A Formal Model for Self-Adaptive and Self-Healing Organizations

Robrecht Haesevoets, Danny Weyns, Tom Holvoet, Wouter Joosen
Department of Computer Science
Katholieke Universiteit Leuven
3001 Heverlee, Belgium
robrecht.haesevoets@cs.kuleuven.be

Abstract

Multi-agent systems typically consist of autonomous entities, capable of adapting their behavior and interaction patterns in dynamic environments, making them an interesting approach for modeling self-adaptive systems. The interactions among agents, a key challenge in engineering multi-agent systems, are often structured and managed by means of organizations.

In previous work we have built a prototype of an organization middleware, which encapsulates the management of dynamic organizations as a reusable service and offers organizations as first-class programming abstractions to application developers. To develop a mature middleware, we face two key challenges: realizing the integration of the middleware with the rest of the system in a disciplined way and assuring properties, such as self-adaptivity and self-healing, of services offered by the middleware.

This paper presents a formal specification of an organization and management model for dynamic organizations, a first step in facing these challenges. Both models contribute to the integration of the middleware with the rest of the system. The organization model rigorously describes the main programming abstractions to which application developers have to conform, while the management model can be used to derive specific monitoring and control points required by the middleware to realize self- properties. In addition, the management model offers a foundation to reason about self-* properties.*

1. Introduction

Self-adaptive systems adapt their structure and behavior to cope with changing environment conditions. Multi-agent systems, consisting of autonomous entities capable of adapting their behavior and interaction patterns, are an interesting approach to model self-adaptive systems. The dynamic interactions and collaborations among agents are typically structured and managed by means of roles and organizations. In most existing approaches, however, agents have the dual responsibility of managing the organizations and their dynamics, and providing the functionality in the organizations by playing roles, making these systems hard to design and understand. We continue this discussion in Sect. 6, covering related work.

In previous work [20], we have proposed an organization middleware to address this problem. The middleware, called MACODO[1], focuses on context-driven dynamic organizations [11], and encapsulates the management of these organizations as a reusable service.

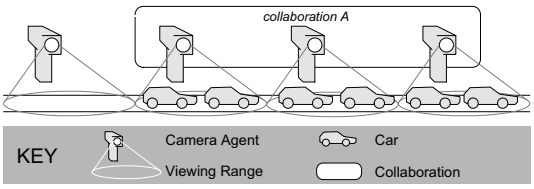

Figure 1. Context-driven collaborations in a decentralized traffic monitoring case.

Context-driven dynamic organizations are a particular class of dynamic organizations, which can be used in dynamic and open environments in which distributed resources may fail and degrade. They allow us to group autonomous entities, such as agents, in organizations, which represent dynamic collaborations that are driven by a dynamic context. An example application domain is the decentralized traffic monitoring case shown in Fig. 1. A number of intelligent cameras are distributed over a road network, and responsible for monitoring the traffic. Because each camera has a limited viewing range, cameras have to collaborate to monitor phenomena such as traffic jams. The dynamic nature of the traffic phenomena requires dynamic

[1] MACODO: Middleware Architecture for Context-driven Dynamic Organizations.

978-1-4244-3724-5/09/$25.00

collaborations between the cameras. The traffic monitoring case is further described in Sect. 2 and is used as a running example throughout the rest of this paper.

In building and validating a prototype of the MACODO middleware, we identified two key challenges in developing a mature organization middleware. A first challenge is realizing the integration of the middleware with the rest of the system in a disciplined way. This includes two important issues. (1) To achieve self-* properties, the middleware needs monitoring and control access to other system infrastructure. Until now we have relied on ad-hoc interfaces. (2) Application developers have to conform to the organization model, offered by the middleware. This model should be complete and communicated to developers in an unambiguous way. A second challenge is to assure properties of the behavior and services offered by the middleware, including self-* properties such as self-adaptive and self-healing organizations.

This paper presents a formal specification of an organization and management model for dynamic organizations. The organization model, Sect. 3, describes the main abstractions to which application developers have to conform. The management model, Sect. 4, describes the desired behavior of the middleware and is applied to the traffic monitoring case in Sect. 5. The formal specification is an important step in facing the challenges of developing a mature middleware. First, it enhances clearness, completeness and eliminates ambiguity to both application developers and the developers of the middleware itself. Self-* properties have to be formally specified. Instead of saying things like "self-adaptive organizations are organizations which adapt to changing environment conditions", we have to precisely specify what it is that changes in the environment and how the organizations are adapted. Second, a formal specification can be used as a foundation for further development of the middleware, to derive concrete monitoring and control interfaces and to check whether the distribution of the middleware conforms to the specification. Finally, it can be used to validate the organization and management model and allows developers to reason about properties and services offered by the middleware.

This paper only presents a first step in developing a mature middleware: the formal specification of an organization and management model for dynamic organizations. The use of this specification for the actual development and validation of the middleware is outside the scope of this paper, but indications are given in the conclusions, Sect. 7. We use Z as formal specification language and CZT tools [3] to edit and type check the specification. Z is a standardized, highly expressive and accessible formal language, based on set theory and first order predicate calculus. It is regularly used for describing and modeling computing systems, including multi-agent systems [5].

