

Z specifications of FORMS - FOrmal Reference Model for Self-adaptation

Danny Weyns

Sam Malek

Jesper Andersson

Report CW 579, June 2010



Katholieke Universiteit Leuven
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Z specifications of FORMS - FOrmal Reference Model for Self-adaptation

Danny Weyns
Sam Malek
Jesper Andersson

Report CW579, June 2010

Department of Computer Science, K.U.Leuven

Abstract

This report gives a specification of the FOrmal Reference Model for Self-adaptation (FORMS). FORMS consists of three perspectives that focus on different concerns in self-adaptive systems. Subsequently, we present the reflection perspective, the unification with the distribution perspective, and the unification with the MAPE-K perspective (Monitor, Analysis, Plan, Execute + Knowledge). The model is formally specified in Z notation. Z builds on set theory and first-order logic to precisely specify the primitives without delving into the implementation details. The formal specification is type checked using Community Z Tools.

Keywords : Software architecture, formal methods, self-adaptative systems.

Z specifications of FORMS - FOrmal Reference Model for Self-adaptation

Danny Weyns, Katholieke Universiteit Leuven, Belgium
Sam Malek, George Mason University, USA
Jesper Andersson, Linnaeus University, Sweden

June 10, 2010

Abstract

This report gives a specification of the FOrmal Reference Model for Self-adaptation (FORMS). FORMS consists of three perspectives that focus on different concerns in self-adaptive systems. Subsequently, we present the reflection perspective, the unification with the distribution perspective, and the unification with the MAPE-K perspective (Monitor, Analysis, Plan, Execute + Knowledge). The model is formally specified in Z notation. Z builds on set theory and first-order logic to precisely specify the primitives without delving into the implementation details. The formal specification is type checked using Community Z Tools.

1 introduction

FORMS, short for FOrmal Reference Model for Self-adaptation, enables engineers to effectively describe, study, and evaluate alternative architectural choices for self-adaptive software systems. FORMS builds on the existing frameworks and established principles, such as MAPE-K [2], computational reflection [4], and architecture-based adaptation [3, 5]. The reference model consists of a small number of primitives and a set of relationships among them that delineates the rules of composition. The model is formally specified, which enables the engineers to precisely define the key characteristics of self-adaptive software systems, and compare alternative solutions.

FORMS resulted from an extensive study of the literature, through which we have developed a candidate set of primitives necessary for formally specifying self-adaptive systems. Through applying FORMS to several existing systems we have confirmed its ability to illuminate the key characteristics of these systems. However, we do not argue FORMS is a conclusive reference model. In fact, one of the key contributions of FORMS is its ability to accommodate future extensions. To ensure extensibility, as well as technology and implementation independence, the primitives are intentionally high-level (i.e., remain at the architectural level) and could be specialized for specific application domains. The primitives refined in this manner enable the engineers to derive and document a catalog of known solutions (e.g., in the form of architectural patterns) for different domains.

An environment comprises a non-empty set of attributes and a set of processes that can modify the attributes. Environment is defined:

$\begin{array}{l} \textit{Environment} \\ \textit{attributes} : \mathbb{P} \textit{Attribute} \\ \textit{processes} : \mathbb{P} \textit{Process} \end{array}$
$\textit{attributes} \neq \emptyset$

The change operation defines how a process changes a set of attributes of the environment:

$\begin{array}{l} \textit{ChangeOp} \\ \Delta \textit{Environment} \\ \textit{attrs?} : \mathbb{P} \textit{Attribute} \\ \textit{p?} : \textit{Process} \end{array}$
$\begin{array}{l} \textit{attrs?} \subseteq \textit{attributes} \\ \textit{p?} \in \textit{processes} \\ \textit{attributes}' = (\textit{attributes} \setminus \textit{attrs?}) \cup \textit{p?}(\textit{attrs?}) \end{array}$

We define context as a set of accessible attributes of the environment:

$$\textit{Context} == \mathbb{P} \textit{Attribute}$$

2.2 Base-Level Subsystem

To define a base-level subsystem, we first introduce models. A model comprises representations that describes something of interest in the physical world and/or cyber world. Models are defined:

$\begin{array}{l} \textit{Model} [\textit{Representation}] \\ \textit{representations} : \mathbb{P} \textit{Representation} \end{array}$
$\textit{representations} \neq \emptyset$

Different models may have different types of representations.

An environment representation is a representation of attributes in the environment. The set of environment representations is defined:

$$[\textit{EnvironmentRepresentation}]$$

A domain model describes a domain of interest for one or more stakeholders. Domain model is defined:

<i>DomainModel</i> <i>Environment</i> <i>Model</i> [<i>EnvironmentRepresentation</i>] <i>mapping</i> : $\mathbb{P} \textit{Attribute} \leftrightarrow \textit{EnvironmentRepresentation}$
$\text{dom } \textit{mapping} \subseteq \{ \textit{attrs} : \mathbb{P} \textit{Attribute} \mid \textit{attrs} \subseteq \textit{attributes} \}$ $\text{ran } \textit{mapping} = \{ r : \textit{EnvironmentRepresentation} \mid r \in \textit{representations} \}$

A domain model maps representations to attribute sets.

To define computations, we introduce the type state. State represents the current status of a computation and is defined:

[*State*]

A computation is an activity in a software system that manages its own state. Computations are defined:

<i>Computation</i> <i>state</i> : $\mathbb{P} \textit{State}$ <i>compute</i> : $\mathbb{P} \textit{State} \rightarrow \mathbb{P} \textit{State}$
$\text{dom } \textit{compute} = \{ s : \mathbb{P} \textit{State} \mid s \subseteq \textit{state} \}$

The computation operation is defined as:

<i>ComputationOp</i> $\Delta \textit{Computation}$ <i>s?</i> , <i>s!</i> : $\mathbb{P} \textit{State}$
$s! = \textit{compute}(s?) \wedge$ $\textit{state}' = \textit{state} \setminus s? \cup s!$

A base-level computation can act upon a set of domain models and can perceive a context in the environment and effect this context.

<i>BaseLevelComputation</i> <i>Computation</i> <i>read</i> : $\mathbb{P} \textit{DomainModel} \times \mathbb{P} \textit{State} \rightarrow \mathbb{P} \textit{State}$ <i>write</i> : $\mathbb{P} \textit{State} \times \mathbb{P} \textit{DomainModel} \rightarrow \mathbb{P} \textit{DomainModel}$ <i>perceive</i> : $\mathbb{P} \textit{State} \times \textit{Context} \rightarrow \mathbb{P} \textit{State}$ <i>effect</i> : $\mathbb{P} \textit{State} \times \textit{Context} \rightarrow \textit{Context}$

A base-level subsystem is a software system that provides some functionality for a stakeholder or set of stakeholders. Base-level subsystem is defined:

$$\begin{array}{l}
 \textit{BaseLevelSubsystem} \\
 \hline
 \textit{models} : \mathbb{P} \textit{DomainModel} \\
 \textit{computations} : \mathbb{P} \textit{BaseLevelComputation} \\
 \hline
 \forall c : \textit{computations} \bullet \\
 \quad \text{dom } c.\textit{read} = \{ \textit{mdls} : \mathbb{P} \textit{DomainModel} \mid \textit{mdls} \subseteq \textit{models} \bullet \\
 \quad \quad (\textit{mdls}, c.\textit{state}) \} \wedge \\
 \quad \text{dom } c.\textit{write} = \{ \textit{mdls} : \mathbb{P} \textit{DomainModel} \mid \textit{mdls} \subseteq \textit{models} \bullet \\
 \quad \quad (c.\textit{state}, \textit{mdls}) \}
 \end{array}$$

A base-level subsystem comprises a set of domain models and a set of base-level computations. The computations can act upon the domain models.

The read operation defines how a base-level subsystem computation reads a set of domain models and updates its state:

$$\begin{array}{l}
 \textit{ReadOp} \\
 \hline
 \Delta \textit{BaseLevelSubsystem} \\
 \textit{c?}, \textit{c!} : \textit{BaseLevelComputation} \\
 \textit{ms?} : \mathbb{P} \textit{DomainModel} \\
 \hline
 \textit{c?} \in \textit{computations} \wedge \\
 \textit{ms?} \subseteq \textit{models} \wedge \\
 \textit{c!}.\textit{state} = \textit{c?}.\textit{read}(\textit{ms?}, \textit{c?}.\textit{state}) \wedge \\
 \textit{c!}.\textit{compute} = \textit{c?}.\textit{compute} \wedge \\
 \textit{models}' = \textit{models} \wedge \\
 \textit{computations}' = \textit{computations} \setminus \{ \textit{c?} \} \cup \{ \textit{c!} \}
 \end{array}$$

The compute operation defines how a base-level subsystem computation performs a computation on its state:

$$\begin{array}{l}
 \textit{ComputeOp} \\
 \hline
 \Delta \textit{BaseLevelSubsystem} \\
 \textit{c?}, \textit{c!} : \textit{BaseLevelComputation} \\
 \textit{s!} : \mathbb{P} \textit{State} \\
 \hline
 \textit{c?} \in \textit{computations} \wedge \\
 \textit{s!} = \textit{c?}.\textit{compute}(\textit{c?}.\textit{state}) \\
 \textit{c!}.\textit{state} = \textit{s!} \wedge \\
 \textit{c!}.\textit{compute} = \textit{c?}.\textit{compute} \wedge \\
 \textit{models}' = \textit{models} \wedge \\
 \textit{computations}' = \textit{computations} \setminus \{ \textit{c?} \} \cup \{ \textit{c!} \}
 \end{array}$$

The write operation defines how a base level computation acts upon a set of domain models:

$$\begin{array}{l}
 \text{WriteOp} \\
 \hline
 \Delta \text{BaseLevelSubsystem} \\
 c? : \text{BaseLevelComputation} \\
 ms? : \mathbb{P} \text{DomainModel} \\
 ms! : \mathbb{P} \text{DomainModel} \\
 \hline
 c? \in \text{computations} \wedge \\
 ms? \subseteq \text{models} \wedge \\
 ms! = c?.\text{write}(c?.\text{state}, ms?) \wedge \\
 \text{models}' = \text{models} \setminus ms? \cup ms! \wedge \\
 \text{computations}' = \text{computations}
 \end{array}$$

2.3 Reflective Subsystem

A reflection model representation reifies the entities (e.g., subsystem constructs, environment attributes) needed for reasoning about adaptation. It is analogous to meta-level information from the domain of computational reflection [4]. A self-adaptive system has a set of reflection model representations:

$$[\text{ReflectionModelRepresentation}]$$

A reflection model comprises reflection model representations:

$$\begin{array}{l}
 \text{ReflectionModel} \\
 \text{Model}[\text{ReflectionModelRepresentation}]
 \end{array}$$

Reflection models are used by reflective computations.

