MEDIATING AGENTS' ACTIVITIES IN SITUATED MULTI-AGENT SYSTEMS

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

January 2, 2008

Abstract

Situated multi-agent systems are decentralized software systems that consist of autonomous entities (agents) that have an explicit position in the environment. The emphasis of situated agents is on direct coupling of perception to action, modularization of behavior, and dynamic interaction with the environment. These properties make situated multi-agent systems particularly suitable to deal with highly dynamic operating conditions. To achieve the overall system functionalities and qualities, situated agents have to coordinate their behavior. The agent environment provides a means for agents to share information and coordinate their behavior. Laws embedded in the agent environment allow to mediate the activities of the agents in the system by defining application specific constrains on agents' activities. In this paper, we declaratively specify the semantics of laws for perception, action, and communication in situated multi-agent systems. We illustrate the use of 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 multi-agent systems and helps to manage complexity, especially in open and pervasive environments.

Keywords: Mediated interaction, laws, situated multi-agent systems

1 INTRODUCTION

Research in multi-agent systems is concerned with the study, behavior, and construction of a collection of autonomous agents that interact with each other and their environment [35]. Agents are reflective software components that are capable to act autonomously to achieve their design objectives. Individual agents have only limited knowledge and control over the system as a whole. To achieve the overall system functionalities and qualities, agents interact and coordinate their behavior. Multi-agent systems are known for quality attributes such as adaptability, robustness, and scalability, making multi-agent systems particularly interesting to deal with the demanding challenges of complex distributed software applications.

Since the late 1980s, the major trend in multi-agent system research spawns from the study of BDI-agents [30]. 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 multi-agent systems investigates interaction-oriented agent systems. Historically, the idea of situated multi-agent systems originates from the domain of reactive robotics [5]. Later the field spawned into behavior-based approaches [2], stigmeric agents [29], and situated multi-agent systems [15, 3]. The emphasis of situated agency is on direct coupling of perception to action, modularization of behavior, and dynamic interaction with the environment. This contrasts with deliberative approaches that emphasize explicit knowledge and rational choice [44]. A situated multi-agent system consists of a set of agents that cooperate to solve a complex problem in a decentralized way. A situated agent has an explicit position in the environment and has local access to the environment, i.e. each agent is placed in a local context which it can perceive and in which it can act and interact with other agents. A situated agent uses a behavior-based action selection mechanism [40]. Behavior-based action selection is driven by stimuli perceived in the environment as well as internal stimuli.

Direct coupling of perception to action allows situated agents to react rapidly to changing conditions, making the approach particularly appropriate for systems that have to operate in highly dynamic and unpredictable environments.

In situated multi-agent systems, mediated interaction is essential. Classical examples of mediated interaction are digital pheromones and computation fields which situated agents use to coordinate their behavior. A digital pheromone [6] is a dynamic structure that aggregates with additional pheromone that is dropped, diffuses in space and evaporates over time. Agents can use digital pheromones to dynamically form paths to locations of interest. Evaporation of digital pheromones prevent agents to follow paths to outdated information, contributing to the adaptability of the system. A computational field [21] is an abstract force field that is spread in the agent environment by the agents or the agent environment. Agents coordinate their behavior by following the shape of the fields. Computational fields dynamically reshape with topological changes in the underlying spatial or logical structure, contributing to the system's adaptability.

Originating from research on situated multi-agent systems, the *agent environment* has recently been put forward as an explicit design abstraction to support mediated interaction in multi-agent systems [42, 41]. The agent environment enables to shield the complexity of the underlying deployment context to agents, and it provides a means to mediate the interactions among the agents. 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 multi-agent systems and helps to manage complexity, especially in open and pervasive environments.

The contribution of this paper is twofold. First, we give a declarative specification of the semantics of laws for perception, action, and communication in situated multi-agent systems. Such a specification is required to support the disciplined engineering of situated multi-agent systems. Second, we give a number of concrete laws in a situated multi-agent system that we have developed for the control of automatic guided vehicles (AGV). Perception, action, and communication are generally acknowledged as the primary means for situated agents to act in the environment and interact with one another [14, 3]. Specific laws for perception, action, and communication enable engineers to deal with these different concerns explicitly and separately. Existing agent-based approaches for mediated coordination such as coordination infrastructures [26] and electronic institutions [25] focus on communicative interactions among knowledge-oriented agents while the approach proposed in this paper focuses on mediated coordination of interaction-oriented agents that have an explicit position in the environment.

