The MACODO Middleware for Context-Driven Dynamic Agent Organizations

DANNY WEYNS, ROBRECHT HAESEVOETS, ALEXANDER HELLEBOOGH, TOM HOLVOET, and WOUTER JOOSEN
DistriNet Labs, Katholieke Universiteit Leuven

One of the major challenges in engineering distributed multiagent systems is the coordination necessary to align the behavior of different agents. Decentralization of control implies a style of coordination in which the agents cooperate as peers with respect to each other and no agent has global control over the system, or global knowledge about the system. The dynamic interactions and collaborations among agents are usually structured and managed by means of roles and organizations. In existing approaches agents typically have a dual responsibility: on the one hand playing roles within the organization, on the other hand managing the life-cycle of the organization itself, for example, setting up the organization and managing organization dynamics. Engineering realistic multiagent systems in which agents encapsulate this dual responsibility is a complex task.

In this article, we present a middleware for context-driven dynamic agent organizations. The middleware is part of an integrated approach, called MACODO: Middleware Architecture for COntext-driven Dynamic agent Organizations. The complementary part of the MACODO approach is an organization model that defines abstractions to support application developers in describing dynamic organizations, as described in Weyns et al. [2010].

The MACODO middleware offers the life-cycle management of dynamic organizations as a reusable service separated from the agents, which makes it easier to understand, design, and manage dynamic organizations in multiagent systems. We give a detailed description of the software architecture of the MADOCO middleware. The software architecture describes the essential building blocks of a distributed middleware platform that supports the MACODO organization model. We used the middleware architecture to develop a prototype middleware platform for a traffic monitoring application. We evaluate the MACODO middeware architecture by assessing the adaptability, scalability, and robustness of the prototype platform.

Categories and Subject Descriptors: I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

General Terms: Design, Experimentation

This research is supported by the DiCoMAS project, which is funded by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT) and the GOA project, which is funded by the Katholieke Universiteit Leuven. Danny Weyns is funded by the Research Foundation Flanders (FWO).
Authors' address: DistriNet Labs, Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium; email: danny.weyns@cs.kuleuven.be.

DOI 10.1145/1671948.1671951 http://doi.acm.org/10.1145/1671948.1671951

Additional Key Words and Phrases: Context, middleware, organization, role, software architecture

ACM Reference Format:

Weyns, D., Haesevoets, R., Helleboogh, A., Holvoet, T., and Joosen, W. 2010. The MACODO middleware for context-driven dynamic agent organizations. ACM Trans. Autonom. Adapt. Syst. 5, 1, Article 3 (February 2010), 28 pages.
DOI = 10.1145/1671948.1671951 http://doi.acm.org/10.1145/1671948.1671951

1. INTRODUCTION

One of the major challenges in software engineering of distributed multiagent systems is the coordination necessary to align the behavior of different agents. Since coordination determines whether agents cooperate effectively, it has a direct impact on the satisfaction of a multiagent system's functional requirements. Furthermore, since coordination in a distributed setting is basically realized by communication, coordination has a large impact on quality attributes such as efficiency and resource usage. Decentralization of control implies a style of coordination in which the agents cooperate as peers with respect to each other, and no agent has global control over the system, or global knowledge about the system. As a result, complex interactions are necessary to achieve consensus since there is no single agent that can make a centralized decision. In the case of mobile applications, agents have to take into account the distribution of the nodes in physical space and other properties of the environment, which adds extra complexity to the realization of coordination.

A typical way to structure and manage interactions among agents is by means of organizations [Kendall 2000; Omicini 2001; Odell et al. 2003; Zambonelli et al. 2003]. In an organization, agents work together based on well-defined roles, each role being responsible for a particular functionality of an organization. Changes in the environment in which the system is situated can trigger an organization to dynamically reorganize [Dignum et al. 2004]. Dynamic reorganizations include intraorganization adaptations such as assigning and revoking roles (e.g. an agent that enters or leaves an e-market), and interorganization adaptations such as merging and splitting organizations (e.g. a mobile sensor network in which two groups of sensors get connected to each other or disconnected from each other). Engineering an organization-oriented multiagent system is a challenging task. Most of the existing work on organizations takes an agent-centered perspective [Ferber and Gutknecht 1998; Dignum et al. 2004; Sims et al. 2008], endowing agents with a dual responsibility: on the one hand playing roles within the organization, on the other hand managing the life-cycle of the organization itself, for example, setting up the organization and managing organization dynamics. Engineering realistic multiagent systems in which agents encapsulate this dual responsibility is a complex task.

To support engineers of organization-oriented multiagent systems, we present a middleware for context-driven dynamic agent organizations. The middleware is part of an integrated approach, called MACODO: Middleware Architecture for COntext-driven Dynamic agent Organizations. The complementary part of the MACODO approach is an organization model that defines abstractions to support application developers in describing dynamic organizations, as described in Weyns et al. [2010].

The MACODO middleware encapsulates the life-cycle management of dynamic organizations as a reusable service, clearly separated from the functionality of agents playing roles within these organizations. Driven by changes in the context,[1] the middleware initiates, maintains, and adapts organizations and actively advertises role positions to the agents, supporting the necessary collaborations between agents needed in the current context. Separating life-cycle management of dynamic organizations from the agents playing roles in the organizations promotes reuse and makes it easier to understand, design, and manage dynamic organizations in multiagent systems. We give a detailed description of the software architecture of the MADOCO middleware. The software architecture describes the essential building blocks of a distributed middleware platform that supports the MACODO organization model. We used the middleware architecture to develop a prototype middleware platform for a traffic monitoring application. We evaluate the MACODO middeware architecture by assessing the adaptability, scalability and robustness of the prototype platform.

Overview. This article is structured as follows. In Section 2, we introduce the MACODO organization model and we illustrate it with a traffic monitoring scenario that we use as a running example in the article. Section 3 describes the MACODO software architecture. In Section 4, we describe the setup of the experiments and we evaluate the MACODO prototype platform applied to the traffic monitoring system. In Section 5, we discuss related work. Finally, we draw conclusions and outline issues for future research in Section 6.

2. MACODO ORGANIZATION MODEL

In this section, we give an overview of the MACODO organization model. First, we introduce the traffic monitoring application that we use as a running example. Then we introduce the MACODO organization model. We use a concrete scenario in the traffic monitoring application to illustrate the basic abstractions of the MACODO organization model. The same scenario will be used to clarify several aspects of the MACODO software architecture in Section 3. In Section 4 we discuss a prototype implementation of the traffic monitoring application that we used to evaluate the MACODO programming abstractions and the software architecture. For a complete formal specification of the MACODO organization model we refer you to Weyns et al. [2010].

2.1 Coordinated Monitoring of Traffic Jams

The monitoring application we consider fits in the domain of intelligent transportation systems, a worldwide initiative to exploit information and

[1] In line with Hirschfeld et al. [2008], we use the term context for "any information which is computationally accessible and upon which behavioral variations depend." In MACODO, the behavioral variations are organization dynamics, and the computationally accessible information is the information in the environment that can be observed and upon which organization dynamics depend.

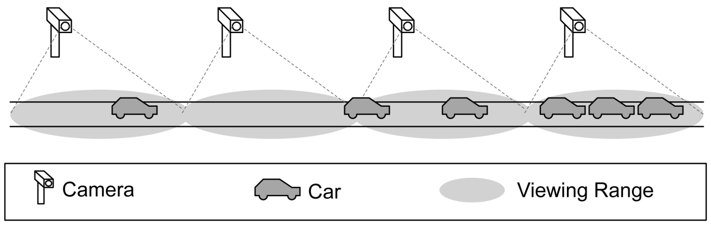


Fig. 1. An example of a highway with traffic cameras.

communication technology to improve traffic [ITS 2008; ERTICO 2008]. The system consists of a set of intelligent cameras distributed evenly along the road. An example of a highway is shown in Figure 1. Each camera has a limited viewing range and cameras are placed to get optimal coverage of the highway with a minimum of overlap.

Cameras are equipped with a data processing unit capable of processing the monitored data, and a communication unit to communicate with other cameras. A camera is able to measure two traffic variables within its viewing range: the current density—the number of vehicles per length unit—and the average speed of the vehicles. These two variables can be used to determine the current congestion level and decide whether there is a traffic jam in the viewing range of a camera [Kerner 2004].

The task of the cameras is to detect and monitor traffic jams on the highway in a decentralized way, avoiding the bottleneck of a centralized control center. Possible clients of the monitoring system are traffic light controllers, driver assistance systems such as systems that inform drivers about expected travel time delays, systems for collecting data for long term structural decision making, and so forth.

Traffic jams can span the viewing range of multiple cameras and can dynamically grow and dissolve. By default, each camera monitors the traffic state within its viewing range, which makes up its context. When a traffic jam occurs, the camera has to collaborate with other cameras detecting the same traffic jam. Because there is no central point of control, cameras have to aggregate the data monitored by each of the cameras to determine the position of the traffic jam on the basis of the head and tail of it. One of the cameras will be responsible to distribute the aggregated data of the traffic jam to the interested clients. Cameras will enter or leave the collaboration whenever the traffic jam enters or leaves their viewing range.

2.2 Overview of the MACODO Organization Model

In this section we explain the basic abstractions MACODO offers to the application developer to describe dynamic organizations. We use a graphical notation and give an informal description of the abstractions. Weyns et al. [2010] gives a detailed formal specification of the organization model.

Figure 2 shows an overview of the basic abstractions of the MACODO organization model. We explain the abstractions using the scenario shown in Figure 3.