2. Traffic Monitoring Case

The traffic monitoring case consists of a number of cameras, distributed over a road network. An intelligent agent is deployed on each camera, responsible for monitoring the traffic. Because there is no central control and because each camera has a limited view, camera agents have to collaborate to observe larger phenomena such as traffic jams. The environment in which the cameras are deployed is very dynamic. Traffic state constantly changes, and sensor and communication hardware may fail at any time.

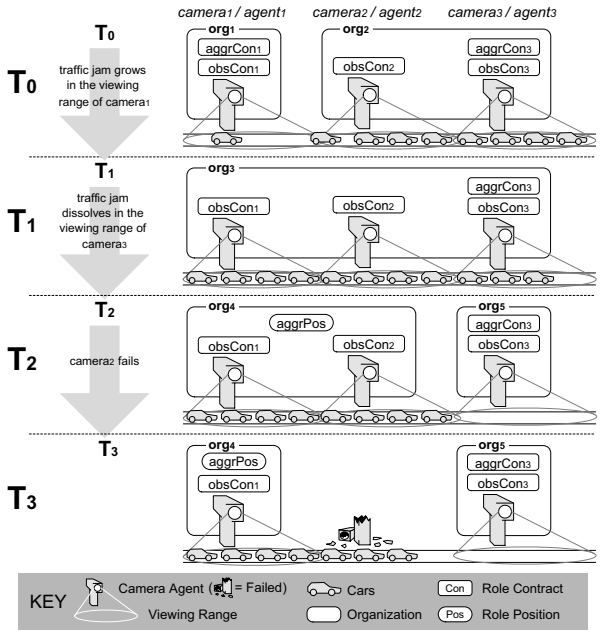

Figure 2. An example of agents collaborating in organizations to monitor traffic jams.

A simple example is shown in Fig. 2. The example shows three cameras along a single-lane highway at four different time steps. Organizations are used to structure the collaborations between the agents, and as the context of the cameras changes, the organizations are adapted accordingly. Most of the dynamics take place at the level of organizations. Organizations with related context are merged together and organizations with a differentiated context are split up. At T_0 in the example, there is a traffic jam in the viewing range of $camera_2$ and $camera_3$, and they are collaborating in one organization. But as the traffic jam grows into the viewing range of $camera_1$, the organization of $camera_1$ is merged with the organization of $camera_2$ and $camera_3$. After a while, the traffic jam dissolves in the viewing range of $camera_3$ and the organization is split up. When disaster strikes, $camera_2$ is destroyed, and at T_3, $camera_2$ is no longer part of organization org_4.

3. Organization Model

This section presents the basic abstractions to group entities with related context in organizations. Application developers, using the organization middleware, have to conform to these abstractions, to allow their agents to collaborate in organizations by playing roles. The management and dynamics of organizations are described in Sect. 4. The rest of this section gradually introduces the organization model and illustrates the core concepts in the traffic monitoring case. An overview of the organization model is shown in Fig. 3. A complete description of the model can be found in [12].

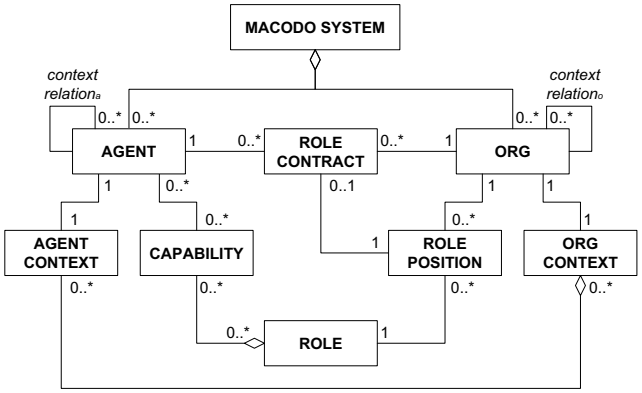

Figure 3. Conceptual organization model for context-driven dynamic organizations.

Names, Capabilities and Roles. We start by defining a number of basic sets representing the names of agents and organizations, and the capabilities of agents. Capabilities represent the abilities of agents, relevant to performing functionalities in organizations and are used to support self-healing in the management model.

$$[AGENTNAME, ORGNAME, CAPABILITY]$$

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

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

It's important to note, that the concepts of roles and capabilities are probably not rich enough to model all aspects of agent interactions. However, as we show in the following sections, the concepts are sufficient for describing the management of dynamic organizations.

Traffic Monitoring Example: In the traffic monitoring case, there are three relevant capabilities: to monitor traffic, to aggregate data and to send data. These capabilities can be combined into two types of roles.

$$monitor, aggregate, send : CAPABILITY$$
$$obsRole, aggrRole : ROLE$$
$$obsRole = \{monitor, send\}$$
$$aggrRole = \{aggregate, send\}$$

The observation role (*obsRole*) observes the traffic and sends its data to an aggregation role (*aggrRole*). The aggregation role aggregates all data received from observation roles, and sends the aggregated data to interested clients, such as traffic controllers or driver assistance systems.

