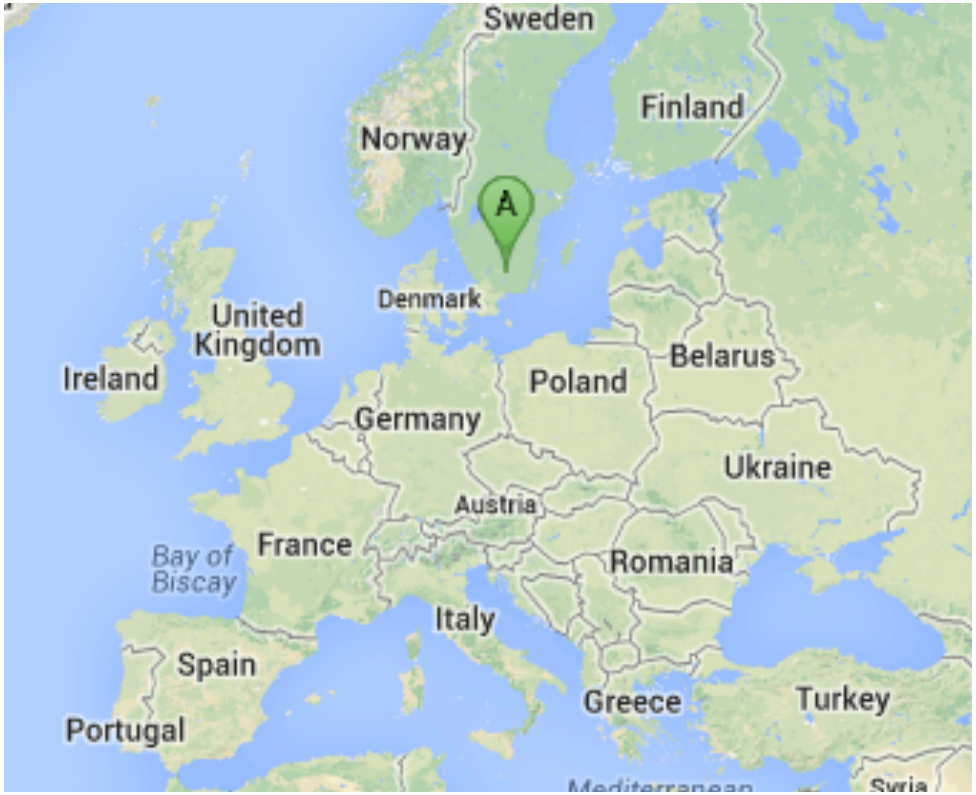
Assurances for Self-Adaptive Systems

SERENE Autumn School
October 2, 2013 Kiev

Danny Weyns, Linnaeus University Sweden
danny.weyns@lnu.se
http://homepage.lnu.se/staff/daweaa/index.htm

Your tutor this afternoon



Linnaeus University Växjö campus – Sweden

Research team focusing on software architecture and self-adaptive systems

Motivation

- Engineering contemporary software systems is complex due to uncertainties at design time
  - Changing availability of resources
  - Faults that are difficult to predict
  - Changing or new user goals

- How to engineer such systems and guarantee system goals regarding of the uncertainties?

Promise of self-adaptive systems*

Self-adaptive systems are able to adjust their behavior in response to their perception of the environment and the system itself

to become more resilient, dependable, robust, energy-efficient […]

*B. Cheng et al., Software Engineering for Self-Adaptive Systems: A Research Roadmap, Lecture Notes in Computer Science, vol. 5525, 2009

Promise of formal approaches for self-adaptive systems*

Formal methods offer a means to

provide evidence that the system requirements are satisfied during operation

regarding the uncertainty of changes that may affect the system, its environment or its goals

*Software Engineering for Self-Adaptive Systems: Assurances
www.dagstuhl.de/de/programm/kalender/semhp/?semnr=13511

Goals of this tutorial

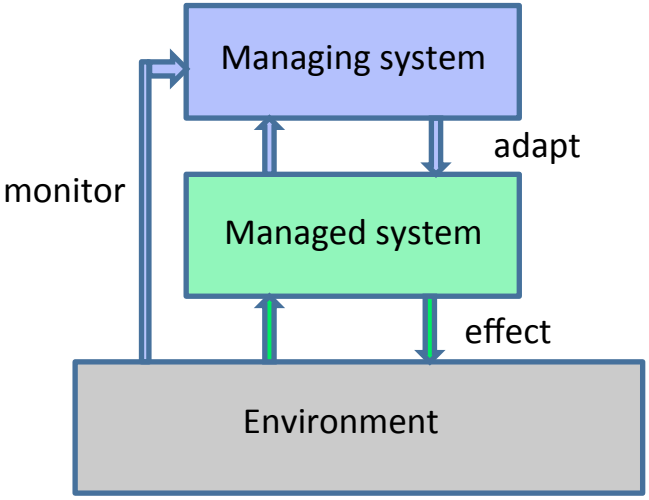
- Understand the notion of self-adaptation
- Get familiar with references approaches for architecture-based self-adaptation
- Get familiar with state of the art in formal methods for self-adaptive systems
- Understand the challenges in formal methods at runtime for self-adaptive systems

Overview

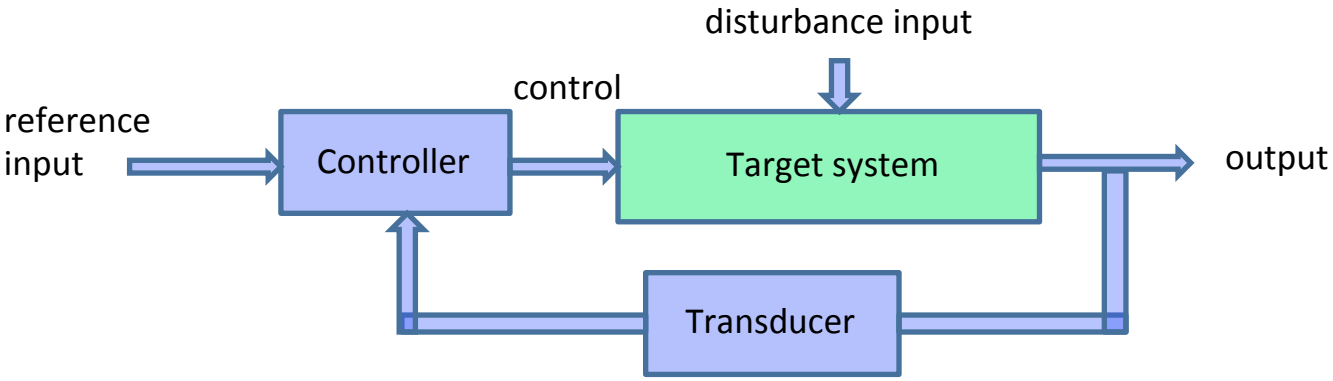
- Architecture-based self-adaptation vs. control-based self-adaptation
- Reference approaches for architecture-based self-adaptation
- Formal methods for self-adaptive systems
- Active formal methods for self-adaptation
- Wrap up

Self-adaptation

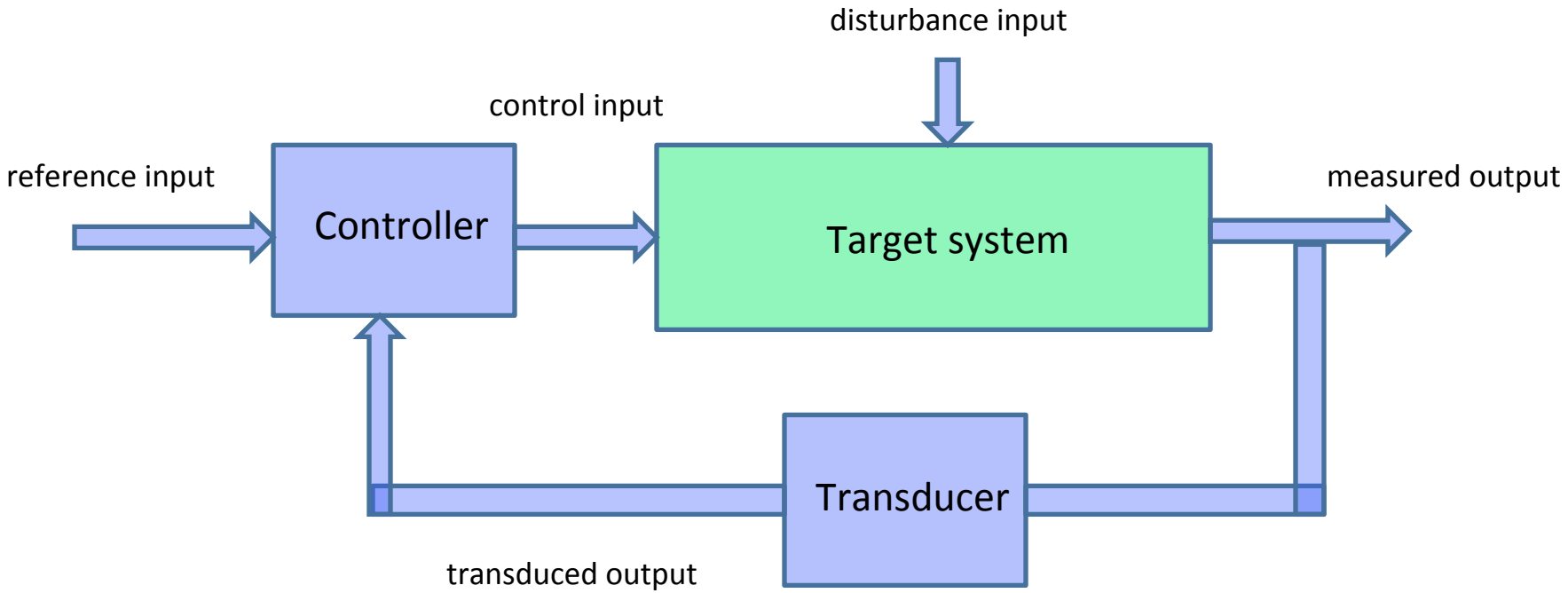
Architecture-based self-adaptation



Control-based self-adaptation

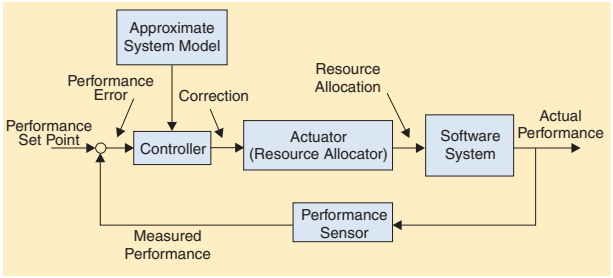


Basic model control-based self-adaptation

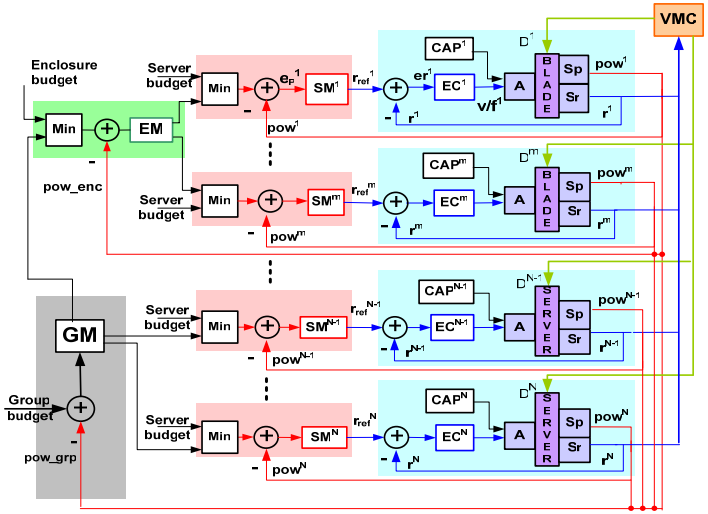


Discrete time dynamic system
x(k+1) = f(x(k),u(k),dx(k))
x: state; u: input; dx: state disturbances
y(k) = g(x(k),u(k),dy(k))
y: output; u: input; dy: output disturbances

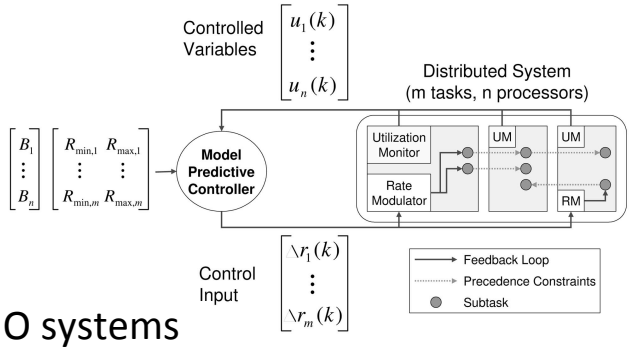
Control-based self-adaptation



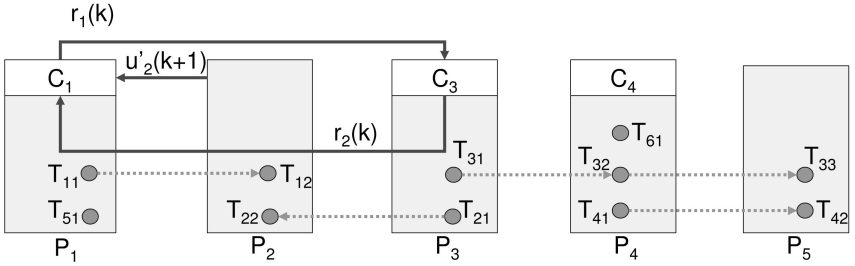
Classic controllers
(Abdelzaher et al. 2003)



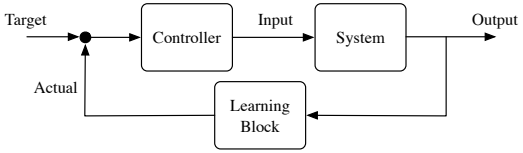
Nested and layered architectures
(Zhu et al, 2006; Kusic et al. 2009)

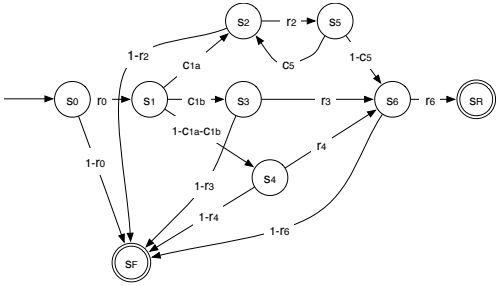


MIMO systems
(Dio et al., 2002, Lu et al. 2005)