Overview. 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, and shows how we have applied this model in an automated transportation system. In section 3, we specify laws for perception, action, and communication, and we illustrate the laws with concrete examples in the automated transportation system. 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 multi-agent system 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 multi-agent systems [38]. This model extends architectures for situated agents from mechanisms for action selection to integral architectures that provides support for different concerns of agents, including support for selective perception enabling situated agents to perceive the environment according to their current tasks, support for protocol-based communication enabling agents to exchange messages according to well-defined communication protocols, and support for explicit social interaction with roles and situated commitments enabling agents to set up explicit collaborations and play different roles in such collaborations. In addition, the model introduces the notion of *agent environment* as a first-class design abstraction in situated multi-agent systems. The agent environment provides an abstract representation of resources in the external world and keeps the state of this virtualization in synchrony with the state of the actual resources. The agent environment also supports mediated interaction, regulating the access to shared resources and governing interactions between agents. In addition, the agent environment can provide a reflective mechanism that enables the composition and function of the agent environment to be modified at runtime. The use of reflection to support self-managing properties such as self-healing and self-optimization is subject of our future research.

The focus of this paper is on the agent environment, in particular on support for mediated interaction. In this section, we give a high-level model of the agent environment and we briefly explain how we have applied the

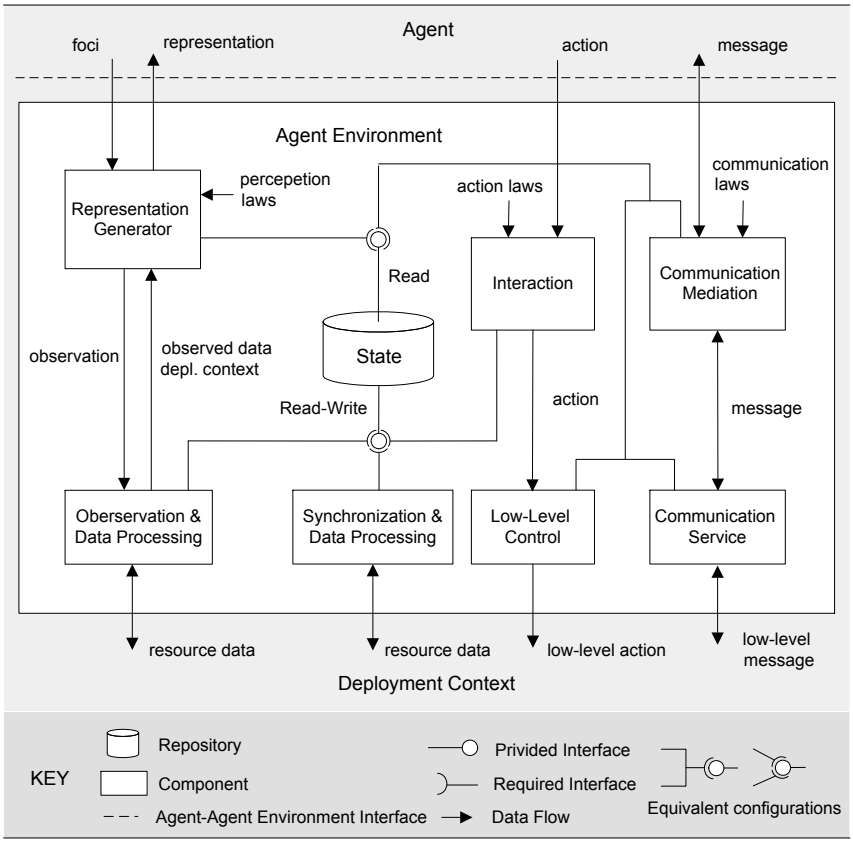

Figure 1: Model of the agent environment

agent environment in an automated transportation system. Support for reflection in the agent environment is outside the scope of the presented model.

2.1 Model for the 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 processors, a network, the operating system, distributed middleware, sensors and actuators, a printer, 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 elements of the agent environment in turn.

The `State` repository contains data that is shared between the components of the agent environment via the `Read` and `Read-Write` interfaces. 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. Foci allow an agent to sense the environment (agent environment and deployment context) for specific types of information. The representation generator uses the current state of the agent environment (via the Read interface of the State repository) and possibly state collected (via `observation`) from the deployment context (`observed data depl. 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 resource 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 setting. The observation & data processing component can provide additional functions to pre-process data, examples are sorting, filtering, and integration of observed data. To provide its functionalities, observation & data processing may use data of the state repository via the Read-Write interface (e.g., to translate logic locations to actual addresses in a network).

Interaction is responsible to deal with agents' actions. 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. 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 (via the Read-Write interface). 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. An example is a robot agent that instructs the robot to drive in a particular direction. This action will be translated by the low-level control component into corresponding actuator commands. The low-level control component decouples the interaction component from the details of the deployment context.

Communication Mediation mediates the communicative interactions (i.e. exchange of messages) among agents. Communication mediation 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 messages coded in a high-level message description language used by agents into low-level message 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 [4].

