

**A Formal Specification of an  
Organization Model and Management  
Model for Context-Driven Dynamic  
Organizations**

*Robrecht Haesevoets*

*Danny Weyns*

*Tom Holvoet*

*Report CW 535, January 2009*



**Katholieke Universiteit Leuven**  
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# A Formal Specification of an Organization Model and Management Model for Context-Driven Dynamic Organizations

*Robrecht Haesevoets*

*Danny Weyns*

*Tom Holvoet*

*Report CW 535, January 2009*

Department of Computer Science, K.U.Leuven

## Abstract

An organization middleware encapsulates the management of dynamic organizations and offers roles and organizations as high-level abstractions to application developers. This report presents a complete formal specification in Z of an organization model and management model for context-driven dynamic organizations, a particular class of dynamic organizations. The organization model describes the abstractions offered by the organization middleware, while the management model describes the desired behavior of the middleware with respect to the management of dynamic organizations.

Context-driven dynamic organizations, allow developers to group agents in organizations which represent collaborations that are driven by a dynamic context. As the context changes, the required collaborations change and the organizations have to be adapted. The management model identifies external inputs, such as changes in context, and a set of reflective adaptation processes, which will adapt the organizations accordingly. We use a decentralized traffic monitoring case to illustrate the organization and management model.

**Keywords :** Software Engineering, Software Architectures, Formal methods, Multi-Agent Systems, Middleware.

**CR Subject Classification :** D.2 , D.2.11, D.2.4

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Organization Model</b>	<b>4</b>
2.1	Basic sets	4
2.2	Roles, Role Positions and Role Contracts	4
2.3	Agent and Organization Context	5
2.4	Agents and Organizations	5
2.5	The MACODO System	6
<b>3</b>	<b>Management Model</b>	<b>7</b>
3.1	External Events	8
3.1.1	Context Update	8
3.1.2	Capability Update	8
3.1.3	Agent Addition/Removal	9
3.1.4	Start New Organization	9
3.1.5	Begin/End Role Contract	10
3.2	Reflective Adaptation Processes and Laws	11
3.2.1	Merge Law	11
3.2.2	Split Law	12
3.2.3	Role Law	12
3.2.4	Self-Healing Law	13
3.2.5	Organization Clean Up Law	14
<b>4</b>	<b>Additional Help Functions</b>	<b>15</b>
4.1	Active In	15
4.2	Between	15
4.3	Local To	15
4.4	Mergeable State	15
4.5	Differentiated State	15
4.6	Update Role Positions	16
4.7	Update Agents and Organizations	16
4.8	Paths and Disjoint	16
4.9	Split Agent Group	17
4.10	Outdated Role Positions and Role Contracts	17
4.11	Update Organization Role Positions	18
4.12	Obsolete Role Contracts	18
4.13	Invalid Role Contracts	18
<b>5</b>	<b>Traffic Monitoring Example</b>	<b>20</b>
5.1	Agent Names, Organization Names, Context, Capabilities and Roles	21
5.2	The MACODO System at $T_0$	21
5.2.1	Start Organization	22
5.2.2	Role Law	23
5.2.3	Context Update	24
5.3	The MACODO System at $T_1$	25
5.3.1	Split Law	25
5.3.2	Role Law	26
5.3.3	Context Update	27
5.4	The MACODO System at $T_2$	28
5.4.1	Merge Law	28

5.4.2	Role Law	29
5.5	The MACODO System at $T_3$	30
5.5.1	Capability Update	30
5.5.2	Self-Healing Law	30

# 1 Introduction

An organization middleware [4, 3] encapsulates the management of dynamic organizations and offers roles and organizations as high-level abstractions to application developers. In this report we present a complete formal specification in  $Z$  of an organization model and management model for context-driven dynamic organizations, a particular class of dynamic organizations. The organization model describes the abstractions offered by the organization middleware, while the management model describes the desired behavior of the middleware with respect to the management of organizations.

Context-driven dynamic organizations, allow developers to group agents in organizations which represent collaborations that are driven by a dynamic context. As the context changes, the required collaborations change and the organizations have to be adapted. The management model identifies external inputs, such as changes in context, and a set of reflective adaptation processes, which will adapt the organizations accordingly.

An example application domain is decentralized traffic monitoring. A number of cameras are distributed over a road network, each with a limited viewing range. Because there is no central control, cameras have to collaborate to observe larger phenomena such as traffic jams. The dynamic nature of traffic requires dynamic collaborations. This report uses a traffic monitoring case to illustrate the organization and management model.

We use  $Z$  as formal specification language.  $Z$  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 [2]. We used CZT tools [1] to edit and type check the specification.

**Overview** Section 2 introduces the formal organization model for context-driven dynamic organizations. Section 3 presents the formal management model, which uses the organizations model. Section 4 explains the additional help functions, used by the organization model and management model. Section 5 illustrates the organization model and management model in a traffic monitoring case.

# Contents

## 2 Organization Model

An overview of the organization model is shown in Fig. 1.

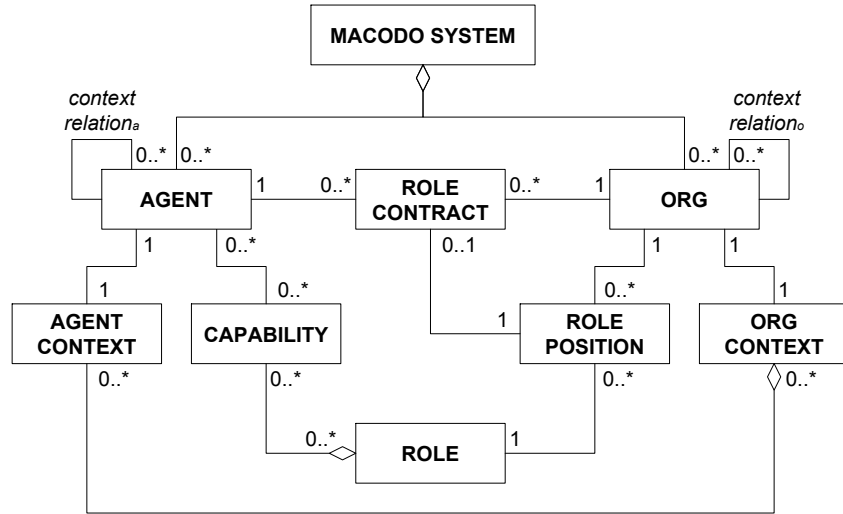


Figure 1: Organization model for context-driven dynamic organizations.

### 2.1 Basic sets

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.

$$\begin{aligned}
 & [AGENTNAME, ORGNAME, CAPABILITY, STATE, LOCATION] \\
 & CONTEXTRELATIONS_a == AGENT \leftrightarrow AGENT \\
 & CONTEXTRELATIONS_o == ORG \leftrightarrow ORG
 \end{aligned}$$

### 2.2 Roles, Role Positions and Role Contracts

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

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

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  $Z$ .

<i>ROLEPOS</i>
<i>role</i> : <i>ROLE</i>
<i>orgName</i> : <i>ORGNAME</i>
<i>role</i> $\neq \emptyset$

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

<p><i>ROLECONT</i></p> <p><i>agentName</i> : <i>AGENTNAME</i></p> <p><i>rolePosition</i> : <i>ROLEPOS</i></p>
---

### 2.3 Agent and Organization Context

The context of an agent consists of the current state of their local environment and a location. The location refers to the locality of the context and can be physical or virtual.

<p><i>AGENTCONTEXT</i></p> <p><i>state</i> : <i>STATE</i></p> <p><i>location</i> : <i>LOCATION</i></p>
--

The context of an organization is the aggregation of the context of all agents that are active in the organization. An agent is active in an organization if it has one or more contracts in the organization.

$ORGCONTEXT == \mathbb{P} AGENTCONTEXT$

### 2.4 Agents and Organizations

Agents have a number of capabilities, allowing them to realize certain roles. In the model, available capabilities are assumed to be unlimited resources of the agent. Before agent can enter into a contract for a role position, they must have the required capabilities. When an agent fails or degrades, it loses some of its capabilities. The abstraction of capability is used to support self-healing in the management model.

<p><i>AGENT</i></p> <p><i>name<sub>a</sub></i> : <i>AGENTNAME</i></p> <p><i>context<sub>a</sub></i> : <i>AGENTCONTEXT</i></p> <p><i>capabilities<sub>a</sub></i> : <math>\mathbb{P} CAPABILITY</math></p> <p><i>roleContracts<sub>a</sub></i> : <math>\mathbb{P} ROLECONT</math></p> <hr/> <p><math>\forall rc : roleContracts_a \bullet rc.agentName = name_a</math></p>
---

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

<p><i>ORG</i></p> <p><i>name<sub>o</sub></i> : <i>ORGNAME</i></p> <p><i>context<sub>o</sub></i> : <i>ORGCONTEXT</i></p> <p><i>roleContracts<sub>o</sub></i> : <math>\mathbb{P} ROLECONT</math></p> <p><i>rolePositions<sub>o</sub></i> : <math>\mathbb{P} ROLEPOS</math></p> <hr/> <p><math>\forall rp : rolePositions_o \bullet rp.orgName = name_o</math></p> <p><math>\forall rc : roleContracts_o \bullet rc.rolePosition.orgName = name_o</math></p>
---

Agents can only participate in organization dynamics if they are active in an organization. To become active, 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. From here on, 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.