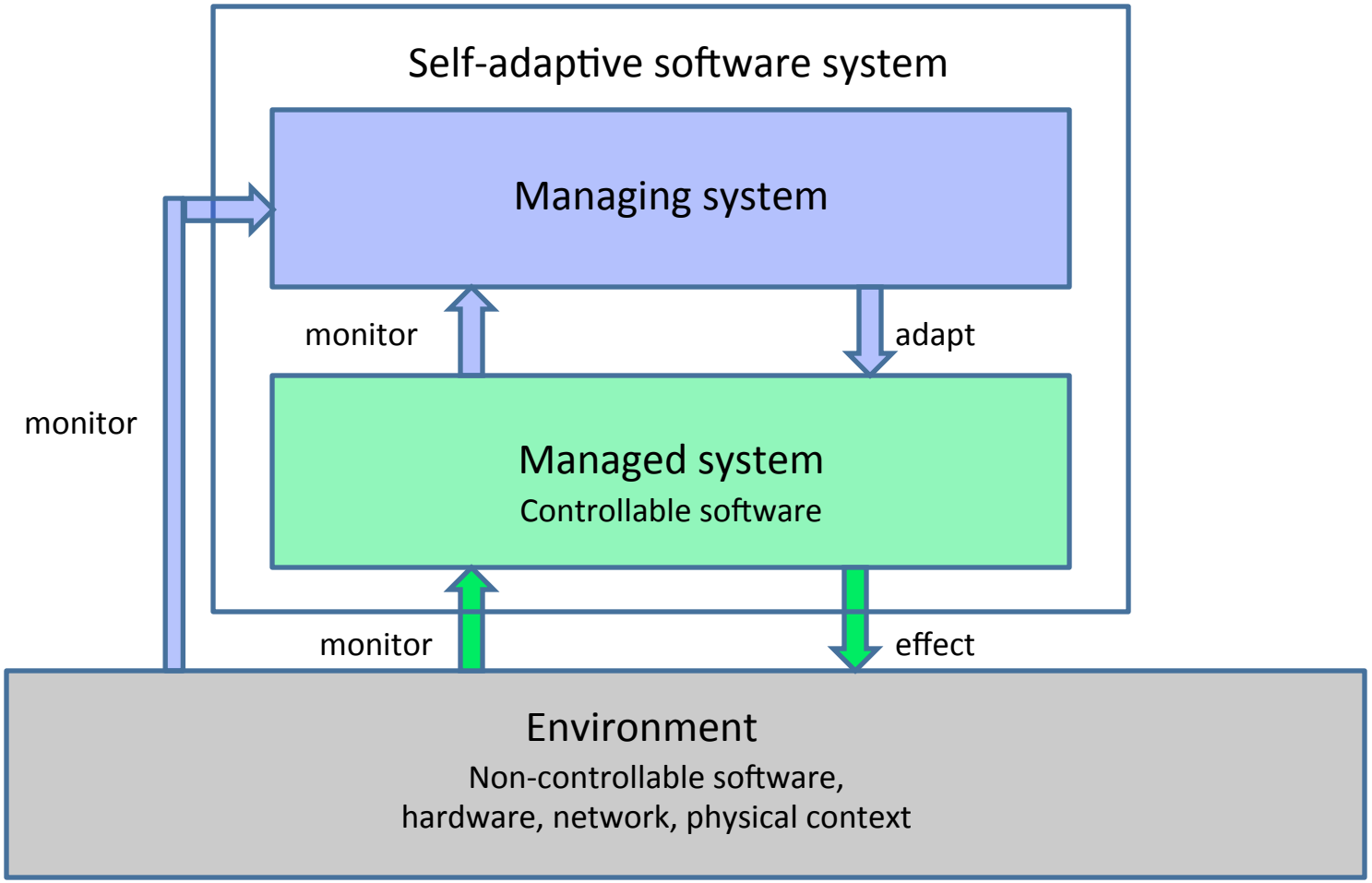
Decentralized control
(X. Wang et al., 2007; R. Wang et al 2012)





Controlling software vs. resources
(Filieri et al. 2011; Maggio et al. 2012)

Basic model architecture-based self-adaptation



Overview

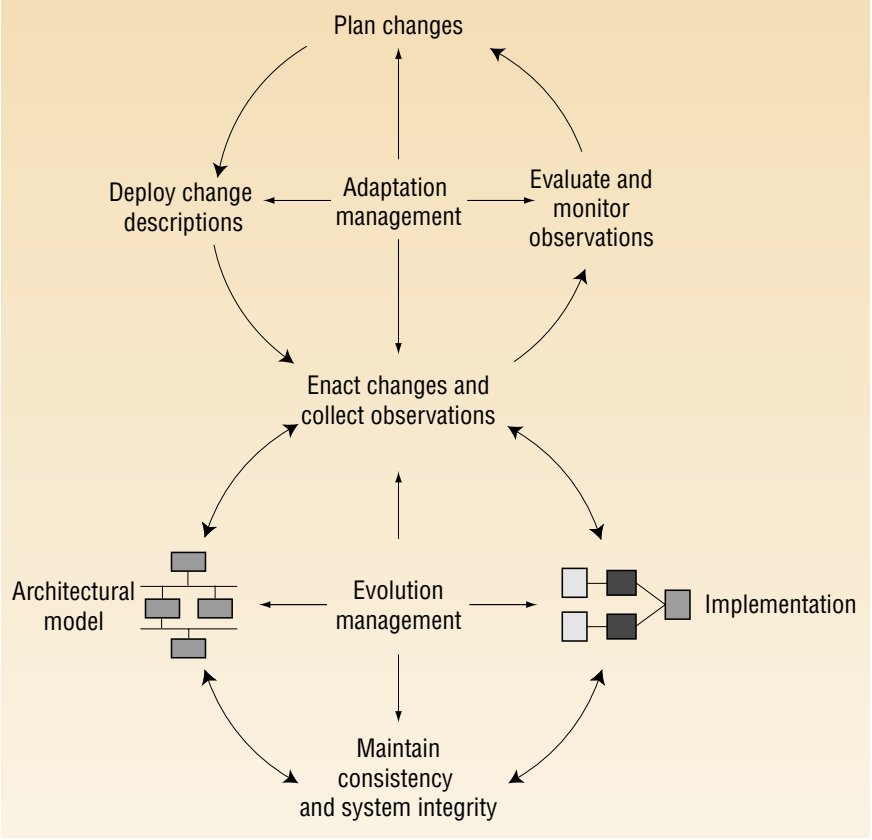
- Architecture-based self-adaptation vs. control-based self-adaptation
- Reference approaches for architecture-based self-adaptation
- Formal methods for self-adaptive systems
- Active formal methods for self-adaptation
- Wrap up

Reference approaches for architecture-based self-adaptation

- 1999: Oreizy et al.
- 2003: MAPE-K IBM
- 2004: Rainbow
- 2007: 3-layer reference model
- 2012: FORMS

Reference approaches for self-adaptation
Oreizy et al. 1999

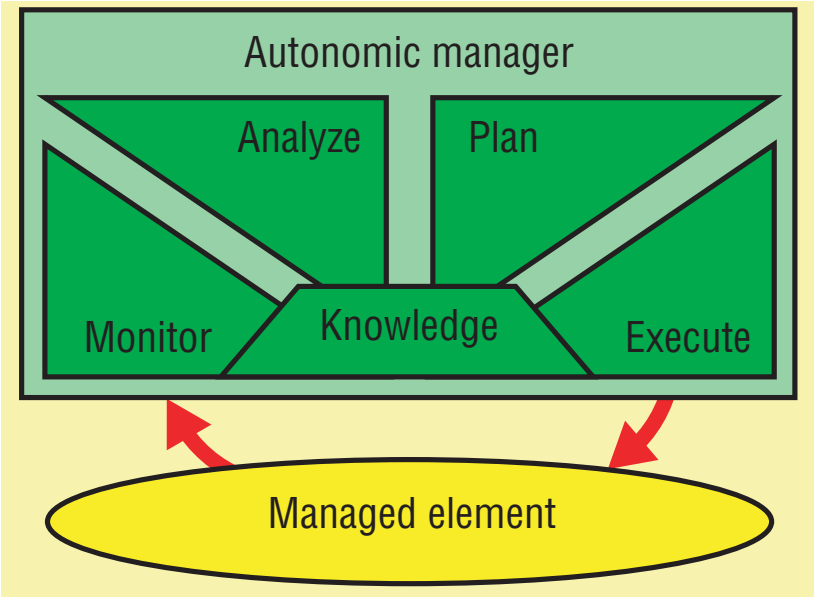
- Adaptation management
  - Life cycle of self-adaptation
  - System monitors and adapts itself
- Evolution management
  - Change of application software
  - Maintain consistency



P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, An Architecture-Based Approach to Self-Adaptive Software, IEEE Intelligent Systems, May/June 1999

Reference approaches for self-adaptation
IBM MAPE-K 2003

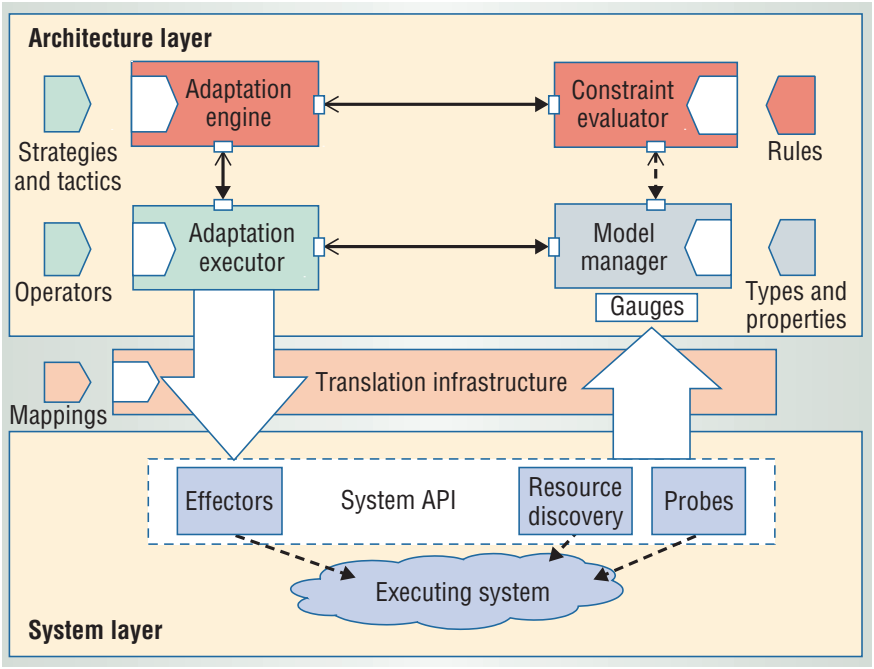
- Autonomic manager
  - Reference model
  - Four key functions + knowledge
- Four types of self-adaptations
  - Self-configuration
  - Self-optimization
  - Self-healing
  - Self-protection



Kephart and Chess, The vision of autonomic Computing, IEEE Computer, January 2003

Reference approaches for self-adaptation
Rainbow 2004

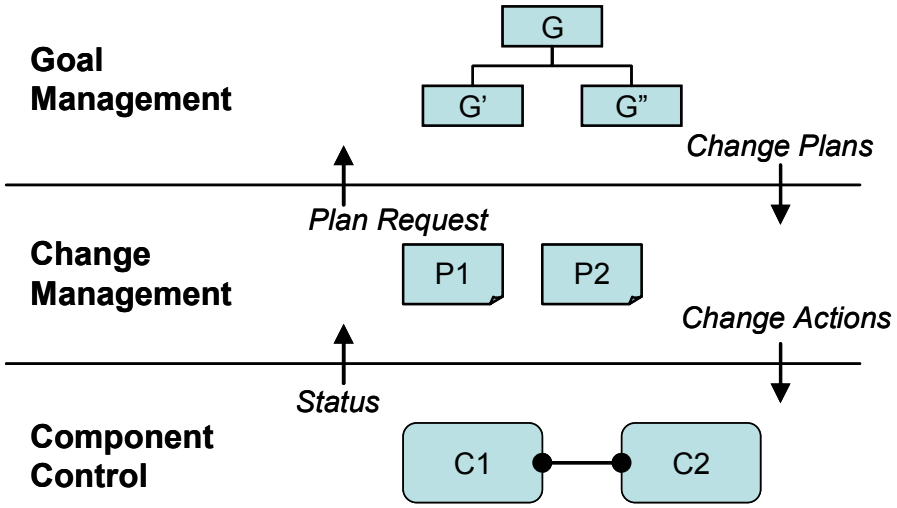
- Framework realizes MAPE control loop
- Uses architecture model of system and context
- Checks constraints
- Adapts running system if violation is detected



D. Garlan, S-W. Cheng, A.C. Huang, B. Schmerl, P. Steenkiste, Rainbow: Architecture- Based Self-Adaptation with Reusable Infrastructure, IEEE Computer, October 2004

Reference approaches for self-adaptation
3-Layer reference model 2007

- Reference model based on Gat's 3-layer robotics model
- Component control realizes application functions
- Change management handles adaptations of component layer based on set of plans
- Goal management produces change management plans when needed (e.g., to deal with new conditions or goals)



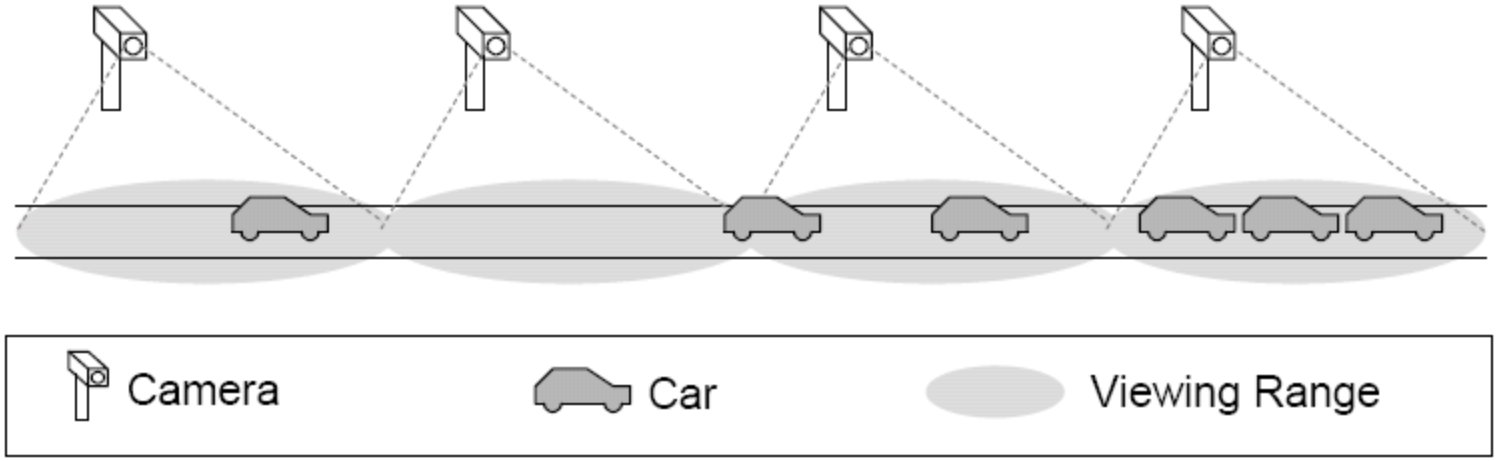
Kramer and Magee, Self-adaptation: an architecture challenge, Future of Software Engineering, FOSE 2007

