The MACODO Organization Model for Context-driven Dynamic Agent Organizations

DANNY WEYNS, ROBRECHT HAESEVOETS, and ALEXANDER HELLEBOOGH
DistriNet Labs, Katholieke Universiteit Leuven, Belgium

Today's distributed applications such as sensor networks, mobile multimedia applications, and intelligent transportation systems pose huge engineering challenges. Such systems often comprise different components that interact with each other as peers, as such forming a decentralized system. The system components and collaborations change over time, often in unanticipated ways. Multi-agent systems belong to a class of decentralized systems that are known for realizing qualities such as adaptability, robustness, and scalability in such environments. A typical way to structure and manage interactions among agents is by means of organizations. Existing approaches usually endow agents with a dual responsibility: on the one hand agents have to play roles providing the associated functionality in the organization, on the other hand agents are responsible for setting up organizations and managing organization dynamics. Engineering realistic multi-agent systems in which agents encapsulate this dual responsibility is a complex task.

In this paper, we present an organization model for context-driven dynamic agent organizations. The model defines abstractions that support application developers to describe dynamic organizations. The organization model is part of an integrated approach, called MACODO: Middleware Architecture for COntext-driven Dynamic agent Organizations. The complementary part of the MACODO approach is a middleware platform that supports the distributed execution of dynamic organizations specified using the abstractions, as described in [Weyns et al. 2009].

In the model, the life-cycle management of dynamic organizations is separated from the agents: organizations are first-class citizens, and their dynamics are governed by laws. The laws specify how changes in the system (e.g. an agent joins an organization) and changes in the context (e.g. information observed in the environment) lead to dynamic reorganizations. As such, the model makes it easier to understand and specify dynamic organizations in multi-agent systems, and promotes reusing the life-cycle management of dynamic organizations. The organization model is formally described to specify the semantics of the abstractions, and ensure its type safety. We apply the organization model to specify dynamic organizations for a traffic monitoring application.

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: Theory, Design

Additional Key Words and Phrases: Context, intelligent transportation systems, middleware, multi-agent system, organization, role

1. INTRODUCTION

Decentralized systems are typically used in environments that contain a lot of dynamism, and where resources have a high degree of physical distribution. Example application do-

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

mains are ad-hoc sensor networks, mobile multimedia applications, and intelligent transportation systems [Roman et al. 2004; Want 2005; Wischhof et al. 2005]. In such situations, central control can be difficult to achieve or even be unfeasible. A decentralized system realizes its goals by relying on local interactions between its components. In addition, adaptation is a key quality as changes in the environment in which the system is situated typically affect the nature and structure of interactions among system components. As such, interaction design is a major challenge for engineering decentralized systems.

Multi-agent systems belong to a class of decentralized systems that are known for realizing quality attributes such as adaptability, robustness and scalability in environments with a lot of dynamism and a high degree of resource distribution. A typical way to structure and manage interactions among agents is by means of organizations [Kendall 2000; Omicini 2001; Odell et al. 2003b; 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 intra-organization adaptations such as assigning and revoking roles (e.g. an agent that enters or leaves an e-market), and inter-organization 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 multi-agent system is a challenging task. Most of the existing work on organizations defines roles and organizations at the level of agents [Ferber and Gutknecht 1998; Dignum et al. 2004; Sims et al. 2008]. As such, agents are endowed with a dual responsibility: on the one hand agents have to play roles providing the associated functionality in the organization, on the other hand agents are responsible for setting up and managing the organizations, and for realizing dynamic reorganizations to deal with changes in the environment. Engineering realistic multi-agent systems in which agents encapsulate this dual responsibility is a complex task.

To support engineers of organization-oriented multi-agent systems, we present an organization model for context-driven dynamic agent organizations. The organization model is part of an integrated approach, called MACODO: Middleware Architecture for COntext-driven Dynamic agent Organizations. The complementary part of the MACODO approach is a middleware platform that supports the distributed execution of dynamic organizations specified using the abstractions, as described in [Weyns et al. 2009].

The organization model defines the MACODO abstractions that allow application developers to describe dynamic organizations. In the model, the life-cycle management of dynamic organizations is separated from the agents: organizations are first-class citizens, and their dynamics are governed by laws. The laws specify how changes in the system (e.g. an agent joins an organization) and changes in the context[1] (e.g. information observed in the environment) lead to dynamic reorganizations. As such, the model makes it easier to understand and specify dynamic organizations in multi-agent systems, and promotes reusing the life-cycle management of dynamic organizations. The model is formally described to specify the semantics of the abstractions, and ensure its type safety. We apply the organization model to specify dynamic organizations for a traffic monitoring application.

[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.

Overview. This paper is structured as follows. In section 2, we give an intuitive explanation of the basic abstractions of the MACODO organization model using a traffic monitoring scenario that we use as a running example in the paper. Subsequently, Section 3 gives the formal specification of the MACODO organization model and explains the semantics of the model abstractions. In section 4, we discuss related work. Finally, we draw conclusions in section 5.

2. BASIC ABSTRACTIONS OF THE MACODO ORGANIZATION MODEL

In this section, we give an intuitive explanation of the basic abstractions of the MACODO organization model. First, we introduce the traffic monitoring application that we use as a running example in the paper. Then we present a concrete scenario that we use to introduce the basic abstractions of the MACODO organization model and organization dynamics. The same scenario will be used to illustrate the formal specification of the MACODO organization model in section 3.

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 communication technology to improve traffic [ITS 2008; ERTICO 2008]. The system consists of a set of intelligent cameras which are 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 an optimal coverage of the highway with a minimum in overlap.

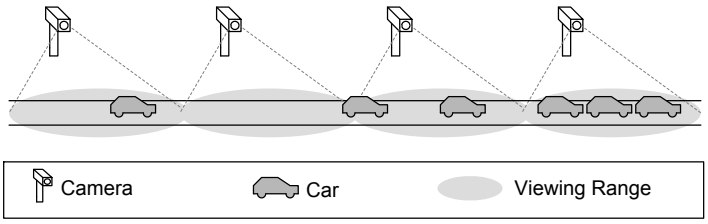

Fig. 1. An example of a highway with traffic cameras

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, i.e. 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 or not in the viewing range of a camera [Kerner 2004]. The congestion level has three possible values: free flow, bound flow, and congestion.

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, etc.

Traffic jams can span the viewing range of multiple cameras and can dynamically grow and dissolve. Because there is no central point of control, cameras have to collaborate and distribute the aggregated data to the clients. By default each camera monitors the traffic

within its viewing range. The traffic conditions in the viewing range make up its context. When a traffic jam occurs, the camera has to collaborate with other cameras detecting the same traffic jam. In the collaboration, the data each camera is monitoring is aggregated in order 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 report the traffic jam to the interested clients. Cameras will enter or leave the collaboration whenever the traffic jam enters or leaves their viewing range. Changes in the traffic conditions may require dynamic adaptations of multiple organizations. For example, when the traffic becomes congested in two neighboring organizations, these organizations have to merge into a single organization. Likewise, an organization has to split when the traffic jam is partially dissolved.

2.2 Basic Abstractions of the MACODO Organization Model

We introduce the basic abstractions of the MACODO organization model using the scenario shown in figure 2. New introduced abstractions are indicated in `teletype` font. `Agents` are the active entities in the system. In the traffic monitoring application, a software agent is deployed on each camera. In the scenario, *agent*1 is deployed on camera $C1$, *agent*2 on camera $C2$, etc. Each camera agent is capable of playing three different `roles`: data observer, data pusher, and data aggregator. An agent can play multiple roles simultaneously. A role describes a coherent set of `capabilities` that are required to realize a particular functionality that is useful in a collaboration. The data observer role is responsible for monitoring the two traffic variables (density and average speed) and deciding whether the congestion level is high 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, such as traffic lights, driver assistance systems, etc. 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.

An `organization` enables camera agents to collaborate in order to detect and monitor traffic jams. At time T1, figure 2 shows two organizations in the scenario. Organization *org*3 consists of *agent*3 and *agent*4 that collaborate to inform the interested clients of a traffic jam that spans the viewing range of cameras $C3$ and $C4$. Organization *org*2 consists of only *agent*2 that monitors the traffic in its viewing range.

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 the role position. When the agent's capabilities match with the required capabilities of the role position, the agent gets a `role contract` for the role position in the organization. A role contract is a mutual agreement between the agent and the organization that allows the agent to play the role of a 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 relevant information for the organization with the organization, e.g. changes in the monitored traffic state.