## 2.5 The 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.

$$\begin{array}{l}
 \text{MACODOSYSTEM} \\
 \text{agents} : \mathbb{P} \text{AGENT} \\
 \text{organizations} : \mathbb{P} \text{ORG} \\
 \text{contextRelations}_a : \text{CONTEXTRELATIONS}_a \\
 \text{contextRelations}_o : \text{CONTEXTRELATIONS}_o \\
 \text{uniquenames}_o : \mathbb{P} \text{ORGNAME} \\
 \text{uniquenames}_o = \{n : \text{ORGNAME} \mid \forall o : \text{organizations} \bullet n \neq o.\text{name}_o\} \\
 \forall x, y : \text{agents} \bullet x.\text{name}_a = y.\text{name}_a \Leftrightarrow x = y \\
 \forall x, y : \text{organizations} \bullet x.\text{name}_o = y.\text{name}_o \Leftrightarrow x = y
 \end{array}$$

Context relations 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. This is because it does not only depend on the context of both entities involved in the relation, but on the context of all entities within the MACODO system.

An example of a locality relation is the neighbor relation in the traffic monitoring case. the MACODO system with context relations for the traffic monitoring case, defined as functions on all the agents and organizations in the MACODO system is shown below. The context relation only considers agents which are currently active in an organization, because these are the only cameras engaged in monitoring traffic. Two agents are neighboring when they are local to each other and there is no other active agents with a location between them. Two organizations are neighboring if there exists at least one neighbor relation between two agents, one of each organization.

$$\begin{array}{l}
 \text{MACODOSYSTEM}_{\text{traffic}} \\
 \text{MACODOSYSTEM} \\
 \forall x, y : \text{AGENT} \bullet (x, y) \in \text{contextRelations}_a \Leftrightarrow \\
 \quad x \neq y \wedge \exists ox, oy : \text{organizations} \bullet x \text{ activeIn } ox \wedge y \text{ activeIn } oy \wedge \\
 \quad x.\text{context}_a.\text{location} \text{ localTo } y.\text{context}_a.\text{location} \wedge \\
 \quad \neg \exists z : \text{agents} \bullet \exists oz : \text{organizations} \bullet z \text{ activeIn } oz \wedge \\
 \quad \quad z.\text{context}_a.\text{location} \text{ between } (x.\text{context}_a.\text{location}, y.\text{context}_a.\text{location}) \\
 \forall x, y : \text{ORG} \bullet (x, y) \in \text{contextRelations}_o \Leftrightarrow \\
 \quad x \in \text{organizations} \wedge y \in \text{organizations} \wedge \\
 \quad x \neq y \wedge \exists ax, ay : \text{agents} \bullet ax \text{ activeIn } x \wedge ay \text{ activeIn } y \wedge (ax, ay) \in \text{contextRelations}_a
 \end{array}$$

### 3 Management Model

An overview of the management model is shown in Fig. 2. External events 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 of organizations and the realization of self-adaptive and self-healing organizations, is the result of an interplay between the laws, the agents and the dynamic environment. The laws specify the desired behavior of the organization middleware.

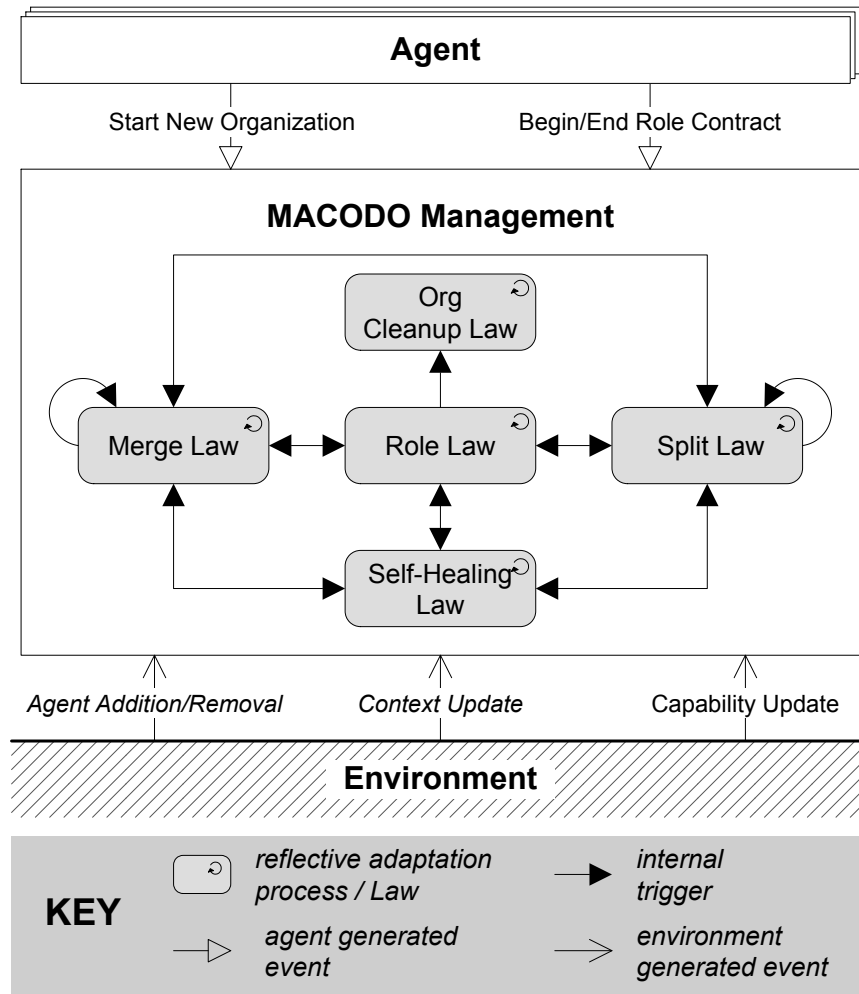


Figure 2: A management model for context-driven dynamic organizations.

Within the domain of context-driven dynamic organizations, there are five types of external events and five laws. The external events, generated by the agents and the environment, drive the laws, by bringing the MACODO system in undesired or inconsistent states. The laws, in turn, ensure the MACODO system returns to a desired and consistent state. Each of the laws has its own specific responsibility. For example, the merge law 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.

## 3.1 External Events

### 3.1.1 Context Update

A context update <sup>1</sup> represents an update to the local context of an agent, active in the MACODO system. Because the context of an organization is the aggregation of agent context, the update also affects all organizations in which the agent is active.

The operation schema specifying a context update is given below. 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 adding a question mark to the names of input variables in operation schemas. The agent name should belong to an agent active in the MACODO system, which is the condition for the operation schema to apply. The operation schema replaces the agent and the affected organizations in the MACODO system with an updated version. The primed components in the schema denote the changes after the operation. Because context relations are defined as a function on the sets of agents and organizations in the MACODO system, they are automatically adapted as these sets change.

<i>ContextUpdate</i>
$\Delta$ MACODOSYSTEM <i>agentname?</i> : AGENTNAME <i>context?</i> : AGENTCONTEXT
$\exists oagent : agents \bullet oagent.name_a = agentname? \wedge$ $\exists uagent : AGENT \bullet uagent = updateAgent_{context}(oagent, context?) \wedge$ $\exists affectedorgs, uorgs : \mathbb{P} ORG \bullet$ $affectedorgs = \{o : organizations \mid oagent \text{ activeIn } o\} \wedge$ $uorgs = \{ao : affectedorgs \bullet$ $updateOrg_{context}(ao, ao.context_o \setminus \{oagent.context_a\} \cup \{context?\})\} \wedge$ $organizations' = organizations \setminus affectedorgs \cup uorgs \wedge$ $agents' = agents \setminus \{oagent\} \cup \{uagent\}$

### 3.1.2 Capability Update

A capability update <sup>1</sup> is an abstract representation of the failure, degradation or recovery of an agent and its resources. A failing or degrading agent loses capabilities, while a recovering agent regains capabilities. The operation schema takes an agent name and an updated set of capabilities as input, and replace the corresponding agent with an updated version. If the agent loses capabilities, required by its current role contracts, this update triggers the self-healing law.