Reference approaches for self-adaptation FORMS 2012

- FOrmal Reference Model for Self-adaptation
- Integrates different perspectives on self-adaptation
  - Reflection perspective
  - MAPE-K perspective
  - Distribution perspective

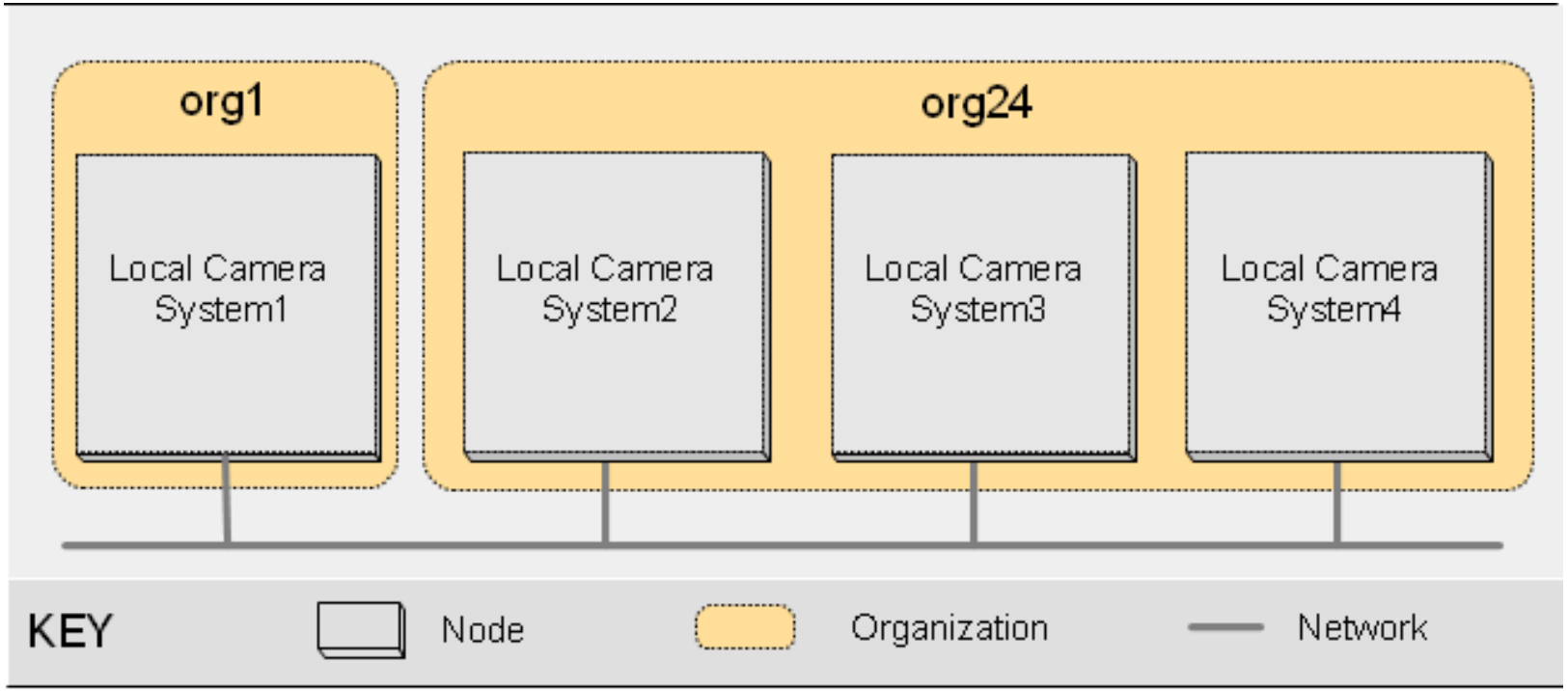
D. Weyns, S. Malek, J. Andersson, FORMS: Formal reference model for self-adaptation, ACM Transactions on Autonomous and Adaptive Systems, TAAS 7(1), 2012

FORMS: Running Example
Traffic jam monitoring

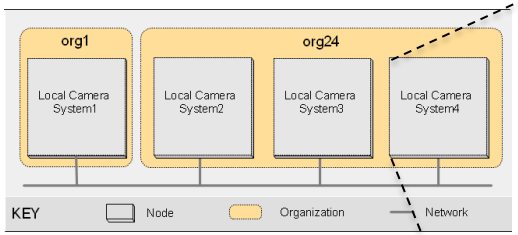


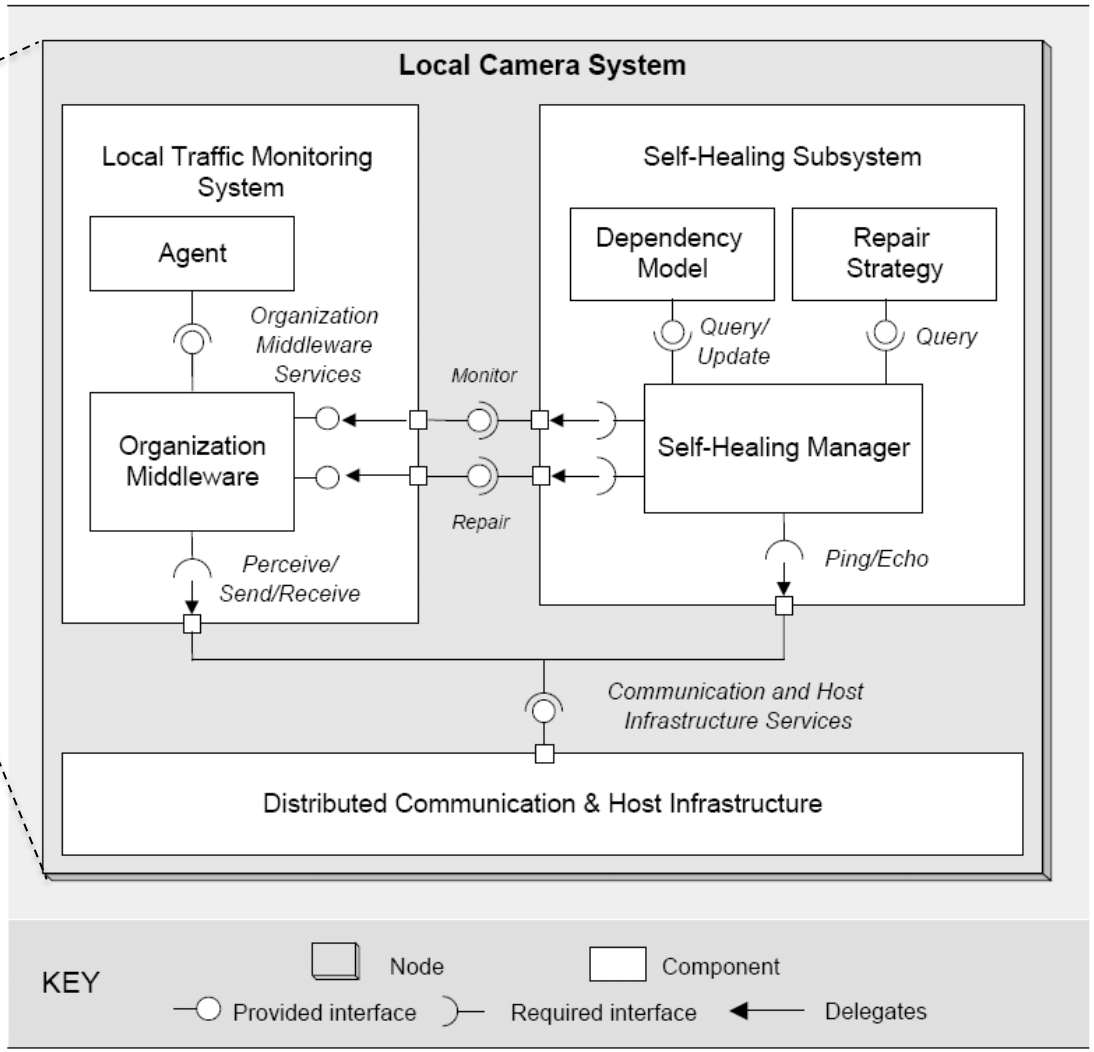
D. Weyns, R. Haesevoets, A. Helleboogh, T. Holvoet, W. Joosen, The MACODO Middleware for Context-Driven Dynamic Agent Organzations, ACM Transaction on Autonomous and Adaptive Systems, 5(1):3.1–3.29, 2010.