Organizations dynamically adapt based on the changing context of the highway. `Context` comprises information that can be observed in the environment that is relevant for organization dynamics; examples are the traffic state monitored by the cameras and the neighborhood of cameras that are involved in an organization. In MACODO, we refer to operations that change the composition of organizations as `laws`. Currently, the

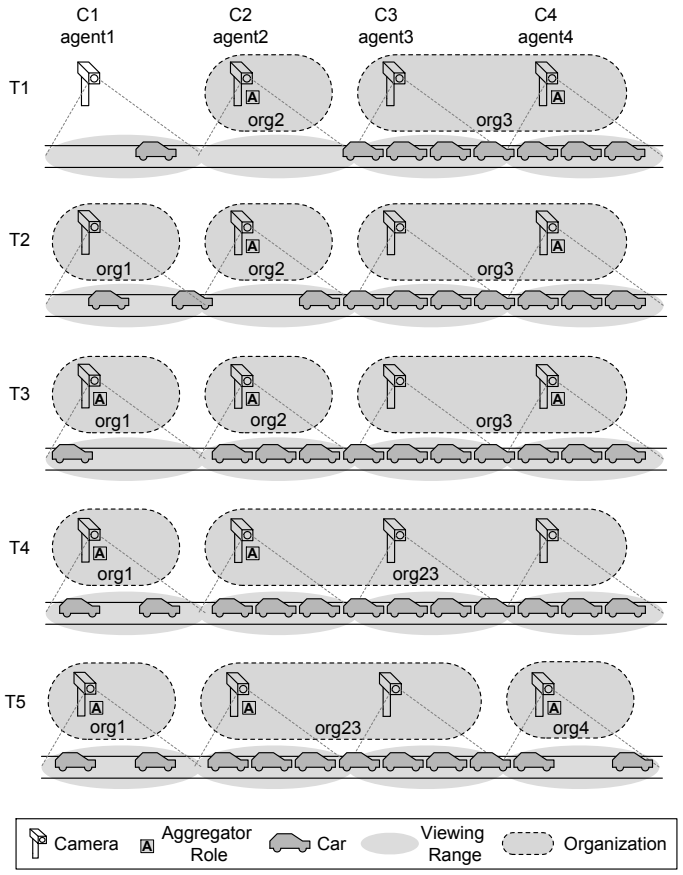

Fig. 2. Scenario to illustrate the basic abstractions of the MACODO organization model.

model supports laws for joining, leaving, merging and splitting organizations.

A join law defines how an agent can join an organization. For example, at time T2 in the scenario (figure 2), *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 neighbor 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.

Finally, we define a `MACODO system` as a system consisting of a set of agents and organizations that comply to the MACODO model. The scenario shows a MACODO system for a traffic monitoring application consisting of four camera agents and a number of traffic organizations.

3. FORMAL SPECIFICATION OF MACODO ORGANIZATION MODEL

In this section, we give a formal specification of the MACODO organization model. We use Z as a specification language. Z is a standardized, highly expressive formal language that is regularly used for describing and modeling computing systems, including multi-agent systems [d'Inverno and Luck 2004]. Z is an accessible formal language that is based on set theory and first order predicate calculus. We used the CZT tools [CZT 2008] to edit and type check the specification.

The formal specification of the MACODO organization model consists of two main parts. The first part introduces various sets and schemas to describe a data model that represents state in a MACODO system. The second part introduces a number of functions and operation schemas to describe laws that represents the behavior of a MACODO system[2]. The definitions of helper functions are omitted from the specification in this paper. For a specification of these functions, we refer to the electronic appendix.

3.1 Data Model

The data model consists of three parts. In the first part, we introduce basic sets for names, context, capabilities, and roles. The second part defines role positions, role contracts, and agents. In the third part, we define organization context, organizations and MACODO systems. In each part, we start with definitions. Then a specialization of the definitions for the traffic monitoring case follows. Finally, we give concrete examples in the running case.

3.1.1 *Names, Context, Capability, and Role.*

Definitions. We start by defining basic sets of names for agents and organizations. Each agent and organization in MACODO has a unique name that can be used to refer to a particular agent or organization.

$$[AGENTNAME, ORGNAME]$$

The context of an agent is defined as:

$$\begin{array}{|l} \hline CONTEXT\ [STATE] \\ state_a : STATE \\ interactcandidates_a : \mathbb{P}\, AGENTNAME \\ \hline \end{array}$$

Context represents information in the environment of an agent that is relevant for the organizations in which the agent participates. We describe context as a state schema with the name *CONTEXT*. Schemas model states as collections of state components. This schema has two state components: $state_a$ and $interactcandidates_a$. The $state_a$ component stores the actual state monitored by an agent in the environment. The type of the $state_a$ component is application-specific and represented by a generic parameter, indicated by [*STATE*] in the heading of the schema. The double bar indicates a generic schema definition. The generic

[2]We use the following name conventions: for set and schema names, all letters are uppercase; function names are all lower case. For the running case, names of schemas are CamilCase.

parameter allows the schema to be instantiated for different types of the $state_a$ component. The $interactcandidates_a$ component stores the names of agents with which an agent has a physical or logical relation in the environment, relevant to the organizations in which the agent participates. Note that component declarations are local to the schema. However, we can use the schema name to refer to its content.

We define a basic set for capabilities of agents:

$$[CAPABILITY]$$

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. We make abstraction of the concrete structure of capabilities.

A role is defined as a set of capabilities:

$$ROLE == \mathbb{P}\, CAPABILITY$$

A role describes a coherent set of capabilities that are required to realize a functionality that is useful in an organization.

Specializations. We now give specializations of the definitions for the traffic monitoring case. To represent context in the traffic monitoring system, we first define a type for traffic state:

$$TrafficState ::= undefined \mid freeflow \mid boundflow \mid congested \mid differentiated$$

The traffic state is undefined when the actual traffic condition is not known. Free flow, bound flow, and congested are regular traffic states. In a free flow state, vehicles can drive at the maximum allowed speed. In a bound flow state this speed is limited. Finally, and in a congested state, vehicles are standing still or can only drive at a minimum speed. The traffic state differentiated is only used by organizations and is explained below.

Context for traffic agents is defined as an instantiation of the *CONTEXT* schema with a domain-specific type *TrafficState*:

$$TrafficContext == CONTEXT[TrafficState]$$

The $state_a$ component now represents the actual state of the traffic in the viewing range of the camera. The $interactcandidates_a$ component represents the names of agents deployed on neighboring cameras.

Examples. We illustrate the definitions with concrete examples, based on the scenario shown in figure 2. The names of the agents and organizations in the scenario are:

$$cam_1, cam_2, cam_3, cam_4 : AGENTNAME$$
$$trorg_1, trorg_2, trorg_3, trorg_{23}, trorg_4 : ORGNAME$$

In the scenario, we consider the following agent capabilities:

$$observ, recogn, calc, aggr, push, present : CAPABILITY$$
$$trafficcapabilities : \mathbb{P}\, CAPABILITY$$
$$\hline$$
$$trafficcapabilities = \{observ, recogn, calc, aggr, push, present\}$$

We use an axiomatic definition to specify the capabilities. The part of the definition above the line declares the variables, the part below the line defines predicates that constrain the

variables. The predicate defines *trafficcapabilities* as the set of all the capabilities in the traffic monitoring case.

In the scenario, the following roles are defined:

$$dataobserver, datapusher, dataaggregator : ROLE$$

$$dataobserver = \{observ, recogn\}$$
$$datapusher = \{observ, calc, push\}$$
$$dataaggregator = \{observ, aggr, present\}$$

Each role consists of a set of agent capabilities that are required to play that role in a traffic monitoring organization.

At $T1$ in the scenario, the traffic context of the camera agents are:

$TrafficContext_{T1}$

$$cont_1, cont_2, cont_3, cont_4 : TrafficContext$$

$$(cont_1.state_a = undefined \land cont_1.interactcandidates_a = \{cam_2\})$$
$$(cont_2.state_a = freeflow \land cont_2.interactcandidates_a = \{cam_1, cam_3\})$$
$$(cont_3.state_a = congested \land cont_3.interactcandidates_a = \{cam_2, cam_4\})$$
$$(cont_4.state_a = congested \land cont_4.interactcandidates_a = \{cam_3\})$$

This schema contains two parts. Above the line, components are declared and the predicates below the line constrain their values. The index T1 in the schema name indicates that the state of the schema holds at time T1 in the scenario (see figure 2). The traffic context $cont_1$ is the context of $agent1$, i.e. the agent with name cam_1 (see figure 2), $cont_2$ is the context of $agent2$ with name cam_2, etc. We assume that at time T1, $agent1$ knows that $agent2$ is an interaction candidate (both agents are deployed on neighboring cameras), but $agent1$ does not know the actual traffic state in its viewing range.

3.1.2 *Role Position, Role Contract, and Agent.*

Definitions. A role position is a vacancy for a particular role in a particular organization. Role positions are defined as:

ROLEPOS

$$role : ROLE$$
$$orgname : ORGNAME$$

$$role \neq \varnothing$$

The predicate tells that each role position must contain at least one capability.

