Laws for Mediating Agents' Activities in Situated Multiagent Systems

Danny Weyns and Tom Holvoet
Katholieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium
{Danny.Weyns, Tom.Holvoet}@cs.kuleuven.be

Abstract

Research on situated multiagent systems (situated MAS) investigates decentralized architectures for software systems that have to deal with highly dynamic operating conditions. To realize the system requirements, the agents of a situated MAS have to coordinate their behavior. The agent environment provides a means to mediate (i.e., enable and constrain) agents' activities in the system. Laws embedded in the agent environment allow to define application specific constrains on agents activities. In this paper, we declaratively specify the semantics of laws for perception, action, and communication in situated MAS. We illustrate the laws with concrete examples in an automated transportation system that we have developed. Mediation of agents' activities via the agent environment improves separation of concerns in MAS and helps to manage complexity, especially in open and pervasive environments.

1. Introduction

Research on multiagent systems (MAS) is concerned with the study, behavior, and construction of a collection of autonomous agents that interact with each other and their environment [14]. MAS architectures are assigned quality attributes such as flexibility, openness, and robustness. Since the late 1980s, the major trend in MAS research spawns from the study of BDI-agents [11]. The basic philosophy of BDI-agents is to reflect the practical reasoning of humans, who act in order to achieve their intentions. Typical BDI approaches are knowledge-oriented, and direct communication is employed for knowledge sharing and agent coordination. Research in the area of situated MAS investigates interaction-oriented agent systems [6, 4, 2, 17, 8]. In situated MAS, the system requirements are realized through interaction between cooperative agents rather than as the result of advanced cognitive capabilities of individual agents.

Originating from research on situated MAS, the *agent environment* has recently been put forward as an explicit design abstraction in MAS [18]. The agent environment enables to shield the complexity of the underlying deployment context to agents, and it provides a means to mediate the agents' activities in the system. Laws embedded in the agent environment allow to define application specific constrains on agents' activities. Mediation of agents' activities via the agent environment improves separation of concerns in MAS and helps to manage complexity, especially in open and pervasive environments. The contribution of this paper is a declarative specification of the semantics of laws for perception, action, and perception in situated MAS. Such a specification is required to support the disciplined engineering of situated MAS.

The remainder of the paper is structured as follows. Section 2 gives a high-level overview of a model for the agent environment that we have developed. In section 3, we specify laws for perception, action, and communication, and we illustrate the laws with concrete examples in an automated transportation system that used automatic guided vehicles (AGV). Section 4 discusses a number of related approaches. Finally, we draw conclusions in section 5.

2. Agent environment

Over the last five years, we have developed various situated MAS applications, ranging from a prototypical peer-to-peer file sharing system up to an industrial automatic transportation. In the course of building these applications, we have developed an advanced model for situated MAS. In this section, we give a high-level description of the agent environment in this model and we briefly explain how we have applied the agent environment in an AGV transportation system.

2.1. Model for agent environment

Fig. 1 shows the model of the agent environment as a set of interacting components that share a data repository. The agent environment provides functionality to *agents* on top of the *deployment context*. The deployment context consists of the given hardware and software and external resources such as sensors and actuators, a printer, a network, a database, a web service, etc. For a distributed application, the deployment context consists of multiple processors deployed on different nodes that are connected through a network. Each node provides an agent environment to the agents located at that node. We briefly discuss the responsibilities of each of the

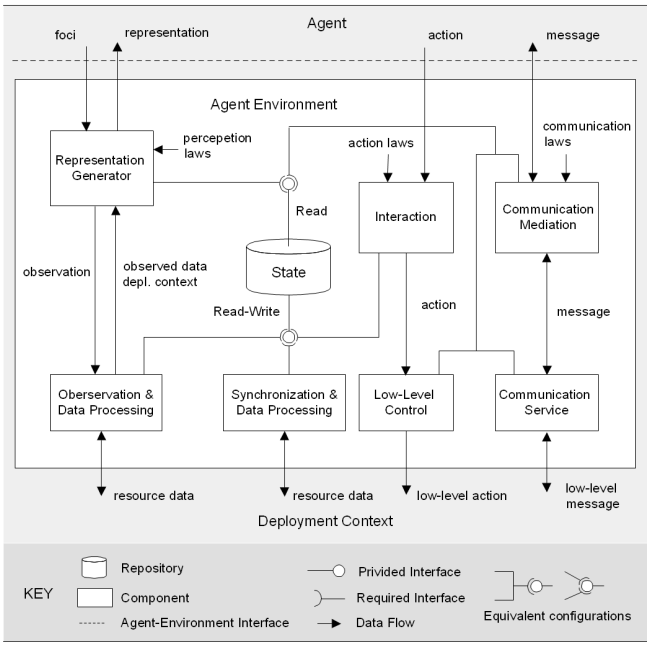

Figure 1. Model of the agent environment

elements of the agent environment in turn.