Synchronization & Data Processing collects resource data from the deployment context and synchronizes particular state of the agent environment (via the Read-Write interface of the State repository) with the state of resources as well as state of the agent environment on different nodes. State synchronization 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. Dedicated middleware can provide support to spread and collect data in a distributed setting. Examples of middleware for mobile network environments are discussed in [31, 34, 21]. Synchronization & data processing converts the resource data collected 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 automatic guided vehicle (AGV) transportation system is a fully automated transport system that uses multiple AGVs to provide logistic services 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 multi-agent system to develop a decentralized AGV control system aiming to improve the flexibility in the system [43]. Fig. 2 shows the model of AGV application.

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 the transport base, i.e. a

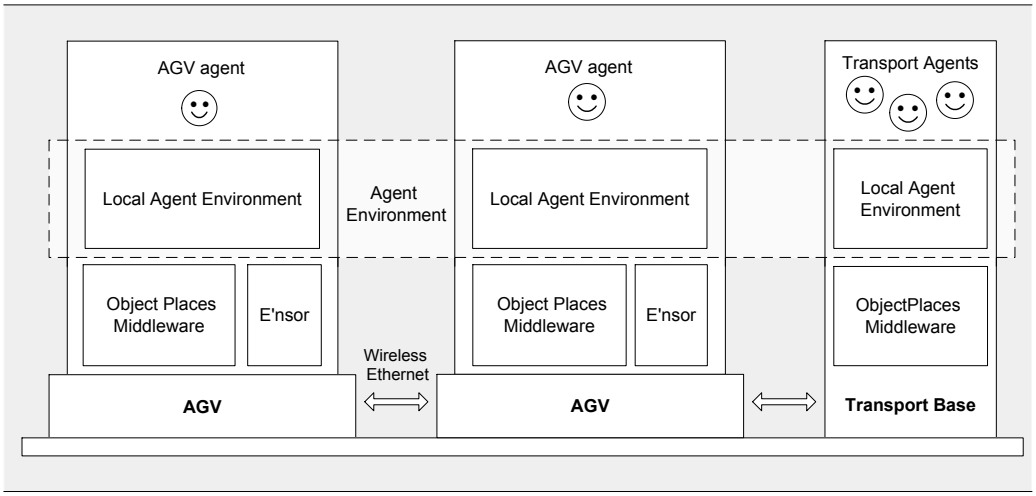

Figure 2: Model of the AGV application

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. Since the agent environment is necessarily distributed over the AGVs and the transport base, an instance of the local agent environment is deployed on each node in the system (mobile AGVs and the transport base).

The states of local agent environments on neighboring nodes are synchronized with each other opportunistically, as the need arises. State synchronization is supported by the ObjectPlaces middleware. ObjectPlaces proposes two programming abstractions, views and roles, to simplify mobile application development. The first abstraction, a view, is an automatically up-to-date collection of data objects, that are copies or representations of data objects available on a set of nodes in the network. The middleware automates gathering the data objects from a set of nodes, and maintains the view in the face of dynamically changing availability of the data objects. The second abstraction, a role, encapsulates the behavior of a component of the application engaging in a protocol. The middleware automates the setup and maintenance of an interaction session between a number of participating components in the mobile network, in the face of a continuously changing number of participants. The ObjectPlaces middleware encapsulates the tedious management tasks associated with distribution and mobility in the AGV transportation system. It significantly reduces the complexity of tackling distributed coordination problems such as collision avoidance, deadlock detection, and task assignment in the AGV transportation system. A detailed discussion of ObjectPlaces is out of scope of this paper. We refer the interested reader to [34].

AGVs are equipped with low-level control software that is called Ensor (Egemin Navigation System On Robot). Ensor provides an interface to command the AGV and to monitor its state. We fully reused the control software in the project. The local agent environment uses Ensor to steer the vehicle based on the commands of the AGV agent, and to regularly poll the vehicles status and adjust its own state appropriately.

Coordination Medium. The local agent environment offers high-level primitives to agents to act in the environment, perceive the environment, and communicate with other agents. This enables agents to share information and coordinate their behavior. As an illustration, we explain how agents exploit the agent environment to avoid collisions by coordinating with other agents through the local agent environment. AGV agents mark the path they are going to drive in their local agent environment using *hulls*. A hull is the physical area an AGV occupies and a series of hulls describe the physical area the AGV occupies along a certain path. If an AGV's hulls do not intersect with hulls of other AGVs, the AGV can drive over the reserved path. In case of a conflict, the involved local agent environments use the priorities of the transported loads to determine which AGV can move on. AGV agents monitor the local agent environment and only instruct the AGV to move on when they are allowed. Afterwards, the AGV agents remove the markings in the agent environment. This example shows that the local agent environments serve as a flexible co-

ordination medium: agents coordinate by putting marks in the local agent environment, and observing marks from other agents.