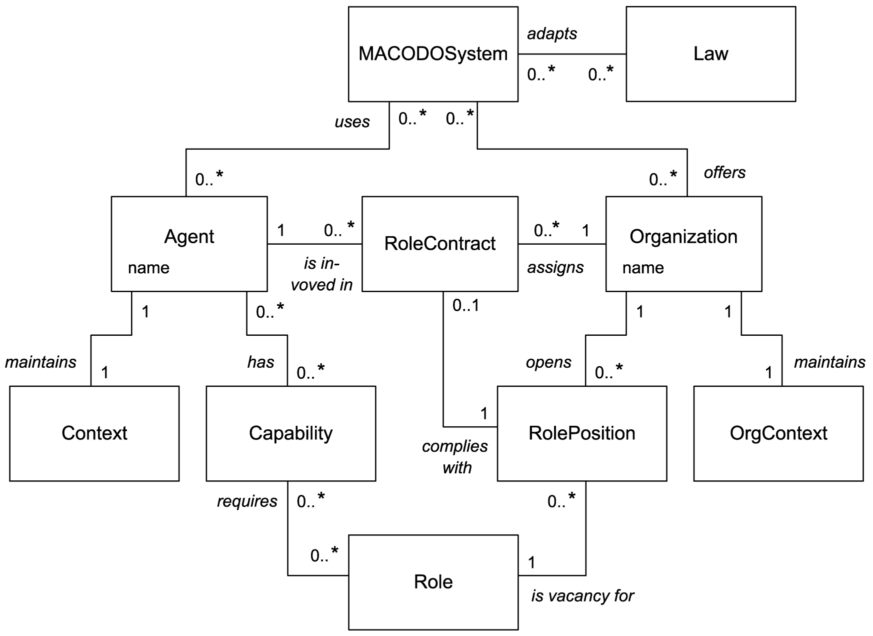


Fig. 2. MACODO organization model.

Context. *Context* represents information in the environment of an agent that is relevant for the organizations in which the agent participates. In this scenario, context includes the actual traffic state in the viewing range of the camera and the names of the agents on neighboring cameras. The traffic state has three possible values: free flow, bound flow, and congestion. In a free flow state, vehicles can drive at the maximum allowed speed. In a bound flow state this speed is limited. Finally, in a congested state, vehicles are standing still or can only drive at a minimum speed. For example, at time T1 in Figure 3, the traffic state monitored by camera $C2$ is free flow, while the state monitored by $C3$ is congestion. The neighbors of *agent*3 are *agent*2 and *agent*4.

Capability. A capability refers to ability of an agent to perform tasks. Capabilities describe required qualifications of an agent to participate in an organization. How an agent uses its capabilities to achieve its goals is an issue private to the agent. Examples of capabilities in the traffic monitoring case are *observe* (monitor the local traffic conditions), *aggregate* (integrate the traffic state of different cameras of an organization), and *present* (distribute information of traffic jams to a client).

Role. A *role* describes a coherent set of capabilities that are required to realize a functionality that is useful in an organization. In this scenario, each camera agent is capable of playing three different roles: data observer, data pusher, and data aggregator. An agent can play multiple roles simultaneously. The data observer role is responsible for monitoring the two traffic variables (density and average speed) and deciding whether the congestion level is high

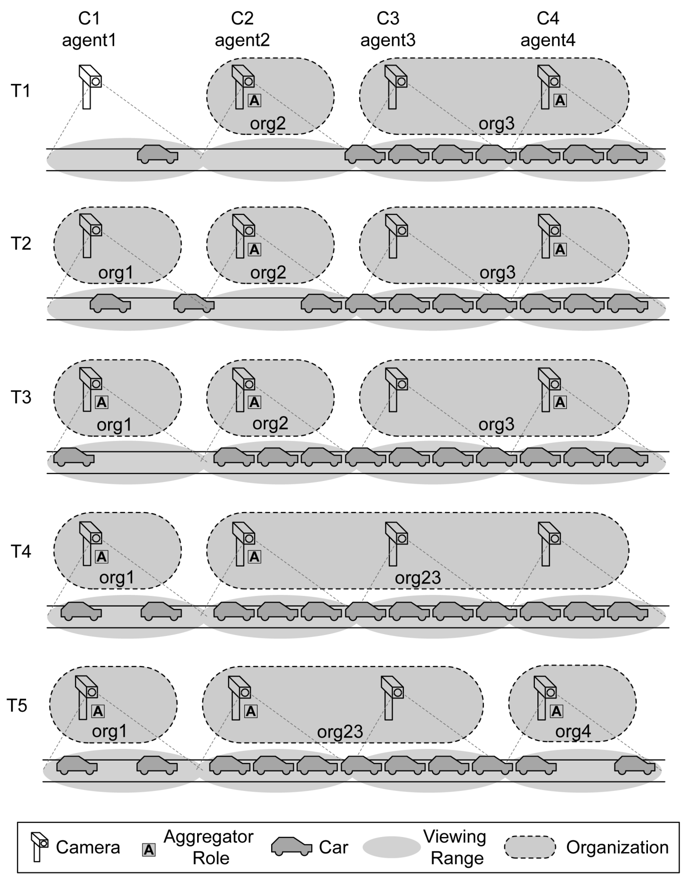


Fig. 3. Scenario to illustrate the basic abstractions of the MACODO organization model.

enough to be considered as a traffic jam. The data pusher role is responsible for pushing the observed data to the data aggregator role, which in turn is responsible for aggregating the data and distributing it to the interested clients. While there can be several agents playing the role of data observer and data pusher (on each camera in the organization), there can be at most one agent playing the role data aggregator in an organization.

Role Position. Organizations attract agents by means of *role positions*. A role position specifies a vacancy for a particular role in a particular organization. To start playing a particular role, the agent must have the required capabilities to play the role in that role position. When the agent's capabilities match the required capabilities of that role position, the agent gets a contract for that role position in the organization. At time T2 in the scenario, the organization *org*1 has an open role position for the data observer role. Later on, *agent*1 gets a role contract for this role position.

Role Contract. A *role contract* is a mutual agreement between the agent and the organization that allows the agent to play the role specified by the role contract in that organization. An agent playing a role in an organization receives services associated with the organization such as support for organization adaptations (see below). On the other hand, a role contract also implies responsibilities; for example, the agent has to share the relevant context for the organization (e.g. changes in the monitored traffic state) with the organization. As an example, at time T1, *agent*3 and *agent*4 have role contracts with organization *org*3 to play the role of data observer. Additionally, *agent*4 has a role contract to play the role of data aggregator (indicated by the boxed letter A).

Agent. *Agents* are the active entities that can play roles in organizations. A MACODO agent has a name, a set of capabilities, a context, and a set of role contracts. These are the social assets of a MACODO agent. Social assets comprise information that the agent uses to play the roles in its role contracts. The agent shares this information with the organizations in which it is involved. We make abstraction of the private structures of an agent which is out of the scope of the MACODO organization model. In the traffic monitoring application, a software agent is deployed on each camera. In this scenario, *agent*1 is deployed on camera *C*1, *agent*2 on camera *C*2, and so forth.

OrgContext. *Organization context* comprises domain-specific information that is relevant for organization dynamics. In this scenario, the organization context consists of the overall traffic state monitored by the cameras in the organization and the names of neighboring organizations. For example, at time T2, the traffic state of organization *org*1 is free-flow; the state of *org*3 is congested. The traffic state is free-flow, bound flow, or congested when all the agents in the organization monitor the corresponding traffic state. The traffic state of an organization is undefined when the actual traffic condition is not known. The traffic state is differentiated when different agents of the organization monitor different traffic conditions. Two traffic organizations are neighbors if there exist two agents (1) each with a role contract in one of the organizations, and (2) that are deployed on neighboring cameras. The neighbors are those organizations with which an organization may interact during organization dynamics. At time T1 in the scenario, the neighbor of organization *org*2 consists of the name of organization *org*3. However at time T2, the interaction candidates also include the name of organization *org*1.

Organization. An *organization* enables agents to collaborate. MACODO organizations have a name, an organization context, a set of role positions, and a set of role contracts. We refer to the actual role positions of an organization as *open* role positions. When an agent applies for an open role position and fulfils the conditions, we say that the role position is closed. A role contract is then created between the organization and the agent for that role position. A traffic organization in the scenario enables camera agents to collaborate in order to detect and monitor traffic jams. At time T1, Figure 3 shows two organizations. Organization *org*3 consists of *agent*3 and *agent*4, which collaborate to inform the interested clients of a traffic jam that spans the viewing range of cameras *C*3 and *C*4. Organization *org*2 consists of only *agent*2, which monitors the traffic in its viewing range.

Law. *Laws* describe the dynamic adaptation of organizations and define how a MACODO system maintains a consistent state. In MACODO, organization dynamics are directly or indirectly triggered by external events (e.g. an agent stops playing a role) and changes in the context of the agents in the organizations (e.g. the traffic state in the viewing range of an agent that collaborates in a traffic monitoring organization changes). There are different purposes why organizations should adapt, reflected in two types of laws. A first type of laws refers to intraorganization dynamics and basically describes how an agent can join and leave an organization dynamically, which is important for open organizations. A second type of laws describes interorganization dynamics, including merging and splitting of organizations. These laws support dynamic restructuring of a set of organizations, which is particularly relevant for collaborations in mobile systems where organizations of agents may get connected and disconnected dynamically. We illustrate both types of laws in the traffic monitoring scenario.