Role Positions and Role Contracts. A role that is required in an organization is represented by a role position. As for other non-basic concepts we use a schema for its representation in Z.

ROLEPOS
$$role : ROLE$$
$$orgName : ORGNAME$$
$$role \neq \varnothing$$

A role position, much like a job opening in a company, can be open or there can be a contract. In the model, such a contract is represented by a role contract between an agent and an organization for a certain role position.

ROLECONT
$$agentName : AGENTNAME$$
$$rolePosition : ROLEPOS$$

Agents and Agent Context. Agents have a number of capabilities, allowing them to realize certain roles. In the model, available capabilities are assumed to be arbitrary resources of the agent. Before agents can enter into a contract for a role position, they must have the required capabilities, which is defined in the management model. When an agent fails or degrades, it loses some of its capabilities.

AGENT
$$name_a : AGENTNAME$$
$$context_a : AGENTCONTEXT$$
$$capabilities_a : \mathbb{P}\, CAPABILITY$$
$$roleContracts_a : \mathbb{P}\, ROLECONT$$
$$\forall rc : roleContracts_a \bullet rc.agentName = name_a$$

The context of an agent consists of the current state of the local environment and a location.

$$[STATE, LOCATION]$$

$$
\begin{array}{|l}
\hline
MACODOSYSTEM_{traffic} \\
MACODOSYSTEM \\
\hline
\forall x, y : AGENT \bullet (x, y) \in contextRelations_a \Leftrightarrow \\
\quad x \neq y \land \exists ox, oy : organizations \bullet x\ activeIn\ ox \land y\ activeIn\ oy \land x.context_a\ localTo\ y.context_a \land \\
\quad \neg\ \exists z : AGENT \bullet \exists oz : organizations \bullet z\ activeIn\ oz \land z.context_a\ between\ (x.context_a, y.context_a) \\
\forall x, y : ORG \bullet (x, y) \in contextRelations_o \Leftrightarrow x \in organizations \land y \in organizations \land x \neq y \land \\
\quad \exists ax, ay : AGENT \bullet ax\ activeIn\ x \land ay\ activeIn\ y \land (ax, ay) \in contextRelations_a \\
\hline
\end{array}
$$

Schema 1: The MACODO system with context relations for the traffic monitoring case.

The location refers to the locality of the context and can be physical or logical.

$$
\begin{array}{|l}
\hline
AGENTCONTEXT \\
state : STATE \\
location : LOCATION \\
\hline
\end{array}
$$

Traffic Monitoring Example: In the traffic monitoring case, location refers to the physical position of a camera and the area covered by its viewing range. The context state is the traffic state observed in the viewing range, which can be: free flow, bounded flow or congested.

$$
freeflow, boundedflow, congested : STATE
$$

Organizations and Organization Context. Organization context is the aggregation of the context of all agents active in the organization. An agent is active in an organization if it has one or more contracts in the organization.

$$
ORGCONTEXT == \mathbb{P}\, AGENTCONTEXT
$$

In addition to the context, organizations consist of the set of open role positions, available to the agents active in the organization, and the set of current role contracts.

$$
\begin{array}{|l}
\hline
ORG \\
name_o : ORGNAME \\
context_o : ORGCONTEXT \\
roleContracts_o : \mathbb{P}\, ROLECONT \\
rolePositions_o : \mathbb{P}\, ROLEPOS \\
\hline
\forall rp : rolePositions_o \bullet rp.orgName = name_o \\
\forall rc : roleContracts_o \bullet \\
\quad rc.rolePosition.orgName = name_o \\
\hline
\end{array}
$$

To participate in organization dynamics, agents can start new organizations in which they get a default role contract. This is the only way for agents to join a new organization. The newly started organization can be merged with other existing organizations. Once an agent has a contract in an organization, it can apply for other open role positions in that organization. Agents can leave an organization by ending all contracts it has in that organization. The concrete mechanics will be explained in the management model.

MACODO System. The state of the complete system in terms of agents and organizations is represented by the MACODO system. It consists of the set of agents, currently active in the system, and the set of organizations, representing the current collaborations between the agents.

$$
\begin{array}{|l}
\hline
MACODOSYSTEM \\
agents : \mathbb{P}\, AGENT \\
organizations : \mathbb{P}\, ORG \\
contextRelations_a : CONTEXTRELATIONS_a \\
contextRelations_o : CONTEXTRELATIONS_o \\
uniquenames_o : \mathbb{P}\, ORGNAME \\
\hline
uniquenames_o = \{n : ORGNAME \mid \\
\quad \forall o : organizations \bullet n \neq o.name_o\} \\
\forall x, y : agents \bullet x.name_a = y.name_a \Leftrightarrow x = y \\
\forall x, y : organizations \bullet \\
\quad x.name_o = y.name_o \Leftrightarrow x = y \\
\hline
\end{array}
$$

In a addition, it contains a number of context relations, which represent a relation between two agents or organizations, with respect to the locality of their context. A context relation is explicitly represented in the MACODO system itself and not in the agents or organizations, because it does not only depend on the context of the entities involved in the relation, but on the context of all entities in the MACODO system. Together with the context, context relations are the main drivers for the management of organizations.