A reflective computation is defined:

$$\begin{array}{l}
 \text{ReflectiveComputation} [\text{Subsystem}] \\
 \text{Computation} \\
 \text{read} : \mathbb{P} \text{ReflectionModel} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State} \\
 \text{write} : \mathbb{P} \text{State} \times \mathbb{P} \text{ReflectionModel} \rightarrow \mathbb{P} \text{ReflectionModel} \\
 \text{perceive} : \text{Context} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State} \\
 \text{sense} : \mathbb{P} \text{Subsystem} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State} \\
 \text{adapt} : \mathbb{P} \text{Subsystem} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{Subsystem} \\
 \text{trigger} : \mathbb{P} \text{State} \times \mathbb{P} \text{ReflectiveComputation}[\text{Subsystem}] \rightarrow \\
 \quad \mathbb{P} \text{ReflectiveComputation}[\text{Subsystem}]
 \end{array}$$

A reflective computation reasons and acts upon a subset of reflection models by reading from, and writing to the models. It also perceives certain environmental context. However, note that unlike a base-level computation, a reflective computation does not effect changes in the environment. Moreover, reflective computation not only senses (monitors) and adapts the subsystem, but also triggers other reflective computations.

A reflective subsystem is composed of reflection models and reflective computations. This is formally specified as follows:

$$\begin{array}{l}
 \text{== } \mathit{ReflectiveSubsystem} [\mathit{Subsystem}] \text{==} \\
 \text{models} : \mathbb{P} \mathit{ReflectionModel} \\
 \text{computations} : \mathbb{P} \mathit{ReflectiveComputation}[\mathit{Subsystem}] \\
 \hline
 \forall c : \text{computations} \bullet \\
 \quad \text{dom } c.\text{read} = \{ mds : \mathbb{P} \mathit{ReflectionModel} \mid mds \subseteq \text{models} \bullet \\
 \quad \quad (mds, c.\text{state}) \} \wedge \\
 \quad \text{dom } c.\text{write} = \{ mds : \mathbb{P} \mathit{ReflectionModel} \mid mds \subseteq \text{models} \bullet \\
 \quad \quad (c.\text{state}, mds) \} \wedge \\
 \quad \text{dom } c.\text{trigger} = \{ ct : \mathbb{P} \mathit{ReflectiveComputation}[\mathit{Subsystem}] \mid \\
 \quad \quad ct \subseteq \text{computations} \setminus \{c\} \bullet (c.\text{state}, ct) \}
 \end{array}$$

2.4 Self-Adaptive System

A self-adaptive system comprises a set of base-level and reflective subsystems. As an example, we consider a self-adaptive system with two reflective levels. We model a meta-level subsystem (i.e. a reflective systems on top of a base-level subsystem) as follows:

$$\mathit{MetaLevelSubsystem} == \mathit{ReflectiveSubsystem}[\mathit{BaseLevelSubsystem}]$$

Similarly, a meta-meta-level subsystem can be defined:

$$\mathit{MetaMetaLevelSubsystem} == \mathit{ReflectiveSubsystem}[\mathit{MetaLevelSubsystem}]$$

We can now model the self-adaptive system as follows:

SelfAdaptiveSystem

baseLevelSubsystems : \mathbb{P} *BaseLevelSubsystem*
metaLevelSubsystems : \mathbb{P} *MetaLevelSubsystem*
metaMetaLevelSubsystems : \mathbb{P} *MetaMetaLevelSubsystem*

$\#baseLevelSubsystems \geq 1$
 $\#metaLevelSubsystems \geq 1$
 $\#metaMetaLevelSubsystems \geq 1$
 $\forall mls : metaLevelSubsystems; cm, ce : ReflectiveComputation \bullet$
 $cm \in mls.computations \wedge ce \in mls.computations \wedge$
 $dom\ cm.sense = \{bls : \mathbb{P}\ BaseLevelSubsystem \mid$
 $bls \subseteq baseLevelSubsystems \bullet (bls, cm.state)\} \wedge$
 $dom\ ce.adapt = \{bls : \mathbb{P}\ BaseLevelSubsystem \mid$
 $bls \subseteq baseLevelSubsystems \bullet (bls, cm.state)\}$
 $\forall mmls : metaMetaLevelSubsystems;$
 $cm, ce : ReflectiveComputation \bullet$
 $cm \in mmls.computations \wedge ce \in mmls.computations \wedge$
 $dom\ cm.sense = \{mls : \mathbb{P}\ MetaLevelSubsystem \mid$
 $mls \subseteq metaLevelSubsystems \bullet (mls, cm.state)\} \wedge$
 $dom\ ce.adapt = \{mls : \mathbb{P}\ MetaLevelSubsystem \mid$
 $mls \subseteq metaLevelSubsystems \bullet (mls, ce.state)\}$

The specification states that meta-level subsystems can sense and adapt base-level subsystems, while meta-meta-level subsystems can sense and adapt meta-level subsystems.

A self-adaptive system situated in an environment is specified:

$\textit{SituatingSelfAdaptiveSystem}$ <hr/> $\textit{Environment}$ $\textit{SelfAdaptiveSystem}$ $\textit{context} : \textit{Context}$ <hr/> $\textit{context} \subseteq \textit{attributes}$ $\forall \textit{bls} : \textit{baseLevelSubsystems}; \textit{c} : \textit{BaseLevelComputation} \bullet$ $\textit{c} \in \textit{bls.computations} \wedge$ $\text{dom } \textit{c.perceive} = \{\textit{attrs} : \textit{Context} \mid$ $\textit{attrs} \subseteq \textit{context} \bullet (\textit{c.state}, \textit{attrs})\} \wedge$ $\text{dom } \textit{c.effect} = \{\textit{attrs} : \textit{Context} \mid$ $\textit{attrs} \subseteq \textit{context} \bullet (\textit{c.state}, \textit{attrs})\}$ $\forall \textit{mls} : \textit{metaLevelSubsystems}; \textit{cu} : \textit{ReflectiveComputation} \bullet$ $\textit{cu} \in \textit{mls.computations} \wedge$ $\text{dom } \textit{cu.perceive} = \{\textit{attrs} : \textit{Context} \mid$ $\textit{attrs} \subseteq \textit{context} \bullet (\textit{attrs}, \textit{cu.state})\}$ $\forall \textit{mmls} : \textit{metaMetaLevelSubsystems};$ $\textit{cu} : \textit{ReflectiveComputation} \bullet$ $\textit{cu} \in \textit{mmls.computations} \wedge$ $\text{dom } \textit{cu.perceive} = \{\textit{attrs} : \textit{Context} \mid$ $\textit{attrs} \subseteq \textit{context} \bullet (\textit{attrs}, \textit{cu.state})\}$

The specification states that base-level subsystems can perceive and effect the context in which the self-adaptive system is situated, while reflective subsystems can only perceive the context.

Finally, we can now formally specify how a meta-level subsystem adapts a base-level subsystem:

$\textit{MetaLevelAdaptationOp}$ <hr/> $\Delta \textit{SituatingSelfAdaptiveSystem}$ $\exists \textit{Environment}$ $\textit{e?} : \textit{ReflectiveComputation}[\textit{BaseLevelSubsystem}]$ $\textit{bls?}, \textit{bls!} : \textit{BaseLevelSubsystem}$ $\textit{mls?}, \textit{mls!} : \textit{MetaLevelSubsystem}$ <hr/> $\textit{bls?} \in \textit{baseLevelSubsystems} \wedge$ $\textit{mls?} \in \textit{metaLevelSubsystems} \wedge$ $\textit{e?} \in \textit{mls?.computations} \wedge$ $\{\textit{bls!}\} = \textit{e?.adapt}(\{\textit{bls?}\}, \textit{e?.state}) \wedge$ $\textit{baseLevelSubsystems}' = \textit{baseLevelSubsystems} \setminus \{\textit{bls?}\} \cup \{\textit{bls!}\}$ $\textit{metaLevelSubsystems}' = \textit{metaLevelSubsystems}$ $\textit{metaMetaLevelSubsystems}' = \textit{metaMetaLevelSubsystems}$
--

The specification states that self-adaptation changes the self-adaptive system, but

does not effect the environment. The adaptation is performed by one of the meta-level reflective computations ($e?$) which adapts one or more base-level subsystems.

3 Unification with Distribution Perspective

Fig. 2 shows a graphical overview of the FORMS elements and relations for the unification of the reflection and the distribution perspective.

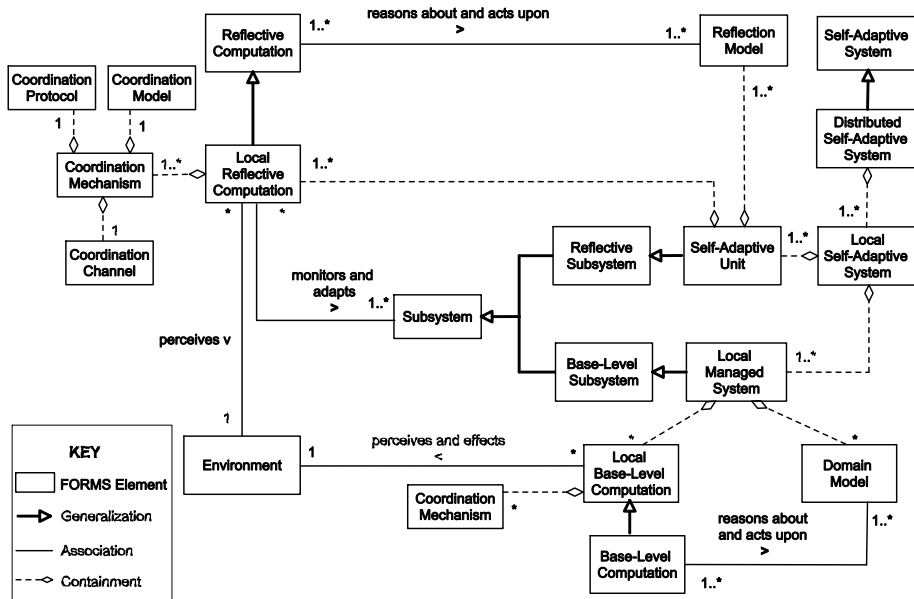
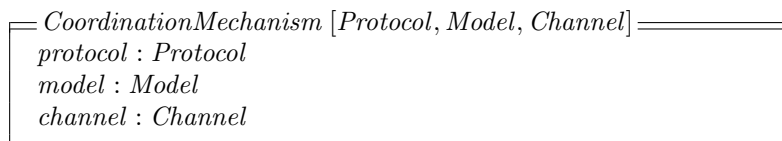


Figure 2: FORMS: unification with distribution perspective.

3.1 Coordination Mechanism

We define a coordination mechanism as follows:



A coordination mechanism comprises a coordination protocol, a coordination model, and a coordination channel.