A join law defines how an agent can join an organization. For example, at time T2 in the scenario, *agent*1 starts a new organization *org*1. This new organization opens a role position for the role of data observer. Subsequently *agent*1 joins *org*1 by accepting this role position. After the join, the role position is closed and *agent*1 has a role contract with *org*1 allowing it to play the role of data observer in the organization. A leave law defines how an agent can leave an organization. A leave law terminates a role contract between the agent and the organization. Depending on the particular situation, the organization may or may not open a new role position for the role in the terminated role contract.

A merge law defines how two organizations can merge. For example, at time T3 in the scenario, the traffic state monitored by camera $C2$ becomes congested. As a result, at time T4, the two neighboring organizations, *org*2 and *org*3, merge together in a single organization *org*23. Since an organization has only one agent in the role of data aggregator, one of the contracts in this role will be terminated by the merge law. A split law defines how an organization is split in two organizations. At time T5, the traffic state in the viewing range of camera $C4$ is no longer congested. The split law will split off *org*4 from *org*23. The split law reorganizes the role contracts of the involved agents ensuring that there will be at most one agent with a contract in the role of data aggregator in both resulting organizations.

MACODO System. Finally, a *MACODO system* consists of a set of agents, a set of organizations, and a set of laws that comply to the MACODO organization model. The scenario shows a MACODO system for a traffic monitoring application consisting of four camera agents and a number of traffic organizations, with laws for joining, leaving, merging, and splitting organizations.

3. SOFTWARE ARCHITECTURE

In this section, we describe a software architecture that supports the MACODO organization model. The software architecture captures the essential building blocks of a distributed middleware platform to support the MACODO programming abstractions.

We start by explaining the functional and quality requirements that underpin the MACODO software architecture (Section 3.1). Then we describe the software architecture using several architectural views. We give an overview of the different layers of the system (Section 3.2). We zoom in on the organization middleware layer and explain interorganization and intraorganization dynamics (Section 3.3). We use the running scenario of the traffic monitoring application introduced in Section 2 as an illustrative case.

3.1 Requirements

We discuss the main functional and quality requirements that are the drivers for the MACODO software architecture.

3.1.1 *Functional Requirements.* Functionality is the ability of the system to perform the tasks for which it is intended. The main functional requirement of the organization middleware is that it should provide the functionality to support the MACODO organization model described in Section 2. This functionality concerns various aspects of organization life-cycle management, including:

—handling role positions and managing role contracts: which is necessary to enable agents to join organizations and to update contracts in case of reorganizations;
—enforcing laws; which includes checking the conditions when laws are applicable as well as enforcing the correct application of the laws;
—maintaining the organization context in a distributed setting.

3.1.2 *Quality Requirements.* *Quality* is the degree to which a system meets the nonfunctional requirements in the context of the required functionality. The main quality requirements that drive the MACODO software architecture are the following.

—Adaptability. The organization middleware should be capable of dynamically reorganizing organizations in response to particular changes in the environment.
—Robustness. The organization middleware should be robust to node failures.
—Scalability. The organization middleware should scale linearly with the size of the organizations in the system.
—Portability. It should be possible to add the organization middleware on top of existing middleware. The organization middleware should augment, rather than replace, existing middleware.

3.2 Layered View of a MACODO System

Figure 4 shows a layered view of the software architecture of a MACODO system.

The layering is strict: each layer is only allowed to use services offered by the layer directly beneath it.

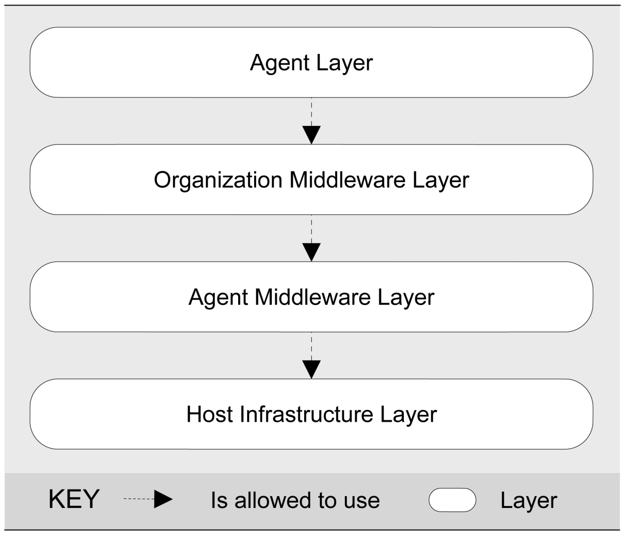


Fig. 4. Layered view of a MACODO system.

Elements. We give an overview of the different layers in a MACODO system.

—*Host Infrastructure Layer.* The host infrastructure encapsulates common middleware services and basic support for distribution, hiding the complexity of the underlying hardware.

—*Agent Middleware Layer.* The agent middleware layer provides basic services in multiagent systems. We used the model proposed in Weyns et al. [2007], which includes basic support for perception, action, and communication. Perception provides a service to agents for sensing the environment in which they are situated. Action provides a service to act in the environment. The communication service supports the exchange of messages in the distributed setting.

—*Organization Middleware Layer.* The organization middleware layer provides support for dynamic organizations. This layer encapsulates the management of dynamic evolution of organizations and it provides role-specific services to the agents for perception, action, and communication. The organization middleware layer is the main focus of the software architecture described in this section.

—*Agent Layer.* The agents in the agent layer use the organization middleware to interact with the environment and each other through the roles they play in the organizations.

Design Rationale. The key motivations for the layered architecture are the following.

—*Encapsulating the management of organizations.* One of the core architectural decisions is to separate the management of organizations from the functionality provided by the agents. The distinction between an Agent Layer and an Organization Middleware Layer defines this separation at an architectural level.

—*Portability of the Organization Middleware Layer.* The Organization Middleware is a layer that is separated from the underlying Agent Middleware. This increases the portability of the Organization Middleware across different basic middlewares.

3.3 Architecture of the Organization Middleware

The Organization Middleware offers support for the abstractions of the organization model. We describe the architecture of the organization middleware using several architecture views. We start by explaining the components and their interactions at one node (Section 3.3.1), and we zoom in on the component responsible for organization management (Section 3.3.2). Next, we elaborate on intra-organization dynamics (Section 3.3.3). Finally, we explain interorganization dynamics (Section 3.3.4).

3.3.1 *Component and Connector View of the Organization Middleware.*

Figure 5 shows a component and connector view of the organization middleware layer as deployed on each node. A node refers to a computer system that is connected with other nodes in a network. A node in the traffic monitoring system consists of an intelligent camera, comprising a camera sensor, a processing unit, a memory unit, and a communication unit. We say that a node is involved in an organization (and an organization is active on a node) if there is at least one agent deployed on that node with a role contract in the organization. However, one node can be involved in multiple organizations and each organization can be distributed over multiple nodes. For the traffic monitoring case the situation is more restricted since each node deploying one camera agent can only be involved in one organization, however, this organization may span multiple nodes.

Elements. We explain the responsibilities of the various components and their relationships. Central in the organization middleware are four local data repositories. The data in these repositories represent the key abstractions from the organization model. These repositories contain local data that is synchronized with data on other nodes in the following manner.

—*Organization Context Repository*. Organization context contains a partial representation of the organization state and interaction candidates of the organizations active on the node. For the traffic monitoring case, organization context includes the current traffic state the organization is monitoring, and the traffic state of the neighboring organizations (interaction candidates). The *ContextManagement* interface enables components to inspect and modify the content of the Organization Context Repository.

—*Role Positions Repository*. The Role Positions Repository contains a representation of the open role positions available to the local agents. The *PositionQuery* interface enables components to inspect the role positions in the Role Positions Repository. The *PositionManagement* interface enables components to inspect and modify the role positions in the repository.

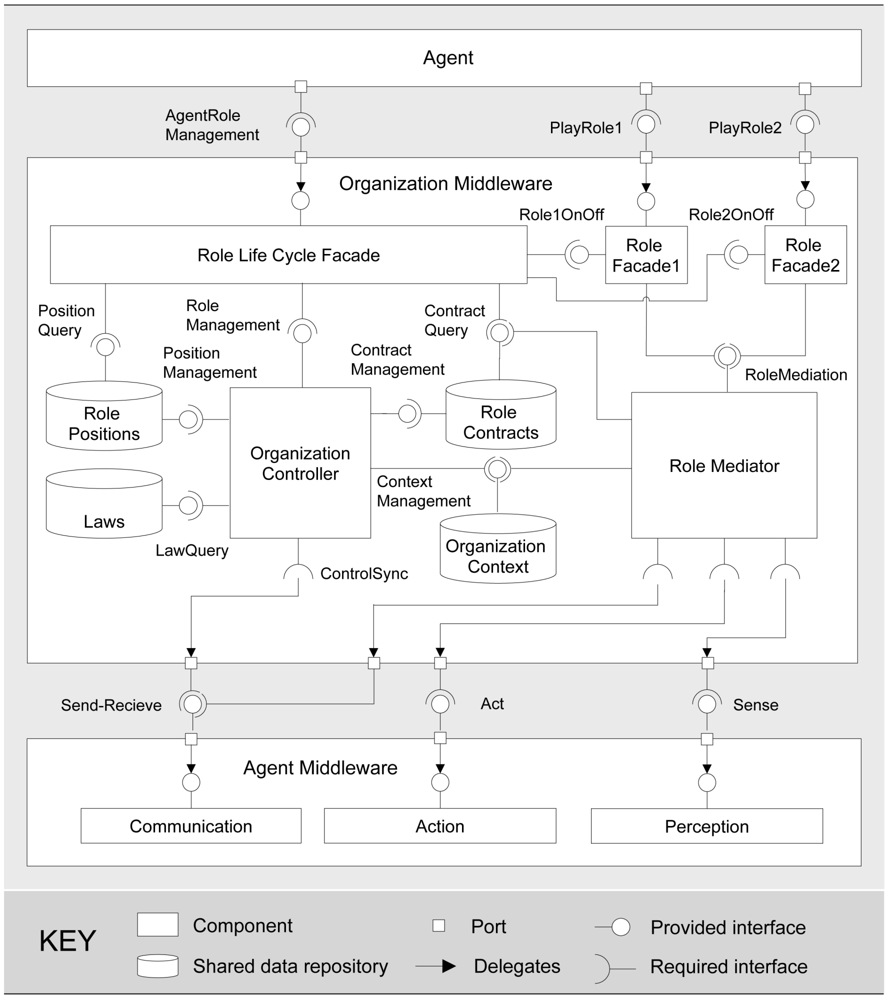


Fig. 5. Component and connector view of the organization middleware layer at one node.