Running Example: traffic jam monitoring

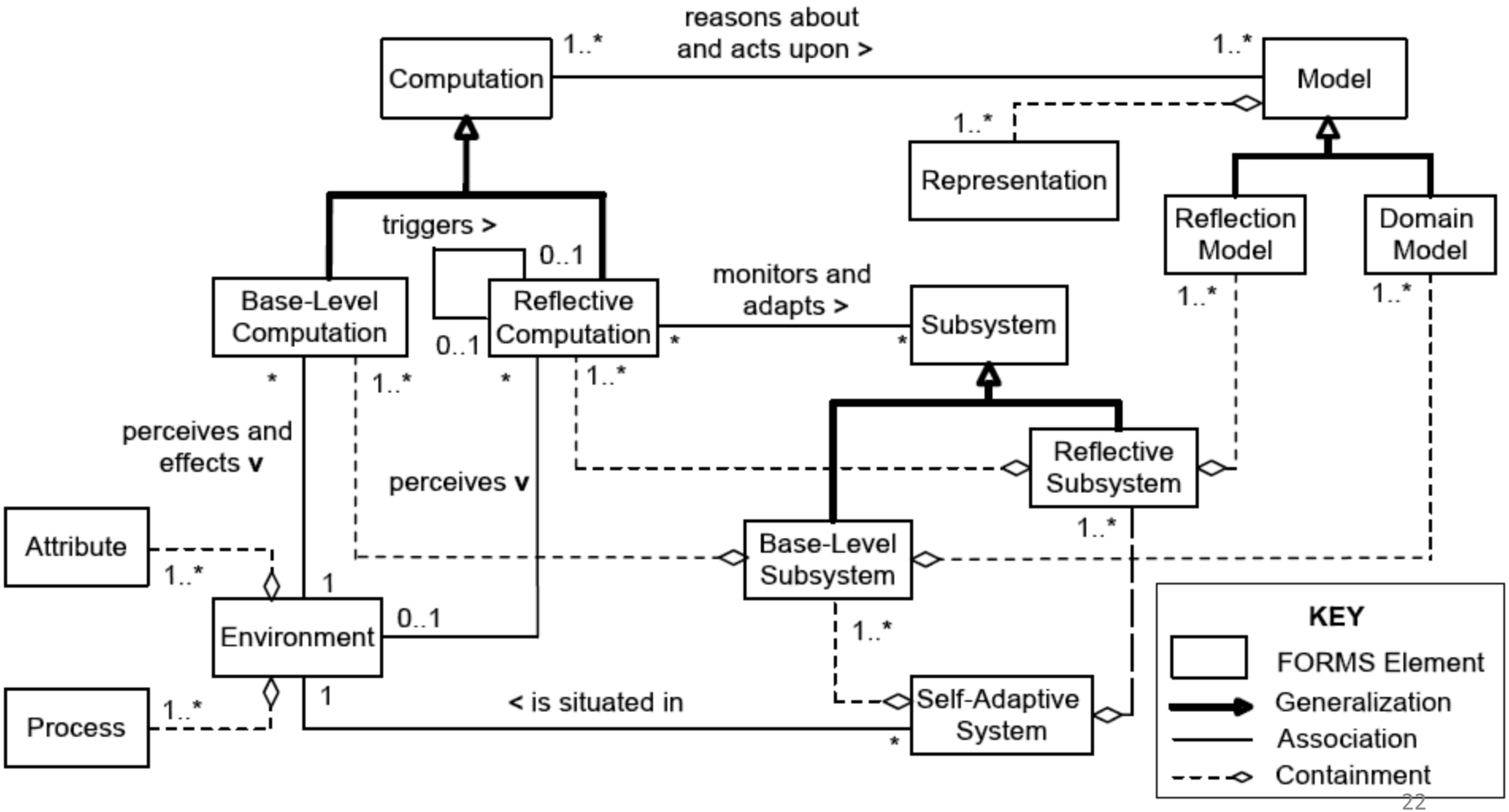


Running Example: traffic jam monitoring



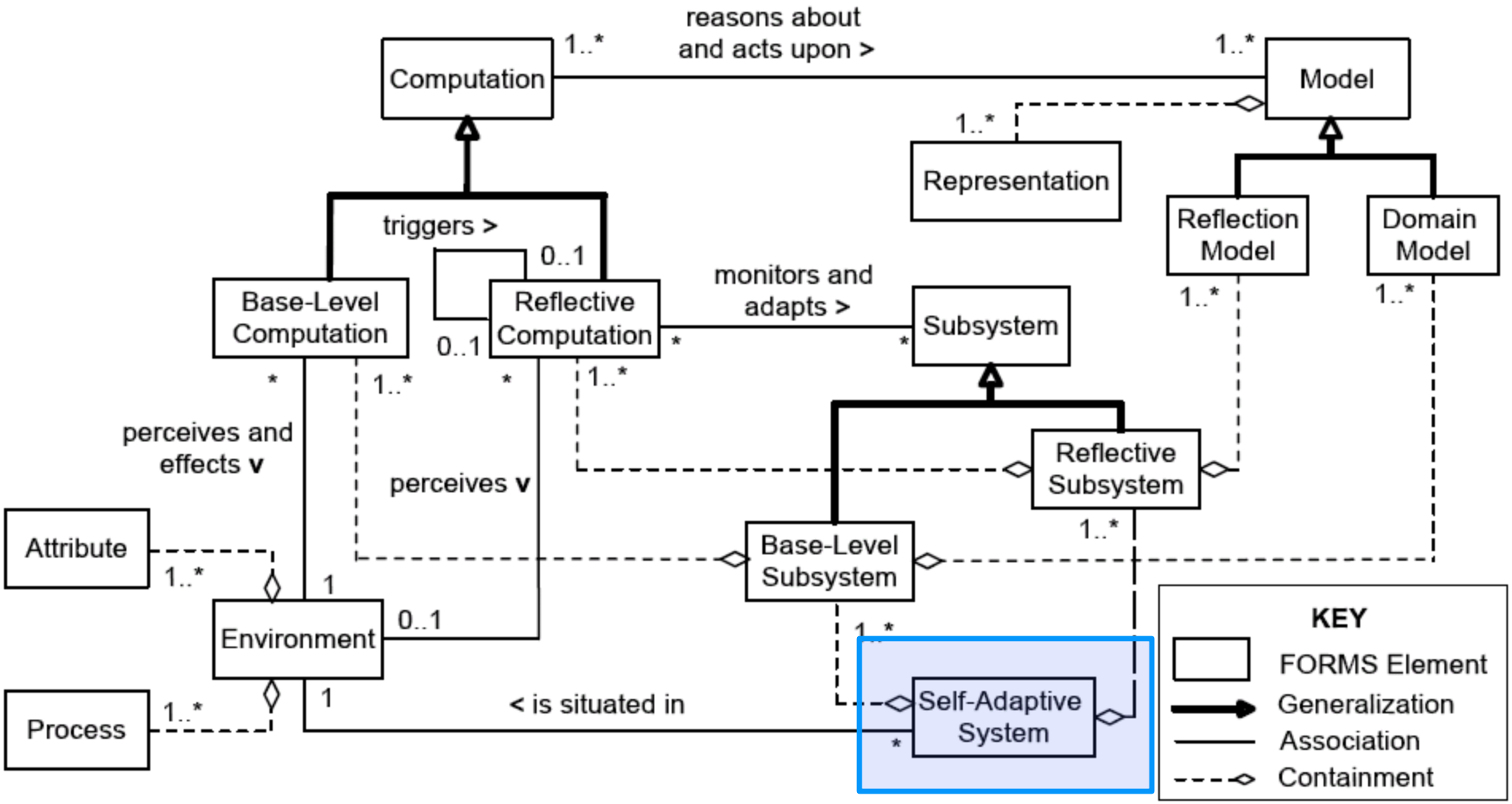


FORMS Reflection Perspective



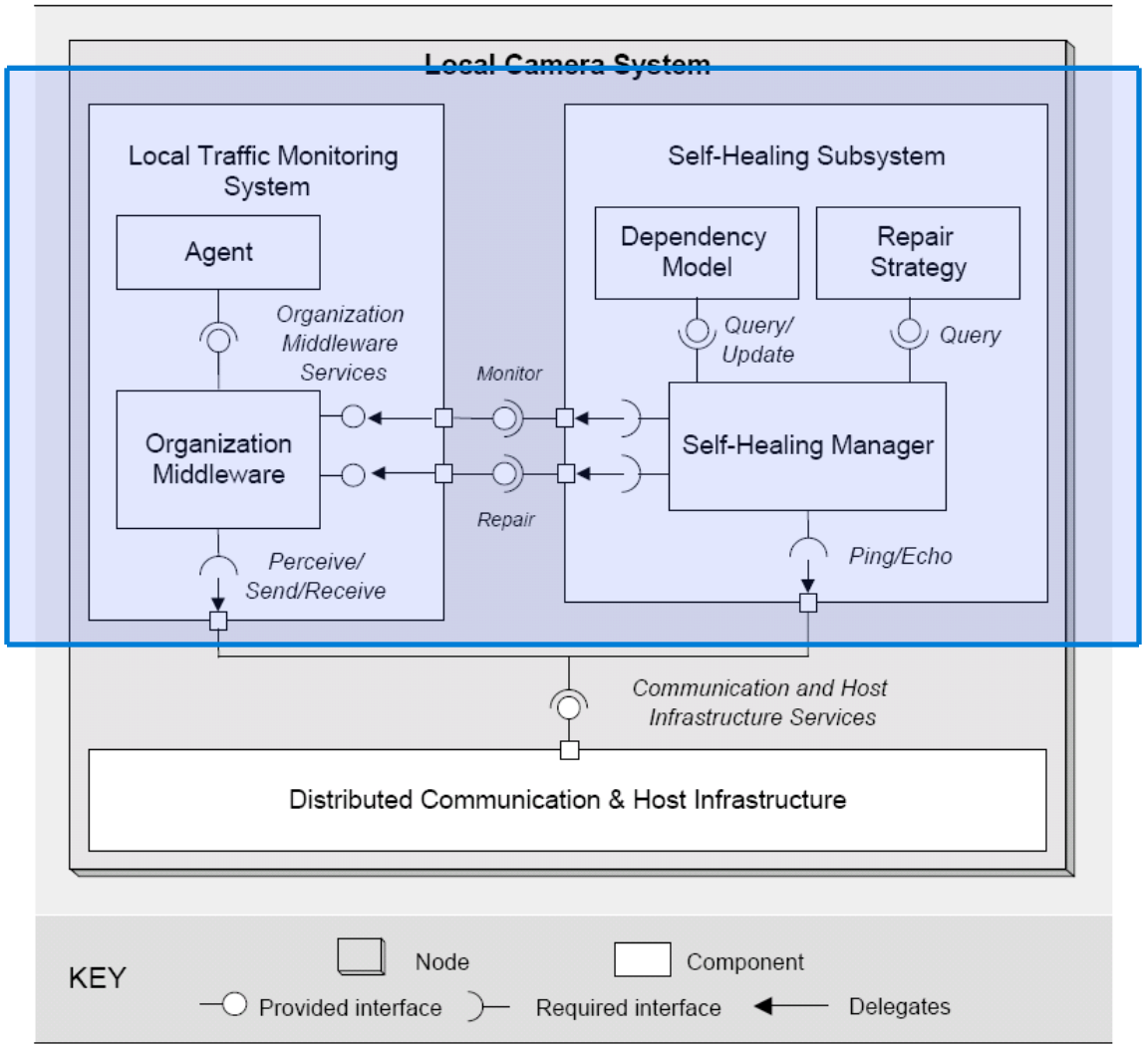
FORMS Reflection Perspective

Self-adaptive system



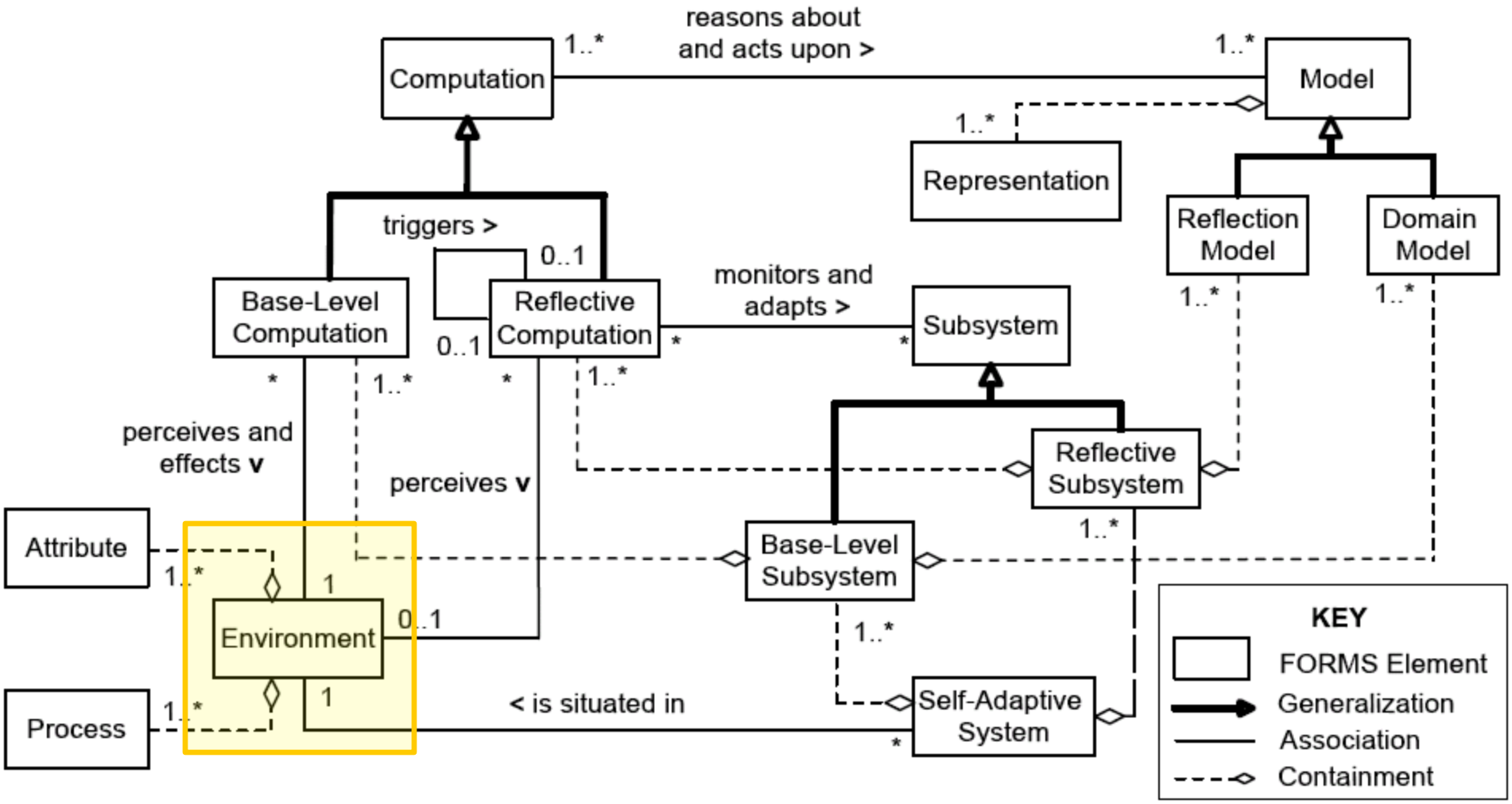
FORMS Reflection Perspective

Self-adaptive system



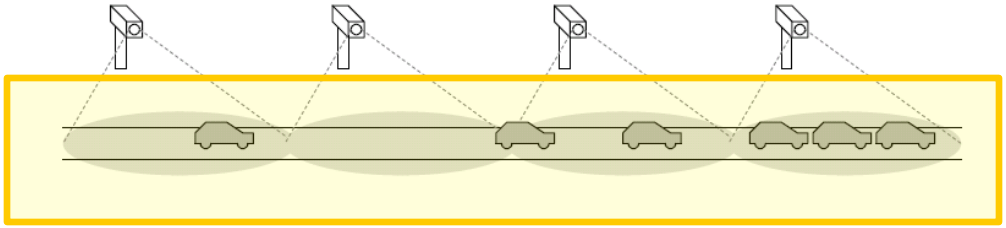