Traffic Monitoring Example: An example of a context relation is the neighbor relation in the traffic monitoring case (e.g. in Fig. 2, $camera_1$ is neighboring $camera_2$ and so

on). Schema 1[2] shows the MACODO system with context relations for the traffic monitoring case, defined as functions on all agents and organizations in the MACODO system. The context relation only considers agents that are currently active in an organization, because these are the only cameras engaged in monitoring traffic. Two agents are neighboring if they are local to each other and if there are no other active agents with a location between them. Two organizations are neighboring if there exists at least one neighbor relation between agents of the two organizations. A typical state of the MACODO system is described in the following schema, showing the state of the example in Fig. 2 at T_0.

$$
\begin{array}{|l}
\hline
\textit{MacodoSystem}T_0 \\
\textit{MACODOSYSTEM}_{traffic} \\
\hline
\exists\, obsCon_1, aggrCon_1, obsCon_2, \\
\qquad obsCon_3, aggrCon_3 : ROLECONT \bullet \\
\quad obsCon_1.rolePosition.role = obsRole \land \\
\quad obsCon_2.rolePosition.role = aggrRole \land \\
\quad \ldots \\
\exists\, agent_1, agent_2, agent_3 : AGENT \bullet \\
\quad agents = \{agent_1, agent_2, agent_3\} \land \\
\quad agent_1.context_a.state = freeflow \land \\
\quad agent_2.context_a.state = congested \land \\
\quad agent_3.context_a.state = congested \land \\
\quad agent_1.roleContracts_a = \{obsCon_1, aggrCon_1\} \land \\
\quad \ldots \\
\exists\, org_1, org_2 : ORG \bullet \\
\quad organizations = \{org_1, org_2\} \land \\
\quad org_1.context_o = \{agent_1.context_a\} \land \\
\quad org_2.context_o = \\
\qquad \{agent_2.context_a, agent_3.context_a\} \land \\
\quad org_1.roleContracts_o = \{obsCon_1, aggrCon_1\} \land \\
\quad \ldots \\
contextRelations_a = \{agent_1 \mapsto agent_2, \\
\quad agent_2 \mapsto agent_3, \ \ldots\ \} \land \\
contextRelations_o = \{org_1 \mapsto org_2, org_2 \mapsto org_1\} \\
\hline
\end{array}
$$

4. Management Model for Dynamic Organizations

An overview of the management model is shown in Fig. 4. Events, external to the MACODO system, alter the state of the MACODO system, while a number of reflective adaptation processes, or laws, reflect over the current state and adapt it accordingly. The actual management and the realization of self-adaptive and self-healing organizations, is the result of an interplay between laws, agents and a dynamic environment.

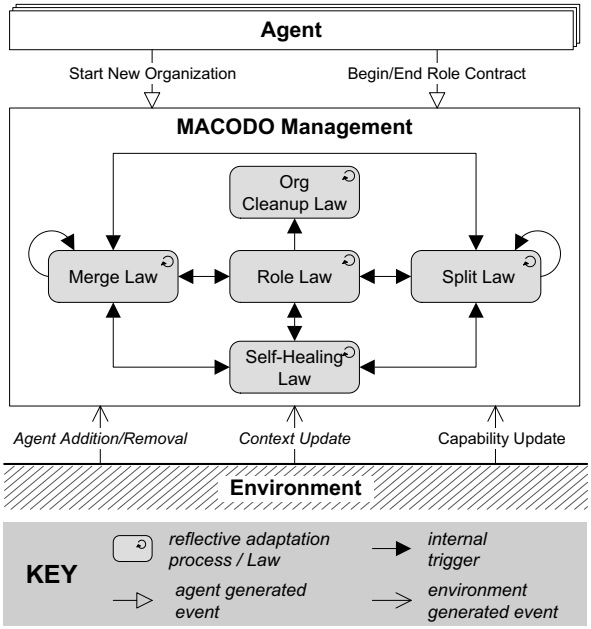

Figure 4. A management model for dynamic organizations.

In the domain of context-driven dynamic organizations, we have identified five types of events and five essential laws. Events represent the monitoring points, required by the organization middleware and laws represent the effect and control of the management on the rest of the system. Each law has its specific responsibility. The merge law, for example, merges organizations together but does not ensure the correct role positions in the merged organization, this is the responsibility of the role law.

We now describe the different types of events and laws. The events and laws are formally specified as operation schemas, which alter the state of the MACODO system. A complete specification can be found in [12].

Events

Context Updates. A context update [3] represents an update to the context of an agent, active in the MACODO system. The update also affects the context of the organizations in which the agent is active. In the traffic monitoring case, a context update can be a change in observed traffic state or a change in camera location. The operation

[2] *localTo* and *between* are application specific functions w.r.t. the locality of agent context. *activeIn*: true if the agent has a role contract in the organization.

[3] Context and capability updates may be passed to the middleware through the agents, but this is not relevant to the model.