—*Role Contracts Repository*. The Role Contracts Repository contains a representation of the current role contracts of the organizations active on the node. The *ContractQuery* interface supports inspecting the Role Contracts Repository, whereas the *ContractManagement* interface supports both inspecting and modifying the contracts in the repository.

—*Laws Repository*. The Laws Repository contains a description of the laws that manage intraorganization dynamics and interorganization dynamics. The *LawQuery* interface supports inspecting the laws.

Sharing the repositories allows coordination among the Organization Controller, Role Mediator, and the agents (via the Role Life Cycle Facade and Role Facade components). An organization itself comprises the combined state of role positions, role contracts and context stored at the repositories on all the nodes that are active in the organization.

Besides the repositories, the organization middleware comprises the following components.

—*Role Life Cycle Facade*. The Role Life Cycle Facade is responsible for the life cycle management of roles. Therefore, the Role Life Cycle Facade enables agents to browse and select open role positions in the local Role Positions repository using the *AgentRoleManagement* interface. As described in the organization model, the open role positions can be positions in an organization where the agent already has a role contract, or positions in an organization that the agent can join. When an agent selects a matching open role position, the Role Life Cycle Facade forwards the request to the Organization Controller, which removes the role position from the Role Positions Repository, makes a role contract for the role position for the agent, and links the new role contract with the organization by updating the Role Contracts Repository. Subsequently, the Role Life Cycle Facade is notified about the new contract and activates a Role Facade component for the role contract to enable the agent playing the role. When an agent stops playing a role, the Role Life Cycle Facade informs the Organization Controller, which terminates the role contract, after which the Role Life Cycle Facade deactivates the corresponding Role Facade component. Depending on the particular laws of the organization, the Organization Controller may open a new role position for the terminated role.

—*Role Facade*. A Role Facade provides a role-specific interface to an agent (*PlayRole*), allowing the agent to play the role in accordance with the capabilities of that role. The Role Facade can be used to perceive and act in the environment and interact with other agents playing roles in the organization. For example, in the traffic case a Role Facade for a data observer role offers an interface for perceiving the local traffic state, whereas a Role Facade for a data pusher offers an interface to transmit the perceived local traffic state to a data aggregator. The Role Facade delegates agent requests, such as the sending of messages to a specific role type, to the Role Mediator.

—*Role Mediator*. The Role Mediator mediates agent activities. Examples of mediation are sending a message to all agents that play a particular role in an organization, and enforcing agents to follow particular interaction patterns (protocols). To this end, the Role Mediator accesses the organization context and role contracts repositories.

—*Organization Controller*. Besides starting and terminating role contracts (see the discussion of the Role Life Cycle Facade), the Organization Controller component has two main responsibilities. On the one hand, the Organization Controller is responsible for enforcing the laws that define organization dynamics. On the other hand, the Organization Controller is responsible for dealing with distribution. We elaborate on the Organization Controller in the following section.

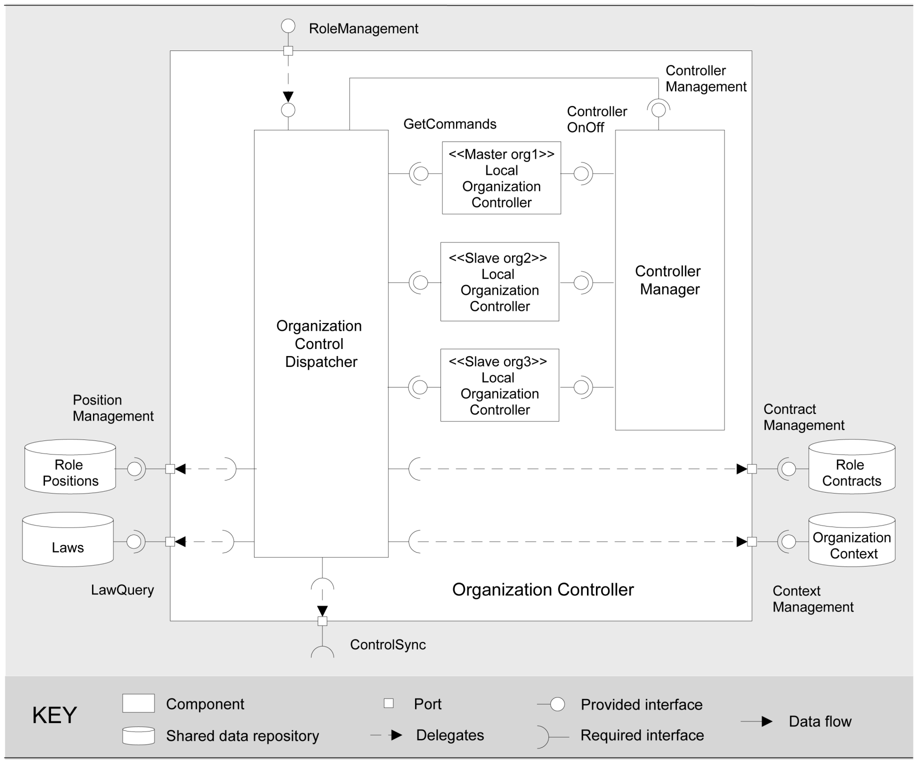


Fig. 6. Component and connector view of the organization controller.

Design Rationale. The architectural design of the organization middleware has the following motivations.

—*First-class role positions, role contracts, organization context, and laws.* The repositories of the organization middleware represent the key abstractions of the organization model as first-class citizens. This results in an easy to understand and modular design. Moreover, the repositories guarantee low coupling between the data accessors.

—*Uniform access the middleware services.* The Role Life Cycle Facade and Role Facades provide uniform interfaces to the agents to access the middleware services, according to the organization model we defined earlier. The facades hide the underlying logic of the organization middleware, which consists of the Organization Controller and Role Mediator, which are responsible for different concerns (managing organization dynamics and mediating agents actions respectively).

3.3.2 *Component and Connector View of the Organization Controller.* Figure 6 zooms in on the internal structure of the Organization Controller. An organization can span across a number of nodes. The *Organization Controllers* are responsible for enforcing, in a distributed setting, the laws that specify organization dynamics.

Elements. The Organization Controller comprises the following components.

—*Local Organization Controller*. A Local Organization Controller component encapsulates the local management strategy of a node with respect to a particular organization. For an organization that spans across multiple nodes, a Local Organization Controller component for that particular organization is present on each node active in that organization. On the other hand, if a node participates in multiple organizations, it maintains one Local Organization Controller component for each of the organizations it is involved in. All Local Organization Controller components of a particular organization need to collaborate to manage the organization in a distributed setting—to correctly enforce the laws (present in the Laws Repository) that specify the dynamics of that organization. We elaborate on the specific collaboration mechanism between the Local Organization Controllers in Section 3.3.3.

—*Organization Control Dispatcher*. The Organization Control Dispatcher is responsible for providing services to support the Local Organization Controllers. These services comprise communication services to enable Local Organization Controllers on different nodes to collaborate, as well as repository access services to enable Local Organization Controllers to manage the organization state (role positions, role contracts, organization context) present in the repositories. At regular times (e.g. on a periodic basis and/or when a particular message is received via the *ControlSync* interface or via the *RoleManagement* interface), the Organization Control Dispatcher dispatches control to the corresponding Local Organization Controller by invoking the *GetCommands* interface. The Local Organization Controller can then issue commands that invoke services offered by the Organization Control Dispatcher (reading or writing to the repositories or sending messages to organization controllers' other nodes).

—*Controller Manager*. The organizations that a particular node is active in, change over time. The Controller Manager is responsible for instantiating and terminating Local Organization Controller components accordingly.

Design Rationale. The main drivers for the design of the Organization Controller are the following.

—*Scalability.* The middleware has a basically decentralized structure, which supports scalability. For each node that is involved in an organization, a Local Organization Controller is instantiated. These controllers together are responsible for dealing with organization dynamics.

—*Adaptability of the synchronization strategy.* The design of the Organization Controller is independent of the strategy of Local Organization Controllers to manage consistency in a distributed setting. Since the Organization Controller makes no assumptions regarding a particular synchronization strategy, different strategies can easily be plugged in. In the next section we show how Organization Controllers use a master/slave strategy to manage organization dynamics. But any other strategy would have been possible.