The `State` repository contains data that is shared between the components of the agent environment. Data typically includes an abstraction of the deployment context together with additional state related to the agent environment. Examples of state related to the deployment context are a representation of the local topology of a network, and data derived from a set of sensors. Examples of additional state are the representation of digital pheromones that are deployed on top of a network, and virtual marks situated on a map of the physical environment.

The `Representation Generator` provides the functionality to agents for perceiving the environment. Agents use foci to sense the environment [21]. Foci allow an agent to sense the environment (agent environment and deployment context) only for specific types of information. The representation generator uses the current state of the agent environment and possibly state collected from the deployment context to produce a representation for the agent. Agents' perception is subject to perception laws that provide the means to constrain perception. We elaborate on perception laws in the next section.

`Observation & Data Processing` provides the functionality to observe the deployment context and collect date from other nodes in a distributed setting. The observation & data processing component translates observation requests into observation primitives that can be used to collect the requested data from the deployment context. Data may be collected from external resources in the deployment context or from the agent environment instances on other nodes in a distributed application. The observation & data processing component can provide additional functions to pre-process data, examples are sorting and integration of observed data.

`Interaction` is responsible to deal with agents' actions in the environment. Agents' actions can be divided in two classes: actions that attempt to modify state of the agent environment and actions that attempt to modify the state of resources of the deployment context. An example of the former is an agent that drops a digital pheromone in the agent environment. An example of the latter is an agent that writes data to an external data base. Agents' actions are subject to action laws [17]. Action laws put restrictions on the actions invoked by the agents, representing domain specific constraints on agents' actions. We elaborate on action laws in the next section. For actions related to the agent environment, the interaction component calculates the reaction, resulting in an update of the state of the agent environment. Actions related to the deployment context are passed to the Low-Level Control component.

`Low-Level Control` converts the actions invoked by the agents into low-level action primitives in the deployment context. This decouples the interaction component from the details of the deployment context.

`Communication Mediation` mediates the communicative interactions among agents [20]. It is responsible for collecting messages, it provides the necessary infrastructure to buffer messages, and it delivers messages to the appropriate agents. Communication mediation regulates the exchange of messages between agents by imposing application specific constraints on communicative interactions. We elaborate on message laws below. To actually transmit the messages, communication mediation makes use of the Communication Service component.

`Communication Service` provides that actual infrastructure to transmit messages. Communication service transfers high-level message descriptions used by agents to communication primitives of the deployment context and vice versa. Depending on the particular application requirements, the communication service may provide specific functionality to enable the exchange of messages in a distributed setting, such as white and yellow page services. An example infrastructure for distributed communication is Jade [3].

`Synchronization & Data Processing` synchronizes state of the agent environment with state of resources in the deployment context as well as state of the agent environment on different nodes. State updates may relate to dynamics in the deployment context and dynamics of state in the agent environment that happens independently of agents or the deployment context. An example of the former is the topology of a dynamic network which changes are reflected in a network abstraction maintained in the state of the agent environment. An example of the latter is the evaporation of digital pheromones. Middleware can provide support to spread and collect data in a distributed setting. Examples

for mobile network environments are discussed in [7, 12]. Synchronization & data processing converts the resource data observed from the deployment context into a format that can be used to update the state of the agent environment. Such conversion typically includes a processing of collected resource data.