FORMS Reflection Perspective

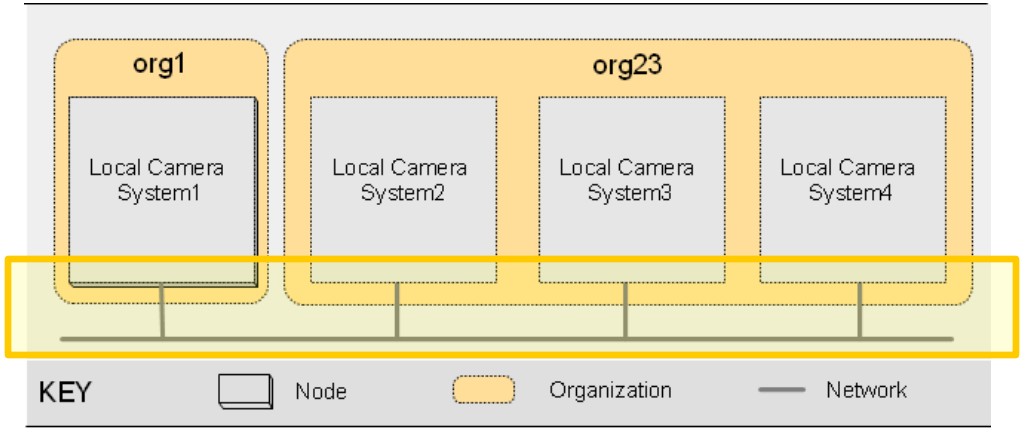
Environment

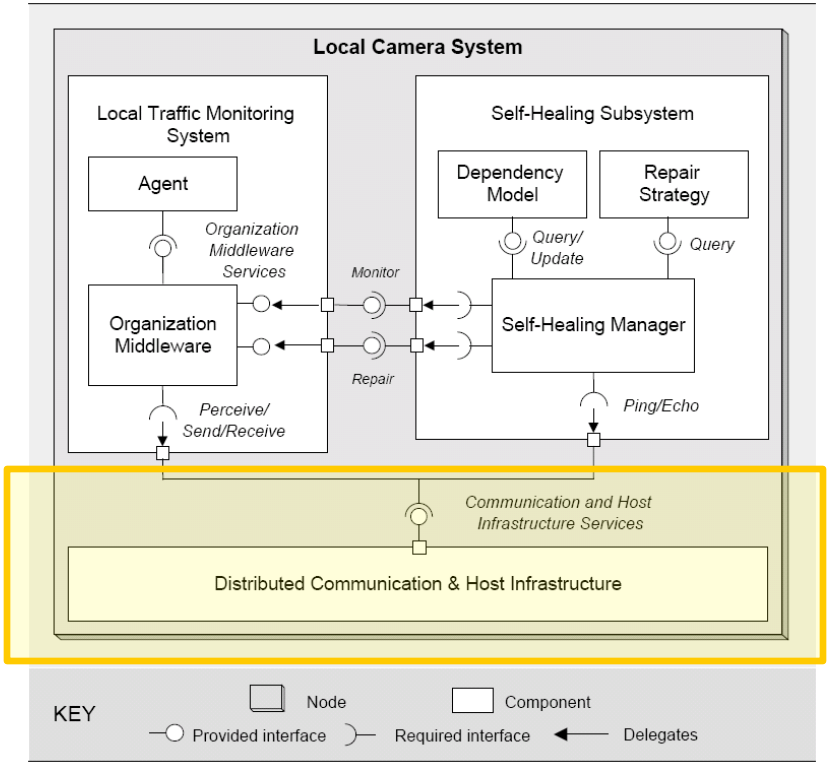


FORMS Reflection Perspective

Environment

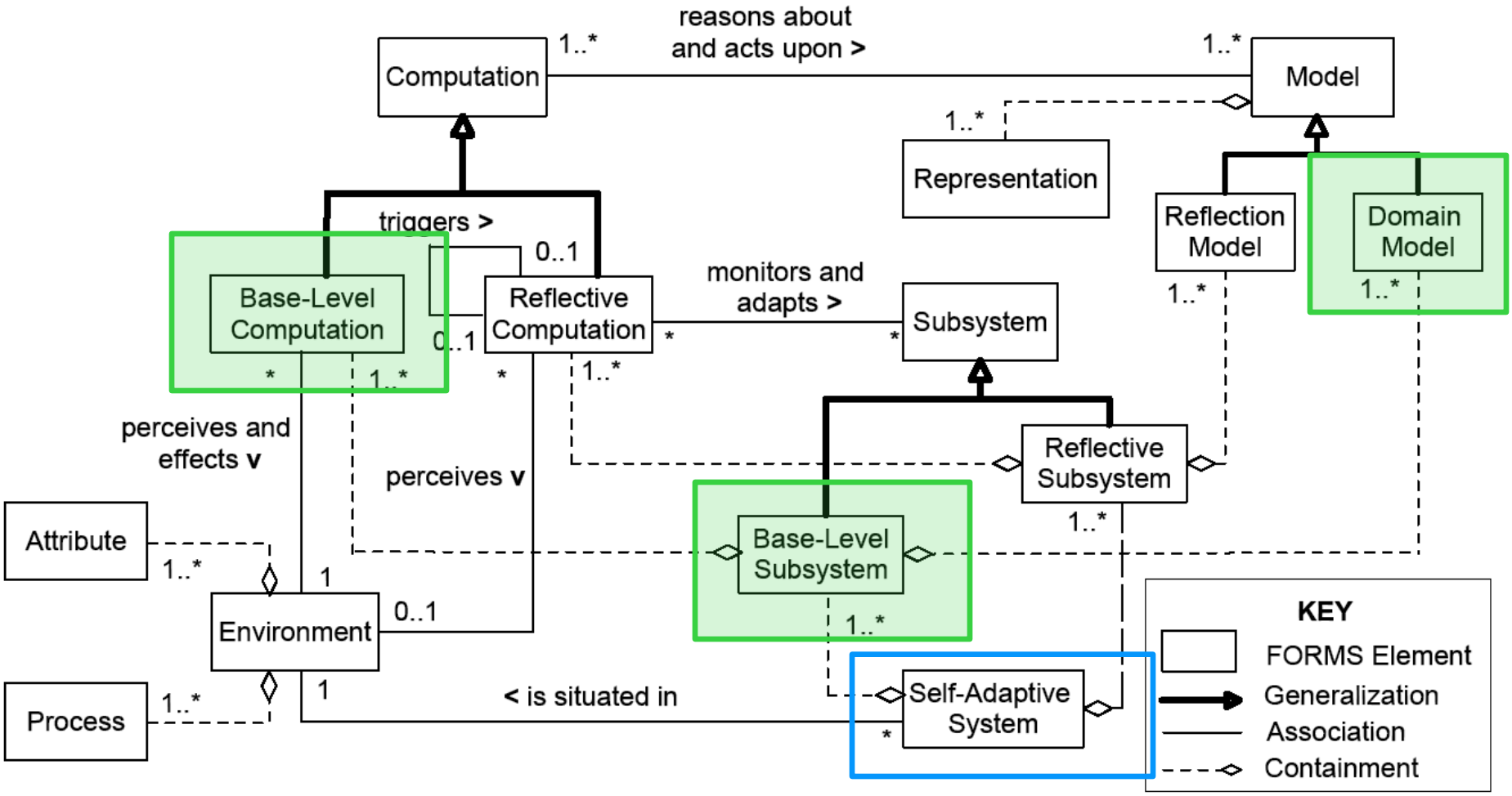






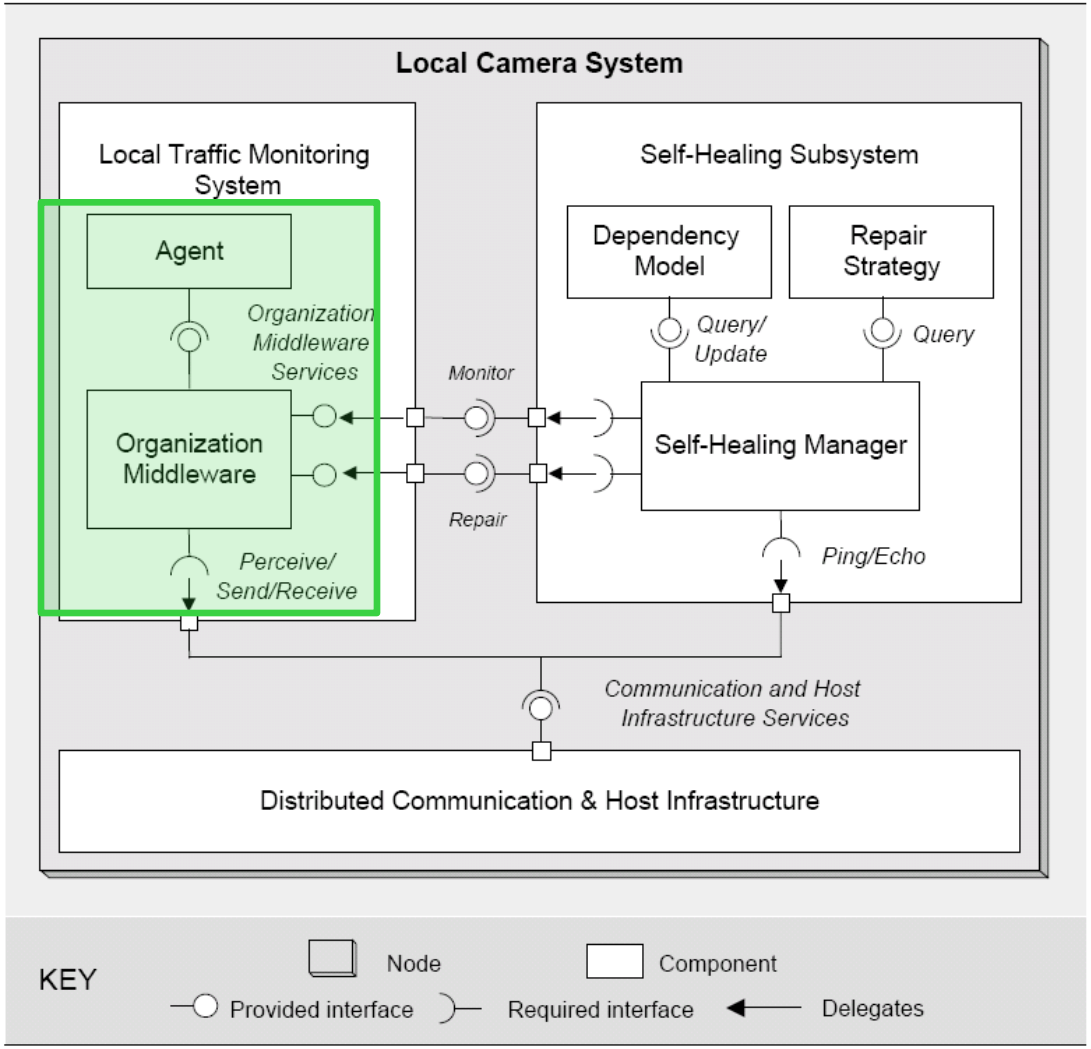
FORMS Reflection Perspective

Base-Level Subsystem



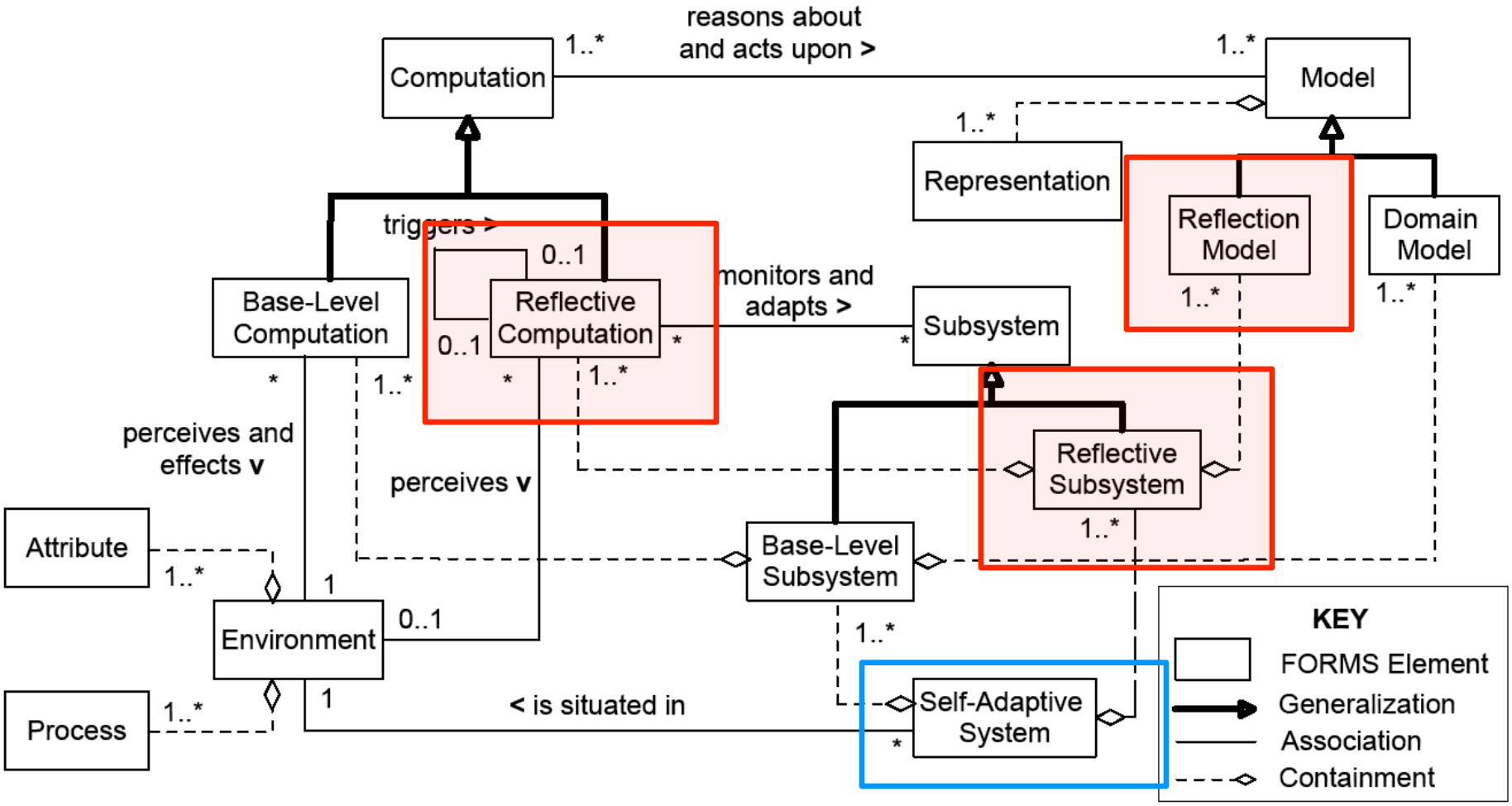
FORMS Reflection Perspective

Base-Level Subsystem



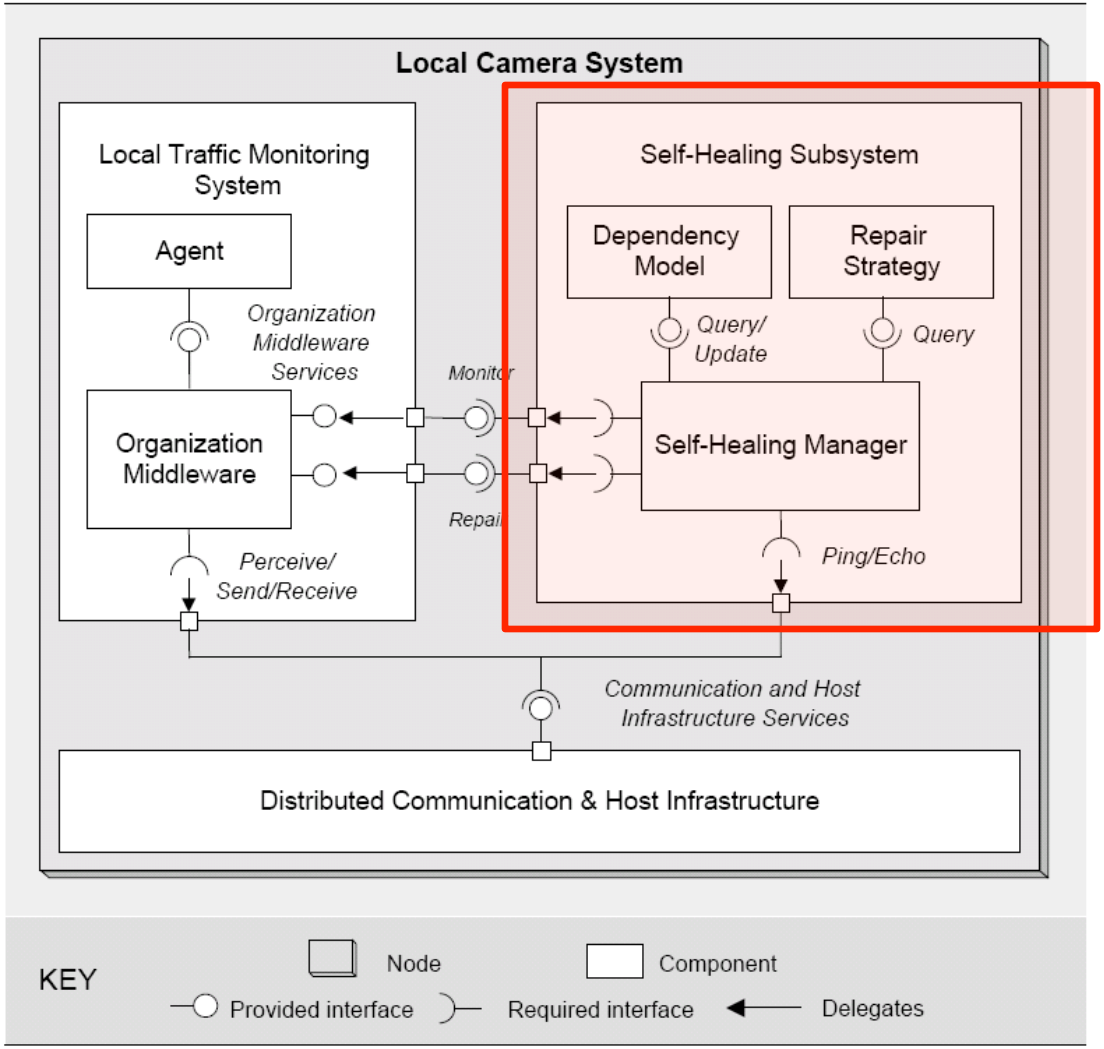