A role contract allows an agent to play a particular role in a particular organization. Role contracts are defined as:

ROLECONT

$$roleposition : ROLEPOS$$
$$agentname : AGENTNAME$$

Agents are defined as:

$$
\begin{array}{|l}
\hline
AGENT\ [STATE] \\
name_a : AGENTNAME \\
capabilities_a : \mathbb{P}\, CAPABILITY \\
context_a : CONTEXT[STATE] \\
rolecontracts_a : \mathbb{P}\, ROLECONT \\
\hline
capabilities_a \neq \varnothing \\
\forall\, rc : rolecontracts_a \bullet \\
\quad rc.agentname = name_a \land rc.roleposition.role \subseteq capabilities_a \\
\hline
\end{array}
$$

A name, a set of capabilities, a context, and a set of role contracts are social assets of a MACODO agent. The social assets comprise information that the agent uses to play the roles in its contracts. The agent shares this information with the organizations in which it is involved. The generic parameter of *CONTEXT* (indicated by [*STATE*] in the heading of the schema) allows the *AGENT* schema to be instantiated for different types of state. The predicates state that an agent must have at least one capability. Furthermore, all the role contracts in which the agent is involved have its name, and an agent can only play roles for which it has the required capabilities. We make abstraction of the private structures of an agent which is out of scope of the MACODO model.

We define a set of agents as:

$$
\begin{array}{|l}
\hline
AGENTS\ [STATE] \\
agents : \mathbb{P}\, AGENT[STATE] \\
\hline
\end{array}
$$

The *AGENTS* schema uses the same generic parameter for context as the *AGENT* schema.

Specializations. Agents in the traffic monitoring case are defined as an instantiation of the *AGENT* schema with a domain-specific type *TrafficState*:

$$CameraAgent == AGENT[TrafficState]$$

The set of camera agents is defined as:

$$CameraAgents == AGENTS[TrafficState]$$

Examples. The role positions in the scenario at time T1 are (see figure 2):

$$
\begin{array}{|l}
\hline
TrafficRolePositions_{T1} \\
rolepos_3, rolepos_4, rolepos_5, rolepos_6, rolepos_7, rolepos_8 : ROLEPOS \\
\hline
(rolepos_3.role = dataobserver \land rolepos_3.orgname = trorg_2) \\
(rolepos_4.role = dataaggregator \land rolepos_4.orgname = trorg_2) \\
(rolepos_5.role = dataobserver \land rolepos_5.orgname = trorg_3) \\
(rolepos_6.role = datapusher \land rolepos_6.orgname = trorg_3) \\
(rolepos_7.role = dataobserver \land rolepos_7.orgname = trorg_3) \\
(rolepos_8.role = dataaggregator \land rolepos_8.orgname = trorg_3) \\
\hline
\end{array}
$$

The role contracts at time T1 in the scenario are:

$$
\begin{array}{|l}
\hline
\textit{TrafficRoleContracts}_{T1} \\
\textit{TrafficRolePositions}_{T1} \\
\textit{rolecon}_3, \textit{rolecon}_4, \textit{rolecon}_5, \textit{rolecon}_6, \textit{rolecon}_7, \textit{rolecon}_8 : \textit{ROLECONT} \\
\hline
(\textit{rolecon}_3.\textit{roleposition} = \textit{rolepos}_3 \land \textit{rolecon}_3.\textit{agentname} = \textit{cam}_2) \\
(\textit{rolecon}_4.\textit{roleposition} = \textit{rolepos}_4 \land \textit{rolecon}_4.\textit{agentname} = \textit{cam}_2) \\
(\textit{rolecon}_5.\textit{roleposition} = \textit{rolepos}_5 \land \textit{rolecon}_5.\textit{agentname} = \textit{cam}_3) \\
(\textit{rolecon}_6.\textit{roleposition} = \textit{rolepos}_6 \land \textit{rolecon}_6.\textit{agentname} = \textit{cam}_3) \\
(\textit{rolecon}_7.\textit{roleposition} = \textit{rolepos}_7 \land \textit{rolecon}_7.\textit{agentname} = \textit{cam}_4) \\
(\textit{rolecon}_8.\textit{roleposition} = \textit{rolepos}_8 \land \textit{rolecon}_8.\textit{agentname} = \textit{cam}_4) \\
\hline
\end{array}
$$

The $\textit{TrafficRoleContracts}_{T1}$ schema includes the $\textit{TrafficRolePositions}_{T1}$ schema. This inclusion indicates that all the declarations and predicates in $\textit{TrafficRolePositions}_{T1}$ apply to $\textit{TrafficRoleContracts}_{T1}$ as well. As an example, the first role contract in the list ($\textit{rolecon}_3$) is an agreement between the organization with name $\textit{trorg}_2$ (see $\textit{rolepos}_3$ above) and the agent with name $\textit{cam}_2$. In this contract the agent plays the role of *dataobserver* (see $\textit{rolepos}_3$).

The state of the four camera agents at time T1 is defined as:

$$
\begin{array}{|l}
\hline
\textit{CameraAgents}_{T1} \\
\textit{CameraAgents} \\
\textit{TrafficContext}_{T1} \\
\textit{TrafficRoleContracts}_{T1} \\
\hline
\exists\, \textit{agent}1, \textit{agent}2, \textit{agent}3, \textit{agent}4 : \textit{CameraAgent} \bullet \\
\quad \textit{agents} = \{\textit{agent}1, \textit{agent}2, \textit{agent}3, \textit{agent}4\} \land \\
\textit{agent}1.\textit{name}_a = \textit{cam}_1 \land \textit{agent}1.\textit{rolecontracts}_a = \varnothing \land \\
\textit{agent}1.\textit{context}_a = \textit{cont}_1 \land \textit{agent}1.\textit{capabilities}_a = \textit{trafficcapabilities} \land \\
\textit{agent}2.\textit{name}_a = \textit{cam}_2 \land \textit{agent}2.\textit{rolecontracts}_a = \{\textit{rolecon}_3, \textit{rolecon}_4\} \land \\
\textit{agent}2.\textit{context}_a = \textit{cont}_2 \land \textit{agent}2.\textit{capabilities}_a = \textit{trafficcapabilities} \land \\
\textit{agent}3.\textit{name}_a = \textit{cam}_3 \land \textit{agent}3.\textit{rolecontracts}_a = \{\textit{rolecon}_5, \textit{rolecon}_6\} \land \\
\textit{agent}3.\textit{context}_a = \textit{cont}_3 \land \textit{agent}3.\textit{capabilities}_a = \textit{trafficcapabilities} \land \\
\textit{agent}4.\textit{name}_a = \textit{cam}_4 \land \textit{agent}4.\textit{rolecontracts}_a = \{\textit{rolecon}_7, \textit{rolecon}_8\} \land \\
\textit{agent}4.\textit{context}_a = \textit{cont}_4 \land \textit{agent}4.\textit{capabilities}_a = \textit{trafficcapabilities} \\
\hline
\end{array}
$$

The *agent1* is not involved in any organization yet. The *agent2* has a role contract for *dataobserver* and *dataaggregator* in the organization with the name $\textit{trorg}_2$. The *agent3* has a contract for *dataobserver* and *datapusher*, and *agent4* has contracts for *dataobserver* and *dataaggregator* both in the same organization with the name $\textit{trorg}_3$. Each of the four agents has the complete set of capabilities in the traffic monitoring application.

3.1.3 *Organization Context, Organization, MACODO System.*

Definitions. Organization context represents all the relevant information upon which the dynamics of the organization depend. Organization context includes the organization state and a set of organization interaction candidates:

$$
\begin{array}{|l}
\hline
\textit{ORGCONTEXT}\ [\textit{ORGSTATE}] \\
\hline
\textit{orgstate}_o : \textit{ORGSTATE} \\
\textit{interactcandidates}_o : \mathbb{P}\, \textit{ORGNAME} \\
\hline
\end{array}
$$

Organization state contains domain-specific information that is relevant for organization dynamics. As such, we make abstraction of the internal structure of organization state and use a generic parameter for the organization state in further definitions. Interaction candidates are the organizations with which an organization may interact during organization dynamics. We give a concrete example below when we discuss the merge of two organizations.

Organizations are defined as:

$$
\begin{array}{|l}
\hline
ORG\ [ORGSTATE] \\
name_o : ORGNAME \\
rolepositions_o : \mathbb{P}\, ROLEPOS \\
rolecontracts_o : \mathbb{P}\, ROLECONT \\
orgcontext_o : ORGCONTEXT[ORGSTATE] \\
\hline
\forall\, rp : rolepositions_o \bullet rp.orgname = name_o \\
\forall\, rc : rolecontracts_o \bullet rc.roleposition.orgname = name_o \\
\hline
\end{array}
$$

The predicates state that all role positions and role contracts in which an organization is involved have its name. 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 assigned to the organization and the agent for that role position (we formalize an agent that joins an organization in section 3.2).