<i>CapabilityUpdate</i>
$\Delta$ MACODOSYSTEM <i>agentname?</i> : AGENTNAME <i>capabilities?</i> : $\mathbb{P} CAPABILITY$
$\exists oa : agents \bullet oa.name_a = agentname? \wedge$ $\exists ua : AGENT \bullet$ $ua.name_a = oa.name_a \wedge$ $ua.capabilities_a = capabilities? \wedge$ $ua.roleContracts_a = oa.roleContracts_a \wedge$ $ua.context_a = oa.context_a \wedge$ $organizations' = organizations \wedge$ $agents' = agents \setminus \{oa\} \cup \{ua\}$

<sup>1</sup>In a real middleware, context updates and capability updates may be passed to the middleware through the agents, instead of directly from the environment, but this is not relevant to the model.

### 3.1.3 Agent Addition/Removal

Agents can be added or removed from the MACODO system. The model only allows the addition or removal of isolated agents, which are not engaged in any role contracts. The addition or removal of agents can be the decision of an administrator or an external policy.

<p><i>AgentAddition</i></p> <p><math>\Delta</math>MACODOSYSTEM</p> <p><i>agent?</i> : AGENT</p>
<p><math>\neg \exists a : agents \bullet a.name_a = agent?.name_a</math></p> <p><math>agent?.roleContracts_a = \emptyset</math></p> <p><math>agents' = agents \cup \{agent?\}</math></p> <p><math>organizations' = organizations</math></p>

<p><i>AgentRemoval</i></p> <p><math>\Delta</math>MACODOSYSTEM</p> <p><i>agent?</i> : AGENT</p>
<p><math>agent? \in agents</math></p> <p><math>agent?.roleContracts_a = \emptyset</math></p> <p><math>agents' = agents \setminus \{agent?\}</math></p> <p><math>organizations' = organizations</math></p>

### 3.1.4 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. An agent gets a default role contract, which is application specific, in the new organization. The event of starting a new organization, may trigger the merge law.

<p><i>StartNewOrganization</i></p> <p><math>\Delta</math>MACODOSYSTEM</p> <p><i>agent?</i> : AGENT</p> <p><i>defaultRole</i> : ROLE</p>
<p><math>agent? \in agents \wedge defaultRole \subseteq agent?.capabilities_a</math></p> <p><math>\exists norg : ORG; rp : ROLEPOS; rc : ROLECONT \bullet</math></p> <p><math>norg.name_o \in uniquenames_o \wedge</math></p> <p><math>rp.role = defaultRole \wedge rp.orgName = norg.name_o \wedge</math></p> <p><math>rc.agentName = agent?.name_a \wedge rc.rolePosition = rp \wedge</math></p> <p><math>norg.context_o = \{agent?.context_a\} \wedge</math></p> <p><math>norg.roleContracts_o = \{rc\} \wedge</math></p> <p><math>norg.rolePositions_o = \emptyset \wedge</math></p> <p><math>\exists uagent : AGENT \bullet</math></p> <p><math>uagent = updateAgent_{contracts}(agent?, agent?.roleContracts_a \cup \{rc\}) \wedge</math></p> <p><math>agents' = agents \setminus \{agent?\} \cup \{uagent\} \wedge</math></p> <p><math>organizations' = organizations \cup \{norg\}</math></p>

An example of an application specific role contract is shown below.

|  $roleA : ROLE$

$StartNewOrganization_{applicationSpecific}$ $StartNewOrganization$
$defaultRole = roleA$

### 3.1.5 Begin/End Role Contract

An agent in the MACODO system 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, and therefor is no longer active in the organization, the agent context is removed from the organization context.

$BeginContract$ $\Delta MACODOSYSTEM$ $roleContract? : ROLECONT$
$\exists oagent : agents \bullet$ $oagent.name_a = roleContract?.agentName \wedge$ $roleContract?.rolePosition.role \subseteq oagent.capabilities_a \wedge$ $\exists oorg : organizations; uorg : ORG \bullet$ $oagent \text{ activeIn } oorg \wedge$ $roleContract?.rolePosition \in oorg.rolePositions_o \wedge$ $\exists uagent : AGENT \bullet$ $uagent.name_a = oagent.name_a \wedge$ $uagent.context_a = oagent.context_a \wedge$ $uagent.capabilities_a = oagent.capabilities_a \wedge$ $uagent.roleContracts_a = oagent.roleContracts_a \cup \{roleContract?\} \wedge$ $\exists uorg : ORG \bullet$ $uorg.name_o = oorg.name_o \wedge$ $uorg.context_o = oorg.context_o \cup \{uagent.context_a\} \wedge$ $uorg.rolePositions_o = oorg.rolePositions_o \setminus \{roleContract?.rolePosition\} \wedge$ $uorg.roleContracts_o = oorg.roleContracts_o \cup \{roleContract?\} \wedge$ $organizations' = organizations \setminus \{oorg\} \cup \{uorg\} \wedge$ $agents' = agents \setminus \{oagent\} \cup \{uagent\}$

*EndContract*

$\Delta$ MACODOSYSTEM

*roleContract?* : ROLECONT

$\exists oagent : agents \bullet oagent.name_a = roleContract?.agentName \wedge$   
 $\exists oorg : organizations \bullet roleContract? \in oorg.roleContracts_o \wedge$   
 $\exists uagent : AGENT \bullet$   
     $uagent.name_a = oagent.name_a \wedge$   
     $uagent.context_a = oagent.context_a \wedge$   
     $uagent.capabilities_a = oagent.capabilities_a \wedge$   
     $uagent.roleContracts_a = oagent.roleContracts_a \setminus \{roleContract?\} \wedge$   
 $\exists uorg : ORG \bullet$   
     $uorg.name_o = oorg.name_o \wedge$   
     $uorg.rolePositions_o = oorg.rolePositions_o \cup \{roleContract?.rolePosition\} \wedge$   
     $uorg.roleContracts_o = oorg.roleContracts_o \setminus \{roleContract?\} \wedge$   
     $uorg.context_o = \{a : AGENT \mid a \text{ activeIn } uorg \bullet a.context_a\} \wedge$   
 $organizations' = organizations \setminus \{oorg\} \cup \{uorg\} \wedge$   
 $agents' = agents \setminus \{oagent\} \cup \{uagent\}$

## 3.2 Reflective Adaptation Processes and Laws

### 3.2.1 Merge Law

The merge law takes two organizations as input and merges these organizations if their context is mergeable and related. The merging itself consists of removing the two input organizations from the MACODO system and adding a new merged organization. The merged organization, consists of the union of context, role positions and role contracts of the two input organizations. If the role contracts and role positions of the merged organization are incorrect, it triggers the role law.

*MergeLaw*

$\Delta$ MACODOSYSTEM

*org1?, org2?* : ORG

$(org1?, org2?) \in contextRelations_o \wedge mergeableState_o (org1?, org2?)$   
 $\exists morg : ORG \bullet$   
     $morg.name_o \in uniquenames_o \wedge$   
     $morg.rolePositions_o =$   
         $updateOrgPositions(org1?.rolePositions_o \cup org2?.rolePositions_o, morg.name_o) \wedge$   
     $morg.roleContracts_o = org1?.roleContracts_o \cup org2?.roleContracts_o \wedge$   
     $morg.context_o = org1?.context_o \cup org2?.context_o \wedge$   
 $\exists oagents, uagents : \mathbb{P}AGENT \bullet$   
     $oagents = \{a : agents \mid a \text{ activeIn } org1? \vee a \text{ activeIn } org2?\} \wedge$   
     $uagents = \{oa : oagents \bullet updateAgent_{contracts}(oa,$   
         $\{mrc : morg.roleContracts_o \mid mrc.agentName = oa.name_a\} \cup$   
         $\{orc : oa.roleContracts_a \mid orc \notin org1?.roleContracts_o \wedge orc \notin org2?.roleContracts_o\})\} \wedge$   
 $organizations' = organizations \setminus \{org1?, org2?\} \cup \{morg\} \wedge$   
 $agents' = agents \setminus oagents \cup uagents$

### 3.2.2 Split Law

The split law splits an organization in one or more suborganizations, if the context state of the agents active in the organization is differentiated, or when the organizations is disjoint with respect to the current context relations in the MACODO system. The organization being split is removed from the MACODO system, and a number of smaller organizations are added to the MACODO system. The split law ensures each of the smaller organizations has a uniform and related context among their active agents. The correct role contracts and role positions in the smaller organizations are the responsibility of the role law. The split is executed at the granularity of role contracts.