3.2 Local Managed System

A local base-level computation is a base-level computation that can send and receive messages:

$$\boxed{\begin{array}{l} \text{LocalBaseLevelComputation} [\text{Protocol}, \text{Model}, \text{Channel}] \\ \text{BaseLevelComputation} \\ \text{coordinationMechanism} : \\ \quad \text{CoordinationMechanism}[\text{Protocol}, \text{Model}, \text{Channel}] \end{array}}$$

A local managed system is defined as:

$$\boxed{\begin{array}{l} \text{LocalManagedSystem} [\text{Protocol}, \text{Model}, \text{Channel}] \\ \text{models} : \mathbb{P} \text{DomainModel} \\ \text{computations} : \\ \quad \mathbb{P} \text{LocalBaseLevelComputation}[\text{Protocol}, \text{Model}, \text{Channel}] \\ \hline \forall c : \text{computations} \bullet \\ \quad \text{dom } c.\text{read} = \{ \text{mdls} : \mathbb{P} \text{DomainModel} \mid \text{mdls} \subseteq \text{models} \bullet \\ \quad \quad (\text{mdls}, c.\text{state}) \} \wedge \\ \quad \text{dom } c.\text{write} = \{ \text{mdls} : \mathbb{P} \text{DomainModel} \mid \text{mdls} \subseteq \text{models} \bullet \\ \quad \quad (c.\text{state}, \text{mdls}) \} \end{array}}$$

A local managed system is a base-level subsystem comprising a set of domain models and a set of local base-level computations.

3.3 Self-Adaptive Unit

A local reflective computation is a reflective computation that comprises a coordination mechanism:

$$\boxed{\begin{array}{l} \text{LocalReflectiveComputation}[\text{Subsystem}, \text{Protocol}, \text{Model}, \text{Channel}] \\ \text{Computation} \\ \text{coordinationMechanism} : \\ \quad \text{CoordinationMechanism}[\text{Protocol}, \text{Model}, \text{Channel}] \\ \text{read} : \mathbb{P} \text{ReflectionModel} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State} \\ \text{write} : \mathbb{P} \text{State} \times \mathbb{P} \text{ReflectionModel} \rightarrow \mathbb{P} \text{ReflectionModel} \\ \text{perceive} : \text{Context} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State} \\ \text{sense} : \mathbb{P} \text{Subsystem} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State} \\ \text{adapt} : \mathbb{P} \text{Subsystem} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{Subsystem} \\ \text{trigger} : \mathbb{P} \text{State} \times \mathbb{P} \text{LocalReflectiveComputation} [\\ \quad \text{Subsystem}, \text{Protocol}, \text{Model}, \text{Channel}] \rightarrow \\ \quad \mathbb{P} \text{LocalReflectiveComputation} [\\ \quad \quad \text{Subsystem}, \text{Protocol}, \text{Model}, \text{Channel}] \end{array}}$$

The self-adaptive unit is defined as:

$$\begin{array}{l}
\boxed{\text{SelfAdaptiveUnit } [Subsystem, Protocol, Model, Channel]} \\
models : \mathbb{P} \text{ ReflectionModel} \\
computations : \mathbb{P} \text{ LocalReflectiveComputation} [\\
\quad Subsystem, Protocol, Model, Channel] \\
\hline
\forall c : computations \bullet \\
\quad \text{dom } c.read = \{mdls : \mathbb{P} \text{ ReflectionModel} \mid mdls \subseteq models \bullet \\
\quad \quad (mdls, c.state)\} \wedge \\
\quad \text{dom } c.write = \{mdls : \mathbb{P} \text{ ReflectionModel} \mid mdls \subseteq models \bullet \\
\quad \quad (c.state, mdls)\} \wedge \\
\quad \text{dom } c.trigger = \{ct : \mathbb{P} \text{ LocalReflectiveComputation} [\\
\quad \quad Subsystem, Protocol, Model, Channel] \mid \\
\quad \quad ct \subseteq computations \setminus \{c\} \bullet (c.state, ct)\}
\end{array}$$

A self-adaptive unit is a reflective subsystem comprising reflection models and local reflective computations.

3.4 Distributed Self-Adaptive System

A local self-adaptive systems comprises a set of local managed systems and a set of self-adaptive units. As an example, we consider a local self-adaptive system with one reflective layer in which all base level computations use a particular coordination mechanism and all reflective computations use a particular coordination protocol:

$$\begin{array}{l}
\boxed{\text{LocalSelfAdaptiveSystem } [BCP, BCM, BCC, ACP, ACM, ACC]} \\
localManagedSystems : \mathbb{P} \text{ LocalManagedSystem} [BCP, BCM, BCC] \\
selfAdaptiveUnits : \\
\quad \mathbb{P} \text{ SelfAdaptiveUnit} [LocalManagedSystem, ACP, ACM, ACC] \\
\hline
\forall sau : selfAdaptiveUnits; lrca : LocalReflectiveComputation \bullet \\
\quad lrca \in sau.computations \wedge lrca \in sau.computations \wedge \\
\quad \text{dom } lrca.sense = \{lms : \mathbb{P} \text{ LocalManagedSystem} \mid \\
\quad \quad lms \subseteq localManagedSystems \bullet (lms, lrca.state)\} \wedge \\
\quad \text{dom } lrca.adapt = \{lms : \mathbb{P} \text{ LocalManagedSystem} \mid \\
\quad \quad lms \subseteq localManagedSystems \bullet (lms, lrca.state)\}
\end{array}$$

The abbreviations BCP, BCM, and BCC refer respectively to the coordination protocol, coordination model, and coordination channel for the base level system. ACP, ACM, and ACC are similar abbreviations for the coordination elements of the self-adaptive unit.

The specification states that self-adaptive units can sense and adapt the local managed systems of the local self-adaptive system.

A situated local self-adaptive system is a local self-adaptive system situated in some context of the environment:

$$\begin{array}{l}
\boxed{\begin{array}{l}
\text{---} \textit{SitatedLocalSelfAdaptiveSystem}[BCP, BCM, BCC, ACP, ACM, ACC]_{||} \\
\textit{Environment} \\
\textit{LocalSelfAdaptiveSystem}[BCP, BCM, BCC, ACP, ACM, ACC] \\
\textit{context} : \textit{Context}
\end{array}} \\
\textit{context} \subseteq \textit{attributes} \\
\forall \textit{lms} : \textit{localManagedSystems}; \textit{c} : \textit{LocalBaseLevelComputation} \bullet \\
\textit{c} \in \textit{lms.computations} \wedge \\
\textit{dom c.perceive} = \\
\{ \textit{attrs} : \textit{Context} \mid \textit{attrs} \subseteq \textit{context} \bullet (\textit{c.state}, \textit{attrs}) \} \wedge \\
\textit{dom c.effect} = \\
\{ \textit{attrs} : \textit{Context} \mid \textit{attrs} \subseteq \textit{context} \bullet (\textit{c.state}, \textit{attrs}) \} \\
\forall \textit{sau} : \textit{selfAdaptiveUnits}; \textit{lrc} : \textit{LocalReflectiveComputation} \bullet \\
\textit{lrc} \in \textit{sau.computations} \wedge \\
\textit{dom lrc.perceive} = \\
\{ \textit{attrs} : \textit{Context} \mid \textit{attrs} \subseteq \textit{context} \bullet (\textit{attrs}, \textit{lrc.state}) \}
\end{array}$$

A distributed self-adaptive system comprises a set of local self-adaptive systems:

$$\boxed{\begin{array}{l}
\text{---} \textit{DistributedSelfAdaptiveSystem}[BCP, BCM, BCC, ACP, ACM, ACC]_{||} \\
\textit{localSelfAdaptiveSystems} : \\
\mathbb{P} \textit{LocalSelfAdaptiveSystem}[BCP, BCM, BCC, ACP, ACM, ACC]
\end{array}}$$

4 Unification with MAPE-K Perspective

Fig. 3 shows a graphical overview of FORMS elements and relations integrated with the MAPE perspective.

4.1 Reflection Models

We distinguish between four types of reflection models: environment model, concern model, mape working model, and subsystem model. To describe reflection models, we first introduce a number of additional types of representations:

$$[\textit{ConcernRepresentation}, \textit{MapeRepresentation}]$$

A concern representation is a representation of a particular concern of interest. Mape representations are used to describe working models used by reflective computations.

A mape working model is a model used by reflective computations to deal with a concern of interest. Mape working models are defined:

$$\boxed{\begin{array}{l} \text{MapeWorkingModel} \\ \text{Model}[\text{MapeRepresentation}] \end{array}}$$

To define a subsystem model we first introduce the concept of feature. Features describe perceivable characteristics of software systems:

$$[\text{Feature}]$$

We define a function *reify* that returns the features for a given subsystem:

$$\boxed{\begin{array}{l} [\text{Subsystem}] \\ \text{reify} : \text{Subsystem} \rightarrow \mathbb{P} \text{Feature} \end{array}}$$

A subsystem model is a model of a subsystem (either a base-level system or a reflective subsystem). Subsystem models are defined:

$$\boxed{\begin{array}{l} \text{SubsystemModel} [\text{Subsystem}] \\ \text{subsystem} : \text{Subsystem} \\ \text{Model}[\text{SubsystemRepresentation}[\text{Subsystem}]] \\ \text{mapping} : \mathbb{P} \text{Feature} \leftrightarrow \text{SubsystemRepresentation}[\text{Subsystem}] \\ \text{dom mapping} \subseteq \\ \quad \{ \text{features} : \mathbb{P} \text{Feature} \mid \text{features} \subseteq \text{reify}(\text{subsystem}) \} \\ \text{ran mapping} = \\ \quad \{ r : \text{SubsystemRepresentation}[\text{Subsystem}] \mid \\ \quad \quad r \in \text{representations} \} \end{array}}$$

A base-level subsystem model is defined:

$$\boxed{\begin{array}{l} \text{BaseLevelSubsystemModel} \\ \text{SubsystemModel}[\text{BaseLevelSubsystem}] \end{array}}$$

We introduce reflection models which groups the sets of models used by a set of reflective computations:

$$\boxed{\begin{array}{l} \text{ReflectionModels} [\text{Subsystem}] \\ \text{environmentModels} : \mathbb{P} \text{EnvironmentModel} \\ \text{concernModels} : \mathbb{P} \text{ConcernModel} \\ \text{mapeWorkingModels} : \mathbb{P} \text{MapeWorkingModel} \\ \text{subsystemModels} : \mathbb{P} \text{SubsystemModel}[\text{Subsystem}] \end{array}}$$

4.2 Reflective Computations

We define five types of reflective computations for self-adaptive system: update, monitor, analyse, plan and execute.

Update

Computation

read : $\mathbb{P} \text{EnvironmentModel} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State}$

write : $\mathbb{P} \text{State} \times \mathbb{P} \text{EnvironmentModel} \rightarrow \mathbb{P} \text{EnvironmentModel}$

perceive : $\text{Context} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State}$

Update computations perceive the environment and update the environment models accordingly.