2.2. Agent environment in AGV system

An AGV transportation system is a fully automated transport system that uses multiple AGVs to transport loads in an industrial environment. The task of the AGVs is to transport loads from one location to another. The task stream is typical irregular and unpredictable. In a joint project with Egemin, we have applied a situated MAS to develop a decentralized AGV control system aiming to improve the flexibility in the system [19]. In the AGV application two types of agents are used: AGV agents and transport agents. Each AGV is controlled by an AGV agent. Transports are represented by transport agents that reside at a stationary computer in the warehouse. Since the physical environment of AGVs is very restricted, it offers little opportunities for agents to use the environment for coordination. Therefore, we introduced an agent environment in which agents are situated. The agent environment offers a medium for agents to share information and coordinate their behavior. An instance of the agent environment is deployed on each node in the system (mobile AGVs and the computer with transport agents). We developed the ObjectPlaces middleware [13] to support communication among nodes and synchronization of the state of neighboring agent environments.

3. Laws for mediating activities

In this section, we declaratively specify the semantics of laws for perception, action, and communication. For the specification we use a simple formal language based on set theory and illustrate the laws with examples from the AGV transportation system. Before we start with specifying the laws, we first introduce a number of general definitions that are further used in the specification.

3.1. General definitions

$\underline{Ag}$ $Ag = \{a_1, .., a_i, .., a_n\}$ is the set of agents;
the index $i \in \{1, .., n\}$ is a unique identifier for agent $a_i \in Ag$; $Y = \{1, .., n\}$ denotes the set of agent identifiers in the system

$\underline{Ont}$ the ontology of the application domain;
Ont defines the terminology of the domain and is specified as a tuple $\langle\, Voc, Rel \,\rangle$ with:
1. Voc: the vocabulary of domain concepts
2. Rel: the set of relationships between concepts of Voc
Voc and Rel are application specific and not further specified

$\underline{S_E^{Ont}}$ the set of state elements of the agent environment; the set of state elements is based on ontology Ont; a state element represents a part of the state of the agent environment or the deployment context and is specified as $\langle\, sname, sfields \,\rangle$ with $sname$ the name of the state element, and $sfields$ a set of fields, each field consisting of a name and a value of an accompanying domain; $sname$ and $sfields$ are application specific and are not further specified; for brevity we use S_E hereafter

$\underline{\Sigma \subseteq 2^{S_E}}$ the powerset of state elements;
we denote the actual set of state elements $\sigma \in \Sigma$ as the *current state* of the agent environment

$\underline{dom(f)}$ the domain of a function f;
for a function $f : D \rightarrow \{v1, .., vn\}$ we use $dom(f)$ to denote the domain of f, thus $dom(f) = D$; we use $dom(f)^{\rightarrow vi}$ to denote the subdomain of elements of $dom(f)$ that map to vi, with $vi \in \{v1, .., vn\}$

3.2. Perception laws

We start with the specification of perception laws. Then we give a practical example of a perception law.

$\underline{Fo}$ the set of foci for agents;
a focus $fo \in Fo$ allows agents to sense the environment selectively and is specified as a 3-tuple $\langle\, i, foname, foparam \,\rangle$ with $i \in Y$ the identity of the agent, $foname$ a name that refers to the type of information the agent aims to observe, and $foparam$ a set of additional scoping variables of the focus; $foname$ and $foparam$ are application specific and not further specified

$\underline{\Theta \subseteq 2^{Fo}}$ the powerset of foci in the system;
$\theta \in \Theta$ is a set of foci of a perception request invoked by an agent

$\underline{Sc}$ the set of perception scopes (or scopes for short);
a scope $sc \in Sc$ is typed as $sc : S_E \rightarrow Bool$; i.e. a scope maps state elements of the environment on booleans; $sc(si)$ returns $true$ for the elements $si \in S_E$ that are within the scope of sc, and $sc(so) = false$ for the elements $so \in S_E$ outside the scope; we call the set of state elements that map on true as the *domain of interest* of a scope, i.e. $dom(sc)^{\rightarrow true}$

$\underline{\mathcal{N} \subseteq 2^{Sc}}$ the powerset of scopes in the system;
$\eta \in \mathcal{N}$ is a set of scopes derived from a set of foci of a perception request; the conversion function $Scoping$ is typed as: $Scoping : \Theta \rightarrow \mathcal{N}$; the domain of interest of a set of scopes η is defined as: $domint(\eta) = \{s \in S_E \mid s \in dom(sc)^{\rightarrow true}$ forall $sc \in \eta\}$

