A Distributed Architecture for Norm-Aware Agent Societies: A Retrospective

Andrés García-Camino[1], Juan-Antonio Rodríguez-Aguilar[2], Carles Sierra[2], and Wamberto W. Vasconcelos[3]

[1] Independent Researcher
andres@garcia-camino.es
[2] IIIA-CSIC, Campus UAB 08193 Bellaterra, Catalunya, Spain
{jar,sierra}@iiia.csic.es
[3] Dept. of Computing Science, Univ. of Aberdeen, Aberdeen AB24 3UE, UK
wvasconcelos@acm.org

Abstract. We provide a retrospective on the research leading to and following our paper "A Distributed Architecture for Norm-Aware Agent Societies" [1], presented at DALT 2005. We do so by giving the context and motivation for that research, listing its contributions, and discussing the main developments of the research and its impact.

1 Introduction

We provide a retrospective on the research reported in the paper "A Distributed Architecture for Norm-Aware Agent Societies" [1], presented at DALT 2005. That paper described a distributed architecture to endow multi-agent systems with a social layer in which normative positions are explicitly represented and managed via *institutional rules*. These rules operate on a representation of the execution states of a multi-agent system. The paper presented the syntax and semantics of institutional rules and an interpreter for them. The approach achieved greater precision and expressiveness by having constraints as part of the rules. Finally, the paper proposed means to connect rules and states in a distributed architecture, whereby a team of administrative agents employ a tuple space to guide the execution of a multi-agent system.

This retrospective is organised as follows. In Section 2 we give the context and motivation for the research. Section 3 reviews the representation of norms and institutional rules, and mechanisms for processing them. We revisit the computational infrastructure based on a shared tuple space in Section 4 and in Section 5 we report on how the research was further developed and its impact in the state-of-the-art. Finally, we draw conclusions in Section 6.

2 Context and Motivation of Research

The work reported in [1] was carried out within García-Camino's PhD research [2]. The work was influenced by research on *electronic institutions* (EIs, for short), especially Esteva's PhD thesis [3], the AMELI middleware [4] and the Electronic

C. Sakama et al. (Eds.): DALT 2011, LNAI 7169, pp. 102–110, 2012.

Institutions Development Environment (EIDE) [5]. Another influence was the Sustainable Lifecycles in Information Ecosystems (SLIE) European project [6]. These efforts helped define a software engineering perspective on electronic institutions, expanding and grounding earlier theoretical work (*e.g.*, [7,8]).

The research reported in [1] was also motivated by a gap between theoretical work on norms for multi-agent systems [9,10] and implementation/engineering concerns. More specifically, we provided clear connections between a declarative formal specification of norms with agents' behaviours, also providing a computational infrastructure to support the implementation of multi-agent systems.

3 Representation and Processing of Norms

Norms are represented in [1] as atomic formulae $obl(S, W, \bar{\mathsf{I}}), per(S, W, \bar{\mathsf{I}})$ and $prh(S, W, \bar{\mathsf{I}})$, standing for, respectively, an obligation, a permission and a prohibition to send a message $\bar{\mathsf{I}}$ in a state W of a scene S[1]. The normative position of an agent is its "social burden", that is, the obligations, permissions and prohibitions associated with the agent. We show in Fig. 1 the architecture proposed in [1] and how its components fit together. The architecture provides a *social layer* for multi-agent systems specified via electronic institutions [3]. EIs specify the kinds and order of interactions among software agents with a view to achieving global and individual goals. The diagram shows a tuple space in which *institutional states* $\Delta_0, \Delta_1, \ldots$ are stored; these states contain all norms and other information that hold in specific points of time during the EI enactment.

The normative positions of agents are updated via *institutional rules*. These are constructs of the form $\mathsf{LHS} \rightsquigarrow \mathsf{RHS}$ where LHS describes a condition of the current institutional state and RHS depicts how it should be updated, giving rise to the next institutional state. The architecture is built around a shared tuple space [11] – a kind of blackboard system that can be accessed asynchronously by different administrative agents. In our diagram our administrative agents are shown in grey: the institutional agent updates the institutional state using the institutional rules; the governor agents work as "escorts" or "chaperons" to the external, heterogeneous software agents, writing onto the tuple space the messages to be exchanged.