Monitor [*Subsystem*]

Computation

read : $\mathbb{P} \text{MapeWorkingModel} \times$

$\mathbb{P} \text{SubsystemModel}[\text{Subsystem}] \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State}$

write : $\mathbb{P} \text{State} \times \mathbb{P} \text{MapeWorkingModel}$

$\times \mathbb{P} \text{SubsystemModel}[\text{Subsystem}] \rightarrow$

$\mathbb{P} \text{MapeWorkingModel} \times \mathbb{P} \text{SubsystemModel}[\text{Subsystem}]$

sense : $\mathbb{P} \text{Subsystem} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State}$

trigger : $\mathbb{P} \text{State} \times \mathbb{P} \text{Analyse}[\text{Subsystem}] \rightarrow \mathbb{P} \text{Analyse}[\text{Subsystem}]$

Monitor computations monitor the underlying subsystem and maintain the subsystem models and possibly mape working models. Monitor computations can trigger analyse computations in particular states.

Analyse [*Subsystem*]

Computation

read : $\mathbb{P} \text{EnvironmentModel} \times \mathbb{P} \text{ConcernModel} \times$

$\mathbb{P} \text{MapeWorkingModel} \times \mathbb{P} \text{SubsystemModel}[\text{Subsystem}] \times$

$\mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State}$

write : $\mathbb{P} \text{State} \times \mathbb{P} \text{MapeWorkingModel} \rightarrow \mathbb{P} \text{MapeWorkingModel}$

trigger : $\mathbb{P} \text{State} \times \mathbb{P} \text{Plan}[\text{Subsystem}] \rightarrow \mathbb{P} \text{Plan}[\text{Subsystem}]$

Plan [*Subsystem*]

Computation

read : $\mathbb{P} \text{EnvironmentModel} \times \mathbb{P} \text{ConcernModel} \times$

$\mathbb{P} \text{MapeWorkingModel} \times \mathbb{P} \text{SubsystemModel}[\text{Subsystem}] \times$

$\mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State}$

write : $\mathbb{P} \text{State} \times \mathbb{P} \text{ConcernModel} \times \mathbb{P} \text{MapeWorkingModel} \rightarrow$

$\mathbb{P} \text{ConcernModel} \times \mathbb{P} \text{MapeWorkingModel}$

trigger : $\mathbb{P} \text{State} \times \mathbb{P} \text{Execute}[\text{Subsystem}] \rightarrow \mathbb{P} \text{Execute}[\text{Subsystem}]$

Analyse and plan computations reason about and act upon the reflection models in order to deal with the concerns of the self-adaptive system. Analyse computations can trigger plan computations in particular states, while plan computations can trigger execute computations.

$\begin{aligned} & \text{Execute } [Subsystem] \\ & \text{Computation} \\ & \text{read} : \mathbb{P} \text{EnvironmentModel} \times \mathbb{P} \text{MapeWorkingModel} \times \\ & \quad \mathbb{P} \text{SubsystemModel}[Subsystem] \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{State} \\ & \text{write} : \mathbb{P} \text{State} \times \mathbb{P} \text{MapeWorkingModel} \times \\ & \quad \mathbb{P} \text{SubsystemModel}[Subsystem] \rightarrow \\ & \quad \mathbb{P} \text{MapeWorkingModel} \times \mathbb{P} \text{SubsystemModel}[Subsystem] \\ & \text{adapt} : \mathbb{P} \text{Subsystem} \times \mathbb{P} \text{State} \rightarrow \mathbb{P} \text{Subsystem} \end{aligned}$
--

Execute computations use environment models and mape working models to adapt the underlying subsystem.

We define the sets of computations of a reflective subsystem for each type of reflective computation.

$\begin{aligned} & \text{Updating } [Subsystem] \\ & \text{updates} : \mathbb{P} \text{Update} \\ & \text{ReflectionModels}[Subsystem] \\ & \forall u : \text{updates} \bullet \\ & \quad \text{dom } u.\text{read} = \{eModels : \mathbb{P} \text{EnvironmentModel} \mid \\ & \quad \quad eModels \subseteq \text{environmentModels} \bullet \\ & \quad \quad (eModels, u.\text{state})\} \wedge \\ & \quad \text{dom } u.\text{write} = \{eModels : \mathbb{P} \text{EnvironmentModel} \mid \\ & \quad \quad eModels \subseteq \text{environmentModels} \bullet \\ & \quad \quad (u.\text{state}, eModels)\} \end{aligned}$

Update computations act upon (a subset of) the environment models.

$\text{Monitoring [Subsystem]}$ $\text{monitors : } \mathbb{P} \text{ Monitor}$ $\text{ReflectionModels[Subsystem]}$ <hr style="border: 0.5px solid black; margin: 5px 0;"/> $\forall m : \text{monitors} \bullet$ $\text{dom } m.\text{read} = \{mModels : \mathbb{P} \text{ MapeWorkingModel};$ $sModels : \mathbb{P} \text{ SubsystemModel[Subsystem]} \mid$ $mModels \subseteq \text{mapeWorkingModels} \wedge$ $sModels \subseteq \text{subsystemModels} \bullet$ $(mModels, sModels, m.\text{state})\} \wedge$ $\text{dom } m.\text{write} = \{mModels : \mathbb{P} \text{ MapeWorkingModel};$ $sModels : \mathbb{P} \text{ SubsystemModel[Subsystem]} \mid$ $mModels \subseteq \text{mapeWorkingModels} \wedge$ $sModels \subseteq \text{subsystemModels} \bullet$ $(m.\text{state}, mModels, sModels)\}$
--

Monitor computations act upon subsystem models and mape working models.

$\text{Analyzing [Subsystem]}$ $\text{analyses : } \mathbb{P} \text{ Analyse}$ $\text{ReflectionModels[Subsystem]}$ <hr style="border: 0.5px solid black; margin: 5px 0;"/> $\forall a : \text{analyses} \bullet$ $\text{dom } a.\text{read} = \{eModels : \mathbb{P} \text{ EnvironmentModel};$ $cModels : \mathbb{P} \text{ ConcernModel};$ $mModels : \mathbb{P} \text{ MapeWorkingModel};$ $sModels : \mathbb{P} \text{ SubsystemModel[Subsystem]} \mid$ $eModels \subseteq \text{environmentModels} \wedge$ $cModels \subseteq \text{concernModels} \wedge$ $mModels \subseteq \text{mapeWorkingModels} \wedge$ $sModels \subseteq \text{subsystemModels} \bullet$ $(eModels, cModels, mModels, sModels, a.\text{state})\} \wedge$ $\text{dom } a.\text{write} = \{mModels : \mathbb{P} \text{ MapeWorkingModel} \mid$ $mModels \subseteq \text{mapeWorkingModels} \bullet$ $(a.\text{state}, mModels)\}$
--

Analyse computations read the different kinds of reflection models and write its analysis results to the mape working models.

$\text{Planning [Subsystem]}$ $plans : \mathbb{P} \text{ Plan}$ $\text{ReflectionModels [Subsystem]}$ <hr style="width: 20%; margin-left: 0;"/> $\forall p : plans \bullet$ $\text{dom } p.read = \{ eModels : \mathbb{P} \text{ EnvironmentModel};$ $cModels : \mathbb{P} \text{ ConcernModel};$ $mModels : \mathbb{P} \text{ MapeWorkingModel};$ $sModels : \mathbb{P} \text{ SubsystemModel [Subsystem]} \mid$ $eModels \subseteq \text{environmentModels} \wedge$ $cModels \subseteq \text{concernModels} \wedge$ $mModels \subseteq \text{mapeWorkingModels} \wedge$ $sModels \subseteq \text{subsystemModels} \bullet$ $(eModels, cModels, mModels, sModels, p.state) \}$ \wedge $\text{dom } p.write = \{ cModels : \mathbb{P} \text{ ConcernModel};$ $mModels : \mathbb{P} \text{ MapeWorkingModel} \mid$ $cModels \subseteq \text{concernModels} \wedge$ $mModels \subseteq \text{mapeWorkingModels} \bullet$ $(p.state, cModels, mModels) \}$

Plan computations use the different reflection models to update the concern models and mape working models.

$\text{Executing [Subsystem]}$ $executes : \mathbb{P} \text{ Execute}$ $\text{ReflectionModels [Subsystem]}$ <hr style="width: 20%; margin-left: 0;"/> $\forall e : executes \bullet$ $\text{dom } e.read = \{ eModels : \mathbb{P} \text{ EnvironmentModel};$ $mModels : \mathbb{P} \text{ MapeWorkingModel};$ $sModels : \mathbb{P} \text{ SubsystemModel [Subsystem]} \mid$ $eModels \subseteq \text{environmentModels} \wedge$ $mModels \subseteq \text{mapeWorkingModels} \wedge$ $sModels \subseteq \text{subsystemModels} \bullet$ $(eModels, mModels, sModels, e.state) \}$ \wedge $\text{dom } e.write = \{ mModels : \mathbb{P} \text{ MapeWorkingModel};$ $sModels : \mathbb{P} \text{ SubsystemModel [Subsystem]} \mid$ $mModels \subseteq \text{mapeWorkingModels} \wedge$ $sModels \subseteq \text{subsystemModels} \bullet$ $(e.state, mModels, sModels) \}$

To perform adaptations, execute computations use the information of the different reflection models. An execute computation can maintain a subsystem model while performing adaptations of the corresponding subsystem.

The reflective computations schema groups the computations of a reflective subsystem:

$$\begin{array}{l}
 \boxed{\text{ReflectiveComputations [Subsystem]}} \\
 \text{Updating[Subsystem]} \\
 \text{Monitoring[Subsystem]} \\
 \text{Analyzing[Subsystem]} \\
 \text{Planning[Subsystem]} \\
 \text{Executing[Subsystem]} \\
 \hline
 \forall m : \text{monitors} \bullet \\
 \quad \text{dom } m.\text{trigger} = \\
 \quad \quad \{as : \mathbb{P} \text{Analyse} \mid as \subseteq \text{analyses} \bullet (m.\text{state}, as)\} \\
 \forall a : \text{analyses} \bullet \\
 \quad \text{dom } a.\text{trigger} = \{ps : \mathbb{P} \text{Plan} \mid ps \subseteq \text{plans} \bullet (a.\text{state}, ps)\} \\
 \forall p : \text{plans} \bullet \\
 \quad \text{dom } p.\text{trigger} = \\
 \quad \quad \{es : \mathbb{P} \text{Execute} \mid es \subseteq \text{executes} \bullet (p.\text{state}, es)\}
 \end{array}$$

Triggers are restricted to (the subsets of) the respective computations of a reflective subsystem.

4.3 IBM's Autonomic Manager Framework

To conclude the MAPE-perspective, we formally describe an example of a hierarchical self-adaptive autonomic system.