A set of organizations is defined as:

$$
\begin{array}{|l}
\hline
ORGS\ [ORGSTATE] \\
organizations : \mathbb{P}\, ORG[ORGSTATE] \\
\hline
\end{array}
$$

Finally, MACODO systems are defined as:

$$
\begin{array}{|l}
\hline
MACODOSYSTEM\ [STATE, ORGSTATE] \\
AGENTS[STATE] \\
ORGS[ORGSTATE] \\
uniquenames_o : \mathbb{P}\, ORGNAME \\
\hline
\forall\, a1, a2 : agents \bullet a1.name_a = a2.name_a \Leftrightarrow a1 = a2 \\
\forall\, o1, o2 : organizations \bullet o1.name_o = o2.name_o \Leftrightarrow o1 = o2 \\
uniquenames_o = \{n : ORGNAME \mid \forall\, o : organizations \bullet n \neq o.name_o\} \\
\hline
\end{array}
$$

A *MACODOSYSTEM* consists of a set of agents and a set of organizations. It has two generic parameters which allows the instantiation of MACODO systems for domain-specific types of context and organization state. The agents and organizations in a MACODO system have unique names. $uniquenames_o$ provides a set of unique names for future organizations that will participate in the MACODO system.

Specialization. Context for traffic organizations is defined as an instantiation of the *ORGCONTEXT* schema with a domain-specific type of organization state, namely *TrafficState*:

$$\mathit{TrafficOrgContext} == \mathit{ORGCONTEXT}[\mathit{TrafficState}]$$

The organization state component of *TrafficOrgContext* represents the overall traffic state monitored by the cameras in the organization. The traffic state of an organization is undefined when the actual traffic condition is not known. 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 is differentiated when different agents of the organization monitor different traffic conditions.

The interaction candidate component of *TrafficOrgContext* represents the names of the neighboring organizations. We formally define organization neighborhood in the traffic monitoring case below when we specify the *TrafficMacodoSystem* schema.

A traffic organization is defined as an instantiation of the *ORG* schema with a domain-specific type of organization state, namely *TrafficState*:

$$\mathit{TrafficOrg} == \mathit{ORG}[\mathit{TrafficState}]$$

A set of traffic organizations is defined as:

$$\mathit{TrafficOrgs} == \mathit{ORGS}[\mathit{TrafficState}]$$

A traffic MACODO system is then defined as:

TrafficMacodoSystem

$$\mathit{MACODOSYSTEM}[\mathit{TrafficState}, \mathit{TrafficState}]$$

$$\begin{array}{l}
\forall\, o1, o2 : \mathit{organizations} \bullet \\
\quad o1 \neq o2 \land \\
\quad o2.\mathit{name}_o \in o1.\mathit{orgcontext}_o.\mathit{interactcandidates}_o \Leftrightarrow \\
\qquad o1.\mathit{name}_o \in o2.\mathit{orgcontext}_o.\mathit{interactcandidates}_o \land \\
\qquad\quad \exists\, a1, a2 : \mathit{agents} \bullet a1\ \mathit{activein}\ o1 \land a2\ \mathit{activein}\ o2 \land \\
\qquad\qquad a2.\mathit{name}_a \in a1.\mathit{context}_a.\mathit{interactcandidates}_a \land \\
\qquad\qquad a1.\mathit{name}_a \in a2.\mathit{context}_a.\mathit{interactcandidates}_a \\
\forall\, a : \mathit{agents} \bullet a.\mathit{name}_a \notin a.\mathit{context}_a.\mathit{interactcandidates}_a \\
\forall\, o : \mathit{organizations};\ rc : \mathit{ROLECONT} \mid rc \in o.\mathit{rolecontracts}_o \bullet \\
\quad \#\,(\{rp : o.\mathit{rolepositions}_o \mid rp.\mathit{role} = \mathit{dataaggregator}\} \cup \\
\qquad \{rp : \mathit{ROLEPOS} \mid rp = rc.\mathit{roleposition} \land \\
\qquad\quad rp.\mathit{role} = \mathit{dataaggregator}\}) \leq 1
\end{array}$$

The *TrafficMacodoSystem* schema includes the *MACODOSYSTEM* schema. We use *TrafficState* as case-specific types for the two generic parameters of the *MACODOSYSTEM* schema. Comparing to a *MACODOSYSTEM*, a *TrafficMacodoSystem* defines an organizational neighborhood and introduces a number of additional constraints for camera agents and traffic organizations. The first predicate says that if two organizations are neighbors there exists an agent with a role contract in each organization that are neighbors of each other. The *activein* function (omitted from the specification) is a helper function that defines whether an agent has a role contract in an organization. The second predicate says a camera agent cannot be a neighbor of itself. The third predicate says in each traffic organization there can only be one role contract or one open role position for the *dataaggregator* role.

Examples. At time T1 (see figure 2) there are two traffic organizations in the scenario:

$$
\begin{array}{|l}
\hline
\textit{TrafficOrgs}_{T1} \\
\textit{TrafficOrgs} \\
\textit{TrafficRoleContracts}_{T1} \\
\hline
\exists\, org2, org3 : \textit{TrafficOrg} \bullet \textit{organizations} = \{org2, org3\} \wedge \\
org2.\textit{name}_o = \textit{trorg}_2 \wedge \\
\quad org2.\textit{rolepositions}_o = \varnothing \wedge \\
\quad org2.\textit{rolecontracts}_o = \{\textit{rolecon}_3, \textit{rolecon}_4\} \wedge \\
\quad org2.\textit{orgcontext}_o.\textit{orgstate}_o = \textit{freeflow} \wedge \\
\quad org2.\textit{orgcontext}_o.\textit{interactcandidates}_o = \{\textit{trorg}_3\} \wedge \\
org3.\textit{name}_o = \textit{trorg}_3 \wedge \\
\quad org3.\textit{rolepositions}_o = \varnothing \wedge \\
\quad org3.\textit{rolecontracts}_o = \{\textit{rolecon}_5, \textit{rolecon}_6, \textit{rolecon}_7, \textit{rolecon}_8\} \wedge \\
\quad org3.\textit{orgcontext}_o.\textit{orgstate}_o = \textit{congested} \wedge \\
\quad org3.\textit{orgcontext}_o.\textit{interactcandidates}_o = \{\textit{trorg}_2\} \\
\hline
\end{array}
$$

Organization *org2* has no open role positions and shares two role contracts with *agent2* (see $\textit{TrafficRoleContracts}_{T1}$). The traffic state of *org2* is *freeflow*, and the organization with name $\textit{trorg}_3$ is an interaction candidate of *org2*. Organization *org3* has no open role positions and shares two role contracts with *agent3* and two contracts with *agent4* (see $\textit{TrafficRoleContracts}_{T1}$). Its traffic state is *congested*, and the organization with name $\textit{trorg}_2$ is an interaction candidate of *org3*.

The state of the *TrafficMacodoSystem* at time T1 is:

$$
\begin{array}{|l}
\hline
\textit{TrafficMacodoSystem}_{T1} \\
\textit{TrafficMacodoSystem} \\
\textit{CameraAgents}_{T1} \\
\textit{TrafficOrgs}_{T1} \\
\hline
\end{array}
$$

The state of the traffic monitoring system at time T1 includes the state of the camera agents at time T1 which includes the state of the four agents in the scenario as shown in figure 2, and the state of the two organizations, $org2$ and $org3$ at time T1.

3.2 Laws

We now shift our attention to laws that represent behavior of a MACODO system. 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 set of laws refer to intra-organization dynamics and basically describe how an agent can join and leave an organization dynamically, which is important for open organizations. A second set of laws describe inter-organization 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.

In this paper, we define two laws: one for inter-organization dynamics and one for intra-

organization dynamics respectively. A join law describes how agents can join organizations, and a merge law describes how two organizations can merge. For an overview of other laws, we refer to [Haesevoets et al. 2008]. A law is defined as an operation schema that changes the state of *MACODOSYSTEM*. For each law, we start with definitions, followed by specializations of these definitions for the traffic monitoring case, and a concrete example in the scenario.

3.2.1 *Join Law.* Agents collaborate in organizations by playing roles. The participation of an agent in an organization is represented by a role contract. As the environment of an agent changes, agents may want to adapt their participation in the existing organizations. This can be done by joining or leaving an organization. A join allows an agent to start a role contract for an open role position in an organization. We specify the mechanism of a join in a join law.