ContextUpdate

$\Delta MACODOSYSTEM$
$agentname? : AGENTNAME$
$context? : AGENTCONTEXT$

$\exists\, oagent : agents \bullet oagent.name_a = agentname? \land$
$\exists\, uagent : AGENT \bullet uagent = updateAgent_{context}(oagent,\ context?) \land$
$\exists\, affectedorgs : \mathbb{P}\, ORG \bullet affectedorgs = \{o : organizations \mid oagent\ activeIn\ o\} \land$
$\exists\, uorgs : \mathbb{P}\, ORG \bullet uorgs = \{ao : affectedorgs \bullet updateOrg_{context}(ao,\ ao.context_o \setminus \{oagent.context_a\} \cup \{context?\})\} \land$
$organizations' = organizations \setminus affectedorgs \cup uorgs \land$
$agents' = agents \setminus \{oagent\} \cup \{uagent\}$

Schema 2: An operation schema specifying a context update event.

schema specifying a context update is given in schema 2[4]. The delta symbol indicates the schema alters the state of the MACODO system. The schema takes an agent name and a corresponding agent context as input. We use the convention of adding a question mark to the names of input variables. If the agent name belongs to an agent active in the MACODO system, the operation schema updates the agent and the affected organizations. Primed components (e.g. $agents'$) denote the changes after the operation. Context relations, defined as functions on the sets of agents and organizations in the MACODO system, are automatically adapted as these sets change. An organization with an updated context may trigger the merge, split or role law.

Capability Updates. A capability update [3] represents the degradation, failure or recovery of an agent and its resources. A failing or degrading agent loses capabilities, while a recovering agent regains capabilities. In the traffic monitoring case, cameras lose capabilities if their sensor or communication unit is damaged. An agent which has lost the capabilities required by its current role contracts triggers the self-healing law.

CapabilityUpdate

$\Delta MACODOSYSTEM$
$agentname? : AGENTNAME$
$capabilities? : \mathbb{P}\, CAPABILITY$

$\exists\, oa : agents \bullet oa.name_a = agentname? \land$
$\exists\, ua : AGENT \bullet ua.name_a = oa.name_a \land$
$\quad ua.capabilities_a = capabilities? \land$
$\quad ua.roleContracts_a = oa.roleContracts_a \land$
$\quad ua.context_a = oa.context_a \land$
$organizations' = organizations \land$
$agents' = agents \setminus \{oa\} \cup \{ua\}$

[4] $updateAgent_{context}$: returns an agent with an updated context. $updateOrg_{context}$: returns an organization with an updated context.

Agent Addition/Removal. Agents can be added or removed from the MACODO system. The model only allows the addition or removal of isolated agents, not engaged in any role contract. The addition or removal of agents can be the decision of an administrator or an external policy.

Start New Organization. Once an agent is added to the system, an agent can decide to start a new organization to participate in the organization dynamics. The agent gets a default role contract in the new organization, which is application specific. In the traffic monitoring case, agents get a default role contract for the observation role. The new organization may trigger the merge or the role law.

Begin/End Role Contract. An agent active in an organization can decide to begin or end a role contract. An agent can only begin a role contract in an organization, if the agent is already active in the organization, if there is a corresponding open role position in the organization, and if the agent has the required capabilities. If an agent ends its last role contract in an organization, the agent context is removed from the organization context.

Laws as Reflective Adaptation Processes

Merge Law. The merge law, shown in schema 3[5], takes two organizations as input and merges these organizations if their context is mergeable and related, which is application specific. The merging consists of removing the two input organizations from the MACODO system and adding a merged organization. The merged organization has a unique

[5] $updateOrgPositions$: updates a given set of role positions with a new organization name. $updateAgent_{contracts}$: returns an agent with an updated set of role contracts.

MergeLaw

$\Delta MACODOSYSTEM$

$org1?, org2? : ORG$

$(org1?, org2?) \in contextRelations_o \land mergeableState_o\ (org1?, org2?)$

$\exists morg : ORG \bullet morg.name_o \in uniquenames_o \land$

$\quad morg.rolePositions_o = updateOrgPositions(org1?.rolePositions_o \cup org2?.rolePositions_o, morg.name_o) \land$

$\quad morg.roleContracts_o = org1?.roleContracts_o \cup org2?.roleContracts_o \land$

$\quad morg.context_o = org1?.context_o \cup org2?.context_o \land$

$\exists oagents, uagents : \mathbb{P}\, AGENT \bullet oagents = \{a : agents \mid a\ activeIn\ org1? \lor a\ activeIn\ org2?\} \land$

$\quad uagents = \{oa : oagents \bullet updateAgent_{contracts}(oa, \{mrc : morg.roleContracts_o \mid mrc.agentName = oa.name_a\} \cup$

$\quad\quad \{orc : oa.roleContracts_a \mid orc \notin org1?.roleContracts_o \land orc \notin org2?.roleContracts_o\})\} \land$

$organizations' = organizations \setminus \{org1?, org2?\} \cup \{morg\} \land$

$agents' = agents \setminus oagents \cup uagents$

Schema 3: An operation schema specifying the merge law.

name and consists of the union of context, role positions and role contracts of the two input organizations. In the traffic monitoring case, organizations have to be observing the same traffic state and there must be a neighbor relation between two of their cameras. A newly merged organization may trigger the role law if its role positions or role contracts are incorrect.