$$\begin{array}{l}
 \textit{SplitLaw} \\
 \Delta \textit{MACODOSYSTEM} \\
 \textit{org?} : \textit{ORG} \\
 \hline
 \textit{differentiatedState}_o \textit{org?} \vee \textit{org?} \in \textit{disjoint}_o(\textit{contextRelations}_a) \\
 \exists \textit{oagents} : \mathbb{P} \textit{AGENT} \bullet \textit{oagents} = \{a : \textit{agents} \mid a \textit{activeIn} \textit{org?}\} \wedge \\
 \exists \textit{splittedagents} : \mathbb{P} \mathbb{P} \textit{AGENT} \bullet \textit{splittedagents} = \textit{splitAgentGroup}(\textit{contextRelations}_a, \textit{oagents}) \wedge \\
 \exists \textit{splittedorgs} : \mathbb{P} \textit{ORG} \bullet \\
 \quad \textit{splittedorgs} = \{\textit{sorg} : \textit{ORG} \mid \exists \textit{group} : \textit{splittedagents} \bullet \\
 \quad \quad \textit{sorg.name}_o \in \textit{uniquenames}_o \wedge \\
 \quad \quad \textit{sorg.context}_o = \{a : \textit{group} \bullet a.\textit{context}_a\} \wedge \\
 \quad \quad \textit{sorg.roleContracts}_o = \\
 \quad \quad \quad \{\textit{rc} : \textit{ROLECONT}; a : \textit{group} \mid \textit{rc} \in a.\textit{roleContracts}_a \bullet \textit{rc}\} \wedge \\
 \quad \quad \textit{sorg.rolePositions}_o = \emptyset\} \wedge \\
 \neg \exists \textit{o1}, \textit{o2} : \textit{splittedorgs} \bullet \textit{o1} \neq \textit{o2} \wedge \textit{o1.name}_o = \textit{o2.name}_o \wedge \\
 \exists \textit{uagents} : \mathbb{P} \textit{AGENT} \bullet \\
 \quad \textit{uagents} = \{\textit{oa} : \textit{oagents} \bullet \textit{updateAgent}_{\textit{contracts}}(\textit{oa}, \\
 \quad \quad \{\textit{src} : \textit{ROLECONT} \mid \exists \textit{so} : \textit{splittedorgs} \bullet \\
 \quad \quad \quad \textit{src} \in \textit{so.roleContracts}_o \wedge \textit{src.agentName} = \textit{oa.name}_a\} \cup \\
 \quad \quad \quad \{\textit{orc} : \textit{oa.roleContracts}_a \mid \textit{orc.rolePosition.orgName} \neq \textit{org?.name}_o\})\} \wedge \\
 \textit{agents}' = \textit{agents} \setminus \textit{oagents} \cup \textit{uagents} \wedge \\
 \textit{organizations}' = \textit{organizations} \setminus \{\textit{org?}\} \cup \textit{splittedorgs}
 \end{array}$$

### 3.2.3 Role Law

The role law ensures the required roles are being played in an organization, or at least provides the required role positions for them. The role law opens role positions for roles which are lacking and close role positions for roles which are no longer needed in an organization. The role law can also close obsolete role contracts, but it can not create any new role contracts. The creation of a role contract still lies within the decision of the agents. The role law can be triggered by all the other laws and different external events. What the required roles in an organization are, depends on the size and context of the organization, and is defined in application specific functions.

*RoleLaw*

$\Delta$ MACODOSYSTEM

$org? : ORG$

$org? \in organizations$

$org? \in outdatedRolePositions_o \vee org? \in outdatedRolePositions_o$

$\exists upositions : \mathbb{P} ROLEPOS \bullet$

$upositions = updateOrgRolePositions(org?) \wedge$

$\exists obsoletecontracts : \mathbb{P} ROLECONT \bullet$

$obsoletecontracts = obsoleteRoleContracts_o(org?) \wedge$

$\exists uorg : ORG \bullet$

$uorg.name_o = org?.name_o \wedge$

$uorg.roleContracts_o = org?.roleContracts_o \setminus obsoletecontracts \wedge$

$uorg.context_o = \{a : agents \mid a \text{ activeIn } uorg \bullet a.context_a\} \wedge$

$uorg.rolePositions_o = upositions \wedge$

$\exists oaagents, uagents : \mathbb{P} AGENT \bullet$

$oaagents = \{a : agents \mid \exists rc : obsoletecontracts \bullet rc \in a.roleContracts_a\} \wedge$

$uagents = \{oa : oaagents \bullet updateAgent_{contracts}(oa, oa.roleContracts_a \setminus obsoletecontracts)\} \wedge$

$organizations' = organizations \setminus \{org?\} \cup \{uorg\} \wedge$

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

### 3.2.4 Self-Healing Law

The self-healing law, ensures the consistency within organizations with respect to agents that no longer have the capabilities required by their current role contracts. It removes all invalid role contracts, for which the agent lacks the required capabilities. It applies to any agent in the MACODO system which has one or more invalid role contract. The schema removes the invalid role contracts from the agent and updates the affected organizations. It may also remove the context of the agent from the context of the affected organizations, possibly making the context of this organization unrelated and triggering the split law. Note that this law does not open any new role positions for the 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 remain in the MACODO system, until removed by an external event, or regain capabilities when its resources have been repaired.

*SelfHealingLaw*

$\Delta$ MACODOSYSTEM

$agent? : AGENT$

$\exists invalidContracts : \mathbb{P} ROLECONT \bullet invalidContracts = invalidroleContracts(agent?) \wedge$

$invalidContracts \neq \emptyset \wedge$

$\exists uagent : AGENT \bullet$

$uagent = updateAgent_{contracts}(agent?, agent?.roleContracts_a \setminus invalidContracts) \wedge$

$\exists affectedorgs, uorgs : \mathbb{P} ORG \bullet$

$affectedorgs = \{o : organizations \mid \exists rc : invalidContracts \bullet rc \in o.roleContracts_o\} \wedge$

$uorgs = \{uo : ORG \mid \exists ao : affectedorgs \bullet$

$uo.name_o = ao.name_o \wedge$

$uo.context_o = \{a : AGENT \mid a \text{ activeIn } uo \bullet a.context_a\} \wedge$

$uo.roleContracts_o = ao.roleContracts_o \setminus invalidContracts \wedge$

$uo.rolePositions_o = ao.rolePositions_o\} \wedge$

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

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

### 3.2.5 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 capabilities required by this role contract.

*OrgCleanupLaw*

$\Delta$ MACODOSYSTEM

*org?* : ORG

*org?*  $\in$  *organizations*

*org?*.*roleContracts*<sub>*o*</sub> =  $\emptyset$

*agents'* = *agents*

*organizations'* = *organizations* \ {*org?*}

## 4 Additional Help Functions

This section introduces a number of additional help functions, used in the organization model and management model.

### 4.1 Active In

A function to determine whether an agent is active in an organization.

relation (*\_ activeIn \_*)

$$\frac{}{| \_ activeIn \_ : AGENT \leftrightarrow ORG} \\ \forall a : AGENT; o : ORG \bullet a \ activeIn o \Leftrightarrow \exists rc : o.roleContracts_o \bullet rc.agentName = a.name_a$$

### 4.2 Between

An application specific function to determine whether one location is located between two others.

relation (*\_ between \_*)

$$| \_ between \_ : LOCATION \leftrightarrow LOCATION \times LOCATION$$

### 4.3 Local To

An application specific function to determine whether one location is close to another.

relation (*\_ localTo \_*)

$$| \_ localTo \_ : LOCATION \leftrightarrow LOCATION$$

### 4.4 Mergeable State

A function to determine whether two organizations have a mergeable context state.

relation (*mergeableState\_o \_*)

$$\frac{}{| mergeableState_o \_ : ORG \leftrightarrow ORG} \\ \forall o1, o2 : ORG \bullet mergeableState_o (o1, o2) \Leftrightarrow \\ \neg \exists a1, a2 : AGENT \mid a1 \ activeIn o1 \wedge a2 \ activeIn o2 \bullet a1.context_a.state \neq a2.context_a.state$$

### 4.5 Differentiated State

A function to determine whether an organization have a differentiated context state.

relation (*differentiatedState\_o \_*)

$$\frac{}{| differentiatedState_o \_ : \mathbb{P} ORG} \\ \forall o : ORG \bullet differentiatedState_o o \Leftrightarrow \\ \exists x, y : AGENT \bullet x \ activeIn o \wedge y \ activeIn o \wedge x.context_a.state \neq y.context_a.state$$

A function to determine whether a group of agents have a differentiated context state.

$$\frac{}{| differentiatedState_a : \mathbb{P} \mathbb{P} AGENT} \\ differentiatedState_a = \{group : \mathbb{P} AGENT \mid \exists a1, a2 : group \bullet a1.context_a.state \neq a2.context_a.state\}$$

## 4.6 Update Role Positions

A function to update a set of role positions with a new organization name.

$$\begin{array}{|l} \hline \text{updateOrgPositions} : \mathbb{P} \text{ROLEPOS} \times \text{ORGNAME} \rightarrow \mathbb{P} \text{ROLEPOS} \\ \hline \forall rps : \mathbb{P} \text{ROLEPOS}; \text{on} : \text{ORGNAME} \bullet \\ \text{updateOrgPositions}(rps, \text{on}) = \{x : \text{ROLEPOS} \mid \exists y : rps \bullet x.\text{role} = y.\text{role} \wedge x.\text{orgName} = \text{on}\} \end{array}$$