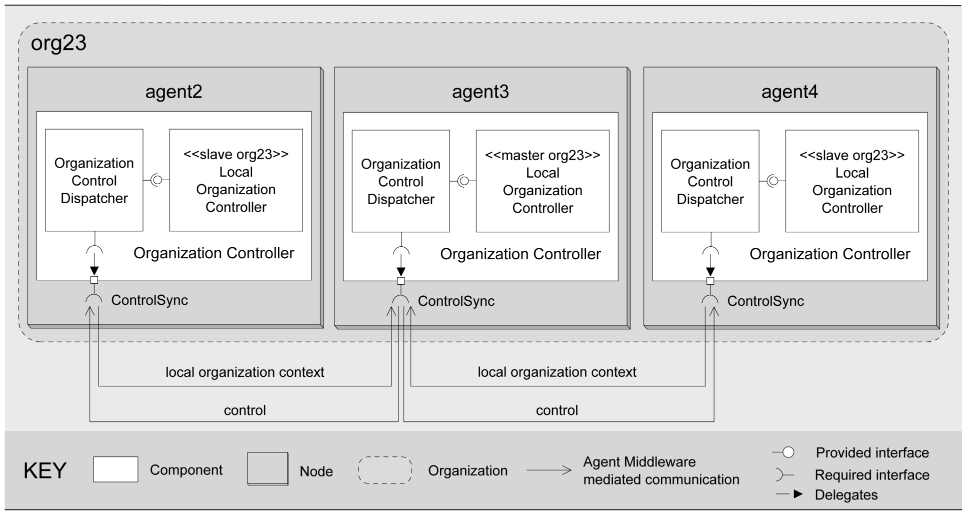

org23
agent2
agent3
agent4
Organization Control Dispatcher
<<slave org23>> Local Organization Controller
<<master org23>> Local Organization Controller
Organization Controller
ControlSync
local organization context
control
KEY
Component
Node
Organization
Agent Middleware mediated communication
Provided interface
Required interface
Delegates

Fig. 7. Master and slave node interaction between the controllers of organization *org*23 at T4 in the running scenario (see Figure 3).

3.3.3 *Intraorganization Dynamics.* To make the system scalable and robust, we have chosen a decentralized approach at the interorganization level (see Section 3.3.4). However, to simplify synchronization issues, we have used a master/slave principle at the intraorganization level. The master/slave principle is a control model, that locally centralizes the control of each organization and the state required for this control.

Figure 7 illustrates the collaboration strategy between the Local Organization Controllers at the intra-organization level. The Local Organization Controller components collaborate using a master/slave strategy. For each organization, one of the Local Organization Controllers is selected as master, whereas the other Local Organization Controllers of that organization are slaves. The master is responsible for managing the dynamics of that organization in a centralized manner, by synchronizing with all slaves.

The master is the central point of control of an organization and needs access to the state required for this control. The required state consists of the organization context, the current role positions and role contracts, data which is spread over the different repositories of all nodes active in the organization. The repositories of slaves only contain a partial representation of the organization context, role positions and role contracts of the organization. This partial representation consists of data that is generated by the local agents on the slave nodes of the organization. Master nodes of an organization, however, require a complete representation to manage the dynamics of the organization. Therefore, whenever the local state changes, slave controllers send their partial representation (*local organization context*) to their master node (see Figure 7). This allows the master nodes to maintain a complete representation of the organization state and context.

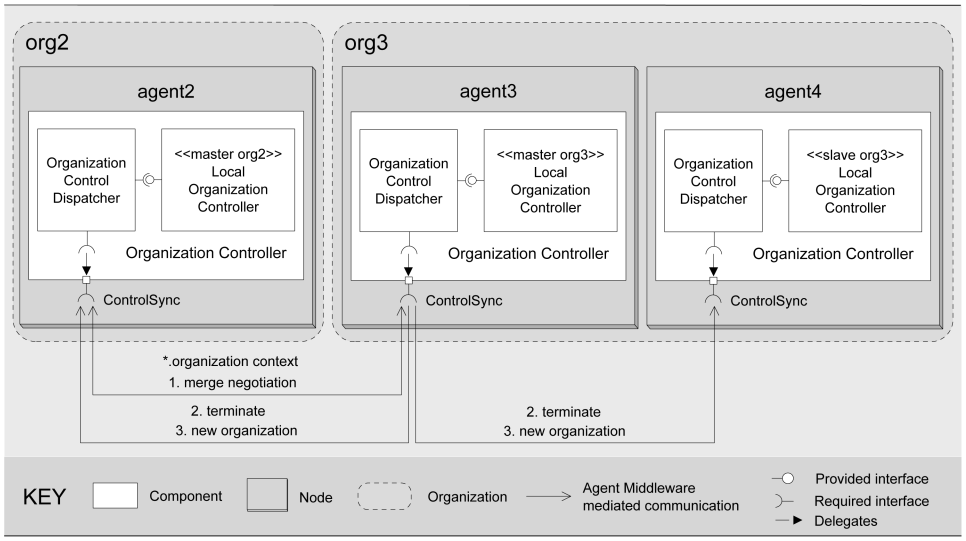


Fig. 8. Merging organizations at the architectural level (messages confirming the safe state have been omitted).

Each Organization Controller has access to a number of application-specific laws, contained in the *Laws* repository (see figure 5). As described in the organization model, these laws define when, for example, an agent joins or leaves an organization. The master controller uses these laws to handle organization dynamics based on a complete representation of the current organization context, role positions, and role contracts of the organization. This complete representation is accessible in the local repositories on the master node of the organization. When the master controller adapts an organization according to a law, it updates the organization context, role positions, and role contracts. The master propagates the changes to the slave nodes (*control*), as shown in Figure 7. In the next section, we explain how the master controller interacts with the slave controllers when the master/slave topology changes, due to interorganization dynamics.

3.3.4 *Interorganization Dynamics.* We now explain how laws are enforced that define interorganization dynamics, such as merging. Enforcing these laws requires information about more than one organization. Consequently, master Local Organization Controllers of multiple organizations need to collaborate. We use a concrete scenario from the traffic monitoring case in Figure 3 to explain how two organizations merge. At time T3, both organizations *org*2 and *org*3 are monitoring congestion, satisfying the conditions for the organizations to merge.

Figure 8 shows the master/slave nodes in the scenario at time T3, and the interactions between the controllers of the nodes involved in the merge. As explained in the Section 3.3.3, a master controller acquires a complete representation of its organization context, role positions, and role contracts, by regular updates of slave controllers sending their local representation of the organization context, role positions, and role contracts. From the complete representation of its organization, a master controller derives a reduced representation that is regularly exchanged with the master nodes of neighboring organizations (**.organization context*) (see Figure 8). In the case of the traffic example, this reduced representation contains the current traffic state of the organization.

In the scenario, at T3, the condition of the merge law is satisfied for both organizations. As a result, the master controller of both organizations will initiate a merge negotiation (*1. merge negotiation*). During this negotiation the controllers will select the new master of the merged organization.[2] In the scenario, the master controller of *org*3 is chosen as new master. Next, to ensure a safe state, the newly chosen master signals to terminate the current activities within both organizations (*2. terminate*). The safe state includes bringing all critical role interactions to a stop. After the safe state has been reached, the newly chosen master notifies all other nodes of the new organization (*3. new organization*). This information includes the identity of the new master, a list of the other nodes in the organization and the updated role positions and role contracts. The result of the merge in the scenario is shown at time T4 in Figure 3. The division of master/slave nodes resulting from the merge at T4 are shows in Figure 7.

3.4 Adding Robustness to Node Failures

One of the quality requirements of the MACODO software architecture is robustness to node failures. The software architecture as presented so far does not include support for dealing with node failures. In a realistic distributed setting, we must consider failures as an essential part of the dynamic environment. Failures will bring the system in an inconsistent state and probably disrupt its services. To make the system capable of dealing with failure dynamics, we have extended the system with a self-healing subsystem. For the architecture description of the self-healing subsystem we refer to the electronic appendix.

4. EVALUATION

To evaluate the validity of the middleware architecture, we used the MACODO organization model and software architecture to develop a prototype middleware platform for a traffic monitoring application. The evaluation focuses on three main quality requirements of the middleware architecture: adaptability, scalability, and robustness. Since we use a simulation for evaluation purposes, portability was evaluated.

In the first part, we evaluate the adaptability of the system. We use a Manhattan scenario and show that organizations dynamically merge and split according to the laws for organization dynamics. Since the main cost of organization dynamics is a communication overhead, we also look at the communication cost to manage the dynamics. In the second part, we evaluate scalability of the system. We study scalability with respect to the number of cameras deployed in the system and the traffic density. Our particular interest is in the communication

[2]The selection is based on a standard election protocol [Malpani et al. 2000].

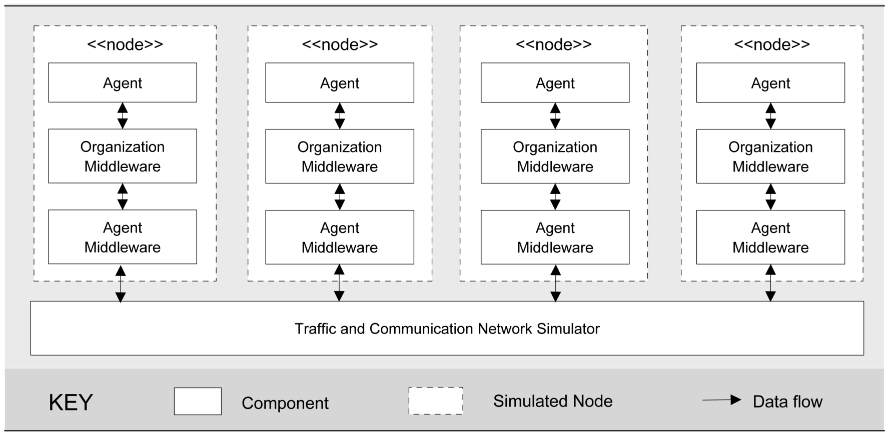


Fig. 9. Simulated camera nodes deployed on a traffic and communication network simulator.