3 LAWS FOR MEDIATING ACTIVITIES

Now, we declaratively specify the semantics of laws for perception, action, and communication in the agent environment as shown in Fig. 1. For the specification we use a simple formal language based on set theory. Each type of law is illustrates with an example 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 Voc: the vocabulary of domain concepts, and 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 environment;
the set of state elements is based on ontology Ont; a state element is (1) a part of the state of the agent environment, or (2) it represents a part of the state of resources in the deployment context. A state element 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 set of state elements $\sigma \in \Sigma$ of the environment under consideration as the *current state* of the environment (i.e. the actual state of the agent environment plus possible representations of state of resources in the deployment context)

$\underline{dom(f)}$ 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 an agent 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 are 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 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 environment σ. $ApplyL_P$ results in a restricted perception scope η'.

For η' holds:

$\forall s \in S_E, \sigma \in \Sigma :$
$s \in domint(\eta')$ iff $(s \in domint(\eta)) \wedge$
$(\forall\, lp \in L_P, \forall\, sc \in \eta' :$
$(\forall\, co \in lp\,(sc, \sigma) : co(s) = false))$
$s \notin domint(\eta')$ otherwise

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 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 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. As we have explained above, AGV agents mark the path they are going to drive in their environment using hulls. A series of hulls mark 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. 3 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 collision range of an AGV_{req} is defined as:

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

$in\text{-}col\text{-}range_{req}$ denotes the actual set of AGVs within collision range of the requesting AGV; $(AGV \backslash \{AGV_{req}\})$ represents the set of AGVs in the system excepting the requesting AGV; the $dist$ function calculates the distance between two AGVs, 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: $OperationGeneration : Act \rightarrow G$

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 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 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 $(g(s) = true) \wedge$

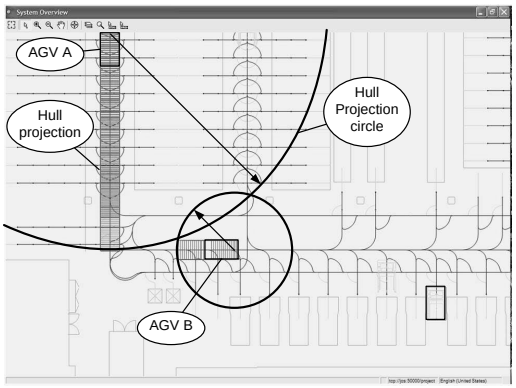

Figure 3: Determining AGVs in collision range

$(\forall\, la \in L_A : (\forall\, co \in la(g,\sigma) : co(s) = false))$
$g'(s) = false$ 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 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 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 agent removes the markings in the agent environment. The constraint that determines the safe condition of a hull projected by AGV_{req} is defined as follows:

$$\begin{aligned} &AGV_{req}.hullstate = safe \text{ iff} \\ &\quad in\text{-}col\text{-}range_{req} = \emptyset \vee \\ &\quad \forall AGV_{other} \in in\text{-}col\text{-}range_{req} : \\ &\qquad AGV_{other}.hullstate \neq safe \\ &\qquad \wedge prior(AGV_{req}.hullprio, AGV_{other}.hullprio) \end{aligned}$$

The constraint defines that it is safe for a requesting AGV to move on if there is no other AGV in collision range, or if none of the AGVs in collision range is in the safe state and all the AGVs in collision range have a lower priority than the requesting AGV. $hullstate \in \{safe, unsafe\}$ denotes the actual state of a hull projection; $hullprio$ denotes the actual hull priority of an AGV; the function $prior(AGV_{req}.hullprio, AGV_{other}.hullprio)$ determines whether an AGV_{req} has priority over an AGV_{other} to move on, i.e. $prior$ returns $true$ if the hull priority of AGV_{req} is higher than that of AGV_{other} 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;
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 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 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 agent environment to interact with. Such a law is for example applied when a transport agent is searching for a suitable AGV to execute its task. To enable adaptive task assignment, we have developed a dynamic contract net protocol called DynCNET [39]. DynCNET allows both AGV and transport agents to revise task assignment when opportunities occur while AGVs drive toward loads. Since it is not scalable to let all agents in the system interact, the agent environment restricts the communication to nearby candidates. E.g., in Fig. 4, there are currently three tasks within the scope of AGV A to execute the transport: u, w, and w. In a similar way, the scope for possible AGVs is restricted for transport agents. The scope of possible candidates may dynamically grow, e.g., when the priority of a transport increases. The constraint that restricts the set of candidate AGVs for a task agent TA_t is defined as:

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

$candid_t$ defines the actual set of candidate AGVs for task assignment. $comrange$ is the maximal communication range for task assignment, $TA_i.taskprio$ is the actual priority of transport agent TA_i. Applying the law to the situation shown in Fig. 4 for transport x yields $\{AGV_E, AGV_F, AGV_G\}$. Similarly, a law is defined that restricts the set of candidate tasks for AGV agents.