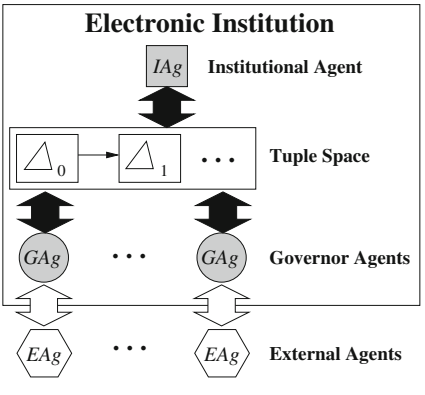

Fig. 1. Architecture Proposed in [1]

[1] States and scenes are means to break down complex EIs [3,8]. For instance, a virtual auction institution has scenes addressing agents' registration, the actual auction room, payment/delivery scenes and departure scenes. Each scene is decomposed into states, connected by expected illocutions (messages) from/to the various concerned parties.

An important contribution of [1] to research on normative systems has been the use of rules to update normative positions, adding and removing norms to/from institutional states. Constraints [12] play a special role in our approach, allowing a fine-grained and precise representation of the context in which rules should apply, and how these match with institutional states. A constraint C is of the form $\mathsf{T} \lhd \mathsf{T}'$, where T, T' are first-order terms, and $\lhd \in \{=, \neq, >, \geq, <, \leq\}$; $\Gamma = \{\mathsf{C}_1, \ldots, \mathsf{C}_n\}$ denotes a set of constraints. $\Gamma_1 \sqsubseteq \Gamma_2$ holds iff $satisfy(\Gamma_1, \Gamma_1')$ and $satisfy(\Gamma_2, \Gamma_2')$ hold and for every constraint $(\bot_1 \lhd X \lhd \top_1)$ in Γ_1', there is a constraint $(\bot_2 \lhd X \lhd \top_2)$ in Γ_2', such that $max(\bot_1, \bot_2) \geq \bot_1$ and $min(\bot_1, \bot_2) \leq \bot_1$, where $\bot_i, \top_i, i = 1, 2$ are arbitrary values. Relation $satisfy(\Gamma, \Gamma')$ holds, for two sets of constraints Γ, Γ' iff Γ can be satisfied and Γ' is the smallest set obtained from Γ such that:

- if both $(\mathsf{T} \lhd X), (X \lhd' \mathsf{T}') \in \Gamma$ then $(\mathsf{T} \lhd X \lhd' \mathsf{T}') \in \Gamma'$.
- if $(X \lhd \mathsf{T}) \in \Gamma$ then $(-\infty < X \lhd \mathsf{T}) \in \Gamma'$.
- if $(\mathsf{T} \lhd X) \in \Gamma$ then $(\mathsf{T} \lhd X < \infty) \in \Gamma'$.

Γ' contains a syntactic variation of the elements in Γ in which the constraints of each variable are *expanded* to be within an interval – two limits, $-\infty, \infty$, represent the lowest and highest value any variable may have.

We extended a conventional rule interpreter [13] to handle constraints in rules, shown in Fig. 2 as a logic program, interspersed with built-in Prolog predicates (each clause is shown with a number on its left). Clause 1 contains the top most

1. $\mathbf{s}^*(\Delta, \mathsf{Rs}, \Delta') \leftarrow$
   $\texttt{findall}(\langle \mathsf{RHS}, \Sigma \rangle, (\texttt{member}((\mathsf{LHS} \rightsquigarrow \mathsf{RHS}), \mathsf{Rs}), \mathbf{s}_l^*(\Delta, \mathsf{LHS}, \Sigma)), \mathsf{RHSs}),$
   $\mathbf{s}_r'(\Delta, \mathsf{RHSs}, \Delta')$

2. $\mathbf{s}_l^*(\Delta, \mathsf{LHS}, \Sigma) \leftarrow \texttt{findall}(\sigma, \mathbf{s}_l(\Delta, \mathsf{LHS}, \sigma), \Sigma)$
3. $\mathbf{s}_l(\Delta, (\mathsf{A} \wedge \mathsf{LHS}), \sigma_1 \cup \sigma_2) \leftarrow \mathbf{s}_l(\Delta, \mathsf{A}, \sigma_1), \mathbf{s}_l(\Delta, \mathsf{LHS}, \sigma_2)$
4. $\mathbf{s}_l(\Delta, \neg \mathsf{LHS}, \sigma) \leftarrow \neg \mathbf{s}_l(\Delta, \mathsf{LHS}, \sigma)$
5. $\mathbf{s}_l(\Delta, \mathsf{B}, \sigma) \leftarrow \texttt{member}(\mathsf{B} \cdot \sigma, \Delta), constrs(\Delta, \Gamma), satisfy(\Gamma \cdot \sigma, \Gamma')$
6. $\mathbf{s}_l(\Delta, \mathsf{C}, \sigma) \leftarrow constrs(\Delta, \Gamma), \{\mathsf{C} \cdot \sigma\} \sqsubseteq \Gamma$