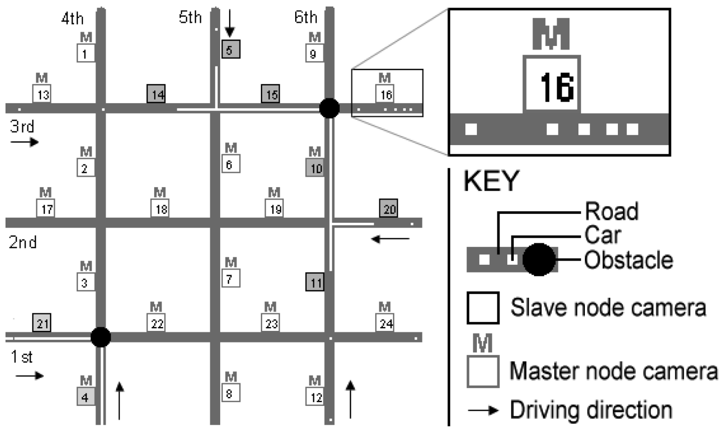


Fig. 10. Manhattan scenario.

cost since this is the main concern with respect to scalability in the middleware. For the evaluation of robustness, see the electronic appendix.

4.1 Experimental Setting

Before we go into the evaluation details, we explain the experimental setting of the evaluation. Figure 9 gives a high-level overview of the prototype application. In comparison with the layered architecture shown in Figure 4, the host infrastructure and the underlying hardware, including cars, cameras, and communication infrastructure are replaced by a simulator that is based on Bolay [1999]. The simulator consists of two parts: a traffic simulator and a communication network simulator.

The traffic simulator simulates the traffic situation on a single-lane road network and offers virtual cameras to observe the traffic. The simulator enables adapting to the car in-flow and adding and removing obstacles on the road to introduce congestion. A snapshot of this simulator is shown in Figure 10.

Roads are represented by grey bars, cars by white dots, and cameras by squares next to the roads. The M above a square represents a master node in an organization. A white camera is observing normal traffic and belongs to a single-member organization. Gray shaded cameras are observing congested traffic. Cameras with the same coloring belong to the same organization.

The communication network simulator simulates a LAN network. It allows introducing custom message delays and monitoring the communication traffic as a function of time. To get an approximation of the distributed system, nodes are implemented as separate threads.

4.2 Adaptability

In this section, we evaluate the adaptability of the system. We show how dynamic traffic situations in an urban setting lead to dynamic organizations that merge and split, and we look at the communication cost to manage the dynamics.

Dynamic Organizations. During this experiment, we use the Manhattan network, with unidirectional roads, as shown in Figure 11(a). The traffic direction of the roads is indicated by an arrow. Vehicles travel through the network in a random manner. A total of 24 cameras are evenly deployed along the road, each monitoring a part of the road network.

During the experiment, an obstacle first occurs on the crossing of the 1st and 4th streets, (see Figure 11(a)). As a result, a traffic jam arises in both streets. Camera 4 and camera 21 have spotted the congestion and are now in the same organization, with camera 4 as master node. Next, an obstacle occurs at the crossing of the 3rd and 6th streets (see Figure 11(b). Camera 10 and camera 15 have merged their organizations to monitor the resulting congestion. This congestion, however, grows further into the viewing ranges of other cameras, as shown in Figure 11(c). Corresponding to the merge law, the organization of camera 10 and camera 15 is expanded accordingly.

In Figure 11(d) the obstacle at the crossing of the 3rd and 6th streets has been removed. As a result, the congestion starts to dissolve in the front, while it continues to grow at the end. Camera 10 and camera 15, however, no longer observe any congestion. Based on the split law, the organization of the previous step is split up. Camera 10 and camera 15 are now each in a single-member organization, while camera 5 and camera 14, and camera 11 and camera 20 are in two separate organizations. Finally, Figure 11(e) shows how the congestion in the north east has completely vanished and only cameras 4 and 21 are still merged in one organization. This experiment shows how dynamic traffic situations lead to dynamic organizations, as described by the laws.

Communication Cost. Figure 12 shows the bandwidth usage during the previous experiment. The increased bandwidth usage in intervals A and B is caused by the merging of camera 4 and camera 21, and the merging of camera 10 and camera 15 (see Figures 11(a) and 11(b)). Since only two cameras were involved in each merge, the network overhead is limited. Interval D shows the communication cost for the merging of organizations that took place between

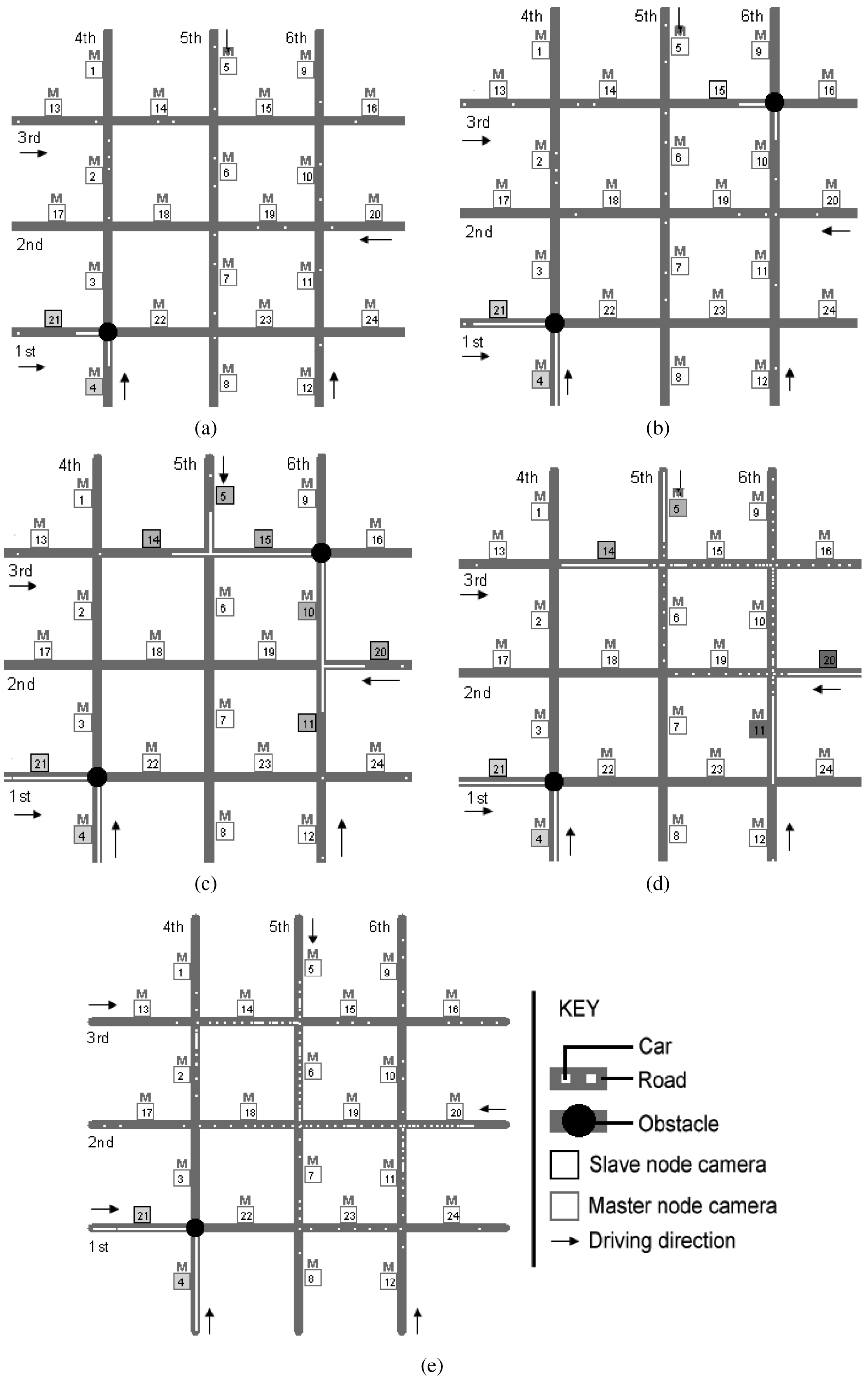


Fig. 11. Manhattan scenario.

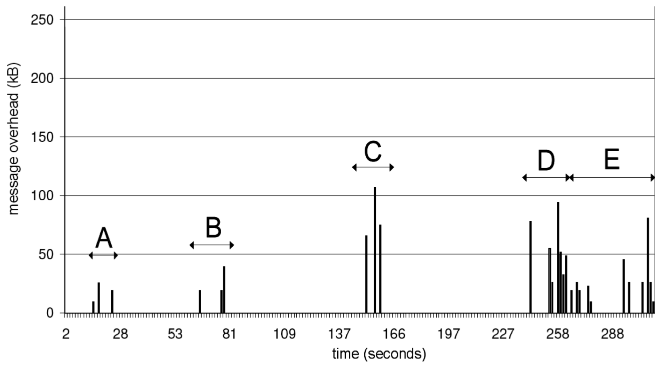


Fig. 12. Bandwidth usage in the Manhattan scenario.