We have tested the effect of variable communication ranges for transport agents. The test results are shown in Fig. 5. The test were performed on the map of a simulated real-world AGV transportation system [1]. The size of the physical layout is 134 m x 134 m. The map has 56 pick and 50 drop locations. We used a standard transport profile that generates about 140 transports per hour real time. Each transport is assigned a random priority $[0.1 \ldots 1.0]$ that increases over time with a maximum of 1.0. The system uses 14 AGVs. Tests were run for 2 hours in real time. The figure on the right hand side shows that the number of messages increases with increasing communication range. However, from a maximum communication range of 60 m the increase is marginal. The figure on the right hand side shows that for low values of maximal communication range, the number of finished transports significantly increases. However, from a maximal range of 40 m, the gain becomes marginal. The results indicate that for the given system, 40 m is a suitable value for the maximal communication range of transport agents.

4 RELATED APPROACHES

Mediated coordination is an extensively studied field. In this section, we discuss a number of related agent-based approaches for mediated coordination. But first we put our work in a broader setting of coordination and perversive computing research.

4.1 Position with respect to coordination and pervasive computing

In the seminal work, Malone and Crowston [20] refer to coordination as the processes for managing shared resources, producer/consumer relationships, simultaneity constraints, and task/subtask dependencies. The agent environment in our approach provides concrete instances of these processes that are specific to the requirements for mediated coordination in situated multi-agent systems. Ciancarini defines a coordination model as a triple (E,M,L) [11, 7]. E are the coordinable entities (components). In our context the agents are the coordinable entities. M are the coordinating media (connectors). In our work M is instantiated by the various means for mediated activity provided by the agent environment. Finally, L are the coordination laws ruling actions by coordinable entities with respect to the coordination media. In our work, we have defined laws for perception, action, and communication, specific for situated multi-agent systems.

In [37], Mark Weiser introduced the idea of ubiquitous computing as "the seamlessly integration of computers into the world at large." According to Saha and

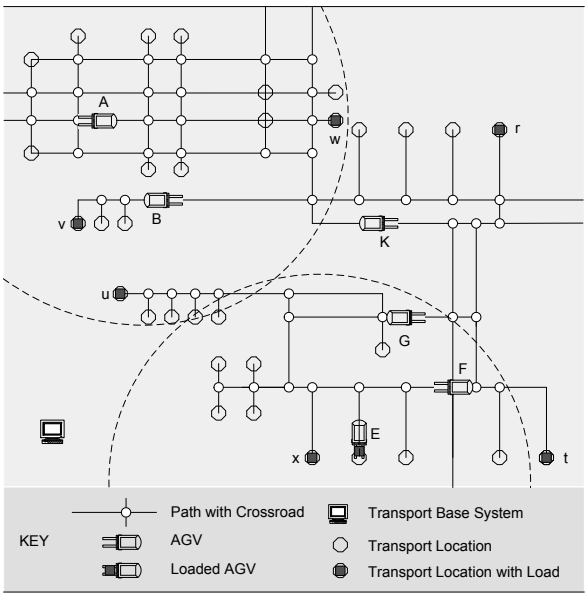

Figure 4: Candidates for task assignment with the dynamic contract net protocol

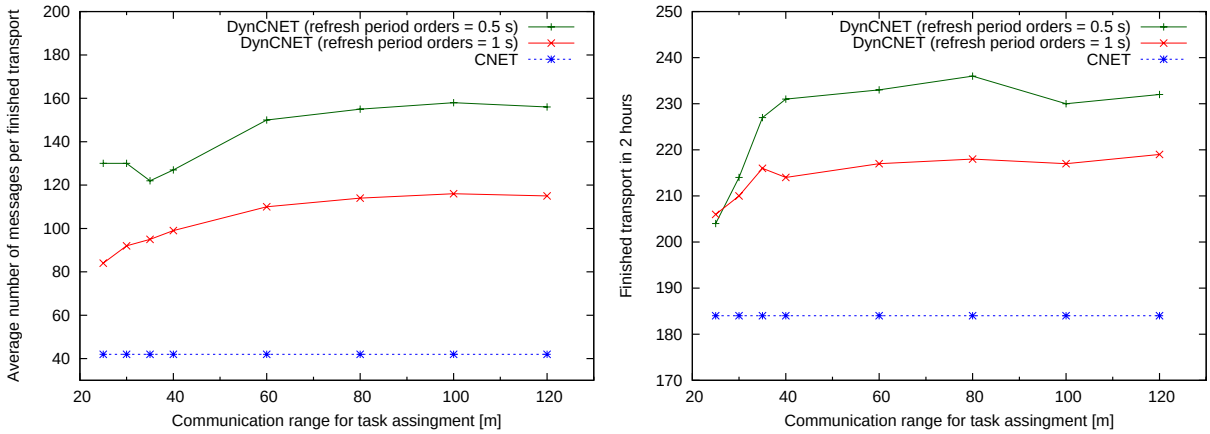

Figure 5: Communication load (left) and throughput (right) for variable communication range of task agents