Split Law. The split law splits any organization in the MACODO system with a differentiated or unrelated context into a set of sub-organizations. The split law ensures each of the sub-organizations has a uniform and related context. Whether context is differentiated or unrelated is defined in application specific functions. In the traffic monitoring case, an organization is split up if its agents are not observing the same traffic state, or when the agents are disjoint with respect to the neighbor relations. The state of the sub-organizations may trigger the merge or the role law.

Role Law. The role law opens role positions for roles that are lacking in organizations and closes role positions and role contracts for roles that are no longer needed in organizations. The role law cannot create new role contracts, this lies within the decision of the agents. The correct configuration of roles is application specific. In the traffic monitoring case, each agent active in an organization should have a role contract for the observation role, and each organization should have exactly one role contract, or at least an open role position, for the aggregation role.

Self-Healing Law. When an agent loses capabilities through a capability update, the agent may no longer have the capabilities required by its current role contracts, making the MACODO system inconsistent with the state of the real world. The self-healing law restores the consistency by removing all invalid role contracts from the MACODO system and updating the affected organizations. The operation schema is given in schema 4[6] and applies to any agent in the MACODO system which has one or more invalid role contracts. The self-healing law does not open new role positions for removed role contracts, this is the responsibility of the role law. When an agent or its resources completely fail, it is not automatically removed from the MACODO system. Instead, it loses all of its capabilities and remains in the MACODO system until removed by an external event, or regains capabilities when its resources have been repaired.

Organization Clean Up Law. The organization clean up law removes isolated organizations from the MACODO that no longer have any role contracts. This law is triggered when an agent ends the last role contract in an organization, or loses the required capabilities for this contract.

5. Applying the Management Model to the Traffic Monitoring Case

We now apply the management model to the traffic monitoring case introduced in Sect. 2. Driven by events, generated by the dynamic traffic environment and a set of camera agents, the laws adapt the organization structure. At the different time steps of the example in Fig. 5, we show the state of the MACODO system represented in Z. A more comprehensive example can be found in [12].

[6] *invalidroleContracts* is an additional help function which returns the role contracts of a given agent, for which the agent no longer has the required capabilities.

SelfHealingLaw

$\Delta MACODOSYSTEM$

$agent? : AGENT$

$\exists\, invalidContracts : \mathbb{P}\, ROLECONT \bullet invalidContracts = invalidroleContracts(agent?) \land invalidContracts \neq \varnothing \land$

$\exists\, uagent : AGENT \bullet uagent = updateAgent_{contracts}(agent?, agent?.roleContracts_a \setminus invalidContracts) \land$

$\exists\, affectedorgs : \mathbb{P}\, ORG \bullet affectedorgs = \{o : organizations \mid \exists\, rc : invalidContracts \bullet rc \in o.roleContracts_o\} \land$

$\exists\, uorgs : \mathbb{P}\, ORG \bullet uorgs = \{uo : ORG \mid \exists\, ao : affectedorgs \bullet$

$\quad uo.name_o = ao.name_o \land uo.context_o = \{a : AGENT \mid a\ activeIn\ uo \bullet a.context_a\} \land$

$\quad uo.roleContracts_o = ao.roleContracts_o \setminus invalidContracts \land uo.rolePositions_o = ao.rolePositions_o\} \land$

$organizations' = organizations \setminus affectedorgs \cup uorgs \land$

$agents' = agents \setminus \{agent?\} \cup \{uagent\}$

Schema 4: An operation schema specifying the self-healing law.

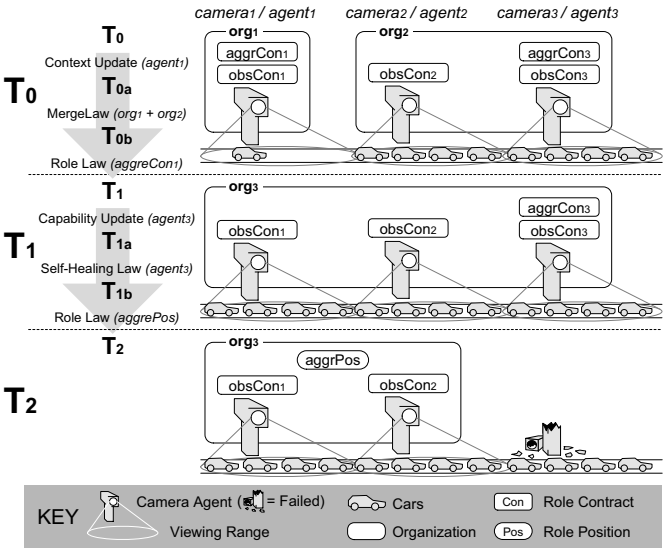

Figure 5. The management model applied to a traffic monitoring example.

At T_0, $camera_2$ and $camera_3$ are observing congested traffic and grouped in one organization, while $camera_1$, observing freeflow traffic, is in a separate organization.