Figures 11(b) and 11(c). The higher bandwidth usage is caused by the fact that more cameras were involved. During interval E, the obstacle on the crossing of the 3rd and 6th streets was removed. The increased network load is caused by the splitting of the organizations to reach the state shown in Figure 11(d). Finally, interval F shows the network usage to reach the situation shown in Figure 11(e). The measurements show that the bandwidth usage correlates with the changes in the traffic situations.

4.3 Scalability

To evaluate the scalability of the organization middleware we measure the communication costs for managing organization dynamics. Communication cost is an important criterion with respect to quality of service of a middleware and a determining factor with respect to scalability [Issarny et al. 2007]. We evaluate scalability with respect to the number of vehicles in the network (traffic density) and the number of deployed cameras.

Scalability and Traffic Density. Increasing traffic density leads to an increase in the number and size of congestions. As a result, more and larger organizations have to be merged and split. In this experiment we evaluate the effect of increasing density by measuring the network load to split and merge organizations as a function of the size of the organizations.

We use a Y-shaped intersection as shown in Figure 13(a). An obstacle is introduced on each lane. As a result, two traffic jams appear. Eventually the traffic jams merge at the intersection, as shown in Figure 13(b). Next, the obstacles in the two lanes are removed. As a result, the traffic jam starts to dissolve.

We measure the network load when the two organizations merge and when the merged organization splits after the obstacles are removed. Figure 14(a) shows the communication costs to merge the two organizations and Figure 14(b) shows the communication cost to split the merged organization after the obstacles are removed. The experiments are repeated for an increasing number of cameras of the initial organizations on both lanes. The results indicate

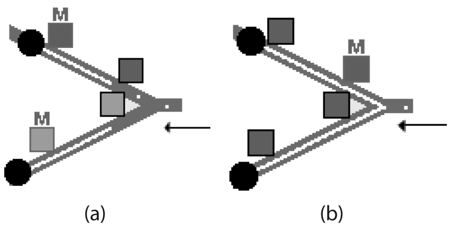
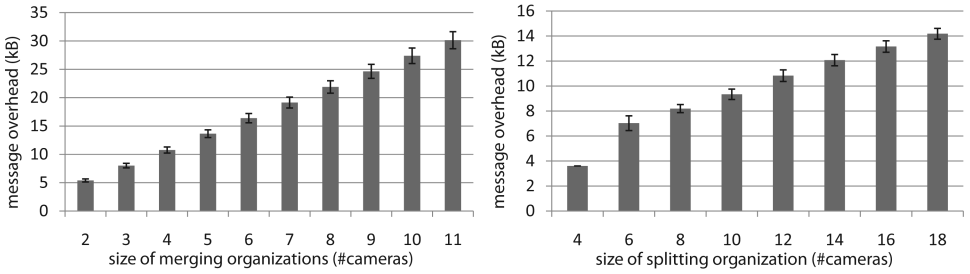


Fig. 13. Merge with 4 nodes.



(a) Network load for merging organizations (b) Network load for splitting organizations

Fig. 14. Effects of traffic density, showing the average of 30 measurements for each organization size and a 95% confidence interval.

that the network load scales linearly with respect to the size of the initial organizations.

Scalability and the Number of Cameras. In this experiment, we evaluate the influence of the number of cameras deployed in the road network on the total network load in three different scenarios. In scenario (a), Figure 15(a), seven cameras are deployed on a highway and an obstacle is introduced between camera 5 and camera 6. In the experiment, a stream of vehicles enters an empty highway, causing a traffic jam. When the traffic jam reaches the end of the highway, the obstacle is removed and the input of vehicles is stopped. The scenario ends when all vehicles have left the highway. Figure 16(a) shows the total bandwidth usage during this scenario.

In scenario (b), seven additional cameras are deployed in the road network, as shown in Figure 15(b). These cameras, however, are deployed to the right of the cameras of scenario (a). Everything else is kept the same as in scenario (a). Figure 16(b) shows the total bandwidth usage during this scenario. Comparing this result with the result of scenario (a), indicates a sublinear relation between the communication cost and the number of load-free cameras (not observing congestion) deployed on the road network.

In scenario (c), shown in Figure 15(c), an on-ramp is added to the road topology of scenario (b), generating a second stream of vehicles. Two obstacles are now added simultaneously, instead of one. Figure 16(c) shows the total bandwidth usage during this scenario, and also the bandwidth of cameras 1 to 6 and cameras 7 to 16. The results show that communication between cameras

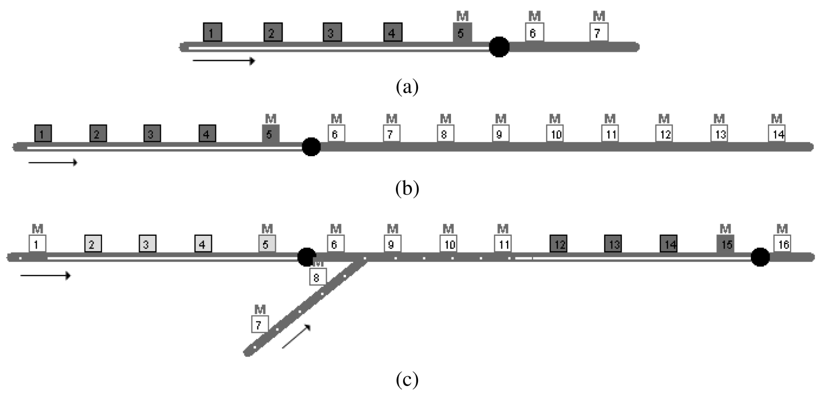


Fig. 15. Scalability scenarios.

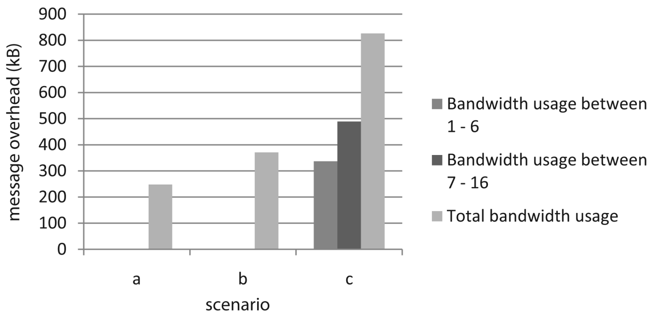


Fig. 16. Bandwidth use in scalability scenarios.

remains localized, indicating that the system scales well with respect to multiple traffic jams in different parts of the road network.

5. RELATED WORK

A vast body of research exists on roles and organizations for multiagent systems. Here we focus on two particular areas of interest. We start by looking at approaches that support organization dynamics. Then we discuss related work on managing organizations and their dynamics using organization infrastructures and middleware.

5.1 Approaches that Support Organization Dynamics

Dynamics are part of any realistic environment. To deal with dynamics, changes in the environment and requirements should be reflected in dynamic organizational structures. Several approaches have been proposed to support such dynamics, focusing at different stages in the life-cycle of multiagent systems, ranging from design and specification to analysis and concrete run-time support. We discuss a number of representative approaches.

Dignum et al. [2004] explore how and why organizations change and how reorganization can be done dynamically, with minimal interference from the system designer. A classification is given of reorganization situations, based on the focus of the reorganization, the authority to modify the organization, and the decisions taken to reorganize. With respect to the latter, two possible approaches are identified. First, the decision to change the organization can be the responsibility of one role in the organization. This corresponds to a master/slave relationship between agents and is called role-based control. Second, some roles are collectively responsible for a change decision. Changes are then achieved by collaboration among the agents. This is called shared control. In MACODO, the organization middleware initiates organization changes based on the dynamic contextual conditions.

AGR(E) models (Agent Group Role-Enviroment) [Ferber and Gutknecht 1998; Ferber et al. 2005] offer run-time support for dynamics at the organizational level. Although the the authors of AGR(E) are strong proponents of an organization-centered perspective on multiagent system design, an agent-centric perspective is taken on the dynamic evolution of groups. This is illustrated by MadKit [Gutknecht et al. 2001], a modular agent infrastructure based on the AGR organizational model. MadKit relies on specialized agents, called kernels, and system agents, for the management of groups, roles, and distribution. It is the responsibility of individual application agents to decide which group they want to join or leave and which role they want to play. This differs from our approach in which the dynamic evolution of organizations is actively managed and driven by the organization middleware instead of the agents themselves. Additionally, AGR(E) offers no explicit support to model evolutional changes at the interorganization level. Our model on the other hand allows modelling interorganizational evolution by means of laws.

McCallum et al. [2008] use the notion of influence to study change and dynamics of agent organizations. The authors argue, that if influence can be examined at the organizational level, instead of at the level of the agents involved, engineers can better understand an organization's robustness to structural, behavioral, and population changes. The authors present a formal model of organizational change (MOChA) as a means to specify and check the impact of influence on the operation of an organization. This formalization of influence is not specific to the presented model, and is adaptable to any organizational model in which explicit relationships among roles of agents are formed. As the MOChA model, the model presented in this article is concerned with organization dynamics. However, our model is targeted at a middleware architecture that supports the life-cycle management of dynamic organizations as a reusable service, clearly separated from the functionality of agents playing roles within these organizations.

In DeLoach et al. [2008], the authors propose a metamodel for multiagent organizations that defines the requisite knowledge of a system's organizational structure to reorganize at runtime, enabling it to achieve its goals effectively in the face of a changing environment. The reorganization is realized by the agents themselves. Their approach shares its runtime support for organizational evolution based on environmental change with the work presented in this article. The authors introduce the concept of a reorganization algorithm that has some similarities with the concept of law. Instead of taking an agent-centric view on the realization of the reorganization algorithms, the work presented in this article, relies on an organization middleware to realize the execution of laws.