Definitions. We define a law for an agent joining an organization:

$$
\begin{array}{l}
\textit{JOINORG}\,[\textit{STATE}, \textit{ORGSTATE}] \\
\hline
\Delta \textit{MACODOSYSTEM}[\textit{STATE}, \textit{ORGSTATE}] \\
\textit{orgupdates}_{\textit{join}}[\textit{STATE}, \textit{ORGSTATE}] \\
\textit{org?} : \textit{ORG}[\textit{ORGSTATE}] \\
\textit{agent?} : \textit{AGENT} \\
\textit{rc?} : \textit{ROLECONT} \\
\hline
\textit{agent?} \in \textit{agents} \land \textit{org?} \in \textit{organizations} \\
\textit{rc?}.\textit{roleposition} \in \textit{org?}.\textit{rolepositions}_o \\
\textit{rc?}.\textit{agentname} = \textit{agent?}.\textit{name}_a \\
\textit{rc?}.\textit{roleposition}.\textit{role} \subseteq \textit{agent?}.\textit{capabilities}_a \\
\exists\, \textit{jorg} : \textit{ORG} \bullet \\
\quad \textit{jorg}.\textit{name}_o = \textit{org?}.\textit{name}_o \land \\
\quad \textit{jorg}.\textit{rolepositions}_o = \textit{org?}.\textit{rolepositions}_o \setminus \{\textit{rc?}.\textit{roleposition}\} \land \\
\quad \textit{jorg}.\textit{rolecontracts}_o = \textit{org?}.\textit{rolecontracts}_o \cup \{\textit{rc?}\} \land \\
\quad \textit{jorg}.\textit{orgcontext}_o = \textit{updateorgcontext}_{\textit{join}}(\textit{org?}.\textit{orgcontext}_o, \textit{agent?}.\textit{context}_a) \land \\
\exists\, \textit{jagent} : \textit{AGENT} \bullet \\
\quad \textit{jagent}.\textit{name}_a = \textit{agent?}.\textit{name}_a \land \\
\quad \textit{jagent}.\textit{rolecontracts}_a = \textit{agent?}.\textit{rolecontracts}_a \cup \{\textit{rc?}\} \land \\
\quad \textit{jagent}.\textit{context}_a = \textit{agent?}.\textit{context}_a \land \\
\quad \textit{jagent}.\textit{capabilities}_a = \textit{agent?}.\textit{capabilities}_a \land \\
\exists\, \textit{interactcandidates}, \textit{jinteractcandidates} : \mathbb{P}\, \textit{ORG} \bullet \\
\quad \textit{interactcandidates} = \{\textit{ic} : \textit{organizations} \mid \\
\quad\quad \textit{ic}.\textit{name}_o \in \textit{jorg}.\textit{orgcontext}_o.\textit{interactcandidates}_o\} \land \\
\quad \textit{jinteractcandidates} = \\
\quad\quad \textit{updateinteractcandidates}_{\textit{join}}(\textit{interactcandidates}, \textit{jorg}.\textit{name}_o) \land \\
\textit{organizations}' = \textit{organizations} \setminus \{\textit{org?}\} \setminus \textit{interactcandidates} \cup \\
\quad \{\textit{jorg}\} \cup \textit{jinteractcandidates} \land \\
\textit{agents}' = \textit{agents} \setminus \{\textit{agent?}\} \cup \{\textit{jagent}\}
\end{array}
$$

The delta symbol tells us that *JOINORG* is an operation schema that changes the state of *MACODOSYSTEM*. The operation schema declares the input variables *org?*, *agent?*, and *rc?* that represent the organization and the agent involved in the join, and the role contract of the join. We use the convention of adding a question mark to the names of input variables

in operation schemas. The primed components in the predicate denote the changes after the operation.

The join law schema includes a schema called $orgupdates_{join}[STATE, ORGSTATE]$ (omitted from the specification) which defines two abstract helper functions: a first function $updateorgcontext_{join}$ which updates a given context of an organization with a given context of an agent; and a second function $updateinteractcandidates_{join}$ which updates the interaction candidates of a given set of organizations by adding an interaction candidate with a given organization name.

The conditions for an agent to join an organization are:

(1) The given agent and organization are part of the set of agents and organizations of the MACODOSYSTEM.
(2) The organization involved in the join holds an open role position, which is the role position of the role contract involved in the join.
(3) This role contract is on the name of the agent involved in the join.
(4) The agent has the required capabilities to play the role of the role contract.

Additional conditions are application-specific. We give a concrete example of a join law for the traffic monitoring case below.

The results of applying a join law are:

(1) The organization's role position associated with the join is closed.
(2) The role contract of the join is added to the set of role contracts of the organization and the agent involved in the join.
(3) The context of the organization is updated.
(4) The context of other organizations, whose interaction candidates change due to the join are updated.

The update of the context of the joining organization and its interaction candidates is application-specific. As explained above, to join an organization, an agent must have the required capabilities to play the role in the role contract, i.e. condition (3) above. However, the decision of how the agent uses its capabilities to achieve its goals is a concern private to that agent and thus out of scope of the organization dynamics in MACODO.

The name of the organization and the agent as well as the context and capabilities of the agent are invariant to the join.

Specializations. A join law for the traffic monitoring case is defined as:

TrafficJoinOrg

$JOINORG[TrafficState, TrafficState]$
$trafficorgupdates_{join}$

$rc?.roleposition.role = dataobserver$
$org?.orgcontext_o.orgstate_o = undefined$
$org?.rolecontracts_o = \varnothing$

The schema includes the $trafficorgupdates_{join}$ schema which is an application-specific instance of the $orgupdates_{join}$ schema (omitted in the specification), adding domain-specific constraints to the helper functions defined for *JOINORG*. In particular, the helper function $updateinteractcandidates_{join}$ updates the context of the given set of traffic organizations by adding a new organization, with the given organization name, to the interaction candidates

in their context. The $updateorgcontext_{join}$ function updates the traffic state of the organization involved in the join and the interaction candidates. For the updated organization state holds: (1) the updated organization state is equal to the given agent state if the given organization state is undefined, (2) the updated organization state is not changed if the give organization state and the given agent state are equal, and (3) the updated organization state is differentiated if the given organization state is undefined and the given organization state and the given agent state differ. The organizations in which the neighbors of the joining agent are active become the interaction candidates of the organization involved in the join (since the set of interaction candidates of a traffic organization before a join is empty).

Besides the basic conditions defined by *JOINORG*, the join law for traffic monitoring requires that:

(1) The role in the role contract of the join is *dataobserver*.

(2) The traffic state of the organization involved in the join is *undefined*.

(3) The organization is not yet involved in any role contract.

In other words, a camera agent can only join a traffic organization that is not involved in a contract yet, and initially the agent has to play the role of data observer. As such, in the traffic monitoring case an organization can only be extended by merging with other existing organizations, and not by agents that directly join the organization.

The results of a join are are similar to the *JOINORG* schema:

(1) The role position for the *dataobserver* role associated with the join is closed.

(2) The role contract in the role of *dataobserver* is added to the set of role contracts of the organization and the agent involved in the join.

(3) The organization state of the organization is updated with the traffic state of the camera agent involved in the join.

(4) The set of neighbors (interaction candidates) of the organization is updated.

(5) The organizations that as a result of the join become a new neighbor (interaction candidate) of the organization involved in the join are also updated with a new neighbor (i.e. the organization involved in the join).

The name of the organization, and the name, the capabilities, and the context of the agent are invariant to the join.

Examples. We illustrate the join in the traffic monitoring scenario by showing how *agent1* joins a new organization *org1* at time T2 (see figure 2). We start by specifying how a new traffic organization can be started, and then how *agent1* starts *org1*. Subsequently, we specify how *agent1* joins the new organization at time T2.

StartNewTrafficOrg

Δ*TrafficMacodoSystem*
sagent? : *CameraAgent*
norg? : *TrafficOrg*
rp : *ROLEPOS*

$sagent? \in agents \land$
$\quad \{observ, recogn\} \subseteq sagent?.capabilities_a \land$
$norg?.name_o \in uniquenames_o \land$
$\quad rp.role = dataobserver \land rp \in norg?.rolepositions_o$
$\quad norg?.rolecontracts_o = \varnothing \land$
$agents' = agents \land$
$organizations' = organizations \cup \{norg?\}$

A new organization can only be started by an agent that belongs to the *TrafficMacodoSystem* and that has the capabilities to play the role of *dataobserver*. To add a new organization to a *TrafficMacodoSystem*, this organization should offer an open role position for the *dataobserver* role. The operation schema to start the new traffic organization *org1* by *agent1* is defined as follows:

TrafficStartOrg$_{T12}$

Δ*TrafficMacodoSystem*$_{T1}$
StartNewTrafficOrg

$\exists\, agent1 : CameraAgent \bullet$
$\quad sagent? = agent1 \land agent1.name_a = cam_1 \land$
$\exists\, org1 : TrafficOrg \bullet$
$\quad norg? = org1 \land org1.name_o = trorg_1$

Now we specify how *agent1* joins *org1*:

TrafficJoinOrg$_{T2}$

Δ*TrafficStartOrg*$_{T12}$
TrafficJoinOrg

$\exists\, agent1 : CameraAgent \bullet$
$\quad agent? = agent1 \land agent1.name_a = cam_1 \land$
$\exists\, rc : ROLECONT;\ org1 : TrafficOrg \bullet$
$\quad rc? = rc \land$
$\quad\quad rc.roleposition.orgname = org1.name_o \land$
$\quad\quad rc.agentname = agent1.name_a \land$
$\quad org? = org1 \land org1.name_o = trorg_1$

The join operation changes the state of *TrafficMacodoSystem*$_{T12}$. As a result of the *TrafficJoinOrg* law, the open role position with the role of *dataobserver* in $org1$ is closed and a role contract with this role is established between *agent1* and *org1*.