The base-level subsystem in an autonomic self-adaptive system is a managed resource and is defined as:

$$\boxed{\begin{array}{l} \text{ManagedResource} \\ \text{BaseLevelSubsystem} \end{array}}$$

Knowledge is defined as:

$$\boxed{\begin{array}{l} \text{Knowledge} \\ \text{ReflectionModel} \end{array}}$$

An autonomic manager is abstractly defined as:

$$\boxed{\begin{array}{l} \text{AutonomicManager} \\ \text{knowledge} : \text{Knowledge} \end{array}}$$

Autonomic manager computation manages a managed element (i.e. either a managed resource or an autonomic manager) and is defined as:

$$\boxed{\boxed{\text{AutonomicManagerComputation [ManagedElement]}} \\ \text{ReflectiveComputation[ManagedElement]}}$$

We distinguish between two types of autonomic managers: orchestrating autonomic manager and resource manager, defined as follows:

$$\boxed{\text{OrchestratingAutonomicManager} \\ \text{AutonomicManager} \\ \text{makeComputations :} \\ \mathbb{P} \text{AutonomicManagerComputation[AutonomicManager]}}$$

$$\boxed{\text{ResourceAutonomicManager} \\ \text{AutonomicManager} \\ \text{makeComputations :} \\ \mathbb{P} \text{AutonomicManagerComputation[ManagedResource]} \\ \text{manage : Knowledge} \times \mathbb{P} \text{ManagedResource} \rightarrow \mathbb{P} \text{ManagedResource}}$$

IBM's autonomic manager framework considers four different types of resource managers that deal with different types of concerns: self-healing, self-optimizing, self-healing, and self-protecting, These managers are defined as:

$$\boxed{\text{SelfConfiguringAutonomicManager} \\ \text{ResourceAutonomicManager}}$$

$$\boxed{\text{SelfOptimizingAutonomicManager} \\ \text{ResourceAutonomicManager}}$$

$$\boxed{\text{SelfHealingAutonomicManager} \\ \text{ResourceAutonomicManager}}$$

<i>SelfProtectingAutonomicManager</i> <i>ResourceAutonomicManager</i>
--

For the example, we define a concrete type of orchestrating autonomic managers that manage resource autonomic manager for a single concern:

<i>SingleConcernAutonomicManager</i> <i>OrchestratingAutonomicManager</i> <i>manage</i> : $\mathbb{P} \text{ Knowledge} \times \mathbb{P} \text{ ResourceAutonomicManager} \rightarrow$ $\mathbb{P} \text{ ResourceAutonomicManager}$
--

Finally, we can specify a concrete self-adaptive autonomic system:

<i>SelfAdaptiveAutonomicSystem</i> <i>Environment</i> <i>context</i> : <i>Context</i> <i>resources</i> : $\mathbb{P} \text{ ManagedResource}$ <i>endpointManagers</i> : $\mathbb{P} \text{ ResourceAutonomicManager}$ <i>systemManager</i> : $\mathbb{P} \text{ SingleConcernAutonomicManager}$ <i>server, client1, client2, network</i> : <i>ManagedResource</i> <i>serverOptimizer, networkOptimizer</i> : <i>ResourceAutonomicManager</i> <i>systemOptimizer</i> : <i>SingleConcernAutonomicManager</i> <i>resources</i> = { <i>server, client1, client2, network</i> } <i>endpointManagers</i> = { <i>serverOptimizer, networkOptimizer</i> } <i>systemManager</i> = { <i>systemOptimizer</i> } <i>dom serverOptimizer.manage</i> = { (<i>serverOptimizer.knowledge</i> , { <i>server</i> }) } <i>ran serverOptimizer.manage</i> = { { <i>server</i> } } <i>dom networkOptimizer.manage</i> = { (<i>networkOptimizer.knowledge</i> , { <i>network</i> }) } <i>ran networkOptimizer.manage</i> = { { <i>network</i> } } <i>dom systemOptimizer.manage</i> = { (<i>systemOptimizer.knowledge</i> , { <i>serverOptimizer, networkOptimizer</i> }) } <i>ran systemOptimizer.manage</i> = { { <i>serverOptimizer, networkOptimizer</i> } }
--

In this example, one resource manager is managing a server, another one is managing a network. In addition, there is the system manager who serves as an orchestrating autonomic manager, managing the two resource managers. The specification describes a hierarchy of autonomic managers and specifies the scope of adaptations of the execute computations (i.e. *manage*) of the autonomic managers in the self-adaptive autonomic system.

5 Traffic Monitoring Case Study

Fig. 1 shows a graphical overview of FORMS model applied to the traffic monitoring example.

5.1 Traffic Environment

We define the following attributes of the traffic environment:

$$\left| \begin{array}{l} camera_1, camera_2, camera_3, freeflow_zone_1, congested_zone_1, \\ freeflow_zone_2, congested_zone_2, freeflow_zone_3, \\ congested_zone_3, congested_zone_4, ping_message_{12}, \\ echo_message_{21} : Attribute \end{array} \right.$$

We introduce two names to group two classes of attributes:

$$\begin{aligned} traffic_domain_attributes == \\ & \{ camera_1, camera_2, camera_3, freeflow_zone_1, \\ & congested_zone_1, freeflow_zone_2, congested_zone_2, \\ & freeflow_zone_3, congested_zone_3 \} \end{aligned}$$

$$\begin{aligned} communication_infrastructure_attributes == \\ & \{ ping_message_{12}, echo_message_{21} \} \end{aligned}$$

The traffic processes are defined as follows:

$$\left| \begin{array}{l} monitor_camera_1, monitor_camera_2, monitor_camera_3, \\ transmit : Process \end{array} \right.$$

$$\begin{aligned} traffic_domain_processes == \\ & \{ monitor_camera_1, monitor_camera_2, monitor_camera_3 \} \end{aligned}$$

$$communication_infrastructure_processes == \{ transmit \}$$

A traffic environment is defined as an environment with traffic attributes and a traffic process:

$\begin{array}{l} TrafficEnvironment \\ Environment \end{array}$
$\begin{array}{l} attributes \subseteq traffic_domain_attributes \cup \\ \quad communication_infrastructure_attributes \\ processes \subseteq traffic_domain_processes \cup \\ \quad communication_infrastructure_processes \end{array}$

The traffic environment at T0 in the example is defined as:

$\begin{array}{l} TrafficEnvironment_{T0} \\ TrafficEnvironment \end{array}$
$\begin{array}{l} attributes = \{ camera_1, camera_2, camera_3, freeflow_zone_1, \\ \quad freeflow_zone_2, congested_zone_3 \} \\ processes = traffic_domain_processes \cup \\ \quad communication_infrastructure_processes \end{array}$

We introduce the type *Event* to model events that cause changes in the environment:

$$\left| \begin{array}{l} \textit{Event} : \mathbb{P} \textit{Attribute} \leftrightarrow \mathbb{P} \textit{Attribute} \end{array} \right.$$

We introduce the type *Shutdown* to model terminations of processes in the environment:

$$\left| \begin{array}{l} \textit{Shutdown} : \mathbb{P} \textit{Process} \end{array} \right.$$

Similarly, we introduce the type *Startup* to model the initiation of new processes in the environment:

$$\left| \begin{array}{l} \textit{Startup} : \mathbb{P} \textit{Process} \end{array} \right.$$

The set of events in the traffic environment is defined as:

$$\left| \begin{array}{l} \textit{events} : \mathbb{P} \textit{Event} \\ \hline \textit{events} = \{\{\textit{freeflow_zone}_2\} \mapsto \{\textit{congested_zone}_2\}, \{\textit{camera}_2\} \mapsto \{\}\} \end{array} \right.$$

We define one shutdown event in the traffic environment:

$$\left| \begin{array}{l} \textit{shutdowns} : \mathbb{P} \textit{Shutdown} \\ \hline \textit{shutdowns} = \{\textit{monitor_camera}_2\} \end{array} \right.$$

The change of the traffic state in zone 2 at T1 is defined as:

$$\boxed{\begin{array}{l} \textit{TrafficEnvironment}_{T1} \text{---} \\ \Delta \textit{TrafficEnvironment}_{T0} \\ e? : \textit{events} \\ \hline e? = \{\textit{freeflow_zone}_2\} \mapsto \{\textit{congested_zone}_2\} \\ \textit{attributes}' = \textit{attributes} \setminus \textit{first}(e?) \cup \textit{second}(e?) \\ \textit{processes}' = \textit{processes} \end{array}}$$

From T1 to T2, the traffic environment does not change:

$$\boxed{\begin{array}{l} \textit{TrafficEnvironment}_{T2} \text{---} \\ \exists \textit{TrafficEnvironment}_{T1} \end{array}}$$

A failure of camera 2 changes the traffic environment as follows:

$TrafficEnvironment_{T_3}$ $\Delta TrafficEnvironment_{T_2}$ $e? : events$ $s? : shutdowns$
$e? = \{camera_2\} \mapsto \{\}$ $s? = monitor_camera_2$ $attributes' = attributes \setminus first(e?) \cup second(e?)$ $processes' = processes \setminus \{s?\}$

5.2 Local Traffic Monitoring System

We consider the following traffic environment representations:

$cam_1, cam_2, cam_3, fflow_zone_1, congest_zone_1, fflow_zone_2,$ $congest_zone_2, fflow_zone_3, congest_zone_3, ping_msg_{12},$ $echo_msg_{21} : EnvironmentRepresentation$

$traffic_environment_representations ==$
 $\{cam_1, cam_2, cam_3, fflow_zone_1, congest_zone_1,$
 $fflow_zone_2, congest_zone_2, fflow_zone_3, congest_zone_3,$
 $ping_msg_{12}, echo_msg_{21}\}$

We introduce the following mappings between attribute sets and environment representations in the traffic monitoring case:

$traffic_attribute_representation_mapping ==$
 $\{\{camera_1\} \mapsto cam_1, \{camera_2\} \mapsto cam_2, \{camera_3\} \mapsto cam_3,$
 $\{freeflow_zone_1\} \mapsto fflow_zone_1, \{congested_zone_1\} \mapsto congest_zone_1,$
 $\{freeflow_zone_2\} \mapsto fflow_zone_2, \{congested_zone_2\} \mapsto congest_zone_2,$
 $\{freeflow_zone_3\} \mapsto fflow_zone_3, \{congested_zone_3\} \mapsto congest_zone_3\}$

A local traffic model is defined as follows:

$LocalTrafficModel$ $TrafficEnvironment$ $Model[EnvironmentRepresentation]$ $mapping : \mathbb{P} Attribute \leftrightarrow EnvironmentRepresentation$
$dom\ mapping \subseteq \{attrs : \mathbb{P} Attribute \mid attrs \subseteq attributes\}$ $ran\ mapping = \{r : EnvironmentRepresentation \mid$ $r \in representations\}$