FORMS Reflection Perspective

Reflective Subsystem

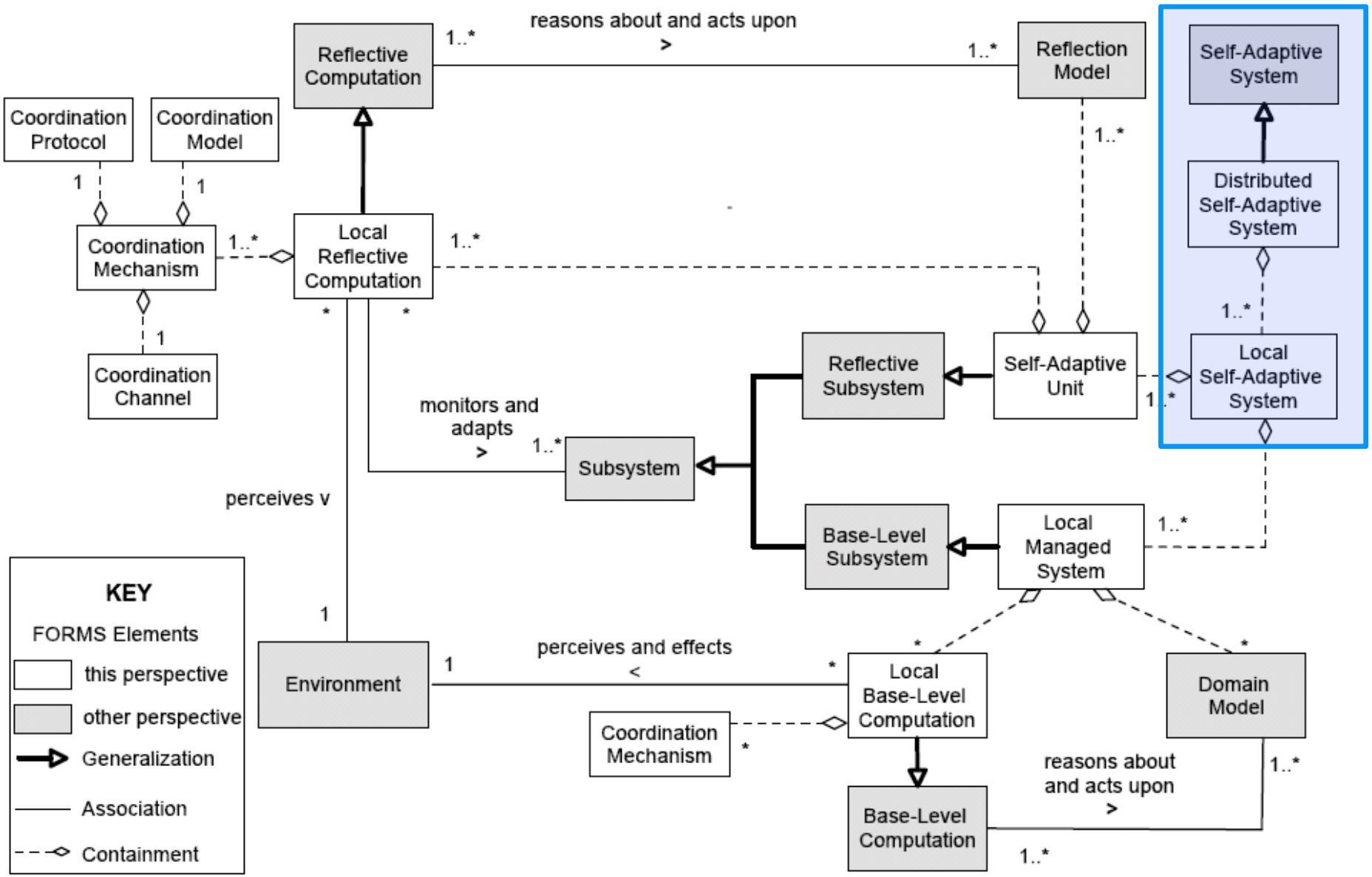


FORMS Reflection Perspective

Reflective Subsystem

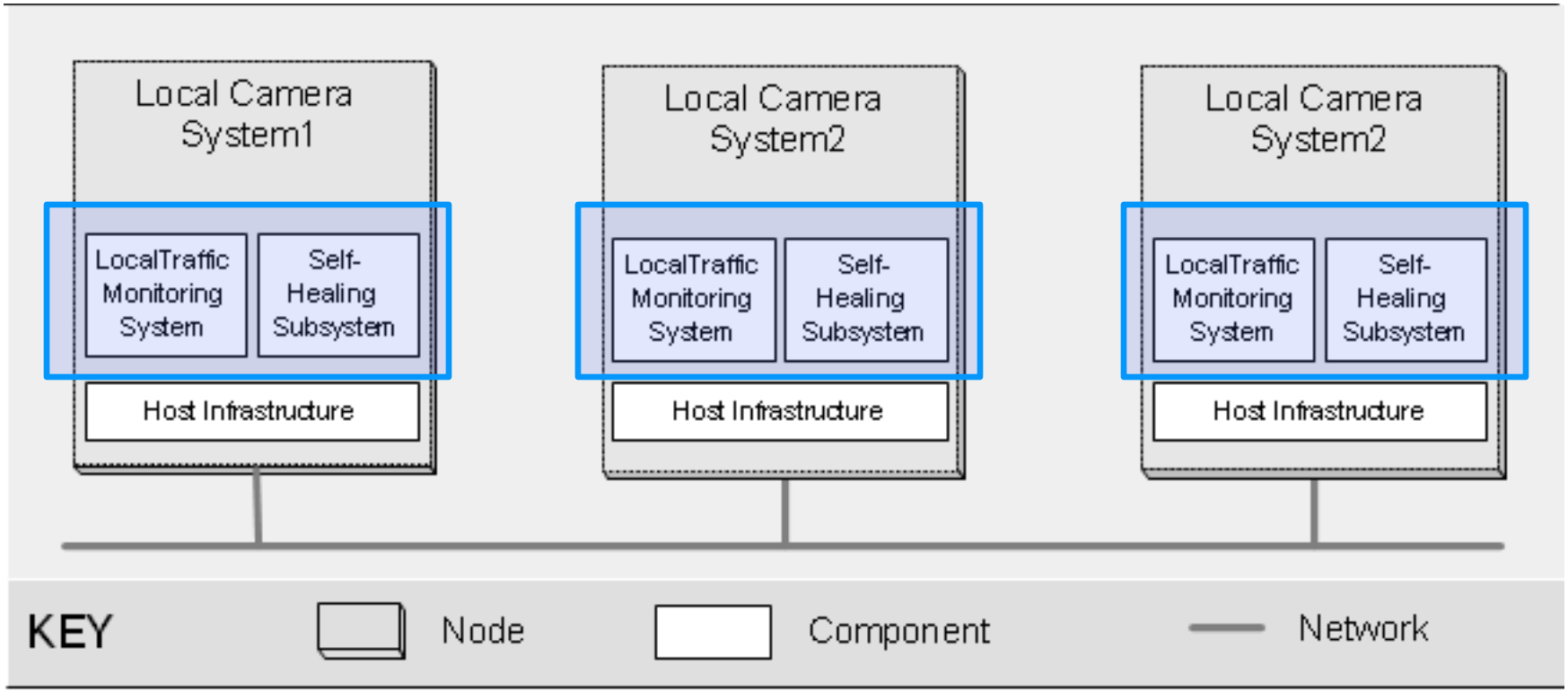


FORMS Distributed Coordination Perspective



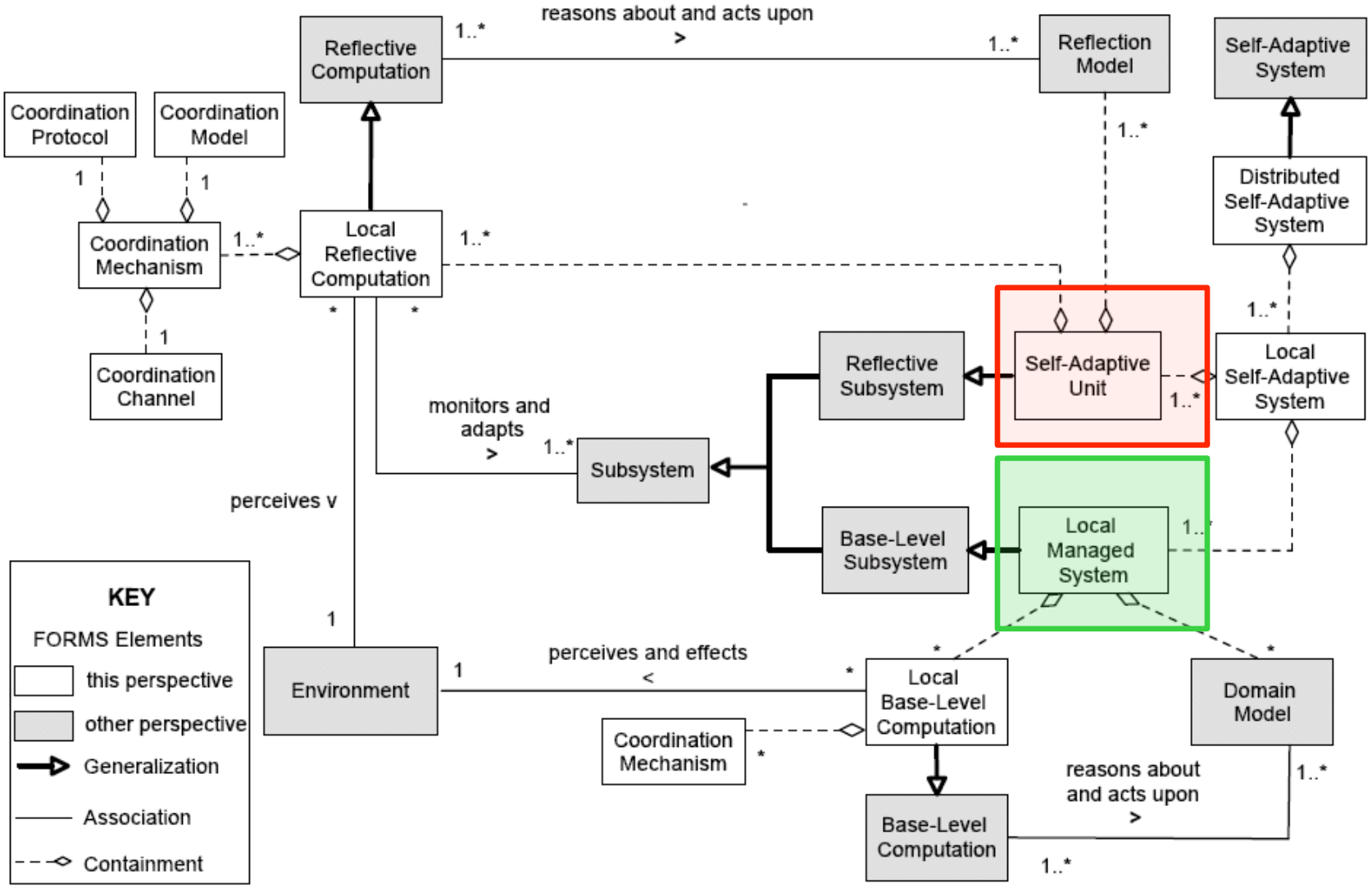
FORMS Distributed Coordination Perspective

Local Self-Adaptive System



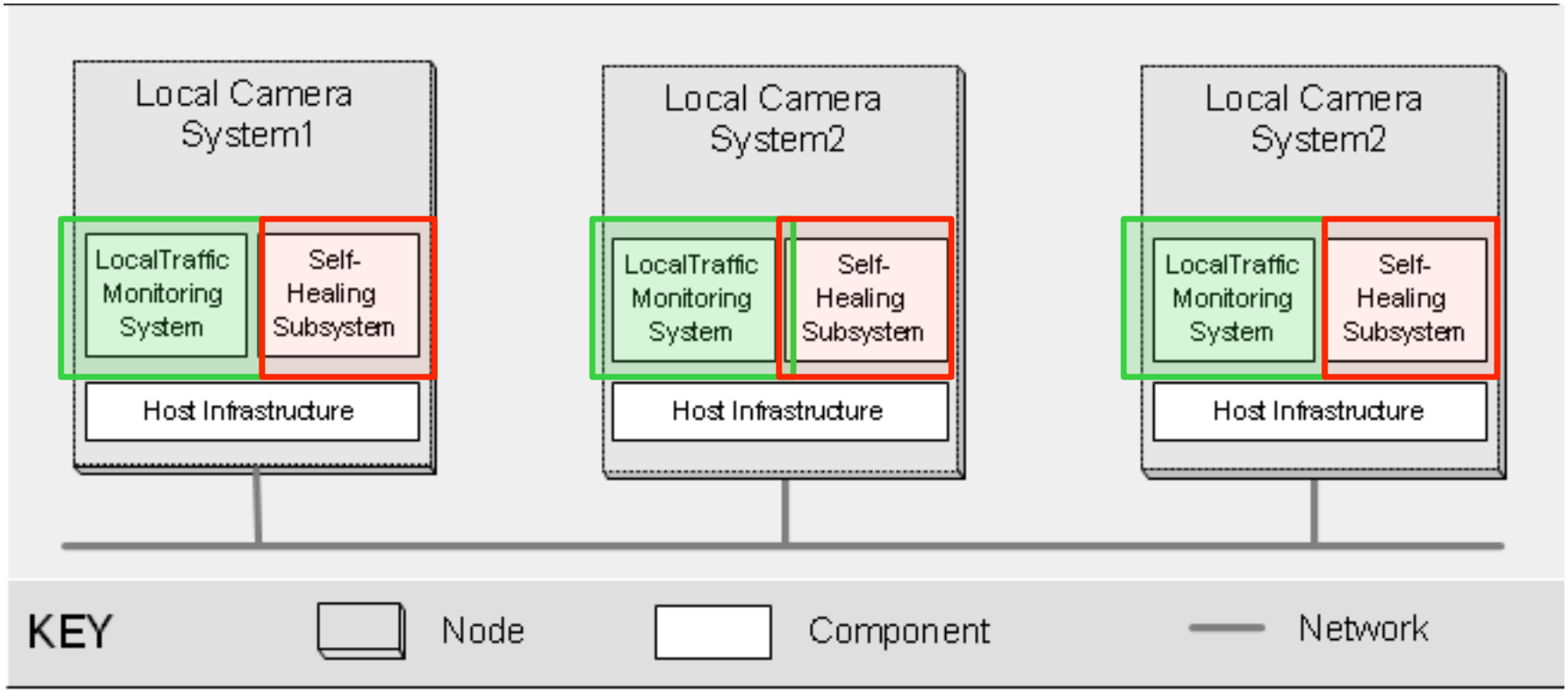
FORMS Distributed Coordination Perspective