3.2.2 *Context Update.* Once an agent participates in an organization, changes in its context will result in an update of the context of the organization. We define the abstract event of a context update of an agent as follows:

CONTEXTUPDATE
$agentname : AGENTNAME$

How and when an agent updates its context is private to the agent and is outside the scope of the MACODO model. The following schema defines a context update of an organization implied by a context update of an agent with a role contract in this organization:

ORGCONTEXTUPDATE $[STATE, ORGSTATE]$
$\Delta ORG[ORGSTATE]$
$orgupdates[STATE, ORGSTATE]$
$event? : CONTEXTUPDATE$
$agent? : AGENT[STATE]$

$event?.agentname = agent?.name_a$
$\exists\, rc : ROLECONT \bullet$
$\quad rc.agentname = agent?.name_a \land rc \in rolecontracts_o$
$orgcontext'_o = updateorgcontext(orgcontext_o, agent?.context_a)$

The $orgupdates[STATE, ORGSTATE]$ schema defines a generic function *updateorgcontext* that updates a given context of an organization with a given context of an agent (omitted in the specification). As a result of an update of its context, an organization may open new role positions or close positions. Such implications are application-specific. The context change may also satisfy the conditions for triggering a particular law such as splitting the organization, or merging with another organization.

Specializations. For the traffic monitoring case, the update of the context of an organization resulting from a context update of a camera agent is defined as:

TrafficOrgContextUpdate
$ORGCONTEXTUPDATE[TrafficState, TrafficState]$
trafficorgupdates

The schema includes the *trafficorgupdates* schema that specializes the *orgupdates* schema. The concrete *updateorgcontext* function updates the traffic state of the traffic organization with the given state of the camera agent following the same rules as for an agent joining an organization (see the specification of the join law above). The organization's interactions candidates are invariant to the context update of a camera agent. However, a context update may trigger a law for splitting the organization or merging with a neighboring organization.

An example of a context change in the traffic monitoring case happens between T2 to T3 when the traffic state in the viewing range of agent 3 change to traffic jam (see figure 2). As a result of the context change, the conditions for a merge between traffic organizations org_2 and org_3 are satisfied. We explain the merge of the traffic organizations next.

3.2.3 *Merge Law.* Given the application domain it can be interesting to merge organizations in certain circumstances. In the traffic monitoring case, for example, two organizations observing the same traffic jam should be merged in one organization. The concrete mechanisms of a merge are defined in a merge law.

Definitions. A law for merging two organizations is defined as:

$$
\begin{array}{l}
\textit{MERGEORGS}\,[\textit{STATE}, \textit{ORGSTATE}] \\
\Delta\textit{MACODOSYSTEM}[\textit{STATE}, \textit{ORGSTATE}] \\
\textit{orgupdates}_{\textit{merge}}[\textit{ORGSTATE}] \\
\textit{org}1?, \textit{org}2? : \textit{ORG}[\textit{ORGSTATE}] \\
\hline
\textit{org}1? \in \textit{organizations} \land \textit{org}2? \in \textit{organizations} \\
\exists\, \textit{morg} : \textit{ORG}[\textit{ORGSTATE}] \bullet \\
\quad \textit{morg.name}_o \in \textit{uniquenames}_o \land \\
\quad \textit{morg.rolepositions}_o = \textit{mergepositions}(\textit{org}1?, \textit{org}2?, \textit{morg.name}_o) \land \\
\quad \textit{morg.rolecontracts}_o = \textit{mergecontracts}(\textit{org}1?, \textit{org}2?, \textit{morg.name}_o) \land \\
\quad \textit{morg.orgcontext}_o = \textit{updateorgcontext}_{\textit{merge}}(\\
\qquad \textit{org}1?.\textit{orgcontext}_o, \textit{org}2?.\textit{orgcontext}_o) \land \\
\exists\, \textit{cagents}, \textit{magents} : \mathbb{P}\,\textit{AGENT} \bullet \\
\quad \textit{cagents} = \{a : \textit{agents} \mid a\ \textit{activein org}1? \lor a\ \textit{activein org}2?\} \land \\
\quad \textit{magents} = \{a : \textit{AGENT} \mid \exists\, ca : \textit{cagents} \bullet \\
\qquad a.\textit{name}_a = ca.\textit{name}_a \land \\
\qquad a.\textit{capabilities}_a = ca.\textit{capabilities}_a \land \\
\qquad a.\textit{rolecontracts}_a = \\
\qquad\quad \{\textit{mrc} : \textit{morg.rolecontracts}_o \mid \textit{mrc.agentname} = a.\textit{name}_a\} \cup \\
\qquad\quad \{\textit{orc} : ca.\textit{rolecontracts}_a \mid \\
\qquad\qquad \textit{orc} \notin \textit{org}1?.\textit{rolecontracts}_o \land \textit{orc} \notin \textit{org}2?.\textit{rolecontracts}_o\} \land \\
\qquad a.\textit{context}_a = ca.\textit{context}_a\} \land \\
\exists\, \textit{interactcandidates}, \textit{minteractcandidates} : \mathbb{P}\,\textit{ORG}[\textit{ORGSTATE}] \bullet \\
\quad \textit{interactcandidates} = \{\textit{ic} : \textit{organizations} \mid \textit{ic.name}_o \in \\
\qquad (\textit{org}1?.\textit{orgcontext}_o.\textit{interactcandidates}_o \cup \\
\qquad \textit{org}2?.\textit{orgcontext}_o.\textit{interactcandidates}_o)\} \land \\
\quad \textit{minteractcandidates} = \\
\qquad \textit{updateinteractcandidates}_{\textit{merge}}(\textit{interactcandidates}, \\
\qquad\quad \{\textit{org}1?.\textit{name}_o, \textit{org}2?.\textit{name}_o\}, \textit{morg.name}_o) \land \\
\textit{organizations}' = \textit{organizations} \setminus \{\textit{org}1?, \textit{org}2?\} \setminus \textit{interactcandidates} \cup \\
\quad \{\textit{morg}\} \cup \textit{minteractcandidates} \land \\
\textit{agents}' = \textit{agents} \setminus \textit{cagents} \cup \textit{magents}
\end{array}
$$

The merge law includes the $\textit{orgupdates}_{\textit{merge}}$ schema that groups a number of abstract helper function (omitted in the specification). The $\textit{updateorgcontext}_{\textit{merge}}$ function returns the updated context of a merged organization given the context of the merging organizations. The $\textit{updateinteractcandidates}_{\textit{merge}}$ function updates the interaction candidates of a given set of organizations, taken into account the given names of the merging organizations and the name of the merged organization. The *mergecontracts* and *mergepositions* functions respectively merge the role contracts or role positions of two organizations, and update these contracts or positions with a given organization name. We can instantiate these functions for a particular application, by including this schema in an application-specific schema, adding domain-specific constraints.

We explain the merge law in three steps, corresponding to the three main parts of the schema: first we explain the results for the organizations, then we explain the results for the agents involved in the merge, finally, we explain the results for the interaction candidates of the involved organizations.

In order to merge, the merging organizations have to belong to the MACODO system. For

the merged organization, the following holds:

(1) The organization has a new unique name.
(2) The open role positions of the merged organization are on its name.
(3) The open role contracts of the merged organization are on its name (which is implied by the definition of organization).
(4) The context of the organization is updated given the context of the merging organizations.

After the merge, all role positions and role contracts of the merging organizations no longer exist in the MACODO system.

The agents involved in the merge are active in one of the merging organizations, i.e. they have one or more role contracts with one of the merging organizations. For each of the agents involved in the merge, the following holds:

(1) The agent may possibly share one or more role contracts with the merged organization (i.e. the role contracts on its name).
(2) The agents' role contracts with other organizations are not changed.

The name of the agents, their capabilities, and contexts are invariant to a merge.

The interaction candidates of the merging organizations are updated given the removal of the merging organizations and the addition of the merged organization to the MACODO system. The update of the interaction candidates is application-specific; we give an example for the traffic monitoring case below.