## 4.7 Update Agents and Organizations

A set of functions to update agents and organizations with a new context, new role contracts or new role positions.

$$\begin{array}{|l} \hline \text{updateAgent}_{\text{context}} : \text{AGENT} \times \text{AGENTCONTEXT} \rightarrow \text{AGENT} \\ \hline \forall a, ua : \text{AGENT}; \text{context} : \text{AGENTCONTEXT} \bullet \\ \text{updateAgent}_{\text{context}}(a, \text{context}) = ua \Leftrightarrow \\ ua.\text{name}_a = a.\text{name}_a \wedge ua.\text{context}_a = \text{context} \wedge \\ ua.\text{capabilities}_a = a.\text{capabilities}_a \wedge ua.\text{roleContracts}_a = a.\text{roleContracts}_a \end{array}$$

$$\begin{array}{|l} \hline \text{updateAgent}_{\text{contracts}} : \text{AGENT} \times \mathbb{P} \text{ROLECONT} \rightarrow \text{AGENT} \\ \hline \forall a, ua : \text{AGENT}; \text{contracts} : \mathbb{P} \text{ROLECONT} \bullet \\ \text{updateAgent}_{\text{contracts}}(a, \text{contracts}) = ua \Leftrightarrow \\ ua.\text{name}_a = a.\text{name}_a \wedge ua.\text{context}_a = a.\text{context}_a \wedge \\ ua.\text{capabilities}_a = a.\text{capabilities}_a \wedge ua.\text{roleContracts}_a = \text{contracts} \end{array}$$

$$\begin{array}{|l} \hline \text{updateOrg}_{\text{context}} : \text{ORG} \times \text{ORGCONTEXT} \rightarrow \text{ORG} \\ \hline \forall o, uo : \text{ORG}; \text{context} : \text{ORGCONTEXT} \bullet \\ \text{updateOrg}_{\text{context}}(o, \text{context}) = uo \Leftrightarrow \\ uo.\text{name}_o = o.\text{name}_o \wedge uo.\text{context}_o = \text{context} \wedge \\ uo.\text{roleContracts}_o = o.\text{roleContracts}_o \wedge uo.\text{rolePositions}_o = o.\text{rolePositions}_o \end{array}$$

$$\begin{array}{|l} \hline \text{updateOrg}_{\text{positions}} : \text{ORG} \times \mathbb{P} \text{ROLEPOS} \rightarrow \text{ORG} \\ \hline \forall o, uo : \text{ORG}; \text{positions} : \mathbb{P} \text{ROLEPOS} \bullet \\ \text{updateOrg}_{\text{positions}}(o, \text{positions}) = uo \Leftrightarrow \\ uo.\text{name}_o = o.\text{name}_o \wedge uo.\text{context}_o = o.\text{context}_o \wedge \\ uo.\text{roleContracts}_o = o.\text{roleContracts}_o \wedge uo.\text{rolePositions}_o = \text{positions} \end{array}$$

## 4.8 Paths and Disjoint

A function to determine whether a sequence of agents is a path in an organization, with respect to the context relations between the agents.

relation ( $\text{path}_o \_$ )

$$\begin{array}{|l} \hline \text{path}_o \_ : (\text{ORG} \times \text{CONTEXTRELATIONS}_a) \leftrightarrow \text{seq AGENT} \\ \hline \forall o : \text{ORG}; \text{relations} : \text{CONTEXTRELATIONS}_a; p : \text{seq AGENT} \bullet \text{path}_o((o, \text{relations}), p) \Leftrightarrow \\ ((\text{head } p) \text{ activeIn } o \wedge (\text{head } p, \text{head } (\text{tail } p)) \in \text{relations} \\ \wedge \text{path}_o((o, \text{relations}), \text{tail } p)) \vee p = \emptyset \end{array}$$

A functions used to determine whether an organization is disjoint, with respect to context relations between the agents.

$$\begin{array}{|l} \hline \text{disjoint}_o : \text{CONTEXTRELATIONS}_a \leftrightarrow \mathbb{P} \text{ORG} \\ \hline \forall o : \text{ORG}; \text{relations} : \text{CONTEXTRELATIONS}_a \bullet o \in \text{disjoint}_o(\text{relations}) \Leftrightarrow \\ \quad \exists x, y : \text{AGENT} \bullet x \text{ activeIn } o \wedge y \text{ activeIn } o \wedge x \neq y \wedge \\ \quad \neg \exists p : \text{seq AGENT} \bullet \text{path}_o((o, \text{relations}), p) \wedge \text{head } p = x \wedge \text{last } p = y \end{array}$$

A function to determine whether a sequence of agents is a path within a group of organizations, with respect to the context relations between the agents.

relation ( $\text{path}_a \_$ )

$$\begin{array}{|l} \hline \text{path}_a \_ : (\mathbb{P} \text{AGENT} \times \text{CONTEXTRELATIONS}_a) \leftrightarrow \text{seq AGENT} \\ \hline \forall \text{group} : \mathbb{P} \text{AGENT}; \text{relations} : \text{CONTEXTRELATIONS}_a; p : \text{seq AGENT} \bullet \\ \quad \text{path}_a((\text{group}, \text{relations}), p) \Leftrightarrow \\ \quad ((\text{head } p) \in \text{group} \wedge (\text{head } p, \text{head } (\text{tail } p)) \in \text{relations} \\ \quad \wedge \text{path}_a((\text{group}, \text{relations}), \text{tail } p)) \vee p = \emptyset \end{array}$$

A functions used to determine whether a group of agents is disjoint, with respect to context relations between the agents.

$$\begin{array}{|l} \hline \text{disjoint}_a : \text{CONTEXTRELATIONS}_a \leftrightarrow \mathbb{P} \mathbb{P} \text{AGENT} \\ \hline \forall \text{group} : \mathbb{P} \text{AGENT}; \text{relations} : \text{CONTEXTRELATIONS}_a \bullet \text{group} \in \text{disjoint}_a(\text{relations}) \Leftrightarrow \\ \quad \exists x, y : \text{AGENT} \bullet x \in \text{group} \wedge y \in \text{group} \wedge x \neq y \wedge \\ \quad \neg \exists p : \text{seq AGENT} \bullet \text{path}_a((\text{group}, \text{relations}), p) \wedge \text{head } p = x \wedge \text{last } p = y \end{array}$$

## 4.9 Split Agent Group

An application specific function to split up a group of agents. The split is based on the individual agent contexts and the current context relations.

$$\begin{array}{|l} \hline \text{splitAgentGroup} : \text{CONTEXTRELATIONS}_a \times \mathbb{P} \text{AGENT} \rightarrow \mathbb{P} \mathbb{P} \text{AGENT} \\ \hline \end{array}$$

For the traffic monitoring case, this function would look like:

$$\begin{array}{|l} \hline \forall \text{relations} : \text{CONTEXTRELATIONS}_a; \text{agents} : \mathbb{P} \text{AGENT} \bullet \\ \quad \exists \text{agentGroups} : \mathbb{P} \mathbb{P} \text{AGENT} \bullet \\ \quad \text{splitAgentGroup}(\text{relations}, \text{agents}) = \text{agentGroups} \wedge \\ \quad \forall \text{group} : \text{agentGroups} \bullet \text{group} \notin \text{disjoint}_a(\text{relations}) \wedge \text{group} \notin \text{differentiatedState}_a \end{array}$$

## 4.10 Outdated Role Positions and Role Contracts

Application specific functions to determine whether an organization needs an update in its current role contracts and role positions

$$\begin{array}{|l} \hline \text{outdatedRolePositions}_o : \mathbb{P} \text{ORG} \\ \text{outdatedRoleContracts}_o : \mathbb{P} \text{ORG} \\ \hline \end{array}$$

For the traffic monitoring case, these functions look like:

$$\begin{aligned}
\text{outdatedRolePositions}_o &= \{o : \text{ORG}; \text{aggrContracts} : \mathbb{P} \text{ROLECONT}; \text{aggrPositions} : \mathbb{P} \text{ROLEPOS} \mid \\
&\text{aggrContracts} = \{rc : o.\text{roleContracts}_o \mid rc.\text{rolePosition.role} = \text{aggrRole}\} \wedge \\
&\text{aggrPositions} = \{rp : o.\text{rolePositions}_o \mid rp.\text{role} = \text{aggrRole}\} \wedge \\
&((\#\text{aggrContracts} \neq 0 \wedge \#\text{aggrPositions} \neq 0) \vee (\#\text{aggrPositions} \geq 2)) \bullet o\}
\end{aligned}$$

$$\begin{aligned}
\text{outdatedRoleContracts}_o &= \{o : \text{ORG}; \text{aggrContracts} : \mathbb{P} \text{ROLECONT} \mid \\
&\text{aggrContracts} = \{rc : o.\text{roleContracts}_o \mid rc.\text{rolePosition.role} = \text{aggrRole}\} \\
&\wedge \#\text{aggrContracts} \geq 2 \bullet o\}
\end{aligned}$$