7. $\mathbf{s}_r'(\Delta, \mathsf{RHSs}, \Delta') \leftarrow$
   $\texttt{findall}(\Delta'', (\texttt{member}(\langle \mathsf{RHS}, \Sigma \rangle, \mathsf{RHSs}), \texttt{member}(\sigma, \Sigma), \mathbf{s}_r(\Delta, \mathsf{RHS} \cdot \sigma, \Delta'')), All\Delta),$
   $merge(All\Delta, \Delta')$
8. $\mathbf{s}_r(\Delta, (\mathsf{U} \wedge \mathsf{RHS}), \Delta_1 \cup \Delta_2) \leftarrow \mathbf{s}_r(\Delta, \mathsf{U}, \Delta_1), \mathbf{s}_r(\Delta, \mathsf{RHS}, \Delta_2)$
9. $\mathbf{s}_r(\Delta, \oplus \mathsf{B}, \Delta \cup \{\mathsf{B}\})) \leftarrow$
10. $\mathbf{s}_r(\Delta, \ominus \mathsf{B}, \Delta \setminus \{\mathsf{B}\})) \leftarrow$
11. $\mathbf{s}_r(\Delta, \oplus \mathsf{C}, \Delta \cup \{\mathsf{C}\}) \leftarrow constrs(\Delta, C), satisfy([\mathsf{Constr}|C], C')$

Fig. 2. Interpreter for Institutional Rules (reproduced from [1])

definition: given a Δ and a set of rules Rs, it shows how we can obtain the next state Δ' by finding (via the built-in `findall` predicate[2]) all those rules in Rs

[2] ISO Prolog built-in `findall`/3 obtains all answers to a query (2nd argument), recording the values of the 1st argument as a list stored in the 3rd argument.

(picked by the `member` built-in) whose LHS holds in Δ (checked via the auxiliary definition $\mathbf{s}_l^*$). This clause then uses the RHS of those rules with their respective sets of substitutions Σ as the arguments of $\mathbf{s}_r'$ to finally obtain Δ'.

Clause 2 implements $\mathbf{s}_l^*$: it finds all the different ways (represented as individual substitutions σ) that the left-hand side LHS of a rule can be matched in an institutional state Δ – the individual σ's are stored in sets Σ of substitutions, as a result of the `findall`/3 execution. In clause 6, $constrs(\Delta, \Gamma), \Gamma \subseteq \Delta$, holds iff for every $\mathsf{C} \in \Delta$ then $\mathsf{C} \in \Gamma$.

Clause 7 shows how $\mathbf{s}_r'$ computes the new state from a list RHSs of pairs $\langle \mathsf{RHS}, \Sigma \rangle$ (obtained in the second body goal of clause 1): it picks out (via predicate `member`/2) each individual substitution $\sigma \in \Sigma$ and uses it in RHS to compute via $\mathbf{s}_r$ a partial new institutional state Δ'' which is stored in $All\Delta$. $All\Delta$ contains a set of partial new institutional states and these are combined together via the *merge*/2 predicate – it joins all the partial states, removing any replicated components. A garbage collection mechanism can be also added to the functionalities of *merge*/2 whereby constraints whose variables are not referred in Δ are discarded.

4 Computational Infrastructure

We refer back to Fig. 1: in its centre we show a tuple space [11] – this is a blackboard system with accompanying operations to manage its entries. Our agents, depicted as a rectangle (labelled *IAg*), circles (labelled *GAg*) and hexagons (labelled *EAg*) interact (directly or indirectly) with the tuple space, reading and deleting entries from it as well as writing entries onto it. We proposed means to represent institutional states with a view to maximising asynchronous aspects (*i.e.*, agents should be allowed to access the tuple space asynchronously) and minimising housekeeping (*i.e.*, not having to move information around).