Local Managed System – Self-Adaptive Unit



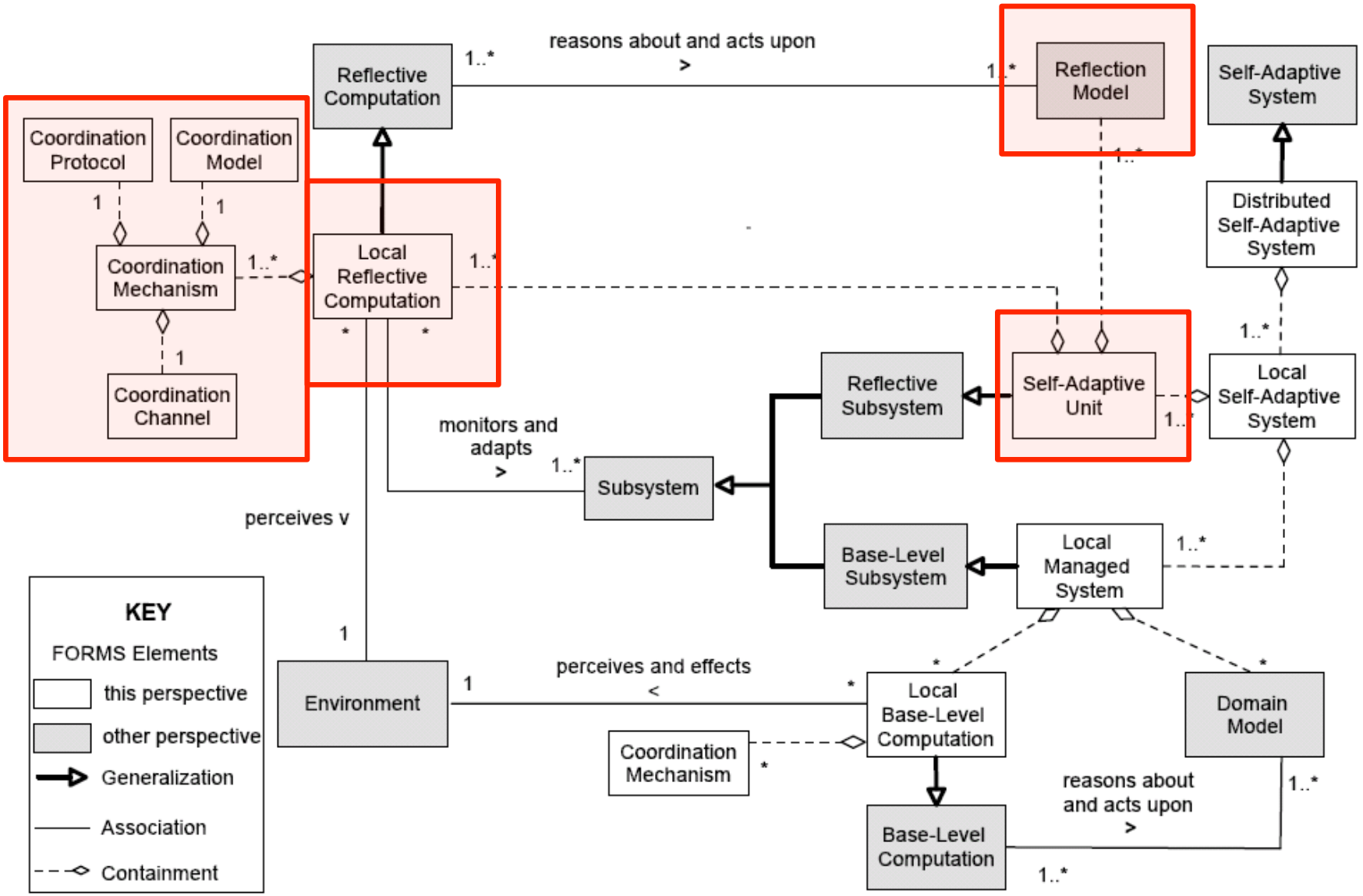
FORMS Distributed Coordination Perspective

Local Managed System – Self-Adaptive Unit



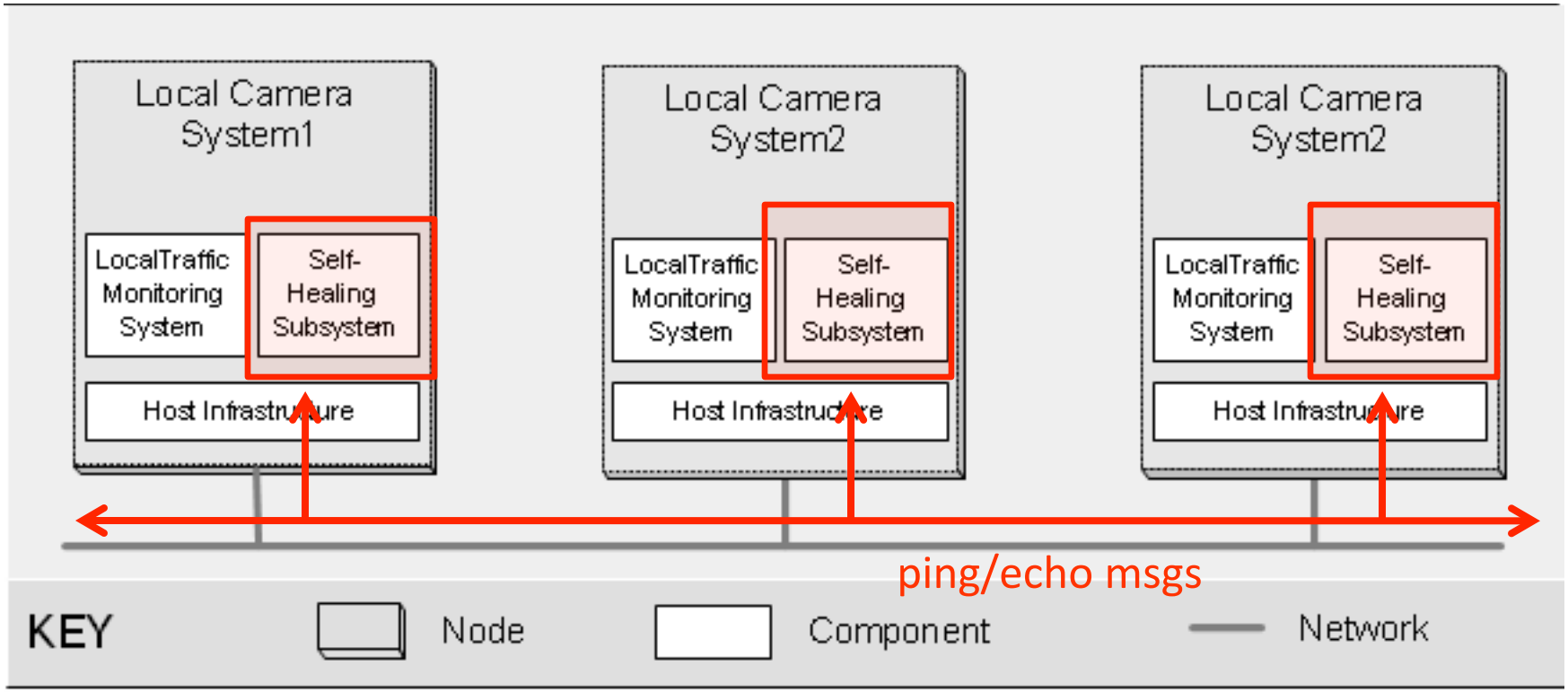
FORMS Distributed Coordination Perspective