Specializations. The merge of two traffic organizations is defined as:

TrafficMergeOrgs

$$\begin{array}{|l} MERGEORGS[TrafficState, TrafficState] \\ trafficorgupdates_{merge} \\ \hline org1?.orgcontext_o.orgstate_o = org2?.orgcontext_o.orgstate_o = congested \\ org1?.name_o \in org2?.orgcontext_o.interactcandidates_o \end{array}$$

We specialize the helper functions of the merge law, by including the $orgupdates_{merge}$ schema in an application-specific schema $trafficorgupdates_{merge}$, adding domain-specific constraints to these helper functions (omitted in the specification). In particular, the *mergecontracts* function merges the role contracts of two organizations by removing all but one role contract for the role of data aggregator and updating the remaining role contracts with the given organization name. Furthermore, a number of additional helper functions are used (omitted in the specification). The *aggregatorcontracts* function returns the role contracts of a given organization in the role of data aggregator. The *selectaggregatorcontract* function takes a set of role contracts (in the role of data aggregator) as input and returns as output a selected role contract. Finally, *updateorgcontracts* is a helper function to update a given set of role contracts with a given organization name.

Besides the basic conditions defined by *MERGEORGS*, the merge law for traffic monitoring requires that:

(1) The traffic state of the merging organizations is congested.
(2) The merging organizations are neighbors.

After the merge operation, the context of the merged organization is updated, as well as the context of the former neighbors of the merging organizations. The context of the camera agents involved in the merge is not changed by the merge.

Examples. We specify how traffic organizations *org2* and *org3* merge at time T4. We assume that the traffic state monitored by the camera of *agent2* has been changed to *congested* at time T3.

$$
\begin{array}{|l}
\hline
\textit{TrafficMergeOrgs}_{T4} \\
\Delta \textit{TrafficMacodoSystem}_{T3} \\
\textit{TrafficMergeOrgs} \\
\hline
\exists\, org1 : \textit{TrafficOrg};\ org2 : \textit{TrafficOrg} \bullet \\
\quad org1? = org1 \land org2? = org2 \land \\
\quad org1.name_o = trorg_2 \land org2.name_o = trorg_3 \\
\hline
\end{array}
$$

The merge operation changes the state of the $\textit{TrafficMacodoSystem}_{T3}$. Applying the *TrafficMergeOrgs* law results in merge of the organizations *org2* and *org3*.

4. RELATED WORK

Discussion of related work focuses on two research areas of multi-agent organizations. We start by looking at research on organizational abstractions, role-based approaches and organization models. Then we discuss related work on formal methods for organizations.

4.1 Organizational Abstractions for Multi-Agent Systems

Roles and organizations are generally acknowledged as valuable abstractions to design interactions in multi-agent systems [Demazeau and Rocha Costa 1996; Jennings 2000; Tambe et al. 2000; Gasser 2001]. An interesting overview of different organizational paradigms in multi-agent systems is given in [Horling and Lesser 2004].

Classical multi-agent systems are typically designed from an agent-centered perspective, in terms of agents mental states. This is reflected in traditional agent development frameworks such as JADE [Bellifemine et al. 2001]. The main interest is the effect of interaction on the internal architecture of the agents. Today, however, it is widely recognized that relying on the agent's individual architecture only is insufficient to model interactions in multi-agent systems. A lot of research has been focusing on separating the internal design of agents from the design of societies [Cabri et al. 2002; Ferber et al. 2004]. To achieve such separation of concerns, many authors have advocated role-based and organization-centered approaches to develop agent societies, claiming effective engineering of multi-agent systems needs high-level agent-independent concepts and abstractions that explicitly define the organization in which agents live [Dignum 2004]. These concepts range from first-class organizational abstractions and rules [Zambonelli et al. 2001], norms and institutions [Dignum and Dignum 2001; Esteva et al. 2001], to the explicit representation of social structures [Parunak and Odell 2002]. We discuss a number of representative examples of role-based approaches and existing organization models.

ROPE [Becht et al. 1999] presents a role-based programming environment for agents, recognizing roles as first-class entities. Recurring cooperation and coordination patterns are captured in cooperation processes, represented by a set of interconnected roles. Roles describe the required cooperative behavior and capabilities of agents and act as an abstraction of the agents within an cooperation process. This allows to decouple the organization

of the agents in a multi-agent system from the structure of cooperation processes, enhancing the separation of concerns. Changes in the agent organization, such as role assignment, do not affect the cooperation process specification and vice versa.

Another example is XRole [Cabri et al. 2002], which also advocates the use of roles to enhance the separation of concerns between agents and their interactions. A role level in a three-level model acts as a intermediary between the application and the environment needs. Roles are defined as the behavior and set of capabilities expected for the agent that plays such role. From the environment point of view, a role imposes a defined behavior to the entity that assumes it. From the application point of view, a role allows a set of capabilities, which can be exploited by agents to carry out their tasks.

In [Odell et al. 2003a] the term role defines a normative repertoire of behavior and other features contextualized according to the group in which the role is being played. The concept of role assignment allows roles to be dynamically assigned to agents and can represent an agent assigned to a specific role as well as an open role position in a specific group. As most of the other existing work on roles there are no explicit concepts for what we call role position. In our work, a separate concept of role position corresponds to a "vacancy" for a role. By using these concepts, we are able to separate the realization of the behavior of an agent associated with a role and the management of organization dynamics.

OperA (Organizations per Agents) [Dignum 2004] describes multi-agent systems at a conceptual level using three types of interrelated models. Each model describes a different dimension of the multi-agent system concerning organizational, social or interactive aspects. Within the social model, social contracts specify the capabilities and responsibilities of agents within the society. The concept of social contract is similar to our concept of role contract and provides a "window" to the agent, through which other agents know what to expect and how to interact with the agent.

AGR (Agent Group Role) [Ferber and Gutknecht 1998] presents a "generic" metamodel of multi-agent systems in which agents, playing roles, are organized into groups. Groups partition organizations and roles represent functional positions of agents in a group. In [Ferber et al. 2005], the AGR model is extended to AGRE with the E standing for environment. Groups are now grouped together into worlds. These worlds offer primitives for agents to join a particular group and to play a particular role.

Whereas most of the discussed approaches recognize separation of concerns in terms of agent design versus organization design, our work goes a step further by offering first-class organization concepts to support a middleware for dynamic organizations.

4.2 Formal Methods for Organizations

Formal models are generally recognized as valuable artifacts within the design and development of agent organizations, supporting formal analysis and facilitating development [Esteva et al. 2001]. We discuss a number of representative examples.

OperA has a formal model based on temporal and deontic logic, building a semantic theory around OperA models. This model allows to verify whether role objectives can be achieved or whether roles are sufficient to realize society objectives.

[Grossi et al. 2007] focus on the structural aspects of organizations, presenting a formal framework which allows to measure, compare and evaluate organizations. The authors distinguish three dimensions in organizational structure: power, coordination and control. Representing these dimensions as multi-graphs and giving the edges formal semantics using description logic, they can study and analyze properties of organizations, such as measuring the completeness and connectedness of single dimensions, or measuring the overlap

and cover between different dimensions. These properties give way for structural evaluation in terms of robustness, flexibility and efficiency of organizations.

Similarly, [Popova and Sharpanskykh 2009] present a formal framework to represent and reason about organizations, enabling formal verification and validation, but also automated analysis and simulation of organizational scenarios. Instead of graphs, a variant of order-sorted predicate logic is used to express temporal relations and dynamics in a number of related views on organizations. Dynamics are viewed from an agent-centric perspective, such as agents adapting their believes in correspondence to what is observed.

Both of the above approaches focus on the verification and analysis of organizational structures. Although [Popova and Sharpanskykh 2009] do consider the environment, both approaches lack explicit support to model relations between the environment and organizational changes and dynamics. The model presented in this paper directly relates organization dynamics to context changes in the environment. Dynamics are not merely described but are specified as desired behavior of organizations as first-class entities.

The concepts of agent ability and capability are widely used in multi-agent systems, referring to an agents' ability to act in an organization and the capabilities required to act in an organization. A number of formal models underpinning these concepts have been proposed, such as [van der Hoek and Wooldridge 2005] using cooperation logic, [Cholvy et al. 2006] based on situation calculus, and logic for agent organizations (LOA) [Dignum and Dignum 2009] as an extension to temporal logic. Such formalizations allow to represent and reason about strategic powers of agents and coalitions in terms of agent abilities and capabilities in a game-theoretic setting. Concepts in these approaches often have a more fundamental and theoretic meaning. In our approach, we used the concept of capability to enable the middleware to reason about the management of organizations.