The top most rectangle in Fig. 1 depicts our *institutional agent IAg*, responsible for updating the institutional state, applying $\mathbf{s}^*$. The circles below the tuple space represent the *governor agents GAg*s, responsible for following the EI "chaperoning" the *external agents EAg*s. The external agents are arbitrary heterogeneous software or human agents that actually enact an EI to ensure that they conform to the required behaviour, each external agent is provided with a governor agent with which it communicates to take part in the EI. Governor agents ensure that external agents fulfil all their social duties during the enactment of an EI. In our diagram, we show the access to the tuple space as black block arrows; communication among agents are the white block arrows.

```
 1 main:-
 2  out(current_state(0)),
 3  time_step(T),
 4  loop(T).

 5 loop(T):-
 6  sleep(T),
 7  no_one_updating,
 8  in(current_state(N)),
 9  get_state(N,Delta),
10  inst_rules(Rules),
11  s*(Delta,Rules,NewDelta),
12  write_onto_space(NewDelta),
13  NewN is N + 1,
14  out(current_state(N)),
15  loop(T).
```

Fig. 3. Institutional Agent

We show in Fig. 3 a Prolog implementation for the institutional agent *IAg*. It bootstraps the architecture by creating an initial value 0 for the current state (lines 2-3); the initial institutional state is empty. In line 3 the institutional agent obtains via `time_step`/1 a value `T`, an attribute of the EI enactment setting up the frequency new institutional states should be computed.

The *IAg* agent then enters a loop (lines 5-14) where it initially (line 6) sleeps for `T` milliseconds – this guarantees that the frequency of the updates will be respected. *IAg* then checks via `no_one_updating`/0 (line 7) that there are no governor agents currently updating the institutional state with their utterances – `no_one_updating`/0 succeeds if there are no `updating`/2 tuples in the space. Such tuples are written by the governor agents to inform the institutional agent it has to wait until their utterances are written onto the space.

When the agent *IAg* is sure there are no more governor agents updating the tuple space then it removes the `current_state`/1 tuple (line 8) thus preventing any governor agent from trying to update the tuple space (the governor agent checks in line 7 of Fig. 4 if such entry exists – if it does not, then the flow of execution is blocked on that line). The agent *IAg* then obtains via predicate `get_state`/2 all those tuples pertaining to the current institutional state `N` and stores them in `Delta`; the institutional rules are obtained in line 10 – they are also stored in the tuple space so that any of the agents can examine them. In line 11 `Delta` and `Rules` are used to obtain the next institutional state `NewDelta` via predicate $\mathbf{s}^*/2$ and its implementation in Fig 2). In line 12 the new institutional state `NewDelta` is written onto the tuple space, then the tuple recording the identification of the current state is written onto the space (line 14) for the next update. Finally, in line 15 the agent loops[3].

```
 1 main:-
 2  connect_ext_ag(Ag),
 3  root_scene(Sc),
 4  initial_state(Sc,St),
 5  loop([Ag,Sc,St,Role]).

 6 loop(Ctl):-
 7  rd(current_state(N)),
 8  Ctl = [Ag|_],
 9  out(updating(Ag,N)),
10  get_state(N,Delta),
11  findall([A,NC],(p(Ctl):-A,p(NC)),ANCs),
12  social_analysis(ANCs,Delta,Act,NewCtl),
13  perform(Act),
14  in(updating(Id,N)),
15  loop(NewCtl).
```

Fig. 4. Governor Agent

Distinct threads will execute the code for the governor agents *GAg* shown in Fig. 4. Each of them will connect to an external agent via predicate `connect_ext_ag`/1 and obtain its identification `Ag`, then find out (line 3) about the EI's root scene (where all agents must initially report to [3]) and that scene's initial state (line 4). In line 5 the governor agent makes the initial call to `loop`/1: the `Role` variable is not yet instantiated at that point, as a role is assigned to the agent when it joins the EI. The governor agents then will loop through lines 6-15, initially checking in line 7 if they are allowed to update the current institutional state, adding their utterances. Only if the `current_state`/1 tuple is on the space

[3] For simplicity we did not show the termination conditions for the loops of the institutional and governor agents. These conditions are prescribed by the EI specification and should appear as a clause preceding the loop clauses of Figs. 3 and 4.

then does the flow of execution of the governor agent move to line 8, where it obtains the identifier `Ag` from the control list `Ctl`; in line 9 a tuple `updating`/2 is written out onto the space. This tuple informs the institutional agent that there are governors updating the space and hence it should wait to update the institutional state. In line 10 the governor agent reads all those tuples pertaining to the current institutional state. In line 11 the governor agent collects all those actions `send`/1 and `receive`/1 in the EI specification which are associated with its current control `[Ag,Sc,St,Role]`. In line 12, the governor agent interacts with the external agent and, taking into account all constraints associated with `Ag`, obtains an action `Act` that is performed in line 14 (*i.e.*, a message is sent or received). In line 14 the agent removes the `updating`/2 tuple and in line 15 the agent starts another loop.