The local traffic model for camera 1 at time T2 is defined:

$$\begin{array}{l}
 \text{LocalTrafficModelOne}_{T_2} \\
 \text{TrafficEnvironment}_{T_2} \\
 \text{Model}[\text{EnvironmentRepresentation}] \\
 \text{mapping} : \mathbb{P} \text{Attribute} \leftrightarrow \text{EnvironmentRepresentation} \\
 \hline
 \text{representations} = \{\text{fflow_zone}_1, \text{cam}_2, \text{cam}_3\} \wedge \\
 \text{mapping} = \{\{\text{freeflow_zone}_1\} \mapsto \text{fflow_zone}_1, \\
 \{\text{camera}_2\} \mapsto \text{cam}_2, \{\text{camera}_3\} \mapsto \text{cam}_3\}
 \end{array}$$

The local traffic model for camera 2 at time T1 is defined:

$$\begin{array}{l}
 \text{LocalTrafficModelTwo}_{T_2} \\
 \text{TrafficEnvironment}_{T_2} \\
 \text{Model}[\text{EnvironmentRepresentation}] \\
 \text{mapping} : \mathbb{P} \text{Attribute} \leftrightarrow \text{EnvironmentRepresentation} \\
 \hline
 \text{representations} = \{\text{congst_zone}_2, \text{cam}_1, \text{cam}_3\} \wedge \\
 \text{mapping} = \{\{\text{congested_zone}_2\} \mapsto \text{congst_zone}_2, \\
 \{\text{camera}_1\} \mapsto \text{cam}_1, \{\text{camera}_3\} \mapsto \text{cam}_3\}
 \end{array}$$

And for camera 3:

$$\begin{array}{l}
 \text{LocalTrafficModelThree}_{T_2} \\
 \text{TrafficEnvironment}_{T_2} \\
 \text{Model}[\text{EnvironmentRepresentation}] \\
 \text{mapping} : \mathbb{P} \text{Attribute} \leftrightarrow \text{EnvironmentRepresentation} \\
 \hline
 \text{representations} = \{\text{congst_zone}_2, \text{congst_zone}_3, \text{cam}_1, \text{cam}_2\} \wedge \\
 \text{mapping} = \{\{\text{congested_zone}_2\} \mapsto \text{congst_zone}_2, \\
 \{\text{congested_zone}_3\} \mapsto \text{congst_zone}_3, \{\text{camera}_1\} \mapsto \text{cam}_1, \\
 \{\text{camera}_2\} \mapsto \text{cam}_2\}
 \end{array}$$

We introduce abstract types for the coordinating elements used by local traffic monitoring computations:

$\text{Role} ::= \text{master} \mid \text{slave}$

$$\begin{array}{l}
 \text{MasterSlave} \\
 \text{role} : \text{Role}
 \end{array}$$

$Name == \mathbb{N}$

<i>OrganizationPartners</i> $partners : \mathbb{P} Name$ $neighborOrganizations : \mathbb{P} Name$
--

A slave has only one partner, i.e. its master. The partners of a master are its slaves and the masters of organizations at neighboring nodes.

<i>MessagePassing</i> $links : Name \leftrightarrow EnvironmentRepresentation$

[*Content*]

<i>Message</i> $from : Name$ $to : \mathbb{P} Name$ $content : Content$
--

$traffic_communication_channel ==$
 $\{1 \mapsto cam_1, 2 \mapsto cam_2, 3 \mapsto cam_3\}$

Local traffic computations use a coordination mechanism based on dynamic agent organizations:

<i>DynamicAgentOrganizations</i> $org_protocol : MasterSlave$ $org_model : OrganizationPartners$ $channel : MessagePassing$
$\forall p : org_model.partners \bullet \exists l : channel.links \bullet first(l) = p \wedge$ $\forall norg : org_model.neighborOrganizations \bullet$ $\exists l : channel.links \bullet first(l) = norg$

we have limited the specification of dynamic agent organizations to the essence of what is needed in the example of a failing camera described below. The interested reader is referred to [6] for a complete formal specification of dynamic agent organizations.

The organizations at T2 are defined:

<i>DynamicAgentOrganizationOne</i> _{T2} <i>DynamicAgentOrganizations</i>
<i>org_protocol.role</i> = <i>master</i> <i>org_model.partners</i> = \emptyset <i>org_model.neighborOrganizations</i> = {3} <i>channel.links</i> = <i>traffic_communication_channel</i> \ {1 \mapsto <i>cam</i> ₁ }

<i>DynamicAgentOrganizationTwo</i> _{T2} <i>DynamicAgentOrganizations</i>
<i>org_protocol.role</i> = <i>slave</i> <i>org_model.partners</i> = {3} <i>org_model.neighborOrganizations</i> = \emptyset <i>channel.links</i> = <i>traffic_communication_channel</i> \ {2 \mapsto <i>cam</i> ₂ }

<i>DynamicAgentOrganizationThree</i> _{T2} <i>DynamicAgentOrganizations</i>
<i>org_protocol.role</i> = <i>master</i> <i>org_model.partners</i> = {2} <i>org_model.neighborOrganizations</i> = {1} <i>channel.links</i> = <i>traffic_communication_channel</i> \ {3 \mapsto <i>cam</i> ₃ }

A local traffic computation is defined as:

<i>LocalTrafficComputation</i> <i>Computation</i> <i>read</i> : <i>LocalTrafficModel</i> \times \mathbb{P} <i>State</i> \rightarrow \mathbb{P} <i>State</i> <i>write</i> : \mathbb{P} <i>State</i> \times <i>LocalTrafficModel</i> \rightarrow <i>LocalTrafficModel</i> <i>perceive</i> : \mathbb{P} <i>State</i> \times <i>Context</i> \rightarrow \mathbb{P} <i>State</i> <i>effect</i> : \mathbb{P} <i>State</i> \times <i>Context</i> \rightarrow <i>Context</i> <i>trafficCoordinationMechanism</i> : <i>DynamicAgentOrganizations</i> <i>send</i> : \mathbb{P} <i>State</i> \rightarrow <i>Message</i> <i>receive</i> : <i>Message</i> \rightarrow \mathbb{P} <i>State</i>
--

Local traffic computations coordinate by exchanging messages.

The local traffic computations at T2 are:

<i>LocalTrafficComputationOne</i> _{T2} <i>LocalTrafficComputation</i> <i>DynamicAgentOrganizationOne</i> _{T2}

*LocalTrafficComputationTwo*_{T2}
LocalTrafficComputation
*DynamicAgentOrganizationTwo*_{T2}

*LocalTrafficComputationThree*_{T2}
LocalTrafficComputation
*DynamicAgentOrganizationThree*_{T2}

A local traffic monitoring system is defined as:

LocalTrafficMonitoringSystem
traffic_model : *LocalTrafficModel*
computation : *LocalTrafficComputation*
 $\text{dom } \textit{computation.read} = \{(\textit{traffic_model}, \textit{computation.state})\} \wedge$
 $\text{dom } \textit{computation.write} = \{(\textit{computation.state}, \textit{traffic_model})\} \wedge$
 $\text{dom } \textit{computation.send} = \{ \textit{computation.state} \}$

The local traffic monitoring systems at T2 are:

*LocalTrafficMonitoringSystemOne*_{T2}
LocalTrafficMonitoringSystem
*LocalTrafficModelOne*_{T2}
*LocalTrafficComputationOne*_{T2}

*LocalTrafficMonitoringSystemTwo*_{T2}
LocalTrafficMonitoringSystem
*LocalTrafficModelTwo*_{T2}
*LocalTrafficComputationTwo*_{T2}

*LocalTrafficMonitoringSystemThree*_{T2}
LocalTrafficMonitoringSystem
*LocalTrafficModelThree*_{T2}
*LocalTrafficComputationThree*_{T2}

5.3 Self-Healing Subsystem

We define two types of reflection models in the traffic monitoring case: dependency model and repair strategy.

To define a dependency model, we introduce the dependency type. For the example we limit the dependencies to neighboring nodes and master slave dependencies.

$$\text{Dependency} ::= \text{neighbor} \mid \text{mymaster} \mid \text{myslave}$$

A dependency model is defined:

$$\begin{array}{l} \text{DependencyModel} \\ \text{dependencies} : \text{Dependency} \leftrightarrow \text{Name} \end{array}$$

The dependency models for the traffic case at T2 are:

$$\begin{array}{l} \text{DependencyModelOne}_{T2} \\ \text{DependencyModel} \\ \text{dependencies} = \{\text{neighbor} \mapsto 2, \text{myslave} \mapsto 0, \text{mymaster} \mapsto 0\} \end{array}$$

$$\begin{array}{l} \text{DependencyModelTwo}_{T2} \\ \text{DependencyModel} \\ \text{dependencies} = \{\text{neighbor} \mapsto 1, \text{neighbor} \mapsto 3, \\ \text{myslave} \mapsto 0, \text{mymaster} \mapsto 3\} \end{array}$$

$$\begin{array}{l} \text{DependencyModelThree}_{T2} \\ \text{DependencyModel} \\ \text{dependencies} = \{\text{neighbor} \mapsto 2, \text{myslave} \mapsto 2, \text{mymaster} \mapsto 0\} \end{array}$$

To model a repair strategy, we introduce a new type of repair actions:

$$\text{RepairActions} == \text{Dependency} \leftrightarrow (\text{Name} \times \text{Name})$$

A repair strategy model is defined as:

$$\begin{array}{l} \text{RepairStrategy} \\ \text{repairActions} : \text{RepairActions} \end{array}$$

The repair strategies for the traffic case are defined as:

$$\frac{\text{RepairStrategyOne}_{T_2}}{\text{RepairStrategy}} \quad \text{repairActions} = \{\text{neighbor} \mapsto (2, 3), \text{neighbor} \mapsto (3, 0)\}$$

$$\frac{\text{RepairStrategyTwo}_{T_2}}{\text{RepairStrategy}} \quad \text{repairActions} = \{\text{neighbor} \mapsto (1, 0), \text{neighbor} \mapsto (3, 0), \text{mymaster} \mapsto (3, 0)\}$$

$$\frac{\text{RepairStrategyThree}_{T_2}}{\text{RepairStrategy}} \quad \text{repairActions} = \{\text{neighbor} \mapsto (2, 1), \text{neighbor} \mapsto (1, 0), \text{myslave} \mapsto (2, 0)\}$$

The coordination model used for fault detection in the traffic monitoring case is defined as:

$$\frac{\text{DependentNodes}}{\text{nodes} : \mathbb{P} \text{Name}}$$

We use a simply model to represent time:

$$\text{Time} == \mathbb{N}$$

The coordination protocol for fault detection is defined as:

$$\frac{\text{PingEcho}}{\quad}$$

The coordination mechanism for fault detection is defined as:

$$\frac{\text{PeerToPeer}}{\text{CoordinationMechanism}[\text{PingEcho}, \text{DependentNodes}, \text{MessagePassing}]}$$

$$\text{ping_time} : \text{Name} \leftrightarrow \text{Time}$$

$$\text{wait_time} : \text{Time}$$

$$\text{dom ping_time} = \text{model.nodes} \wedge$$

$$\forall n : \text{model.nodes} \bullet \exists l : \text{channel.links} \bullet \text{first}(l) = n$$