Mukherjee [32], the technological advances necessary to build a pervasive computing environment fall into four broad areas: devices, networking, middleware, and applications. Pervasive middleware provides the layer between the networking kernel ad the end-users applications running on pervasive devices. The authors state: "this pervasive middleware will mediate interactions with the networking kernel on the users behalf and will keep users immersed in the pervasive computing space." The agent environment in our work provides a domain-specific middleware for situated multi-agent systems that contributes to the objectives of pervasive middleware. Mediating activities of situated agents (i.e. application components) and providing an up-to-date visualization of the surrounding environment are core activities of the agent environment.

Pervasive computing requires context management that is proactive and includes support for location monitoring, real-time information processing, merging data from multiple and possibly disagreeing sensors [32]. The model for the agent environment shown in Fig. 1 provides such support for the domain of situated multi-agent systems. The Observation & Data Processing module provides functionality to observe the surrounding environment and integrates functionality to preprocess the collected data. The Synchronization & Data Processing module collects data from the surrounding environment in a proactive fashion and allows to resolve conflicts among different nodes. Collision avoidance in the AGV application is an example of conflict resolution

among nodes.

Satyanarayanan points out that good system design in pervasive computing requires "the need for reducing interactions between distant entities" [33]. He calls this the need for "localized scalability." Laws in the agent environment allow to manage the scope of interactions among application components in a dynamic fashion.

4.2 Related agent-based approaches

Stigmergic agent systems. Stigmergic agents are simple entities that coordinate their behavior through the manipulation of marks in the agent environment in a similar way as social ants coordinate [18]. A classic example of stigmergic coordination are digital pheromones [6] which software agents use to coordinate their behavior. A digital pheromone is a dynamic structure in the agent environment that aggregates with additional pheromone that is dropped, diffuses into space, and evaporates over time. Agents can use pheromones to dynamically form paths to locations of interest. Digital pheromones combine reinforcement of interesting information (by the agents) with decay of information over time (truth maintenance by the agent environment). Another well-established approach of stigmergic coordination is a computational field [21]. In this approach, the movements of agents are driven by abstract force fields that are spread in the agent environment (by agents or the agent environment itself). Agents coordinate their behavior by following the shape of the fields.

In stigmergic approaches, coordination laws are implicitly defined by the coordination infrastructure. In contrast, our approach proposes explicit laws for mediating agents' activities, including laws for perception, action, and communication. Moreover, stigmergic infrastructures do not support communicative interactions by means of message exchange.

Situated multi-agent systems consider explicitly defined laws that determine the outcome of interactions among situated agents in the environment. We discuss two representative examples.

In [15], Ferber and Müller propose a basic model for situated multi-agent systems. Central to the model is the way actions are modeled. The action model distinguishes between influences and reactions to influences. Influences are produced by agents and are attempts to modify the course of events in the world. Reactions, which result in state changes, are produced by the agent environment by combining influences of all agents, given the local state of the agent environment and the laws of the world. Laws as defined by the authors are first-class concepts. Laws however, are limited to the response of influences performed by the agents. In contrast, our model for agent environment supports laws for perception, action, and communication. The model of Ferber and Müller has mainly been applied in the context of agent-based simulation.

In the Multilayered Multi-Agent Situated System [3] (MMASS) agents and the environment are explicitly modeled. MMASS introduces the notion of agent type which defines agent state, perceptual capabilities and a behavior specification. The environment is modeled as a multi-layered structure, where each layer is represented as a connected graph of sites. Layers may represent abstractions of a physical environment, but can also represent logical aspects, e.g. the organizational structure of a company. Between the layers specific connections (interfaces) can be defined that are used to specify that information generated in one layer, may propagate into other layers. MMASS defines a number of basic primitives, such as emit (starts the diffusion of a field), transport (defines the movement of the agent), or trigger (specifies state change when a particular condition is sensed in the environment). These primitives have characteristics similar to reactions and laws as introduced by Ferber and Müller. In our work, we define laws as first-class concepts, not only for action, but also for perception and communication.