We were able to claim that the resulting society of agents is endowed with norm-awareness because their behaviour is regulated by the governor agents depicted above. The social awareness of the governor agent, in its turn, stems from two features: *i)* its access to the institutional state where obligations, prohibitions and permissions are recorded (as well as constraints on the values of their variables); *ii)* its access to the set of possible actions prescribed in the protocol. With this information, we can define various alternative ways in which governor agents, in collaboration with their respective external agents, can decide on which action to carry out.

We show in Fig. 5 a definition for predicate `social_analysis`/4. Its first subgoal removes from the list `ANCs` all those utterances that are prohibited from being sent, obtaining the list `ANCsWOPrhs`. The second subgoal ensures that obligations are given adequate priority: the list `ANCsWOPrhs` is further refined to get the obligations among the actions and store them in the list `ANCsObls` – if there are no obligations, then `ANCsWOPrhs` is the same as `ANCsObls`. Finally, in the third subgoal, an action is chosen from `ANCsObls` and customised in collaboration with the external agent.

```
social_analysis(ANCs,Delta,Act,NewCtr):-
 remove_prhs(ANCs,Delta,ANCsWOPrhs),
 select_obls(ANCsWOPrhs,Delta,ANCsObls),
 choose_customise(ANCsObls,Delta,Act,NewCtr).
```

Fig. 5. Definition of Social Analysis

5 Developments and Impact

The research reported in [1] was further developed in many ways. We refined, extended and related the rule-based approach to normative-oriented programming desiderata in [14], also exploring the approach in an auction scenario. In [15] we consolidated our approach, offering the pragmatics of our rule-based language, and how to represent and enact protocols; in that paper we also carry out an expressiveness analysis of the language, comparing it with other similar approaches. A shorter version of [1] appears as [16].

Our rule-based approach to norm-oriented programming and the tuple-space centered architecture allowed us to extend AMELI [4]: in [17] we presented

AMELI$^+$, a layered and distributed architecture with a team of administrative agents responsible for the "housekeeping" of rules and propagation of norms among various concurrent scenes. In [18] we developed an algorithm to deal with normative conflicts in a distributed setting.The use of constraints to increase precision and expressiveness of a formalism influenced the work reported in [19,20]. However, in this research constraints were used in the norm themselves (and not in rules adding/removing norms). The approach to use rules in order to give norms an operational semantics was also adopted in [21] and, subsequently, in [22].

The seminal ideas of [1] provided us with a vantage point from which various issues could be conveniently explored. The impact of this research was very noticeable, leading to distributed and highly scaleable architectures for multi-agent systems, also providing an explicit account of normative aspects and mechanisms for their management. Alternative formulations of our distributed architecture were proposed in [23] and in [24]; the concept of administrative agents stemmed from early work on EIs [3,7], but in [1] these were presented in a compact and self-contained fashion, being explicitly related with the information model and representation of norms.

6 Conclusions

This is a retrospective on the work reported [1], listing its context, the main ideas and contributions, and assessing its developments, influence and impact. The proposal of a formalism for norm representation which was expressive yet of practical use, and coupled with conceptual/architectural concerns, paved the way to the development of alternative formalisms and associated mechanisms and architecures.

References

1. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.W.: A Distributed Architecture for Norm-Aware Agent Societies. In: Baldoni, M., Endriss, U., Omicini, A., Torroni, P. (eds.) DALT 2005. LNCS (LNAI), vol. 3904, pp. 89–105. Springer, Heidelberg (2006)
2. García-Camino, A.: Normative Regulation of Open Multi-Agent Systems. PhD thesis, Universitat Autònoma de Barcelona, Spain (2009); IIIA monography, Vol. 35
3. Esteva, M.: Electronic Institutions: From Specification to Development. PhD thesis, Universitat Politècnica de Catalunya, Spain (2003); IIIA monography, Vol. 19
4. Esteva, M., Rosell, B., Rodríguez-Aguilar, J.A., Arcos, J.L.: AMELI: An agent-based middleware for electronic institutions. In: Jennings, N., et al. (eds.) Procs. 3rd Int'l Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS 2004), pp. 236–243. ACM (2004)
5. Esteva, M., Rodriguez-Aguilar, J.A., Arcos, J.L., Sierra, C., Noriega, P., Rosell, B., de la Cruz, D.: Electronic institutions development environment. In: Procs. 7th Int'l Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS 2008), pp. 1657–1658. IFAAMAS, Richland (2008)