MacodoSystemT_0

$MACODOSYSTEM_{traffic}$

$\exists\, org_1, org_2 : ORG \bullet$

$\quad organizations = \{org_1, org_2\} \land$

$\ldots$

$agent_1.context_a.state = freeflow \land$

$agent_2.context_a.state = congested \land$

$agent_3.context_a.state = congested \land$

$\ldots$

After T_0, the traffic jam grows into the viewing range of $camera_1$, generating a context update event which changes the context of $agent_1$.

MacodoSystemT_{0a}

$MACODOSYSTEM_{traffic}$

$\ldots$

$agent_1.context_a.state = congestion \land$

$\ldots$

At T_{0a}, org_1 and org_2 have a mergeable and related context, triggering the merge law with org_1 and org_2 as input organizations. At T_{0b}, the merge law has merged org_1 and org_2 in org_3. Because org_3 has two role contracts for the observation role the role law closes one. At T_1, $Agent_3$ still has all of its capabilities.

MacodoSystemT_1

$MACODOSYSTEM_{traffic}$

$\exists\, org_3 : ORG \bullet organizations = \{org_3\} \land$

$\ldots$

$\exists\, obsCon_1, obsCon_2, obsCon_3,$

$\quad aggrCon_3 : ROLECONT \bullet$

$obsCon_3.rolePosition.role = \{monitor, send\} \land$

$aggrCon_3.rolePosition.role = \{aggregate, send\} \land$

$\ldots$

$agent_3.roleContracts_a = \{obsCon_3, aggrCon_3\} \land$

$agent_3.capabilities_a = \{monitor, aggregate, send\} \land$

$org_3.roleContracts_o =$

$\quad \{cont_{1A}, cont_{2A}, cont_{3A}, cont_{3B}\} \land$

$org_3.rolePositions_o = \varnothing \land$

$\ldots$

After T_1, $camera_3$ fails. A capability update event removes all capabilities of $agent_3$.

$MacodoSystemT_{1a}$

$MACODOSYSTEM_{traffic}$

$\ldots$
$agent_3.capabilities_a = \varnothing \wedge$
$\ldots$

$Agent_3$ now lacks the capabilities required for its role contracts, triggering the self-healing law which removes the invalid role contracts of $agent_3$. As a result, $agent_3$ is no longer active in org_3. The role law opens a new role position for the aggregation role in Org_3.

$MacodoSystemT_2$

$MACODOSYSTEM_{traffic}$

$\exists\, aggrPos : ROLEPOS \bullet$
$\ldots$
$agent_3.roleContracts_a = \varnothing \wedge$
$org_3.roleContracts_o = \{obsCon_1, obsCon_2\} \wedge$
$aggrPos.role = \{aggregate, send\} \wedge$
$org_3.rolePositions_o = \{aggrPos\} \wedge$
$\ldots$

6. Related Work

Roles and organizations are generally acknowledged as valuable abstractions to build multi-agent systems [13]. A number of approaches exist to support organizational evolution and dynamics, such as AGRE [6] and TuCSoN [17]. These approaches, however, do not support inter-organization dynamics and put the responsibility of managing organizations with the agents. The work in this paper uses an organization middleware, which encapsulated the management of organizations as a reusable service. Two particular lines of related research are computational institutions [8] and Law-Governed Interactions [16] (LGI), both using laws, norms or policies to govern interactions among agents. Computational institutions is based on a middleware structure and LGI explicilty supports self-healing.

A number of formal models for roles and organizations have been proposed, for example [4, 14], recognizing formalization as a foundation for analyzing properties such as structure and stability of organizations. These models focus on the behavior and dynamics of agents and organizations. The purpose of the organization model presented in this paper is to provide a basis for organization management.

Formalization is known as a valuable tool in engineering software architectures. In the domain of architectural styles, Abdowd et al. [1] use formalization as a way to enhance effective communication about concepts, allowing formal reasoning about properties, and to help them ask the right questions about architectural concepts. Medvidovic et al. [15] advocate the formal modeling of software architectures at multiple levels, and argue that formal specification of component behavior is an important step towards components as reusable building blocks. Although organizations are not yet recognized as an explicit architectural style and do not directly map to software components, they are a domain specific way of structuring software architectures, and represent reusable abstractions for applications developers.

More recently, formalization has been used as a basis for architecture-based self-management. Bradbury et al. [2] give an overview of formal specification approaches supporting self-management in dynamic software achitectures. In combination with constraint languages, such as Alloy and DynAlloy [7], architectural descriptions have been used to develop self-organizing and self-adaptive systems [10, 9]. Graph grammars and graph transformations have been used to formally describe architectural reconfigurations and change management [18, 19]. The work presented in this paper focuses on a domain specific way of structuring a software system with organizations. Formalized laws, enforced by an organization middleware, allow us to achieve self-adaptivity and self-healing in the domain of dynamic organizations.

7 Conclusions and Future Work

We discuss three topics: (1) what have we learned with respect to self-* properties in general, (2) how has the formal specification contributed to the organization and management model, and (3) how can the formal specification be used as a foundation for maturing the organization middleware. The latter can be considered as future work.