$\underline{Co_P}$ the set of perception constraints in the system;
a perception constraint $cop \in Co_P$ is typed as $cop : S_E \rightarrow Bool$; i.e. a perception constraint maps state elements $s \in S_E$ on booleans restricting agents' perception of the environment; $cop(s)$ returns $true$ for the state elements $s \in S_E$ that are restricted for perception, and $false$ for unconstrained elements

$\underline{L_P}$ the set of perception laws;

a perception law $lp \in L_P$ is typed as $lp : Sc \times \Sigma \rightarrow Co_P$; i.e., a perception law $lp(sc, \sigma) = cop$ takes a scope sc together with the current state of the agent environment σ and produces a perception constraint cop

The application of the perception laws is defined by the $ApplyL_P$ function that is typed as follows:

$$ApplyL_P : \mathcal{N} \times \Sigma \rightarrow \mathcal{N}$$

$$ApplyL_P(\eta, \sigma) = \eta'$$

$ApplyL_P$ applies the set of perception laws L_p to a set of scopes η, given the current state of the agent environment σ. $ApplyL_P$ results in a restricted perception scope η'. For η' holds:

$$\begin{array}{l} \forall s \in S_E, \sigma \in \Sigma : \\ \quad s \in domint(\eta') \text{ iff } (s \in domint(\eta)) \wedge \\ \qquad (\forall\, lp \in L_P, \forall\, sc \in \eta' : \\ \qquad\ (\forall\, co \in lp\,(sc, \sigma) : co(s) = false)) \\ \quad s \notin domint(\eta') \text{ otherwise} \end{array}$$

That is, a state element is within the restricted perception scope if (i) the state element is within the domain of interest of the original set of scopes, and (ii) none of the constraints of the applied perception laws is applicable to the state element. For the domain of interest of η' holds:

$$domint(\eta') = (\bigcup_{sc \in \eta} dom(sc)^{\rightarrow true}) \bigcap (\bigcup_{lp \in L_P,\ sc \in \eta,\ co \in lp(sc,\sigma)} dom(co)^{\rightarrow false})$$

The observable domain of the perception scope (i.e. the subdomain of elements of observable state of the agen environment that map to true) consists of the intersection of the domain of interest of the scopes of the perception request and the subdomain of elements of the state of the agent environment that are not constrained by the perception laws.

Example. AGV agents in the AGV transportation system avoid collisions by coordinating with other agents through the agent environment. AGV agents mark the path they are going to drive in their environment using *hulls*. The hull of an AGV is the physical area the AGV occupies. A series of hulls then describes the physical area an AGV occupies along a certain path. An example of a perception law in the AGV transportation system is a law that determines the AGVs in collision range. To guarantee safety, the subset of AGVs with which a requesting AGV might collide must be included. Yet, the amount of information that needs to be communicated among AGVs must be reasonably small. Fig. 2 illustrates how the safe subset of AGVs is determined. When an AGV agent requests the AGVs in collision range, the agent environment selects the AGVs whose *hull projection circle* overlaps with the hull projection circle of the requesting AGV. The radius of the hull projection circle is equal to the distance between the AGV and the furthest point on its hull projection. The set of AGVs with overlapping circles provides a first approximation of the vehicles that are in collision range. The constraint to select the set of AGVs in

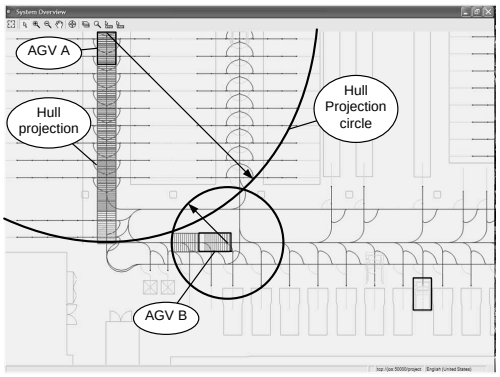

Figure 2. Determining AGVs in collision range

collision range of an AGV_{req} is defined as:

$$\begin{array}{l} in\text{-}col\text{-}range_{req} = \{AGV_{other} \in AGV \mid \\ \quad dist(AGV_{req}.pos, AGV_{other}.pos) \leq \\ \quad AGV_{req}.hullrad + AGV_{other}.hullrad\,\} \end{array}$$

pos denotes the current (x, y) position of an AGV and $hullrad$ its current hull radius.