6. Vasconcelos, W.W., Robertson, D., Agustí, J., Sierra, C., Wooldridge, M.J., Parsons, S., Walton, C.D., Sabater, J.: A Lifecycle for Models of Large Multi-agent Systems. In: Wooldridge, M.J., Weiß, G., Ciancarini, P. (eds.) AOSE 2001. LNCS, vol. 2222, pp. 297–318. Springer, Heidelberg (2002)
7. Esteva, M., Rodríguez-Aguilar, J.-A., Sierra, C., Garcia, P., Arcos, J.-L.: On the Formal Specification of Electronic Institutions. In: Sierra, C., Dignum, F.P.M. (eds.) AgentLink 2000. LNCS (LNAI), vol. 1991, pp. 126–147. Springer, Heidelberg (2001)
8. Vasconcelos, W.: Logic-Based Electronic Institutions. In: Leite, J., Omicini, A., Sterling, L., Torroni, P. (eds.) DALT 2003. LNCS (LNAI), vol. 2990, pp. 221–242. Springer, Heidelberg (2004)
9. Dignum, F.: Autonomous agents with norms. Art. Intell. & Law 7, 69–79 (1999)
10. Verhagen, H.: Norm Autonomous Agents. PhD thesis, Stockholm University (2000)
11. Carriero, N., Gelernter, D.: Linda in context. Comm. of the ACM 32 (1989)
12. Jaffar, J., Maher, M.J., Marriott, K., Stuckey, P.J.: The semantics of constraint logic programs. Journal of Logic Programming 37, 1–46 (1998)
13. Vianu, V.: Rule-based languages. Annals of Mathematics and Artificial Intelligence 19, 215–259 (1997)
14. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: A rule-based approach to norm-oriented programming of electronic institutions. ACM SIGecom Exchanges 5, 33–40 (2006)
15. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Constraint rule-based programming of norms for electronic institutions. Autonomus Agents and Multi-Agent Systems 18, 186–217 (2009)
16. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Norm-oriented programming of electronic institutions. In: 5th Int'l Joint Conf. on Autonomous Agents and Multiagent Systems, AAMAS 2006 (2006)
17. García-Camino, A., Rodríguez-Aguilar, J.-A., Vasconcelos, W.W.: A Distributed Architecture for Norm Management in Multi-Agent Systems. In: Sichman, J.S., Padget, J., Ossowski, S., Noriega, P. (eds.) COIN 2007. LNCS (LNAI), vol. 4870, pp. 275–286. Springer, Heidelberg (2008)
18. Gaertner, D., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A., Vasconcelos, W.: Distributed norm management in regulated multi-agent systems. In: 6th Int'l Joint Conf. on Autonomous Agents and Multiagent Systems, AAMAS 2007 (2007)
19. Kollingbaum, M.J., Vasconcelos, W.W., García-Camino, A., Norman, T.J.: Conflict Resolution in Norm-Regulated Environments Via Unification and Constraints. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2007. LNCS (LNAI), vol. 4897, pp. 158–174. Springer, Heidelberg (2008)
20. Kollingbaum, M.J., Vasconcelos, W.W., García-Camino, A., Norman, T.J.: Managing Conflict Resolution in Norm-Regulated Environments. In: Artikis, A., O'Hare, G.M.P., Stathis, K., Vouros, G.A. (eds.) ESAW 2007. LNCS (LNAI), vol. 4995, pp. 55–71. Springer, Heidelberg (2008)
21. Aldewereld, H., Dignum, F., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A., Sierra, C.: Operationalisation of norms for usage in electronic institutions. In: Procs. 5th Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2006), pp. 223–225. ACM, New York (2006)

22. Aldewereld, H., Álvarez Napagao, S., Dignum, F., Vázquez-Salceda, J.: Making norms concrete. In: Procs. 9th Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010), pp. 807–814. IFAAMAS, Richland (2010)
23. Okuyama, F., Bordini, R., da Rocha Costa, A.: A Distributed Normative Infrastructure for Situated Multi-agent Organisations. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2008. LNCS (LNAI), vol. 5397, pp. 29–46. Springer, Heidelberg (2009)
24. Felicíssimo, C.H., de Lucena, C.J.P., Briot, J.P.: A norm-based approach for the modeling of open multiagent systems. In: Procs. Int'l Conf. on Agents & Artificial Intelligence (ICAART 2009), pp. 540–546 (2009)