Self-* Properties in General. This paper formally specified self-adaptation and self-healing for context-driven dynamic organizations. The specification is based on a particular set of organization dynamics, such as merging and splitting, which support the dynamics required for context-driven dynamic organizations. Although this specification is not applicable to every type of self-* property, it does show some underlying principles for middleware enabled self-* properties. It shows how self-* properties can be realized as an interplay between a number of basic reflective adaptation processes and how there is a great overlap between different self-* properties. For example, adaptation processes used to achieve self-adaptive organizations with respect to changing traffic context, are also used achieve self-healing with respect to failing agents.

Contributions to the Organization and Management Model. The formal specification has clarified and completed a number of concepts. Two important issues were the effects of merging and splitting on context relations, and the relation between self-healing and other self-* properties. These issues have been clarified by formally defining context relations as functions on all agents and organizations in the MACODO system and self-healing and other self-* properties as a set of basic adaptation processes.

Maturing the Organization Middleware. In terms of development, there are three main contributions. First of all, the management model can be used to derive required monitoring and control points from events and laws. These points can be translated into concrete interfaces required by the organization middleware. An example of a monitoring point is the ability to monitor the capabilities of agents. Second, the management model can be used as a basis for distributing the middleware. It shows which information and control is required for realizing the laws. For example, the merging of two organizations, does not require complete knowledge or control of the MACODO system. Finally, the organization and management model can be used to determine which facilities agents require and which facilities agents have to offer to integrate with the organization middleware. For example, agents require access to open role positions but have to provide access to end their role contracts. Application developers can use the formal specification for building application specific self-* properties that rely on organization-related self-* properties.

In terms of validation, the formal specification can be used to evaluate the completeness of the laws and the validity of self-* properties.

References

[1] G. Abowd, R. Allen, and D. Garlan. Formalizing style to understand descriptions of software architecture. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 4(4):319–364, 1995.

[2] J. Bradbury, J. Cordy, J. Dingel, and M. Wermelinger. A survey of self-management in dynamic software architecture specifications. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 28–33. ACM New York, NY, USA, 2004.

[3] CZT, 2008. Community Z Tools. `http://czt.sourceforge.net/`.

[4] V. Dignum, J. Meyer, F. Dignum, and H. Weigand. Formal Specification of Interaction in Agent Societies. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 37–52, 2003.

[5] M. d'Inverno and M. Luck. *Understanding Agent Systems*. SpringerVerlag, 2004.

[6] J. Ferber, F. Michel, and J. Baez. AGRE: Integrating environments with organizations. In *First International Workshop on Environments for Multi-Agent Systems*, volume 3374 of *Lecture Notes in Computer Science*, pages 48–56, New York, NY, USA, 2005. Springer-Verlag.

[7] M. Frias, J. Galeotti, C. Pombo, and N. Aguirre. DynAlloy: upgrading alloy with actions. In *International Conference on Software Engineering: Proceedings of the 27 th international conference on Software engineering*, volume 15, pages 442–451, 2005.

[8] A. Garcia-Camino, P. Noriega, and J. .Rodríguez-Aguilar. Implementing norms in electronic institutions. In *AAMAS '05: Proceedings of the 4th international joint conference on Autonomous agents and multiagent systems*, pages 667–673, New York, NY, USA, 2005. ACM Press.

[9] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *COMPUTER*, pages 46–54, 2004.

[10] I. Georgiadis, J. Magee, and J. Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the first workshop on Self-healing systems*, pages 33–38. ACM Press New York, NY, USA, 2002.

[11] R. Haesevoets, B. Eylen, D. Weyns, A. Helleboogh, T. Holvoet, and W. Joosen. Managing Agent Interactions with Context-Driven Dynamic Organizations. *Engineering Environment-Mediated Multiagent Systems, Lecture Notes in Computer Science. Springer-Verlag*, 2008.

[12] R. Haesevoets, D. Weyns, and T. Holvoet. A formal specification of an organization model and management model for context-driven dynamic organizations. Technical Report CW535, Katholieke Universiteit Leuven, 2009. `http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW535.abs.html`.

[13] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296, 2000.

[14] M. McCallum, W. Vasconcelos, and T. Norman. Organisational change through influence. *Autonomous Agents and Multi-Agent Systems*, 17(2):1–33, 2008.

[15] N. Medvidovic, R. Taylor, and E. Whitehead Jr. Formal Modeling of Software Architectures at Multiple Levels of Abstraction. *ejw*, 714:824–2776.

[16] N. Minsky. On conditions for self-healing in distributed software systems. *Autonomic Computing Workshop*, pages 86–92, 2003.

[17] A. Omicini and A. Ricci. Reasoning about organisation: Shaping the infrastructure. *AI* IA Notizie*, 16(2):7–16, 2003.

[18] G. Taentzer, M. Goedicke, and T. Meyer. Dynamic Change Management by Distributed Graph Transformation: Towards Configurable Distributed Systems. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 179–193, 2000.

[19] M. Wermelinger and J. Fiadeiro. A graph transformation approach to software architecture reconfiguration. *Science of Computer Programming*, 44(2):133–155, 2002.

[20] D. Weyns, R. Haesevoets, B. Van Eylen, A. Helleboogh, T. Holvoet, and W. Joosen. Endogenous versus exogenous self-management. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 41–48. ACM New York, NY, USA, 2008.