3.3. Action laws

We start with the specification of action laws. Then we give a practical example.

Act the set of actions in the system;
an action $act \in Act$ is defined as a 3-tuple $\langle i, aname, aparam \rangle$ with $i \in Y$ the identity of the agent, $aname$ a name that refers to the type of action the agent invokes and $aparam$ is a set of additional parameters of the action; $aname$ and $aparam$ are application specific and not further specified

G the set of operations in the system;
an operation $g \in G$ is typed as: $g : S_E \rightarrow \{true, false, unspec\}$; $g(st) = true$ denotes that $st \in S_E$ is part of the target state of the operation, $g(sf) = false$ denotes that $sf \in S_E$ is *not* part of the target state, and finally $g(su) = unspec$ denotes that $su \in S_E$ is invariable to the operation; the function $OperationGeneration$ converts the selected action into an operation and is typed as follows: $OperationGeneration : Inf \rightarrow G$

Co_A the set of operation constraints in the system;
an operation constraint $coa \in Co_A$ is typed as $coa : S_E \rightarrow Bool$; i.e. an operation constraint maps state elements $s \in S_E$ on booleans restricting agents' actions in the environment; $coa(s)$ returns $true$ for the state elements that are constrained for modification, and $false$ for unconstrained state elements

L_A the set of action laws in the system;
an action law $la \in L_A$ is typed as: $la : G \times \Sigma \rightarrow Co_A$; an action law $la(g, \sigma) = coa$ takes an operation g and the current state of the agent environment σ and returns an operation

constraint coa

The application of the action laws is defined by the $ApplyL_A$ function that is typed as follows:

$$ApplyL_A : G \times \Sigma \rightarrow G$$
$$ApplyL_A(g, \sigma) = g'$$

$ApplyL_A$ applies the set of action laws L_A to operation g, given the current state of the agent environment σ. $ApplyL_A$ restricts the operation g according to the set of applicable action laws. For the restricted g' holds:

$\forall s \in S_E$:
$g'(s) = true$ iff $(g(s) = true) \wedge$
$(\forall\, la \in L_A : (\forall\, co \in la(g, \sigma) : co(s) = false))$
$g'(s) = false$ iff $(g(s) = false) \wedge$
$(\forall\, la \in L_A : (\forall\, co \in la(g, \sigma) : co(s) = false))$
$g'(s) = unspec$ otherwise

That is, (1) a state element of the agent environment is part of the target state of the restricted operation if the state element is part of the target state of the operation and none of the constraints of the applied action laws is applicable to the state element; (2) the restricted operation is invariable to the rest of the state elements. For the restricted operation holds:

$dom(g')^{\rightarrow true} =$
$(dom(g)^{\rightarrow true}) \bigcap (\bigcup_{la \in L_A,\, co \in la(g,\sigma)} dom(co)^{\rightarrow false})$
$dom(g')^{\rightarrow false} =$
$(dom(g)^{\rightarrow false}) \bigcap (\bigcup_{la \in L_A,\, co \in la(g,\sigma)} dom(co)^{\rightarrow false})$
$dom(g')^{\rightarrow unspec} =$
$(dom(g)^{\rightarrow unspec}) \bigcup (\bigcup_{la \in L_A,\, co \in la(g,\sigma)} dom(co)^{\rightarrow true})$

The target domain of the restricted operation (i.e. the subdomains of state elements that map to true or false) consists of the intersection of the target domain of the original operation and the subdomain of state elements of the agent environment that are not constrained by the action laws.

Example. When an AGV agent projects a hull in the agent environment, an action law determines how the agent can proceed. If the area is not marked by other hulls (the AGV's own hulls do not intersect with others), the AGV can move along and actually drive over the reserved path. In case of a conflict, the involved agent environments use the priorities of the transported loads and the vehicles to determine which AGV can move on. Afterward, the AGV removes the markings in the agent environment. The constraint that determines the safe condition of a hull projected by AGV_{req} is defined as:

$AGV_{req}.hullstate = safe$ iff $in\text{-}col\text{-}range_{req} = \emptyset \vee$
$\forall AGV_{other} \in in\text{-}col\text{-}range_{req}$:
$AGV_{other}.hullstate\,! = safe$
$\wedge\, prior(AGV_{req}.hullprio, AGV_{other}.hullprio)$

$hullstate$ denotes the state of a hull projection (safe or unsafe); $prior(AGV_x.hullprio, AGV_y.hullprio) = true$ if the hull priority of AGV_x is higher than that of AGV_y and $false$ otherwise.

