a scene level meaning that resolution criteria involving more than one agent are now possible.

We want to extend normative structures [7], as we use them in our architecture, along several directions: (1) to handle constraints as part of the norm language, in particular constraints related with the notion of time; (2) to capture in the conflict resolution algorithm different semantics relating the deontic notions by supporting different axiomations (*e.g.*, relative strength of prohibition versus obligation, default deontic notions, deontic inconsistencies, etc.).

We also intend to use analysis techniques for Coloured Petri-Nets (CPNs) in order to characterise classes of CPNs (*e.g.*, acyclic, symmetric, etc.) corresponding to families of Normative Structures that are susceptible to tractable off-line conflict detection. The combination of these techniques along with our online conflict resolution mechanisms is intended to endow MAS designers with the ability to incorporate norms into their systems in a principled way.

**Acknowledgements** – This work was partially funded by the Spanish Education and Science Ministry as part of the projects TIN2006-15662-C02-01 and 2006-5-0I-099. García-Camino enjoys an I3P grant from the Spanish National Research Council (CSIC).

## References

1. A. Artikis, L. Kamara, J. Pitt, and M. Sergot. A Protocol for Resource Sharing in Norm-Governed Ad Hoc Networks. In *Declarative Agent Languages and Technologies II*, volume 3476 (LNCS). Springer-Verlag, 2005.

2. S. Cranefield. A Rule Language for Modelling and Monitoring Social Expectations in Multi-Agent Systems. Technical Report 2005/01, University of Otago, 2005.
3. M. Esteva, B. Rosell, J. A. Rodríguez-Aguilar, and J. L. Arcos. AMELI: An agent-based middleware for electronic institutions. In *Procs of 3rd Int'l Conf on Autonomous Agents and Multiagent Systems (AAMAS'04)*, pages 236–243, 2004.
4. M. Esteva, W. Vasconcelos, C. Sierra, and J. A. Rodríguez-Aguilar. Norm consistency in electronic institutions. In *XVII Brazilian Symposium on Artificial Intelligence - SBIA'04*, volume 3171 (LNAI), pages 494–505. Springer-Verlag, 2004.
5. N. Fornara, F. Viganò, and M. Colombetti. An Event Driven Approach to Norms in Artificial Institutions. In *AAMAS05 Workshop: Agents, Norms and Institutions for Regulated Multiagent Systems (ANI@REM)*, Utrecht, 2005.
6. Foundation for Intelligent Physical Agents (FIPA). FIPA-ACL: Message Structure Specification, December 2002.
7. D. Gaertner, A. García-Camino, P. Noriega, J. A. Rodríguez-Aguilar, and W. Vasconcelos. Distributed Norm Management in Regulated Multi-agent Systems. In *Procs of 6th Int'l Conf on Autonomous Agents and Multiagent Systems (AAMAS'07)*, pages 624–631, Hawai'i, May 2007.
8. A. García-Camino, P. Noriega, and J. A. Rodríguez-Aguilar. Implementing Norms in Electronic Institutions. In *Procs of 4th Int'l Conf on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 667–673, Utrecht, July 2005.
9. A. García-Camino, J.-A. Rodríguez-Aguilar, C. Sierra, and W. Vasconcelos. A Distributed Architecture for Norm-Aware Agent Societies. In *Decl. Agent Languages and Technologies III*, volume 3904 (LNAI), pages 89–105. Springer, 2006.
10. J. Habermas. *The Theory of Communication Action, Volume One, Reason and the Rationalization of Society*. Beacon Press, 1984.
11. M. J. Kollingbaum, W. W. Vasconcelos, A. García-Camino, and T. J. Norman. Conflict resolution in norm-regulated environments via uni cation and constraints. In *Fifth International Workshop on Declarative Agent Languages and Technologies (DALT 2007)*, Hawai'i, May 2007.
12. N. Minsky. Law Governed Interaction (LGI): A Distributed Coordination and Control Mechanism (An Introduction, and a Reference Manual). Technical report, Rutgers University, 2005.
13. A. Ricci and M. Viroli. Coordination Artifacts: A Unifying Abstraction for Engineering Environment-Mediated Coordination in MAS. *Informatica*, 29:433–443, 2005.
14. J. Searle. *Speech Acts, An Essay in the Philosophy of Language*. Cambridge University Press, 1969.
15. M. Sergot. A Computational Theory of Normative Positions. *ACM Trans. Comput. Logic*, 2(4):581–622, 2001.
16. Y. Shoham and M. Tennenholtz. On Social Laws for Artificial Agent Societies: Off-line Design. *Artificial Intelligence*, 73(1-2):231–252, 1995.
17. Y. B. Udupi and M. P. Singh. Multiagent policy architecture for virtual bussiness organizations. In *Proceedings of the IEEE International Conference on Services Computing (SCC)*, September 2006.
18. G. H. von Wright. *Norm and Action: A Logical Inquiry*. Routledge and Kegan Paul, London, 1963.