Coordination infrastructures Coordination infrastructures are studied for a long time. Blackboard systems were the first type of mediated interaction models proposed by AI researchers [12]. A blackboard is an intermediary data repository that enables cooperating software modules to communicate indirectly and anonymously. In contrast to blackboard systems, tuple-based technologies use associative access to a shared data space for communication and synchronization purposes. Tuplespaces were first introduced in Linda [17]. Agents in Linda communicate by putting tuples in, and removing them from a shared space, i.e., the tuplespace. Throughout the years, distributed variants appeared, such as Sun's JavaSpaces [16], TuCSoN [28], MARS [8], and LIME [24]. Coordination infrastructures [26] provide reusable solutions for coordination of cognitive agents. These infrastructures define laws that govern the observable behavior of knowledge-oriented agents and regulate the interactions in agent societies via rules, norms, and laws. Coordination artifacts [27] generalize over different coordination models. Coordination artifacts embody the laws of coordination and can be used by engineers to regulate the behavior of multi-agent systems. Coordination artifacts provide a reflective interface that allows adapting the behavior of the artifacts at runtime. The model for the agent environment

presented in this paper provides a coordination infrastructure tailored to situated multi-agent systems. Contrary to existing approaches, our model provides explicit support for agents' perception, action, and communication, enabling engineers to deal with these different concerns explicitly and separately. Existing approaches provide more generic support, but, as a consequence lack specific support and reuse for mediated coordination in situated multi-agent systems.

Low-Governed Interaction (LGI [23]) is an approach for decentralized coordination of agents based on explicitly specified policies. LGI is a mature approach for mediated interaction that has proven to be useful in various domains. However, 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.

Computational institutions. Research on computational institutions such as electronic institutions [25], virtual organizations [9], logic-based institutions [36], and normative multi-agent systems [10] is an extensive studied field of regulating infrastructures. Computational institutions allow multi-agent system engineers to impose laws and norms to agents. Norms can be prescriptive and only allow permitted behaviors, or undesired behaviors can be detected and sanctioned by the infrastructure. Interaction mediation in computational institutions are tailored to cognitive agents and are typically applied in domains such as electronic markets, electronic commerce, and other B2B scenarios. Computational institutions focus on communicative interactions. In contrast, the agent environment in our work focuses on mediated coordination in situated multi-agent systems that are typically applied in domains where agents have an explicit position in the environment, such as automated transportation systems, robotics, manufacturing control, mobile and ad-hoc networks, wireless sensor networks, traffic monitoring and control, etc.

Distributed constraint satisfaction problems. Finally, there are points of similarity between distributed constraint satisfaction algorithms and the work presented in this paper. Distributed constraint satisfaction algorithms are used to solve unbounded problems, i.e. the variables and constraints involved in the problem are not bounded (e.g. in meeting scheduling) or the admissible values and value tuples are unbounded (e.g. in supply chain configuration). In asynchronous backtracking [13], each agent (representing one variable) maintains a view on the set of agents called "agentview". The agentview determines the agents with higher priorities that are relevant for the agent's search. The relative priority between the agents can be considered as a perception law that determines whether an agent belongs to the agentview or not. [22] proposes an ant-based approach for solving dynamic constraint optimization problems. In this approach, a graph environment is used that represents the constraint problem itself. The graph determines how the ants can walk around building a solution to the problem. Ants can put digital pheromones on the edges of the graph that the ants use to approximate traditional variable and value ordering heuristics. The graph environment in this approach is a particular instance of the agent environment as presented in this paper. Whereas laws in the agent environment are defined as first-class concepts, the laws that restrict the behavior of the ants are encoded in the graph structure and the dynamics of the digital pheromones.

5 CONCLUSIONS

The agent environment provides a design abstraction that multi-agent system 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 multi-agent systems. We illustrated the various laws in an AGV transportation system. Embedding laws in the agent environment improves separation of concerns in multi-agent systems and helps to manage complexity.

In ongoing research, we study how the agent environment can be exploited to support dynamic organizations in situated multi-agent systems [19]. The objective is to separate the management of dynamic evolution of organizations (by the agent environment) from the actual functionality provided by the agents playing roles in the organizations. In this approach, a set of organization laws define the way organizations should evolve given the current context. The aim of separating these concerns is to make it easier to understand, design, and manage organizations in multi-agent systems. A long-term goal of our research is to develop a formally founded architectural description language for situated multi-agent systems. Support for defining different types of laws will be an important part of this language.

ACKNOWLEDGEMENTS

Danny Weyns is supported by the Research Funding Flanders (FWO). We are grateful to the anonymous reviewers for the valuable feedback on earlier versions of this paper.

References

[1] AGV Simulator, DistriNet, AgentWise. http://www.cs.kuleuven.ac.be/∼distrinet/taskforces /agentwise/agvsimulator/.

[2] R. Arkin. *Behavior-Based Robotics*. Massachusetts Institute of Technology, MIT Press, Cambridge, MA, USA, 1998.

[3] 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.