3.4. Communication laws

First, we specify communication laws. Then we give a concrete example in the AGV application.

M the set of messages in the system;
a message $m_{i \rightarrow des} \in M$ is a formatted structure of characters that represents a message sent by the agent with identity $i \in Y$ to a set of agents with identities specified in $des \in 2^Y$

Co_C the set of communication constraints in the system;
a communication constraint $coc \in Co_C$ is typed as $coc : Y \rightarrow Bool$, i.e. coc maps identities of agents $i \in Y$ on booleans; $coc(i)$ returns $true$ for identities of agents that are excluded for a particular message, and $false$ for non constrained identities

L_C the set of communication laws in the system;
a communication law $lc \in L_C$ is typed as $lc : M \times \Sigma \rightarrow Co_C$; a communication law $lc(m_{i \rightarrow des}, \sigma) = coc$ takes a message $m_{i \rightarrow des}$ and the current state of the agent environment σ and returns a communication constraint coc

The application of the communication laws is defined by the $ApplyL_C$ function that is typed as follows:

$$ApplyL_C : M \times \Sigma \rightarrow M$$
$$ApplyL_C(m_{i \rightarrow des}, \sigma) = m_{i \rightarrow des'}$$

$ApplyL_C$ applies the set of communication laws L_C to message $m_{i \rightarrow des}$, given the current state of the agent environment σ. For the resulting message $m_{i \rightarrow des'}$ holds:

$\forall\, y \in des'$:
$(y \in des) \wedge$
$(\forall\, lc \in L_C : (\forall\, co \in lc(m_{i \rightarrow des}, \sigma) : co(y) = false))$

That is, the message $m_{i \rightarrow des'}$ will be transmitted to all addressees of the original message that are not constrained by any of the communication laws.

Example. An example of a communication law in the AGV transportation system is a law that determines nearby agents in the environment to interact with. Such a law is for example applied when a transport agent is searching for a suitable AGV to execute its tasks and vice versa. To enable adaptive task assignment, we have developed a dynamic contract net protocol that allows both types of agents to revise task assignment when opportunities occur while AGVs drive toward loads [16]. Since it is not scalable to interact with all agents in the system, the agent environment will restrict the communication to nearby candidates. If an agent does not find a candidate, it may increase its scope of interaction. The constraint that restricts the set of candidate AGVs for task assignment for a task agent TA_t is defined as:

$$candid_t = \{AGV_i \in AGV \mid dist(AGV_i.pos, TA_t.pos) \leq comrange \times prio_i\}$$

$comrange$ is the default communication range for candidates and $1 \leq prio_i$ is the actual priority of transport agent TA_i

4. Related approaches

Stigmergic agents coordinate their behavior through the manipulation of marks in the environment. Classic examples are digital pheromones [4] and computational fields [8]. In stigmergic approaches, coordination laws are implicitly defined by the coordination infrastructure. In contrast, we propose explicit laws for mediating agents' activities. [6, 2] consider explicitly defined laws that determine the outcome of interactions among situated agents in the environment. These approaches however, do not consider explicit laws for perception and communication of situated agents.

Coordination artifacts [10] embody and enact the laws of MAS coordination. The focus of coordination artifacts is on cognitive agents in the first place, while the focus of our work is on situated agents. Low-Governed Interaction (LGI [9]) is an advanced approach for decentralized coordination of agent behavior based on explicitly specified policies. LGI does not make an explicit distinction between laws for perception, action, and communication. On the other hand, LGI emphasizes the necessity of security in open systems.

Research on computational institutions such as electronic institutions [1], logic-based institutions [15], and normative MAS [5] have developed a specific line of regulating infrastructures. The focus of computation institutions in on the regulation of interactions among cognitive agents via laws and norms. These approaches do not consider action, perception, and communication as first-class activities of agents in the environment.

5. Conclusions

The agent environment provides a design abstraction that MAS engineers can exploit to mediate the agents' activities in the system. In this paper, we specified laws that allow to define application specific constraints on agents perception, action, and communication in situated MAS. We illustrated the various laws in an AGV transportation system. Embedding laws in the agent environment improves separation of concerns in MAS and helps to manage complexity. Our long-term goal is to develop a formally founded architectural description language for situated MAS. Support for defining application specific laws will be an important part of this language.