5.2 Organization Infrastructures and Middleware

In traditional agent-centric approaches, agents are considered as autonomous entities pursuing individual goals. To guarantee the desired system behavior, a number of infrastructures and middleware approaches have been proposed that manage organizations according to predefined policies and norms. We discuss a number of representative examples.

In Omicini and Ricci [2003], a TuCSoN node serves as an organization node, hosting programmable tuplespaces (tuple centers) as coordination artifacts/services available to agents. Each node has a special tuple center that hosts the description of the organization, which can be dynamically inspected and changed by agents. This allows agents, for example, to add or delete roles from the organization. However, it is only possible to make changes on the intraorganizational level this way. Reorganizations on the interorganizational level are not explicitly supported. In Omicini et al. [2008], the authors present the agents & artifacts (A & A) metamodel. In this model, agents are the proactive entities in charge of achieving the system goals, whereas artifacts are the reactive entities encapsulating services that enable agents to work together. Artifacts provide an abstraction that could be used to design and implement a MACODO middleware platform.

TeamCore [Pynadath and Tambe 2003] provides an automated infrastructure that allows human operators to integrate heterogeneous teamwork between software agents and humans. TeamCore proxies encapsulate general teamwork functionality such as coordination actions, shielding human developers from low level coordination details. The concept of TeamCore proxy is similar to our concept of roles as first-class entities and the concept of role facade, encapsulating reusable organizational services. TeamCore also supports dynamic plan alteration. TeamCore proxies can make local decisions to use alternative predefined plans, based on the current capabilities of agents and humans.

AMELI [Esteva et al. 2004] proposes a domain-independent distributed infrastructure to mediate agent interactions in multiagent systems, enforcing institutional rules. The proposed infrastructure consists of three layers: an agent layer, a social or AMELI layer, and a communication layer. The authors refer to the AMELI layer as an agent-based middleware because this layer is realized using using four types of agents: institution managers, transition managers, scene managers, and governors. Governors act as access points to the infrastructure for external agents, which is somewhat similar to role facades.

ORA4MAS (Organizational Artifacts for Multi-Agent Systems) [Hübner et al. 2009] presents a middleware approach in which an organization infrastructure offers a set of organizational services. Access to these services is mediated by organizational proxies, which control and enforce organizational constraints. This mediation can be done by regimentation (preventive mechanism) or enforcement (reactive mechanism) of norms. Regimentation is realized by artifacts, while enforcement is detected by artifacts but evaluated by the agents. The basic middleware idea behind ORA4MAS is similar to the middleware approach presented in this article. In contrast to the work presented in this article, ORA4MAS does not provide explicit concepts to describe organizational evolution, such as laws.

Most of the approaches discussed in this section rely on a set of specialized agents to realize the middleware services, such as kernel agents in MadKit, managers and governors in AMELI, and organizational agents in ORA4MAS. Except for a high-level conceptual architecture, a comprehensive description of the software architecture of the infrastructure or middleware is often lacking.

6. CONCLUSIONS

In this article, we presented the MACODO middleware. The complementary part of the MACODO approach is an organization model that defines abstractions that support application developers in describing dynamic organizations, as in Weyns et al. [2010]. Contrary to most existing approaches in which agents have the dual responsibility of playing roles and managing organization dynamics, the MACODO middleware takes the burden of the life-cycle management of dynamic organizations. This separation of concerns promotes reuse, and makes the design and management of dynamic organizations in multiagent systems easier.

The contributions of this article are the following.

(1) The description of the MACODO software architecture. The software architecture describes the essential building blocks of a distributed middleware platform that supports the MACODO organization model.
(2) A prototype middleware platform based on MACODO software architecture for a traffic monitoring application. We used the prototype platform to evaluate the feasibility of the MACODO middleware architecture by demonstrating the adaptability, scalability and robustness of the prototype platform.

The main tracks for future research are: (1) to extend the prototype to a full-fledged reusable middleware platform that implements the MACODO software architecture and supports application developers with the MACODO abstractions defined in the organization model, and (2) to apply the MACODO approach in the domain of collaborative business processes, such as supply chain management and collaborative health care institutions [DiCoMas 2008].

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

REFERENCES

BOLAY, K. 1999. Cellular automaton traffic simulators based on the work of Nagel-Schreckenberg (1992), Takayasu (1993), Helbing and Schreckenberg (1999). http://rcs www.urz.tu-dresden.de/∼helbing/RoadApplet/.

DELOACH, S., OYENAN, W., AND MATSON, E. 2008. A capabilities-based model for adaptive organizations. *Auton. Agents Multi-Agent Syst. 16,* 1, 13–56.

DICOMAS. 2008. Distributed collaboration using multi-agent system architectures. http://distrinet.cs.kuleuven.be/projects/dicomas/.

DIGNUM, V., DIGNUM, F., AND SONENBERG, L. 2004. Towards dynamic reorganization of agent societies. In *Proceedings of the Workshop on Coordination in Emergent Agent Societies at ECAI*. 22–27.

ERTICO. 2008. Intelligent transportation systems for Europe. `http://www.ertico.com/`.

ESTEVA, M., ROSELL, B., RODRIGUEZ-AGUILAR, J., AND ARCOS, J. 2004. AMELI: An agent-based middleware for electronic institutions. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*. Vol. 1. IEEE Computer Society, 236–243.

FERBER, J. AND GUTKNECHT, O. 1998. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the 3rd International Conference on Multi-Agent Systems*. 128–135.

FERBER, J., MICHEL, F., AND BAEZ, J. 2005. AGRE: integrating environments with organizations. In *Proceedings of the 1st International Workshop on Environments for Multi-Agent Systems*. Lecture Notes in Computer Science, vol. 3374. Springer-Verlag, 48–56.

GARLAN, D., CHENG, S., HUANG, A., SCHMERL, B., AND STEENKISTE, P. 2004. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Comput. 37,* 10, 46–54.

GUTKNECHT, O., FERBER, J., AND MICHEL, F. 2001. Integrating tools and infrastructures for generic multi-agent systems. In *Proceedings of the 5th International Conference on Autonomous Agents*. ACM.

HIRSCHFELD, R., COSTANZA, P., AND NIERSTRASZ, O. 2008. Context-oriented programming. *J. Object Tech. 7,* 3, 125–151.

HÜBNER, J., BOISSIER, O., KITIO, R., AND RICCI, A. 2009. Instrumenting multi-agent organisations with organisational artifacts and agents. *Auton. Agents Multi-Agent Syst.* 1–32.

ISSARNY, V., CAPORUSCIO, M., AND GEORGANTAS, N. 2007. A perspective on the future of middleware-based software engineering. In *Proceedings of the Conference on the Future of Software Engineering* (*FOSE*). IEEE Computer Society. 244–258.

ITS. 2008. Intelligent Transportation Society of America. `http://www.itsa.org/`.

KENDALL, E. 2000. Role modeling for agent system analysis, design, and implementation. *IEEE Concurrency 8,* 2, 34–41.

KERNER, B. 2004. *The Physics of Traffic : Empirical Freeway Pattern Features, Engineering Applications, and Theory*. Springer, Berlin.

KRAMER, J. AND MAGEE, J. 2007. Self-managed systems: an architectural challenge. In *Proceedings of the International Conference on Software Engineering*. 259–268.

MALPANI, N., WELCH, J., AND VAIDYA, N. 2000. Leader election algorithms for mobile ad hoc networks. In *Discrete Algorithms and Methods for Mobile Computing and Communications*. ACM.

MCCALLUM, M., VASCONCELOS, W., AND NORMAN, T. 2008. Organisational change through influence. *Auton. Agents Multi-Agent Syst. 17,* 2, 1–33.

ODELL, J., PARUNAK, H. V. D., AND FLEISCHER, M. 2003. The role of roles. *J. Object Tech. 2,* 1, 39–51.

OMICINI, A. 2001. SODA: societies and infrastructures in the analysis and design of agent-based systems. In *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*. Lecture Notes in Computer Science, Vol. 1957. Springer-Verlag, 185–193.

OMICINI, A. AND RICCI, A. 2003. Reasoning about organisation: shaping the infrastructure. *AI* IA Notizie 16,* 2, 7–16.

OMICINI, A., RICCI, A., AND VIROLI, M. 2008. Artifacts in the A&A meta-model for multi-agent systems. *Auton. Agents Multi-Agent Syst.* Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems. *17,* 3, 432–456.

PYNADATH, D. AND TAMBE, M. 2003. An automated teamwork infrastructure for heterogeneous software agents and humans. *Auton. Agents Multi-Agent Syst. 7,* 1, 71–100.

SIMS, M., CORKILL, D., AND LESSER, V. 2008. Automated organization design for multi-agent systems. *Auton. Agents Multi-Agent Syst. 16,* 2, 151–185.

WEYNS, D., HEASEVOETS, R., AND HELLEBOOGH, A. 2010. The MACODO organization model for context-driven dynamic agent organizations. *ACM Trans. Auton. Adapt. Syst.* (To appear).

WEYNS, D., OMICINI, A., AND ODELL, J. 2007. Environment as a first-class abstraction in multiagent systems. *Auton. Agents Multi-Agent Syst. 14,* 1, 5–30.

ZAMBONELLI, F., JENNINGS, N., AND WOOLDRIDGE, M. 2003. Developing Multiagent Systems: The Gaia Methodology. *ACM Trans. Softw. Engin. Method. 12,* 3, 317–370.

Received May 2008; revised December 2008, June 2009; accepted December 2009