#### 4.11 Update Organization Role Positions

An application specific function that updates the set of role positions, needed within an organization, based on the current set of role positions, role contracts and current organization context.

$$\text{updateOrgRolePositions} : \text{ORG} \leftrightarrow \mathbb{P} \text{ROLEPOS}$$

For the traffic monitoring case, this function looks like:

$$\begin{aligned}
\forall \text{org} : \text{ORG} \bullet \\
&\exists \text{obsPositions} : \mathbb{P} \text{ROLEPOS} \bullet \\
&\quad \text{obsPositions} = \{rp : \text{org}.\text{rolePositions}_o \mid rp.\text{role} = \text{obsRole}\} \wedge \\
&\exists \text{aggrPos} : \text{ROLEPOS} \bullet \\
&\quad \text{aggrPos.role} = \text{aggrRole} \wedge \text{aggrPos.orgName} = \text{org.name}_o \wedge \\
&\exists \text{aggrContracts} : \mathbb{P} \text{ROLECONT} \bullet \\
&\quad \text{aggrContracts} = \{rc : \text{org}.\text{roleContracts}_o \mid rc.\text{rolePosition.role} = \text{aggrRole}\} \wedge \\
&\quad \text{updateOrgRolePositions}(\text{org}) = \text{obsPositions} \\
&\quad \cup (\text{if } \#\text{aggrContracts} \neq 0 \text{ then } \emptyset \text{ else } \{\text{aggrPos}\})
\end{aligned}$$

#### 4.12 Obsolete Role Contracts

An application specific function that determines the role contracts of an organization that are obsolete and should be closed. Remark: Contracts are initiated by agents, however, however the closing of a contract can be outside the control of an agent.

$$\text{obsoleteRoleContracts}_o : \text{ORG} \leftrightarrow \mathbb{P} \text{ROLECONT}$$

For the traffic monitoring case, this function looks like:

$$\begin{aligned}
\forall \text{org} : \text{ORG} \bullet \\
&\exists \text{aggrContracts} : \mathbb{P} \text{ROLECONT} \bullet \\
&\quad \text{aggrContracts} = \{rc : \text{org}.\text{roleContracts}_o \mid rc.\text{rolePosition.role} = \text{aggrRole}\} \wedge \\
&\exists \text{obsoleteContracts} : \mathbb{P} \text{ROLECONT} \bullet \\
&\quad \text{obsoleteContracts} \subseteq \text{aggrContracts} \wedge \\
&\quad \#\text{obsoleteContracts} = \text{if } \#\text{aggrContracts} \geq 1 \text{ then } \#\text{aggrContracts} - 1 \text{ else } 0 \wedge \\
&\quad \text{obsoleteRoleContracts}_o(\text{org}) = \text{obsoleteContracts}
\end{aligned}$$

#### 4.13 Invalid Role Contracts

A function to determine the invalid role contracts of an agent, with respect to the current capabilities of the agent, which are required by the role contracts.

$invalidroleContracts : AGENT \rightarrow \mathbb{P} ROLECONT$

$\forall a : AGENT \bullet$

$invalidroleContracts(a) = \{rc : a.roleContracts_a \mid$   
 $\exists cap : rc.rolePosition.role \bullet cap \notin a.capabilities_a\}$

## 5 Traffic Monitoring Example

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 on the road network, 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. A running example is shown in Fig. 3, to which we will apply the organization model and management model.

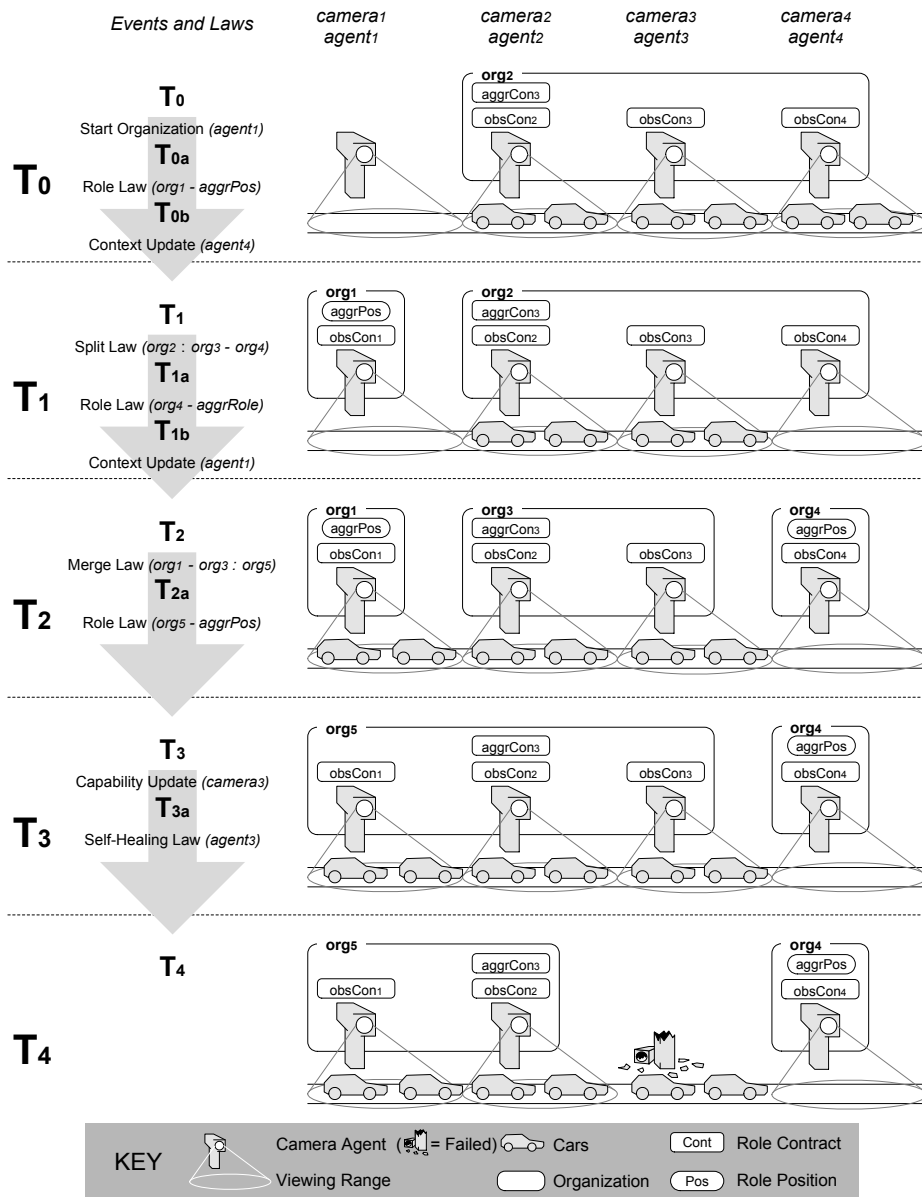


Figure 3: An example of the organization model and management model, applied to the traffic monitoring case.

## 5.1 Agent Names, Organization Names, Context, Capabilities and Roles

$camera_1, camera_2, camera_3, camera_4 : AGENTNAME$   
 $organization_1, organization_2, organization_3, organization_4, organization_5 : ORGNAME$   
 $congested, freeflow, boundedflow : STATE$   
 $loc_1, loc_2, loc_3, loc_4 : LOCATION$   
 $monitor, aggregate, send : CAPABILITY$   
 $obsRole, aggrRole : ROLE$

$loc_1 localTo loc_2 \wedge loc_1 localTo loc_3 \wedge loc_2 localTo loc_3 \wedge loc_2 localTo loc_4 \wedge loc_3 localTo loc_4$   
 $loc_2 between(loc_1, loc_3) \wedge loc_3 between(loc_2, loc_4)$   
 $obsRole = \{monitor, send\}$   
 $aggrRole = \{aggregate, send\}$

## 5.2 The MACODO System at $T_0$

$RoleContracts_{T_0}$   
 $obsCon_2, aggrCon_2, obsCon_3, obsCon_4 : ROLECONT$

$obsCon_2.agentName = camera_2 \wedge obsCon_2.rolePosition.role = obsRole \wedge$   
 $obsCon_2.rolePosition.orgName = organization_2$   
 $aggrCon_2.agentName = camera_2 \wedge aggrCon_2.rolePosition.role = aggrRole \wedge$   
 $aggrCon_2.rolePosition.orgName = organization_2$   
 $obsCon_3.agentName = camera_3 \wedge obsCon_3.rolePosition.role = obsRole \wedge$   
 $obsCon_3.rolePosition.orgName = organization_2$   
 $obsCon_4.agentName = camera_3 \wedge obsCon_4.rolePosition.role = obsRole \wedge$   
 $obsCon_4.rolePosition.orgName = organization_2$