References

[1] J. Arcos, P. Noriega, J. Rodriguez-Aguilar, and C. Sierra. E4MAS through Electronic Institutions. In *Environments for Multi-Agent Systems III*, Lecture Notes in Computer Science, Vol. 4389. Springer-Verlag.

[2] S. Bandini, S. Manzoni, and C. Simone. Dealing with Space in Multiagent Systems: A Model for Situated Multiagent Systems. In *1st Joint Conference on Autonomous Agents and Multiagent Systems*. ACM Press, 2002.

[3] F. Bellifemine, A. Poggi, and G. Rimassa. Jade, A FIPA-compliant Agent Framework. In *4th International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, 1999.

[4] S. Brueckner. *Return from the Ant, Synthetic Ecosystems for Manufacturing Control*. Ph.D Dissertation, Humboldt University, Berlin, Germany, 2000.

[5] C. Castelfranchi, F. Dignum, C. Jonker, and J. Treur. Deliberative Normative Agents: Principles and Architecture. In *6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages*. Springer, 2000.

[6] J. Ferber and J. Muller. Influences and Reaction: a Model of Situated Multiagent Systems. *2nd International Conference on Multi-agent Systems, Japan, AAAI Press*, 1996.

[7] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications with the TOTA middleware. In *2nd International Conference on Pervasive Computing and Communications*. IEEE Computer Society, USA, 2004.

[8] M. Mamei and F. Zambonelli. *Field-Based Coordination for Pervasive Multiagent Systems*. Springer, 2006.

[9] N. Minsky and V. Ungureanu. Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems. *ACM Transactions on Software Engineering Methodologies*, 9(3), 2000.

[10] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In *3rd Joint Conference on Autonomous Agents and Multiagent Systems*, NY, 2004.

[11] A. Rao and M. Georgeff. BDI Agents: From Theory to Practice. In *1st International Conference on Multiagent Systems, 1995, Agents, San Francisco, USA*. MIT Press, 1995.

[12] G. Roman, C. Julien, and A. Murphy. A Declarative Approach to Agent Centered Context-Aware Computing in Ad Hoc Wireless Environments. In *Software Engineering for Large-Scale Multi-Agent Systems, LNCS, Vol. 2603*. Springer Verlag, 2003.

[13] K. Schelfthout. *Supporting Coordination in Mobile Networks: A Middleware Approach*. Ph.D Dissertation, Katholieke Universiteit Leuven, Belgium, 2006.

[14] K. Sycara. Multiagent Systems. *Artificial Intelligence*, 10(2):79–93, 1998.

[15] W. Vasconcelos. Logic-Based Electronic Institutions. Vol. 2990 of Lecture Notes in Computer Science. Springer, 2004.

[16] D. Weyns, N. Boucké, and T. Holvoet. DynCNET: A Protocol for Flexible Task Assignment in Situated Multiagent Systems. In *1st International Conference on Self-Adaptive and Self-Organizing Systems, Boston*, 2007.

[17] D. Weyns and T. Holvoet. Formal Model for Situated Multi-Agent Systems. *Fundamenta Informaticae*, 63(1-2):125–158, 2004.

[18] D. Weyns, A. Omicini, and J. Odell. Environment as a First-Class Abstraction in Multiagent Systems. *Autonomous Agents and Multi-Agent Systems*, 14(1), 2007.

[19] D. Weyns, K. Schelfthout, T. Holvoet, and T. Lefever. Decentralized control of E'GV transportation systems. In *4th Joint Conference on Autonomous Agents and Multiagent Systems, Industry Track*, Utrecht, 2005. ACM Press.

[20] D. Weyns, E. Steegmans, and T. Holvoet. Protocol Based Communication for Situated Multi-Agent Systems. In *3th Joint Conference on Autonomous Agents and Multi-Agent Systems*, New York, USA, 2004. IEEE Computer Society.

[21] D. Weyns, E. Steegmans, and T. Holvoet. Towards Active Perception in Situated Multi-Agent Systems. *Applied Artificial Intelligence*, 18(9-10):867–883, 2004.