[4] 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.

[5] R. Brooks. Achieving artificial intelligence through building robots. *AI Memo 899, MIT Lab*, 1986.

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

[7] N. Busi, P. Ciancarini, R. Gorrieri, and G. Zavattaro. Coordination models: A guided tour. In *Coordination of Internet Agents: Models, Technologies, and Applications*, pages 6–24. 2001.

[8] G. Cabri, L. Leonardi, and F. Zambonelli. MARS: a Programmable Coordination Architecture for Mobile Agents. *IEEE Internet Computing*, 4(4):26–35, 2000.

[9] H. Cardoso, A. Rocha, and E. Oliveira. Virtual organization support through electronic institutions and normative multi-agent systems. In *Handbook of Research on Nature Inspired Computing for Economy and Management II*. Idea Group, Inc., 2006.

[10] 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.

[11] P. Ciancarini. Coordination models and languages as software integrators. *ACM Comput. Surv.*, 28(2):300–302, 1996.

[12] R. S. Englemore and A. Morgan. *Blackboard Systems*. Addison-Wesley, 1988.

[13] B. Faltings. *Distributed Constraint Programming*, pages 699–729. Foundations of Artificial Intelligence. Elsevier, 2006.

[14] J. Ferber. *An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[15] 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.

[16] E. Freeman, S. Hupfer, and K. Arnold. JavaSpaces: Principles, Patterns, and Practice. *Addison-Wesley*, 1997.

[17] D. Gelernter and D. Carrierro. Coordination languages and their significance. *Communications of the ACM*, 35(2), 1992.

[18] P. Grassé. La Reconstruction du nid et les Coordinations Inter-Individuelles chez Bellicositermes Natalensis et Cubitermes sp. La theorie de la Stigmergie. Essai d'interpretation du Comportement des Termites Constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[19] R. Haesevoets, B. V. Eylen, D. Weyns, A. Helleboogh, T. Holvoet, and W. Joosen. Coordinated monitoring of traffic jams with context-driven dynamic organizations. In *Engineering Environment-Mediated Multiagent Systems, Lecture Notes in Computer Science*. Springer, 2008.

[20] T. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26(1):87–119, 1994.

[21] M. Mamei and F. Zambonelli. *Field-based Coordination for Pervasive Multiagent Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[22] K. Mertens. *An Ant-Based Approach for Solving Dynamic Constraint Optimization Problems*. Ph.D Dissertation, Katholieke Universiteit Leuven, Belgium, 2006.

[23] 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.

[24] A. Murphy, G. Picco, and G. Roman. LIME: a Middleware for Physical and Logical Mobility. *21th International Conference on Distributed Computing Systems*, 2001.

[25] P. Noriega and C. Sierra. Electronic Institutions: Future Trends and Challenges. In *6th International Workshop on Cooperative Information Agents*, Lecture Notes in Computer Science, Vol. 2446. Springer-Verlag, London, UK, 2002.

[26] A. Omicini, S. Ossowski, and A. Ricci. Coordination infrastructures in the engineering of multiagent systems. In *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, chapter 14. Kluwer Academic Publishers, 2004.

[27] 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.

[28] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, 1999. Special Issue: Coordination Mechanisms for Web Agents.

[29] H. V. D. Parunak. Go to the Ant: Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 75:69–101, 1997.

[30] 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.

[31] 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.

[32] D. Saha and A. Mukherjee. Pervasive computing: A paradigm for the 21st century. *Computer*, 36(3):25–31, 2003.

[33] M. Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications*, 8(4):10–17, 2001.

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

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

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

[37] M. Weiser. The computer for the 21 st century. *Scintific American*, pages 94–104, 1991. Reprinted in IEEE Pervasive Computing, Jan.–Mar. 2002, pp. 19–25.

[38] D. Weyns. *An Architecture-Centric Approach for Software Engineering with Situated Multiagent Systems*. Ph.D Dissertation, Katholieke Universiteit Leuven, Belgium, 2006.

[39] 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, SASO, Boston, USA*, 2007.

[40] D. Weyns and T. Holvoet. From Reactive Robotics to Situated Multiagent Systems: A Historical Perspective on the Role of Environment in Multiagent Systems. In *Engineering Societies in the Agents World VI, 6th International Workshop, ESAW, Kusadasi, Turkey*, Lecture Notes in Computer Science, Vol. 3963. Springer-Verlag, 2006.

[41] 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.

[42] D. Weyns, V. Parunak, F. Michel, T. Holvoet, and J. Ferber. Environments for Multiagent Systems, State-of-the-Art and Research Challenges. In D. Weyns, V. Parunak, and F. Michel, editors, *Environments for Multiagent Systems*, volume 3374 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[43] 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.

[44] M. Wooldridge and N. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2), 1995.