$Agents_{T_0}$

$RoleContracts_{T_0}$   
 $agent_1, agent_2, agent_3, agent_4 : AGENT$

$agent_1.name_a = camera_1 \wedge agent_1.roleContracts_a = \emptyset \wedge$   
 $agent_1.capabilities_a = \{monitor, send\} \wedge$   
 $agent_1.context_a.state = freeflow \wedge agent_1.context_a.location = loc_1$   
 $agent_2.name_a = camera_2 \wedge agent_2.roleContracts_a = \{obsCon_2, aggrCon_2\} \wedge$   
 $agent_2.capabilities_a = \{monitor, aggregate, send\} \wedge$   
 $agent_2.context_a.state = congested \wedge agent_2.context_a.location = loc_2$   
 $agent_3.name_a = camera_3 \wedge agent_3.roleContracts_a = \{obsCon_3\} \wedge$   
 $agent_3.capabilities_a = \{monitor, aggregate, send\} \wedge$   
 $agent_3.context_a.state = congested \wedge agent_3.context_a.location = loc_3$   
 $agent_4.name_a = camera_4 \wedge agent_4.roleContracts_a = \{obsCon_4\} \wedge$   
 $agent_4.capabilities_a = \{monitor, send\} \wedge$   
 $agent_4.context_a.state = congested \wedge agent_4.context_a.location = loc_4$

Its important to note, that both  $agent_1$  and  $agent_4$ , do not have the required capabilities for the aggregation role.

<i>Organizations</i> $T_0$ <i>Agents</i> $T_0$ <i>RoleContracts</i> $T_0$ <i>org</i> <sub>2</sub> : <i>ORG</i>
<i>org</i> <sub>2</sub> . <i>name</i> <sub>o</sub> = <i>organization</i> <sub>2</sub> <i>org</i> <sub>2</sub> . <i>context</i> <sub>o</sub> = { <i>agent</i> <sub>2</sub> . <i>context</i> <sub>a</sub> , <i>agent</i> <sub>3</sub> . <i>context</i> <sub>a</sub> , <i>agent</i> <sub>4</sub> . <i>context</i> <sub>a</sub> } <i>org</i> <sub>2</sub> . <i>roleContracts</i> <sub>o</sub> = { <i>obsCon</i> <sub>2</sub> , <i>aggrCon</i> <sub>2</sub> , <i>obsCon</i> <sub>3</sub> , <i>obsCon</i> <sub>4</sub> } <i>org</i> <sub>2</sub> . <i>rolePositions</i> <sub>o</sub> = $\emptyset$

<i>MACODOSystem</i> $T_0$ <i>MACODOSYSTEM</i> <sub>traffic</sub> <i>Agents</i> $T_0$ <i>Organizations</i> $T_0$
<i>agents</i> = { <i>agent</i> <sub>1</sub> , <i>agent</i> <sub>2</sub> , <i>agent</i> <sub>3</sub> , <i>agent</i> <sub>4</sub> } <i>organizations</i> = { <i>org</i> <sub>2</sub> }

### 5.2.1 Start Organization

At  $T_0$ , *agent*<sub>1</sub> decides to start an organization and a start new organization event is generated.

<i>StartNewOrganization</i> $T_0$ <i>MACODOSystem</i> $T_0$ <i>StartNewOrganization</i>
<i>agent</i> ? = <i>agent</i> <sub>1</sub> <i>defaultRole</i> = <i>obsRole</i>

The resulting state of the MACODO system is shown below. Only the relevant parts that have been changed, are shown.

<i>RoleContracts</i> $T_{0a}$ <i>obsCon</i> <sub>1</sub> , <i>obsCon</i> <sub>2</sub> , <i>aggrCon</i> <sub>2</sub> , <i>obsCon</i> <sub>3</sub> , <i>obsCon</i> <sub>4</sub> : <i>ROLECONT</i>
<i>obsCon</i> <sub>1</sub> . <i>agentName</i> = <i>camera</i> <sub>1</sub> $\wedge$ <i>obsCon</i> <sub>1</sub> . <i>rolePosition.role</i> = <i>obsRole</i> $\wedge$ <i>obsCon</i> <sub>1</sub> . <i>rolePosition.orgName</i> = <i>organization</i> <sub>1</sub> ...

<i>Agents</i> $T_{0a}$ <i>RoleContracts</i> $T_{0a}$ <i>agent</i> <sub>1</sub> , <i>agent</i> <sub>2</sub> , <i>agent</i> <sub>3</sub> , <i>agent</i> <sub>4</sub> : <i>AGENT</i>
<i>agent</i> <sub>1</sub> . <i>roleContracts</i> <sub>a</sub> = { <i>obsCon</i> <sub>1</sub> } ...

$OrganizationsT_{0a}$ $AgentsT_{0a}$ $RoleContractsT_{0a}$ $org_1, org_2 : ORG$
$org_1.name_o = organization_1 \wedge org_1.context_o = \{agent_1.context_a\} \wedge$ $org_1.roleContracts_o = \{obsCon_1\} \wedge org_1.rolePositions_o = \emptyset$ $\dots$

$MACODOSystemT_{0a}$ $MACODOSYSTEM_{traffic}$ $AgentsT_{0a}$ $OrganizationsT_{0a}$
$agents = \{agent_1, agent_2, agent_3, agent_4\}$ $organizations = \{org_1, org_2\}$

### 5.2.2 Role Law

At  $T_{0a}$ , the role law is triggered for  $org_1$ , because does not have a role contract or role position for the aggregation role.

$RoleLawT_{0a}$ $MACODOSystemT_{0a}$ $RoleLaw$
$org? = org_1$

This results in the following state for the MACODO system.

$RoleContractsT_{0b}$ $obsCon_1, obsCon_2, aggrCon_2, obsCon_3, obsCon_4 : ROLECONT$
$\dots$

$AgentsT_{0b}$ $RoleContractsT_{0b}$ $agent_1, agent_2, agent_3, agent_4 : AGENT$
$\dots$

$OrganizationsT_{0b}$ $AgentsT_{0b}$ $RoleContractsT_{0b}$ $org_1, org_2 : ORG$
$org_1.rolePositions_o = \{aggrPos : ROLEPOS \mid$ $aggrPos.role = aggrRole \wedge aggrPos.orgName = organization_1\}$ $\dots$

$MACODOSystemT_{0b}$ $MACODOSYSTEM_{traffic}$ $AgentsT_{0b}$ $OrganizationsT_{0b}$
$agents = \{agent_1, agent_2, agent_3, agent_4\}$ $organizations = \{org_1, org_2\}$

### 5.2.3 Context Update

At  $T_{0b}$ , the traffic jam dissolves in the viewing range of  $camera_4$ , generating a context update event.

$ContextUpdateT_{0b}$ $MACODOSystemT_{0b}$ $ContextUpdate$
$agentname? = camera_4$ $context?.location = loc_4 \wedge context?.state = freeflow$

This results in the following state for the MACODO system.

$RoleContractsT_1$ $obsCon_1, obsCon_2, aggrCon_2, obsCon_3, obsCon_4 : ROLECONT$ $\dots$
---

$AgentsT_1$ $RoleContractsT_1$ $agent_1, agent_2, agent_3, agent_4 : AGENT$ $agent_4.context_a.state = freeflow$ $\dots$
--

$OrganizationsT_1$ $AgentsT_1$ $RoleContractsT_1$ $org_1, org_2 : ORG$ $\dots$
--

$\overline{MACODOSystemT_1}$ $MACODOSYSTEM_{traffic}$ $AgentsT_1$ $OrganizationsT_1$
$agents = \{agent_1, agent_2, agent_3, agent_4\}$ $organizations = \{org_1, org_2\}$

### 5.3 The MACODO System at $T_1$

#### 5.3.1 Split Law

At  $T_1$ ,  $organization_1$  has a differentiated context, due to the context update at  $T_{0b}$ , triggering the split law.

$\overline{SplitLawT_1}$ $MACODOSystemT_1$ $SplitLaw$
$org? = org_2$

This results in the following state for the MACODO system.