Concrete instances for the traffic monitoring case at T2 are:

<i>PeerToPeerOne_{T2}</i> <i>PeerToPeer</i>
<i>model.nodes</i> = {2, 3} <i>channel.links</i> = <i>traffic_communication_channel</i> \ {1 ↦ <i>cam</i> ₁ } <i>ping_time</i> = {2 ↦ 4430} <i>wait_time</i> = 40

<i>PeerToPeerTwo_{T2}</i> <i>PeerToPeer</i>
<i>model.nodes</i> = {1, 3} <i>channel.links</i> = <i>traffic_communication_channel</i> \ {2 ↦ <i>cam</i> ₂ } <i>ping_time</i> = {1 ↦ 4432, 3 ↦ 4434} <i>wait_time</i> = 40

<i>PeerToPeerThree_{T2}</i> <i>PeerToPeer</i>
<i>model.nodes</i> = {1, 2} <i>channel.links</i> = <i>traffic_communication_channel</i> \ {3 ↦ <i>cam</i> ₃ } <i>ping_time</i> = {1 ↦ 4436, 3 ↦ 4440} <i>wait_time</i> = 40

Self-healing manager is defined as:

<i>SelfHealingManager</i> <i>Computation</i> <i>coordinationMechanism</i> : <i>PeerToPeer</i> <i>readDM</i> : <i>DependencyModel</i> × \mathbb{P} <i>State</i> → \mathbb{P} <i>State</i> <i>writeDM</i> : \mathbb{P} <i>State</i> × <i>DependencyModel</i> → <i>DependencyModel</i> <i>readRS</i> : <i>RepairStrategy</i> × \mathbb{P} <i>State</i> → \mathbb{P} <i>State</i> <i>writeRS</i> : \mathbb{P} <i>State</i> × <i>RepairStrategy</i> → <i>RepairStrategy</i> <i>sense</i> : <i>LocalTrafficMonitoringSystem</i> × \mathbb{P} <i>State</i> → \mathbb{P} <i>State</i> <i>adapt</i> : <i>LocalTrafficMonitoringSystem</i> × \mathbb{P} <i>State</i> → <i>LocalTrafficMonitoringSystem</i> <i>send</i> : \mathbb{P} <i>State</i> → <i>Message</i> <i>receive</i> : <i>Message</i> → \mathbb{P} <i>State</i>
--

Self-healing managers at T2 are defined:

<i>SelfHealingManagerOne</i> _{T2} <i>SelfHealingManager</i> <i>PeerToPeerOne</i> _{T2}

<i>SelfHealingManagerTwo</i> _{T2} <i>SelfHealingManager</i> <i>PeerToPeerTwo</i> _{T2}

<i>SelfHealingManagerThree</i> _{T2} <i>SelfHealingManager</i> <i>PeerToPeerThree</i> _{T2}

A self-healing subsystem is defined as:

<i>SelfHealingSubsystem</i> <i>dependencyModel</i> : <i>DependencyModel</i> <i>repairStrategy</i> : <i>RepairStrategy</i> <i>selfHealingManager</i> : <i>SelfHealingManager</i>
$\text{dom } selfHealingManager.readDM =$ $\{ (dependencyModel, selfHealingManager.state) \} \wedge$ $\text{dom } selfHealingManager.writeDM =$ $\{ (selfHealingManager.state, dependencyModel) \} \wedge$ $\text{dom } selfHealingManager.readRS =$ $\{ (repairStrategy, selfHealingManager.state) \} \wedge$ $\text{dom } selfHealingManager.writeRS =$ $\{ (selfHealingManager.state, repairStrategy) \} \wedge$ $\text{dom } selfHealingManager.send = \{ selfHealingManager.state \} \wedge$ $\forall dependency : dependencyModel.dependencies \bullet \exists l :$ $selfHealingManager.coordinationMechanism.channel.links;$ $d : Dependency; n : Name \bullet$ $dependency = (d, n) \wedge first(l) = n \wedge$ $\forall repairAction : repairStrategy.repairActions \bullet \exists ol, nl :$ $selfHealingManager.coordinationMechanism.channel.links;$ $d : Dependency; on, nn : Name \bullet$ $repairAction = (d, (on, nn)) \wedge$ $first(ol) = on \wedge first(nl) = nn$

Concrete self-healing subsystems at T2 are defined as:

<i>SelfHealingSubsystemOne</i> _{T2} <i>SelfHealingSubsystem</i> <i>DependencyModelOne</i> _{T2} <i>RepairStrategyOne</i> _{T2} <i>SelfHealingManagerOne</i> _{T2}
--

<i>SelfHealingSubsystemTwo</i> _{T2} <i>SelfHealingSubsystem</i> <i>DependencyModelTwo</i> _{T2} <i>RepairStrategyTwo</i> _{T2} <i>SelfHealingManagerTwo</i> _{T2}
--

<i>SelfHealingSubsystemThree</i> _{T2} <i>SelfHealingSubsystem</i> <i>DependencyModelThree</i> _{T2} <i>RepairStrategyThree</i> _{T2} <i>SelfHealingManagerThree</i> _{T2}
--

To model a timeout of a ping message, we introduce a simple clock:

<i>Clock</i> <i>time</i> : <i>Time</i>

The clock at T2 is defined:

<i>Clock</i> _{T3} <i>Clock</i> <i>time</i> = 4444
--

Time passes by as follows:

<i>Tick</i> Δ <i>Clock</i> <i>time'</i> = <i>time</i> + 1
--

A timeout is defined as:

$\frac{\textit{Timeout}}{\exists \textit{SelfHealingManager}$ \textit{Tick} $n! : \textit{Name}$
$\exists n! : \textit{Name}; t : \textit{Time} \bullet$ $(n!, t) \in \textit{coordinationMechanism.ping_time} \wedge$ $t + \textit{coordinationMechanism.wait_time} > \textit{time}'$

The timeout for self-healing manager 1 after the crash of camera 2 is defined as:

$\frac{\textit{Timeout}_1}{\textit{Timeout}$ $\exists \textit{SelfHealingManagerOne}_{T_2}$ \textit{Tick} $n! : \textit{Name}$
$\textit{time} = 4470$ $n! = 1$

5.4 Traffic Jam Monitoring System

A local camera system is defined as:

$\frac{\textit{LocalCameraSystem}}{\textit{localTrafficMonitoringSystem} : \textit{LocalTrafficMonitoringSystem}$ $\textit{selfHealingSubsystem} : \textit{SelfHealingSubsystem}$ $\textit{myName} : \textit{Name}$
$\text{dom } \textit{selfHealingSubsystem.selfHealingManager.sense} =$ $\{(\textit{localTrafficMonitoringSystem},$ $\quad \textit{selfHealingSubsystem.selfHealingManager.state})\} \wedge$ $\text{dom } \textit{selfHealingSubsystem.selfHealingManager.adapt} =$ $\{(\textit{localTrafficMonitoringSystem},$ $\quad \textit{selfHealingSubsystem.selfHealingManager.state})\}$

The concrete local camera systems at T2 are;

<i>LocalCameraSystemOne</i> _{T2} <i>LocalCameraSystem</i> <i>LocalTrafficMonitoringSystemOne</i> _{T2} <i>SelfHealingSubsystemOne</i> _{T2}
<hr/> <i>myName</i> = 1

<i>LocalCameraSystemTwo</i> _{T2} <i>LocalCameraSystem</i> <i>LocalTrafficMonitoringSystemTwo</i> _{T2} <i>SelfHealingSubsystemTwo</i> _{T2}
<hr/> <i>myName</i> = 2

<i>LocalCameraSystemThree</i> _{T2} <i>LocalCameraSystem</i> <i>LocalTrafficMonitoringSystemThree</i> _{T2} <i>SelfHealingSubsystemThree</i> _{T2}
<hr/> <i>myName</i> = 3

A situated local camera system is defined as:

<i>SituatedLocalCameraSystem</i> <i>TrafficEnvironment</i> <i>LocalCameraSystem</i> <i>context</i> : <i>Context</i>
<hr/> $context \subseteq attributes \wedge$ $dom(localTrafficMonitoringSystem.computation.perceive) =$ $\{attrs : Context \mid attrs \subseteq context \bullet$ $(localTrafficMonitoringSystem.computation.state, attrs)\} \wedge$ $dom(localTrafficMonitoringSystem.computation.effect) =$ $\{attrs : Context \mid attrs \subseteq context \bullet$ $(localTrafficMonitoringSystem.computation.state, attrs)\}$

The concrete situated local cameras at T2 are:

<i>SituatedLocalCameraSystemOne</i> _{T2} <i>TrafficEnvironment</i> _{T2} <i>LocalCameraSystemOne</i> _{T2} <i>context</i> : <i>Context</i>
<hr/> $context = \{camera_2, camera_3, freeflow_zone_1\}$

<i>SituatedLocalCameraSystemTwo</i> _{T2} <i>TrafficEnvironment</i> _{T2} <i>LocalCameraSystemTwo</i> _{T2} <i>context</i> : <i>Context</i>
<i>context</i> = { <i>camera</i> ₁ , <i>camera</i> ₃ , <i>congested_zone</i> ₂ }

<i>SituatedLocalCameraSystemThree</i> _{T2} <i>TrafficEnvironment</i> _{T2} <i>LocalCameraSystemThree</i> _{T2} <i>context</i> : <i>Context</i>
<i>context</i> = { <i>camera</i> ₁ , <i>camera</i> ₂ , <i>congested_zone</i> ₃ }

A traffic jam monitoring system is defined as:

<i>TrafficJamMonitoringSystem</i> <i>localCamaraSystems</i> : \mathbb{P} <i>SituatedLocalCameraSystem</i>
$\forall lcs : localCamaraSystems; msgs : \mathbb{P} Message; addressees : \mathbb{P} Name \bullet$ $msgs = ran (lcs.localTrafficMonitoringSystem.computation.send) \wedge$ $addressees = \{n : Name; msg : msgs \mid n = msg.from \bullet n\} \wedge$ $addressees = dom (lcs.localTrafficMonitoringSystem.computation.$ $trafficCoordinationMechanism.channel.links) \wedge$ $\forall lcs : localCamaraSystems; d : Dependency; n : Name \bullet$ $(d, n) \in lcs.selfHealingSubsystem.dependencyModel.dependencies$ $\wedge n \neq lcs.myName \wedge$ $\forall lcs : localCamaraSystems; shmsgs : \mathbb{P} Message; shaddressees : \mathbb{P} Name \bullet$ $shmsgs = ran (lcs.selfHealingSubsystem.selfHealingManager.send) \wedge$ $shaddressees = \{n : Name; msg : msgs \mid n = msg.from \bullet n\} \wedge$ $shaddressees = dom (lcs.selfHealingSubsystem.selfHealingManager.$ $coordinationMechanism.channel.links)$

The specification defines the scope of communication in the system, and the dependencies.