[da Rocha Costa and Dimuro 2008] study social dynamics in organizations by combining a time-variant population-organization model (PopOrg) with a system of exchange values. This allows modeling structural dynamics in a multiagent system as a set of transformations on the systems overall population-organization structure. In contrast to our approach, it does not explicitly specify dynamics in terms of organizational evolution, but rather allows to represent changes within the social structure of PopOrg models.

5. CONCLUSIONS

In this paper, we presented the MACODO organization model. The complementary part of the MACODO approach is a middleware platform that supports the distributed execution of dynamic organizations specified using the abstractions of the MACODO organization model, as described in [Weyns et al. 2009].

The contributions of this paper are:

- A formal model that defines the programming abstractions MACODO offers to the application developer to describe dynamic organizations. In the model, the life-cycle management of dynamic organizations is separated from the agents: organizations are first-class citizens, and their dynamics are governed by laws.
- The model is formally described to specify the semantics of the abstractions, and ensure its type safety.
- The application of the organization model to model dynamic organizations for a traffic monitoring application.

Modeling organizations as first-class citizens separated from the agents makes it easier to understand and specify dynamic organizations in multi-agent systems, and promotes

reusing the life-cycle management of dynamic organizations.

In ongoing research we apply MACODO to the domain of collaborative business processes, such as supply chain management and collaborative health care institutions [DiCoMas 2008]. Collaboration between different business units requires the coordination and allocation of various people and equipment at different locations. A particular focus of this research is on middleware support for secure and dynamic organizations.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library by visiting the following URL: `http://www.acm.org/pubs/citations/journals/jn/2009-V-N/p1-URLend`.

REFERENCES

BECHT, M., GURZKI, T., KLARMANN, J., AND MUSCHOLL, M. 1999. Rope: Role oriented programming environment for multiagent systems. In *COOPIS '99: Proceedings of the 4th IECIS International Conference on Cooperative Information Systems*. IEEE Computer Society, Washington, DC, USA, 325–333.

BELLIFEMINE, F., POGGI, A., AND RIMASSA, G. 2001. JADE: a FIPA2000 Compliant Agent Development Environment. In *Proceedings of the Fifth International Conference on Autonomous agents*. ACM New York, NY, USA, 216–217.

CABRI, G., LEONARDI, L., AND ZAMBONELLI, F. 2002. Separation of concerns in agent applications by roles. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*. IEEE Computer Society, Washington DC, USA, 430–438.

CHOLVY, L., GARION, C., AND SAUREL, C. 2006. Ability in a Multi-Agent Context: a Model in the Situation Calculus. In *Computational Logic in Multi-Agent Systems*. Lecture Notes on Computer Science. Springer, 23–36.

CZT. 2008. Community Z Tools. `http://czt.sourceforge.net/`.

DA ROCHA COSTA, A. AND DIMURO, G. 2008. Semantical Concepts for a Formal Structural Dynamics of Situated Multiagent Systems. In *Coordination, Organizations, Institutions, and Norms in Agent Systems III*. Lecture Notes on Computer Science. Springer, 139–154.

DEMAZEAU, Y. AND ROCHA COSTA, A. 1996. Populations and organizations in open multi-agent systems. In *Proceedings of the 1st National Symposium on Parallel and Distributed AI*.

DICOMAS. 2008. Distributed Collaboration using Multi-Agent System Architectures. http://distrinet.cs.kuleuven.be/research/projects/showProject.do?projectID=DiCoMas.

DIGNUM, V. 2004. A Model for Organizational Interaction: Based on Agents, Founded in Logic. SIKS Dissertation Series.

DIGNUM, V. AND DIGNUM, F. 2001. Modelling Agent Societies: Co-ordination Frameworks and Institutions. In *Progress in Artificial Intelligence*. Lecture Notes on Computer Science. Springer, 7–21.

DIGNUM, V. AND DIGNUM, F. 2009. A Logic for Agent Organizations. In *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, V. Dignum, Ed. Information Science Reference, 220–240.

DIGNUM, V., DIGNUM, F., AND SONENBERG, L. 2004. Towards Dynamic Reorganization of Agent Societies. *Proceedings of Workshop on Coordination in Emergent Agent Societies at ECAI*, 22–27.

D'INVERNO, M. AND LUCK, M. 2004. *Understanding Agent Systems*. SpringerVerlag.

ERTICO. 2008. Intelligent Transportation Systems for Europe, `http://www.ertico.com/`.

ESTEVA, M., RODRÍGUEZ-AGUILAR, J., SIERRA, C., GARCIA, P., AND ARCOS, J. 2001. On the Formal Specifications of Electronic Institutions. In *Agent Mediated Electronic Commerce*. Lecture Notes on Computer Science. Springer, 126–147.

FERBER, J. AND GUTKNECHT, O. 1998. A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. *Third International Conference on Multi Agent Systems*, 128–135.

FERBER, J., GUTKNECHT, O., AND MICHEL, F. 2004. From Agents to Organizations: an Organizational View of Multi-Agent Systems. In *Agent-Oriented Software Engineering IV*. Lecture Notes on Computer Science. Springer, 443–459.

FERBER, J., MICHEL, F., AND BAEZ, J. 2005. AGRE: Integrating environments with organizations. In *First International Workshop on Environments for Multi-Agent Systems*. Lecture Notes in Computer Science, vol. 3374. Springer-Verlag, New York, NY, USA, 48–56.

GASSER, L. 2001. Perspectives on organizations in multi-agent systems. In *Mutli-agents systems and applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1–16.

GROSSI, D., DIGNUM, F., DIGNUM, V., DASTANI, M., AND ROYAKKERS, L. 2007. Structural Aspects of the Evaluation of Agent Organizations. In *Coordination, Organizations, Institutions, and Norms in Agent Systems II*. Lecture Notes on Computer Science. Springer, 3–18.

HAESEVOETS, R., WEYNS, D., AND HOLVOET, T. 2008. A formal specification of an organization model and management model for context-driven dynamic organizations. In *Technical Report, CW535*. Katholieke Universiteit Leuven. http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW535.pdf.

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

HORLING, B. AND LESSER, V. 2004. A survey of multi-agent organizational paradigms. *Knowledge Engineering Review 19,* 4, 281–316.

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

JENNINGS, N. R. 2000. On agent-based software engineering. *Artificial Intelligence 177,* 2, 277–296.

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.

ODELL, J., PARUNAK, H., AND FLEISCHER, M. 2003a. The Role of Roles in Designing Effective Agent Organizations. In *Software Engineering for Large-Scale Multi-Agent Systems*. Lecture Notes in Computer Science, Vol. 2603. Springer, 27–38.

ODELL, J., PARUNAK, H. V. D., AND FLEISCHER, M. 2003b. The Role of Roles. *Journal of Object Technology 2,* 1, 39–51.

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

PARUNAK, H. AND ODELL, J. 2002. Representing Social Structures in UML. In *Agent-Oriented Software Engineering II*. Lecture Notes on Computer Science. Springer, 1–16.

POPOVA, V. AND SHARPANSKYKH, A. 2009. A Formal Framework for Organization Modeling and Analysis. In *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, V. Dignum, Ed. Information Science Reference, 141–171.

ROMAN, G., JULIEN, C., AND PAYTON, J. 2004. A Formal Treatment of Context-Awareness. In *FASE'04: 7th International Conference on Fundamental Approaches to Software Engineering*. Springer, 12–36.

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

TAMBE, M., PYNADATH, D. V., AND CHAUVAT, N. 2000. Building dynamic agent organizations in cyberspace. *IEEE Internet Computing 4,* 2, 65–73.

VAN DER HOEK, W. AND WOOLDRIDGE, M. 2005. On the Logic of Cooperation and Propositional Control. *Artificial Intelligence 164,* 1-2, 81–119.

WANT, R. 2005. System Challenges for Pervasive and Ubiquitous Computing (Intel). In *ICSE '05: Proceedings of the 27th international conference on Software engineering*. ACM, New York, NY, USA, 9–14.

WEYNS, D., HEASEVOETS, R., HELLEBOOGH, A., HOLVOET, T., AND JOOSEN, W. 2009. MACODO: Middleware Architecture for Context-Driven Dynamic Agent Organizations. *ACM Transactions on Autonomous and Adaptive Systems y,* x, nn–nn.

WISCHHOF, L., EBNER, A., AND ROHLING, H. 2005. Information Dissemination in Self-organizing Intervehicle Networks. *IEEE Transactions on Intelligent Transportation Systems 6,* 1, 90–101.

ZAMBONELLI, F., JENNINGS, N., AND WOOLDRIDGE, M. 2001. Organizational Abstractions for the Analysis and Design of Multi-Agent Systems. In *Agent-Oriented Software Engineering*. Lecture Notes on Computer Science. Springer, 407–422.

ZAMBONELLI, F., JENNINGS, N., AND WOOLDRIDGE, M. 2003. Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology 12,* 3, 317–370.