$\overline{RoleContractsT_{1a}}$ $obsCon_1, obsCon_2, aggrCon_2, obsCon_3, obsCon_4 : ROLECONT$
$obsCon_2.rolePosition.orgName = organization_3 \wedge aggrCon_2.rolePosition.orgName = organization_3$ $obsCon_3.rolePosition.orgName = organization_3$ $obsCon_4.rolePosition.orgName = organization_4$ $\dots$

$\overline{AgentsT_{1a}}$ $RoleContractsT_{1a}$ $agent_1, agent_2, agent_3, agent_4 : AGENT$
$\dots$

$\overline{OrganizationsT_{1a}}$ $AgentsT_{1a}$ $RoleContractsT_{1a}$ $org_1, org_3, org_4 : ORG$
$org_3.name_o = organization_3 \wedge org_3.context_o = \{agent_2.context_a, agent_3.context_a\} \wedge$ $org_3.roleContracts_o = \{obsCon_2, aggrCon_2, obsCon_3\} \wedge org_3.rolePositions_o = \emptyset$ $org_4.name_o = organization_4 \wedge org_4.context_o = \{agent_4.context_a\} \wedge$ $org_4.roleContracts_o = \{obsCon_4\} \wedge org_4.rolePositions_o = \emptyset$ $\dots$

$\overline{MACODOSystemT_{1a}}$ $MACODOSYSTEM_{traffic}$ $AgentsT_{1a}$ $OrganizationsT_{1a}$
$agents = \{agent_1, agent_2, agent_3, agent_4\}$ $organizations = \{org_1, org_3, org_4\}$

### 5.3.2 Role Law

After the split at  $T_1$ ,  $org_4$  does not have a role contract or role position for the aggregation role, which triggers the role law.

$\overline{RoleLawT_{1a}}$ $MACODOSystemT_{1a}$ $RoleLaw$
$org? = org_4$

This results in the following state for the MACODO system.

$\overline{RoleContractsT_{1b}}$ $obsCon_1, obsCon_2, aggrCon_2, obsCon_3, obsCon_4 : ROLECONT$
$\dots$

$\overline{AgentsT_{1b}}$ $RoleContractsT_{1b}$ $agent_1, agent_2, agent_3, agent_4 : AGENT$
$\dots$

$\overline{OrganizationsT_{1b}}$ $AgentsT_{1b}$ $RoleContractsT_{1b}$ $org_1, org_3, org_4 : ORG$
$org_4.rolePositions_o = \{aggrPos : ROLEPOS \mid$ $aggrPos.role = aggrRole \wedge aggrPos.orgName = organization_4\}$
$\dots$

$\overline{MACODOSystemT_{1b}}$ $MACODOSYSTEM_{traffic}$ $AgentsT_{1b}$ $OrganizationsT_{1b}$
$agents = \{agent_1, agent_2, agent_3, agent_4\}$ $organizations = \{org_1, org_3, org_4\}$

### 5.3.3 Context Update

At  $T_{1b}$ , the traffic jam grows into the viewing range of  $camera_1$ , generating a context update event.

$\overline{RoleLawT_{1b}}$ $MACODOSystemT_{1b}$ $ContextUpdate$
$agentname? = camera_1$ $context?.location = loc_1 \wedge context?.state = congested$

This results in the following state for the MACODO system.

$\overline{RoleContractsT_2}$ $obsCon_1, obsCon_2, aggrCon_2, obsCon_3, obsCon_4 : ROLECONT$
...

$\overline{AgentsT_2}$ $RoleContractsT_2$ $agent_1, agent_2, agent_3, agent_4 : AGENT$
$agent_1.context_a.state = congested$ ...

$\overline{OrganizationsT_2}$ $AgentsT_2$ $RoleContractsT_2$ $org_1, org_3, org_4 : ORG$
...

$\overline{MACODOSystemT_2}$ $MACODOSYSTEM_{traffic}$ $AgentsT_2$ $OrganizationsT_2$
$agents = \{agent_1, agent_2, agent_3, agent_4\}$ $organizations = \{org_1, org_3, org_4\}$

## 5.4 The MACODO System at $T_2$

### 5.4.1 Merge Law

After the context update of  $T_{1b}$ , organizations  $org_1$  and  $org_3$  have a mergeable and related context, triggering the merge law.

$MergeLaw_2$ $MACODOSystemT_2$ $MergeLaw$
$org1? = org_1 \wedge org2? = org_3$

This results in the following state for the MACODO system.

$RoleContractsT_{2a}$ $obsCon_1, obsCon_2, aggrCon_2, obsCon_3, obsCon_4 : ROLECONT$
$obsCon_1.rolePosition.orgName = organization_5$ $obsCon_2.rolePosition.orgName = organization_5 \wedge aggrCon_2.rolePosition.orgName = organization_5$ $obsCon_3.rolePosition.orgName = organization_5$ $\dots$

$AgentsT_{2a}$ $RoleContractsT_{2a}$ $agent_1, agent_2, agent_3, agent_4 : AGENT$
$\dots$

$OrganizationsT_{2a}$ $AgentsT_{2a}$ $RoleContractsT_{2a}$ $org_5, org_4 : ORG$
$org_5.name_o = organization_5$ $org_5.context_o = \{agent_1.context_a, agent_2.context_a, agent_3.context_a\}$ $org_5.roleContracts_o = \{obsCon_1, obsCon_2, aggrCon_2, obsCon_3\}$ $org_5.rolePositions_o =$ $\{aggrPos : ROLEPOS \mid aggrPos.role = aggrRole \wedge aggrPos.orgName = organization_5\}$ $\dots$

$MACODOSystemT_{2a}$ $MACODOSYSTEM_{traffic}$ $AgentsT_{2a}$ $OrganizationsT_{2a}$
$agents = \{agent_1, agent_2, agent_3, agent_4\}$ $organizations = \{org_5, org_4\}$

## 5.4.2 Role Law

At  $T_{2a}$ , after the merge,  $org_5$  has both a role contract and an open role position for the aggregation role, triggering the role law which will close the open role position.

$RoleLaw_{2a}$ $MACODOSystemT_{2a}$ $RoleLaw$
$org? = org_5$

This results in the following state for the MACODO system.

$RoleContractsT_3$ $obsCon_1, obsCon_2, aggrCon_2, obsCon_3, obsCon_4 : ROLECONT$
$\dots$

$AgentsT_3$ $RoleContractsT_3$ $agent_1, agent_2, agent_3, agent_4 : AGENT$
$\dots$

$OrganizationsT_3$ $AgentsT_3$ $RoleContractsT_3$ $org_5, org_4 : ORG$
$org_5.roleContracts_o = \{obsCon_1, obsCon_2, aggrCon_2, obsCon_3\}$ $org_5.rolePositions_o = \emptyset$ $\dots$

$MACODOSystemT_3$ $MACODOSYSTEM_{traffic}$ $AgentsT_3$ $OrganizationsT_3$
$agents = \{agent_1, agent_2, agent_3, agent_4\}$ $organizations = \{org_5, org_4\}$

## 5.5 The MACODO System at $T_3$

### 5.5.1 Capability Update

At  $T_3$ ,  $camera_3$  fails, generating a capability update event.

$CapabilityUpdate_3$ $MACODOSystemT_3$ $CapabilityUpdate$
$agentname? = camera_3$ $capabilities? = \emptyset$

This results in the following state for the MACODO system.

$RoleContractsT_{3a}$ $obsCon_1, obsCon_2, aggrCon_2, obsCon_3, obsCon_4 : ROLECONT$ ...
--

$AgentsT_{3a}$ $RoleContractsT_{3a}$ $agent_1, agent_2, agent_3, agent_4 : AGENT$ $agent_3.capabilities_a = \emptyset$ ...
--

$OrganizationsT_{3a}$ $AgentsT_{3a}$ $RoleContractsT_{3a}$ $org_5, org_4 : ORG$ ...
---

$MACODOSystemT_{3a}$ $MACODOSYSTEM_{traffic}$ $AgentsT_{3a}$ $OrganizationsT_{3a}$ $agents = \{agent_1, agent_2, agent_3, agent_4\}$ $organizations = \{org_5, org_4\}$
--

### 5.5.2 Self-Healing Law

At  $T_{3a}$ , after the capability update,  $agent_3$  is lacking the capabilities required for its current role contracts, which triggers the self-healing law.

$SelfHealingLaw_{3a}$ $MACODOSystemT_{3a}$ $SelfHealingLaw$
$agent? = agent_3$

This results in the following state for the MACODO system.

$RoleContractsT_4$ $obsCon_1, obsCon_2, aggrCon_2, obsCon_4 : ROLECONT$
$\dots$

$AgentsT_4$ $RoleContractsT_4$ $agent_1, agent_2, agent_3, agent_4 : AGENT$
$agent_3.roleContracts_a = \emptyset$ $\dots$

$OrganizationsT_4$ $AgentsT_4$ $RoleContractsT_4$ $org_5, org_4 : ORG$
$org_5.roleContracts_o = \{obsCon_1, obsCon_2, aggrCon_2\}$ $\dots$

$MACODOSystemT_4$ $MACODOSYSTEM_{traffic}$ $AgentsT_4$ $OrganizationsT_4$
$agents = \{agent_1, agent_2, agent_3, agent_4\}$ $organizations = \{org_5, org_4\}$

## References

- [1] CZT, 2008. Community Z Tools. <http://czt.sourceforge.net/>.
- [2] M. d'Inverno and M. Luck. *Understanding Agent Systems*. SpringerVerlag, 2004.
- [3] 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.
- [4] 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.