At T2 the state of the traffic jam monitoring system is:

<i>TrafficJamMonitoringSystem</i> _{T2} <i>TrafficJamMonitoringSystem</i> <i>SituatedLocalCameraSystemOne</i> _{T2} <i>SituatedLocalCameraSystemTwo</i> _{T2} <i>SituatedLocalCameraSystemThree</i> _{T2}

At T3 when camera 2 fails, the state of the traffic camera system is changed as follows:

$$\begin{array}{l}
\text{TrafficJamMonitoringSystem}_{T_3} \\
\Delta \text{TrafficJamMonitoringSystem}_{T_2} \\
lcs2? : \text{SituatedLocalCameraSystem} \\
\hline
lcs2? \in \text{localCamaraSystems} \wedge \\
lcs2?.myName = 2 \wedge \\
\text{localCamaraSystems}' = \text{localCamaraSystems} \setminus \{lcs2?\}
\end{array}$$

To conclude, we formalize how camera 1 recovers from the failure of camera 2 that happens after the time out of the ping message. First we define two helper functions to update the different parts of the camera system:

$$\begin{array}{l}
\text{adaptLocalTrafficMonitoringSystem} : \text{SituatedLocalCameraSystem} \times \\
\text{Attribute} \times \text{EnvironmentRepresentation} \times \text{Name} \rightarrow \\
\text{LocalTrafficMonitoringSystem} \\
\hline
\forall slcs : \text{SituatedLocalCameraSystem}; \\
\text{ultms} : \text{LocalTrafficMonitoringSystem}; \text{camera} : \text{Attribute}; \\
\text{cam} : \text{EnvironmentRepresentation}; n : \text{Name} \bullet \\
\text{ultms.traffic_model.representations} = \\
\text{slcs.localTrafficMonitoringSystem.traffic_model.representations} \setminus \\
\{ \text{cam} \} \wedge \\
\text{ultms.traffic_model.mapping} = \\
\text{slcs.localTrafficMonitoringSystem.traffic_model.mapping} \setminus \\
\{ \{ \text{camera} \} \mapsto \text{cam} \} \wedge \\
\text{ultms.computation.} \\
\text{trafficCoordinationMechanism.org_protocol.role} = \\
\text{slcs.localTrafficMonitoringSystem.computation.} \\
\text{trafficCoordinationMechanism.org_protocol.role} \wedge \\
\text{ultms.computation.} \\
\text{trafficCoordinationMechanism.org_model.partners} = \\
\text{slcs.localTrafficMonitoringSystem.computation.} \\
\text{trafficCoordinationMechanism.org_model.partners} \setminus \\
\{ n \} \wedge \\
\text{ultms.computation.} \\
\text{trafficCoordinationMechanism.org_model.neighborOrganizations} = \\
\text{slcs.localTrafficMonitoringSystem.computation.} \\
\text{trafficCoordinationMechanism.} \\
\text{org_model.neighborOrganizations} \wedge \\
\text{ultms.computation.} \\
\text{trafficCoordinationMechanism.channel.links} = \\
\text{slcs.localTrafficMonitoringSystem.computation.} \\
\text{trafficCoordinationMechanism.channel.links} \setminus \\
\{ n \mapsto \text{cam} \} \wedge \\
\text{adaptLocalTrafficMonitoringSystem}(slcs, \text{camera}, \text{cam}, n) = \text{ultms}
\end{array}$$

The first helper function takes a situated local camera system and the data of a camera that fails and returns the adapted local traffic monitoring system of the camera system. The function is applicable for situations in which a neighboring camera fails that plays the role of slave. The adaptation includes:

- The representation of the camera is removed from the set of representations;
- The mapping of the representation to the real camera is removed;
- The role of the traffic monitoring system is not changed;
- The failing camera is removed from the list of partners;
- The neighboring organizations are not changed (the failing camera is a slave of a neighboring organization);
- The communication link to the failing camera is removed.

$$\begin{aligned}
& \text{updateSelfHealingSubsystem} : \text{SituatedLocalCameraSystem} \times \\
& \quad \text{Attribute} \times \text{EnvironmentRepresentation} \times \text{Name} \\
& \quad \rightarrow \text{SelfHealingSubsystem} \\
\hline
& \forall \text{slcs} : \text{SituatedLocalCameraSystem}; \\
& \quad \text{ushs} : \text{SelfHealingSubsystem}; \text{camera} : \text{Attribute}; \\
& \quad \text{cam} : \text{EnvironmentRepresentation}; n : \text{Name} \bullet \\
& \exists \text{newneighbor} : \text{Name} \bullet \text{slcs.selfHealingSubsystem.repairStrategy.} \\
& \quad \text{repairActions} \triangleright \{(n, \text{newneighbor})\} = \\
& \quad \quad \{\text{neighbor} \mapsto (n, \text{newneighbor})\} \wedge \\
& \text{slcs.selfHealingSubsystem.dependencyModel.dependencies} = \\
& \quad \text{ushs.dependencyModel.} \\
& \quad \quad \text{dependencies} \oplus \{\text{neighbor} \mapsto \text{newneighbor}\} \wedge \\
& \text{slcs.selfHealingSubsystem.repairStrategy.repairActions} = \\
& \quad \text{ushs.repairStrategy.repairActions} \wedge \\
& \text{slcs.selfHealingSubsystem.selfHealingManager.state} = \\
& \quad \text{ushs.selfHealingManager.state} \wedge \\
& \text{slcs.selfHealingSubsystem.selfHealingManager.} \\
& \quad \text{coordinationMechanism.protocol} = \\
& \quad \quad \text{ushs.selfHealingManager.coordinationMechanism.protocol} \wedge \\
& \text{slcs.selfHealingSubsystem.selfHealingManager.coordinationMechanism.} \\
& \quad \text{model.nodes} = \text{ushs.selfHealingManager.coordinationMechanism.} \\
& \quad \quad \text{model.nodes} \setminus \{n\} \wedge \\
& \text{slcs.selfHealingSubsystem.selfHealingManager.coordinationMechanism.} \\
& \quad \text{channel.links} = \text{ushs.selfHealingManager.coordinationMechanism.} \\
& \quad \quad \text{channel.links} \setminus \{n \mapsto \text{cam}\} \wedge \\
& \exists \text{pt} : \text{Time} \bullet \{n\} \triangleleft \text{slcs.selfHealingSubsystem.selfHealingManager.} \\
& \quad \text{coordinationMechanism.ping_time} = \{(n \mapsto \text{pt})\} \wedge \\
& \text{slcs.selfHealingSubsystem.selfHealingManager.coordinationMechanism.} \\
& \quad \text{ping_time} = \text{ushs.selfHealingManager.coordinationMechanism.} \\
& \quad \quad \text{ping_time} \setminus \{n \mapsto \text{pt}\} \wedge \\
& \text{slcs.selfHealingSubsystem.selfHealingManager.coordinationMechanism.} \\
& \quad \text{wait_time} = \text{ushs.selfHealingManager.coordinationMechanism.} \\
& \quad \quad \text{wait_time} \wedge \\
& \text{updateSelfHealingSubsystem}(\text{slcs}, \text{camera}, \text{cam}, n) = \text{ushs}
\end{aligned}$$

The second helper function updates the self-healing system after a camera fails. This function is applicable for the same type of situations as the first helper function. The update includes:

- The dependencies are updated with the new neighbor;
- The repair actions are not changed;
- The computation state of the self-healing manager is not changed;
- The coordination protocol is not changed;
- The node of the failing camera is removed from the coordination model;
- The communication link to the failing camera is removed;

- The ping time to the failing camera is removed;
- The wait time for ping messages is not changed.

Finally, the recovery is defined as:

$$\begin{array}{l}
\text{CameraOneRecoversFromFailureCameraTwo} \\
\Delta \text{TrafficJamMonitoringSystem}_{T_3} \\
\text{TrafficEnvironment}_{T_3} \\
\text{Timeout}_1 \\
lcs1?, lcs1! : \text{SituatedLocalCameraSystem} \\
\text{camera} : \text{Attribute} \\
\text{cam} : \text{EnvironmentRepresentation} \\
n : \text{Name} \\
\{ \text{camera} \} = \text{first}(e?) \wedge \\
\{ \{ \text{camera} \} \} \triangleleft \text{traffic_attribute_representation_mapping} = \\
\{ \{ \text{camera} \} \mapsto \text{cam} \} \wedge \\
\text{traffic_communication_channel} \triangleright \{ \text{cam} \} = \{ n \mapsto \text{cam} \} \wedge \\
lcs1? \in \text{localCamaraSystems} \wedge lcs1?.\text{myName} = 1 \wedge \\
lcs1!.myName = lcs1?.myName \wedge \\
lcs1!.context = lcs1?.context \setminus \{ \text{camera} \} \wedge \\
lcs1!.selfHealingSubsystem = \\
\quad \text{updateSelfHealingSubsystem}(lcs1?, \text{camera}, \text{cam}, n) \wedge \\
lcs1!.localTrafficMonitoringSystem = \\
\quad \text{adaptLocalTrafficMonitoringSystem}(lcs1?, \text{camera}, \text{cam}, n) \wedge \\
\text{localCamaraSystems}' = \text{localCamaraSystems} \setminus \{ lcs1? \} \cup \{ lcs1! \}
\end{array}$$

The specification *declaratively* specifies how the state of the local camera system is adapted after the crash. The adaptation consists of two parts, an update of the state of the self-healing subsystem and the actual adaptation of the local traffic monitoring system. *Operationally*, the self-healing manager will update its state and apply the adaptation of the local traffic monitoring system using various read and write operations. An analogous specification can be defined for the recovery of camera 3 in the scenario.

References

- [1] CZT. <http://czt.sourceforge.net/>, Jan 2010.
- [2] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.
- [3] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *Int'l Conf. on Software Engineering*, Minneapolis, MN, 2007.
- [4] P. Maes. Concepts and experiments in computational reflection. In *OOPSLA*, Orlando, FL, 1987.

- [5] P. Oreizy et al. Architecture-based runtime software evolution. In Int'l Conf. on Software engineering, Kyoto, Japan, 1998.
- [6] D. Weyns, R. Haesevoets, and A. Helleboogh. The MACODO organization model for context-driven dynamic agent organizations. *TAAS*, 2010. www.cs.kuleuven.be/~danny/papers/2010TAAS-model.pdf.
- [7] D. Weyns, S. Malek, and J. Andersson. FORMS: FOrmal Reference Model for Self-adaptation. In *International Conference on Autonomic Computing and Communication*, Washington, 2010. www.cs.kuleuven.be/~danny/papers/2010ICAC.pdf.
- [8] D. Weyns, S. Malek, and J. Andersson. On decentralized self-adaptation: Lessons from the trenches and challenges for the future. In *International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Cape Town, 2010. www.cs.kuleuven.be/~danny/papers/2010SEAMS.pdf.