Coordination Mechanism

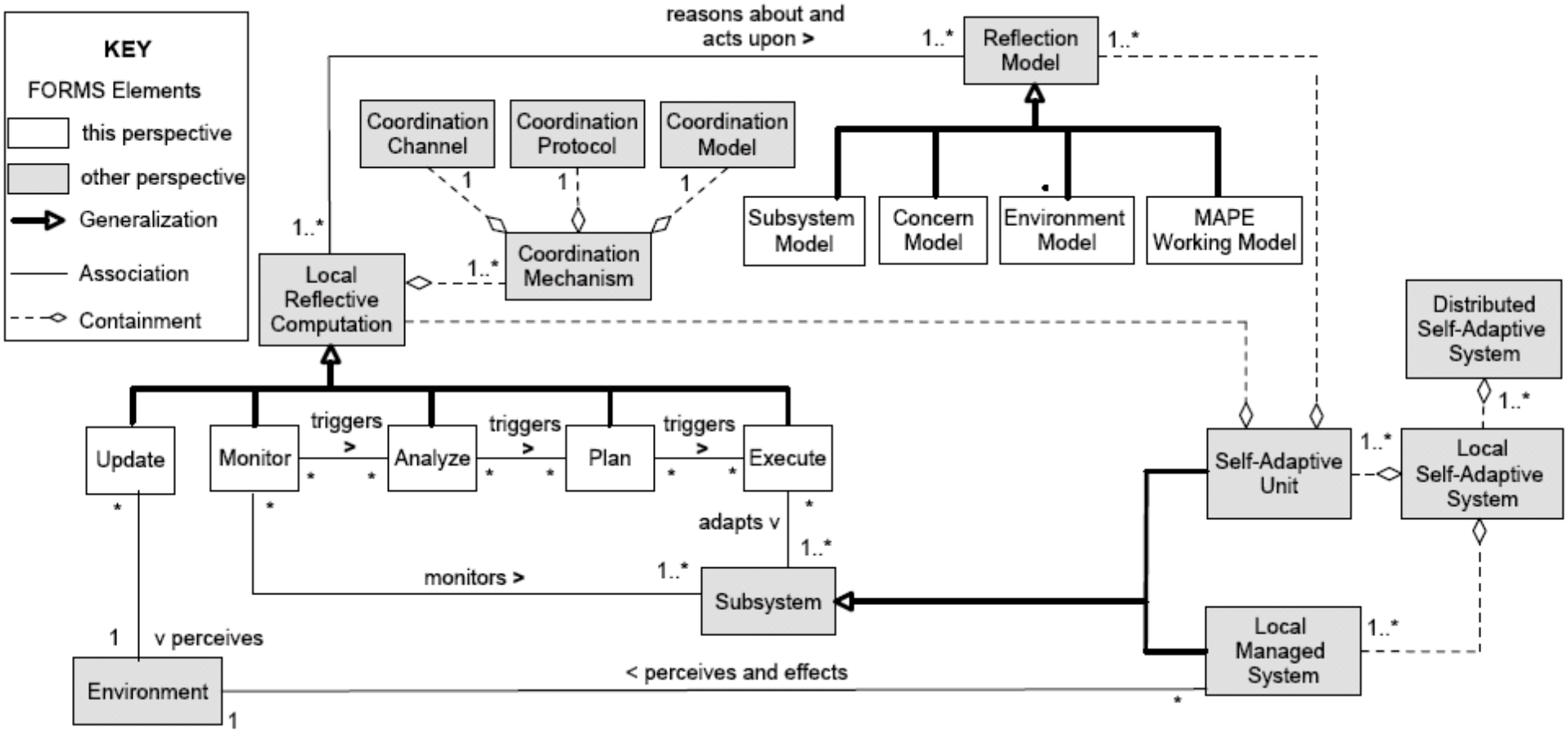


FORMS Distributed Coordination Perspective

Coordination Mechanism

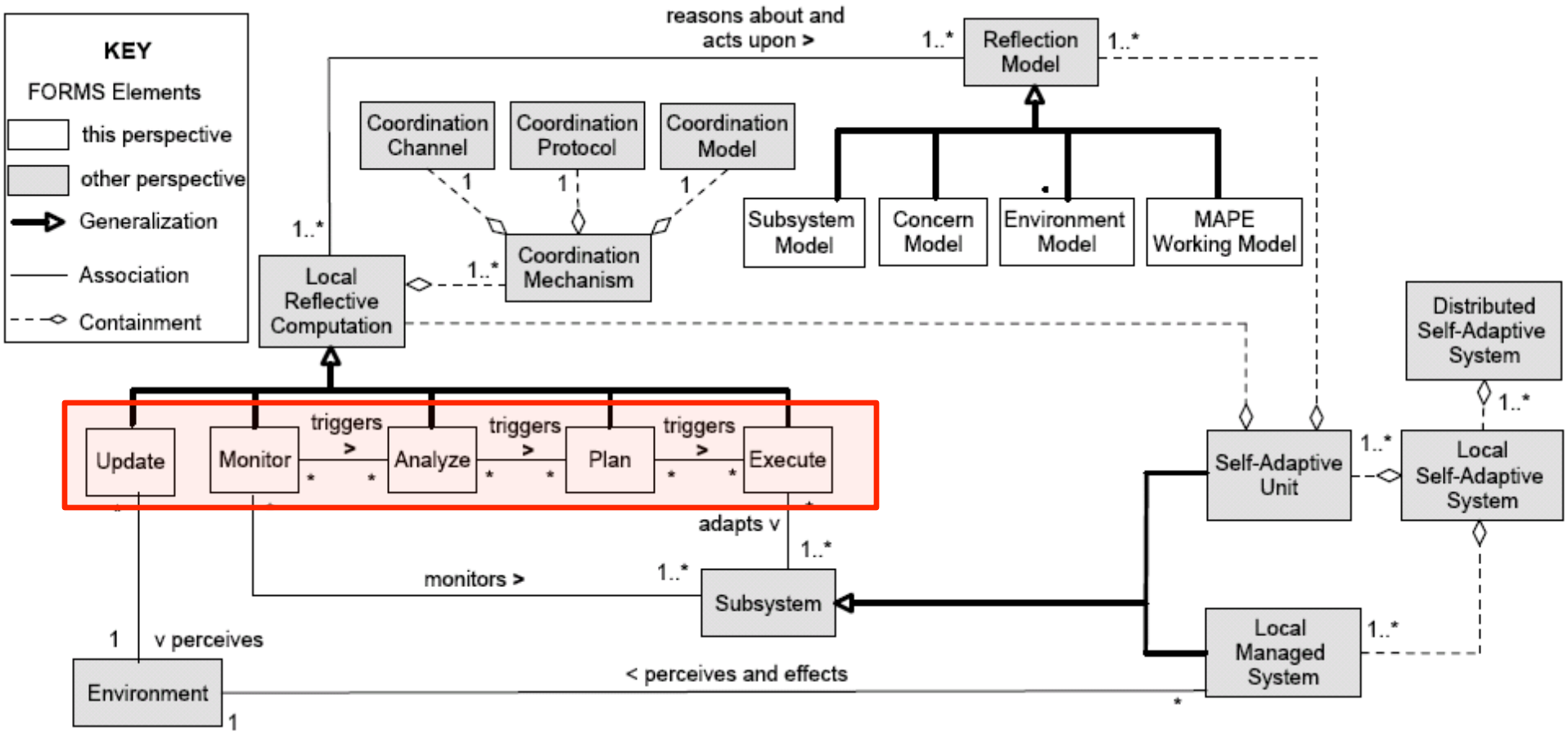


FORMS MAPE Perspective



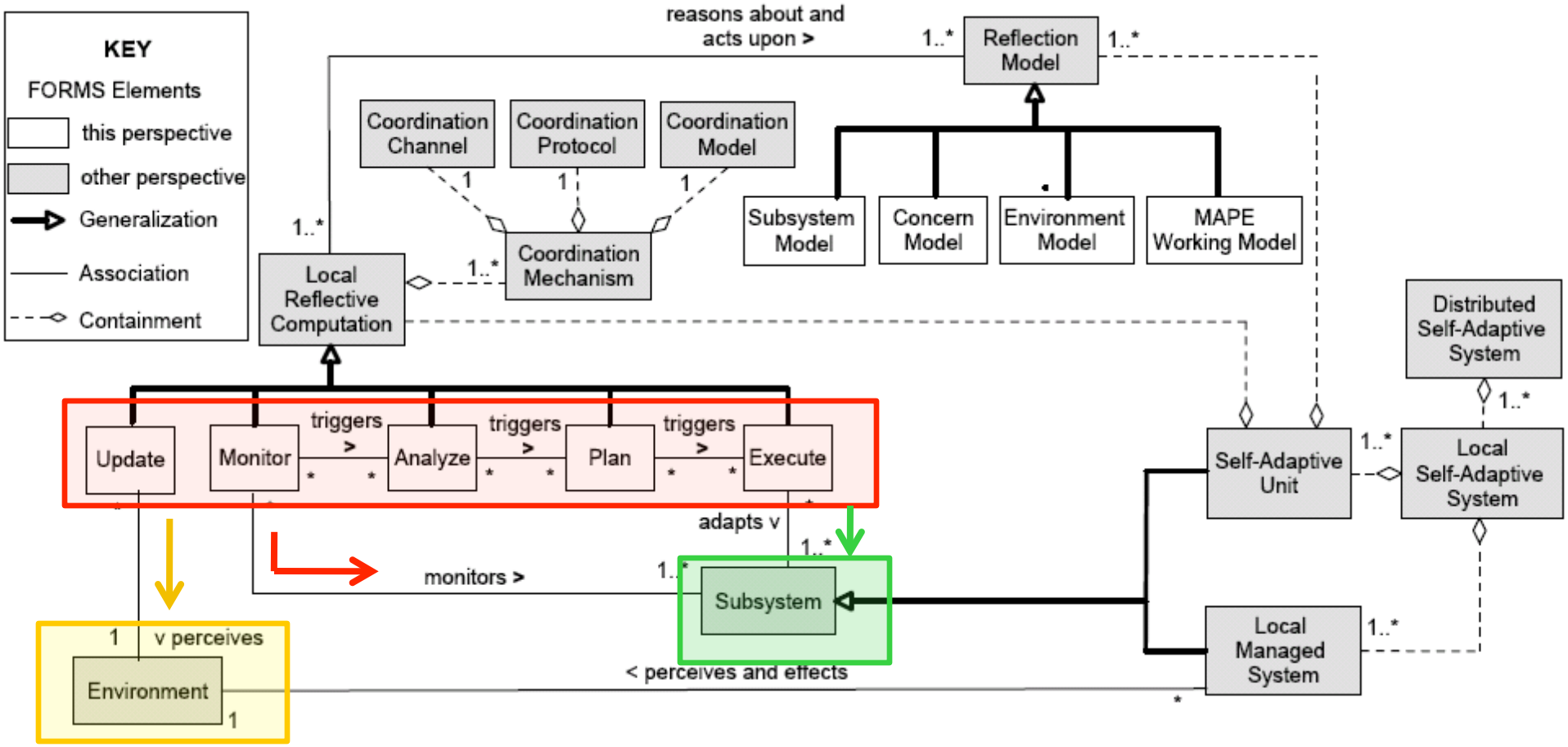
FORMS MAPE Perspective

Local Reflective Computations



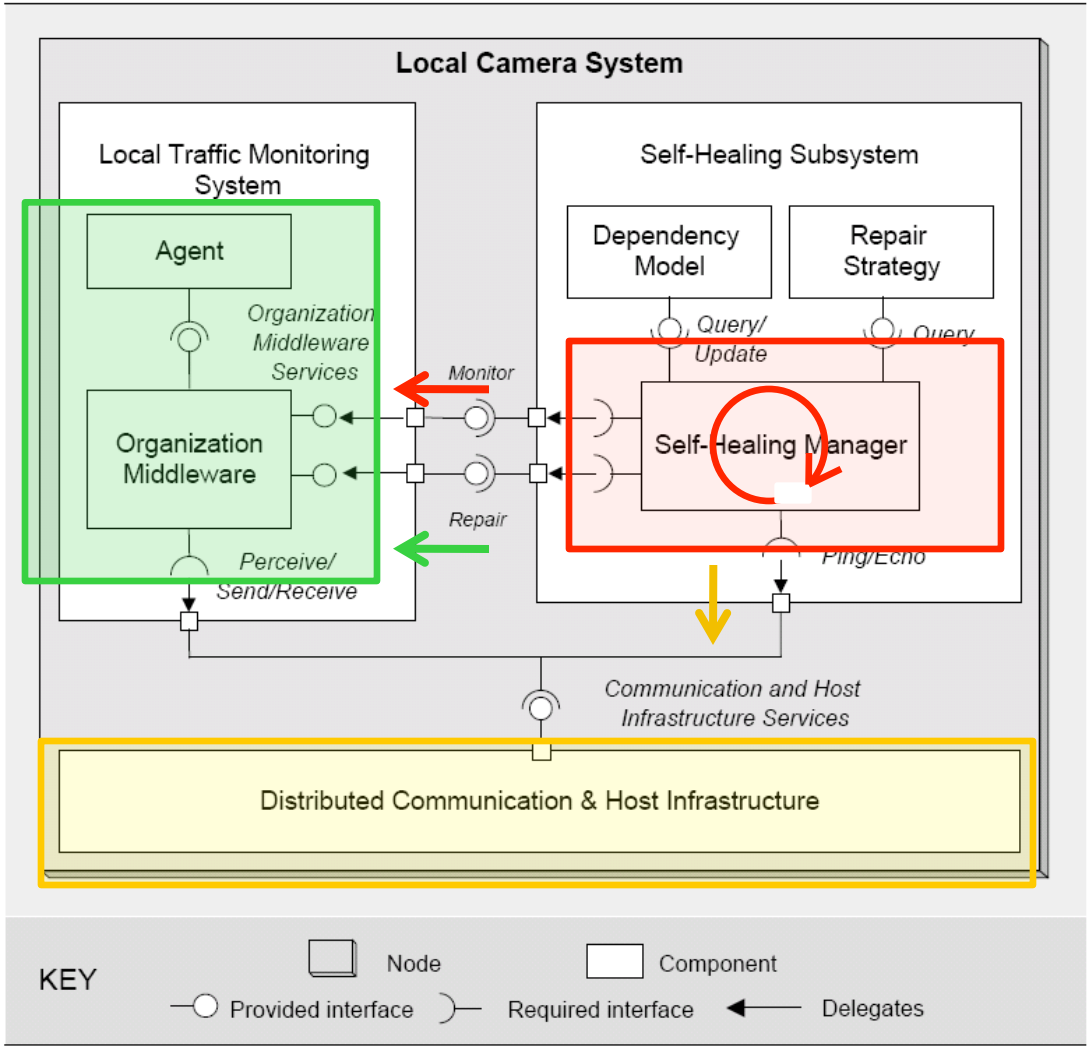
FORMS MAPE Perspective

Local Reflective Computations



FORMS MAPE Perspective

Local Reflective Computations



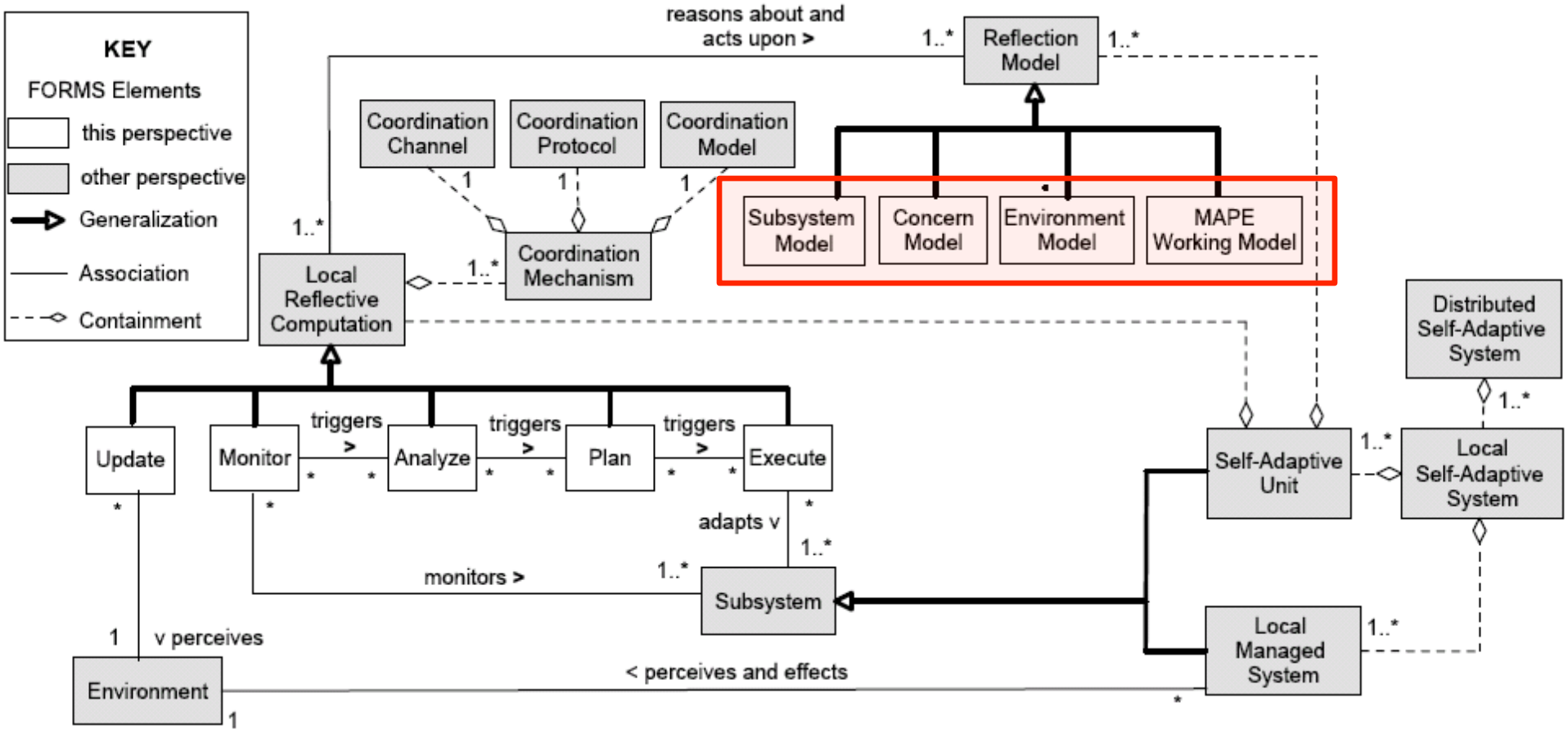
perceive

sense

adapt

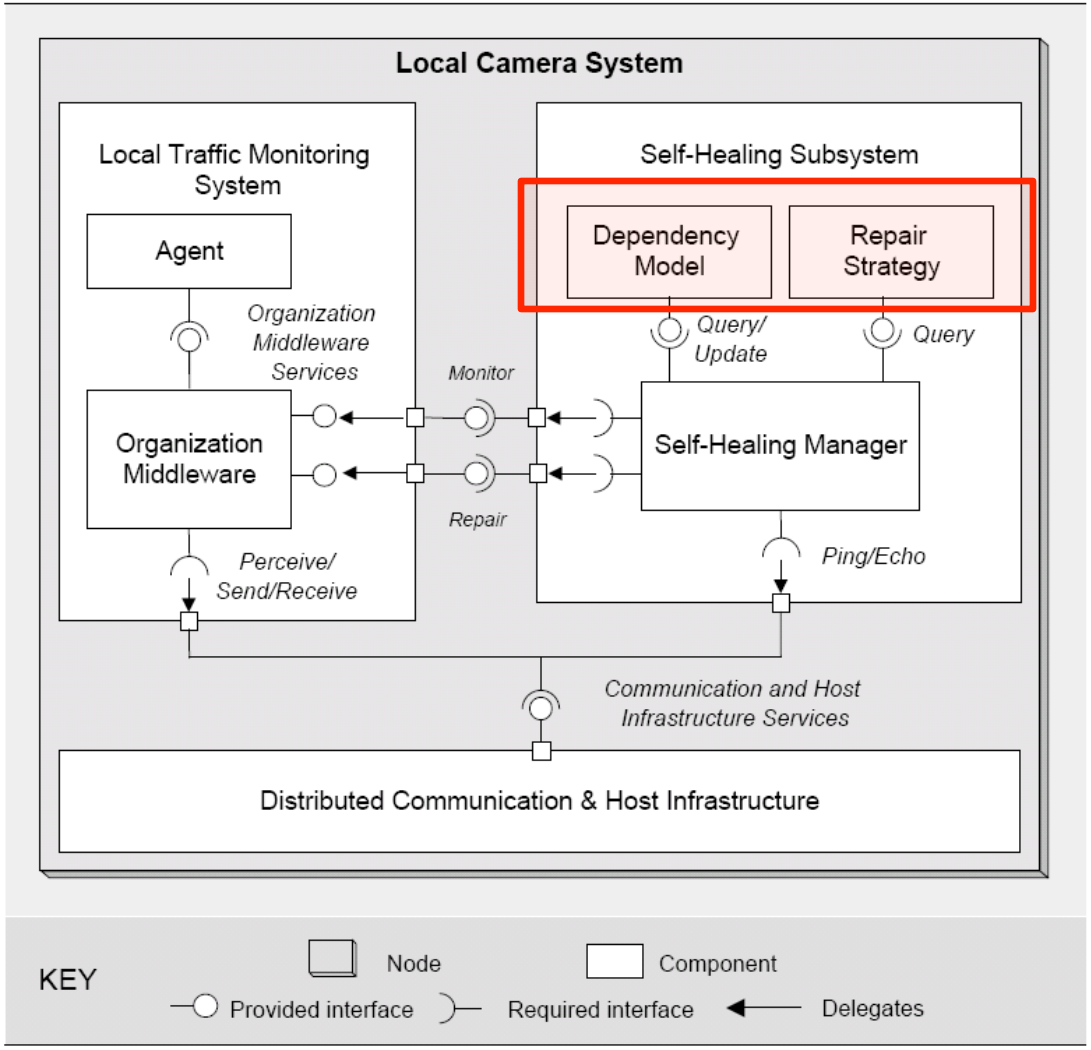
FORMS MAPE Perspective

Reflection Models

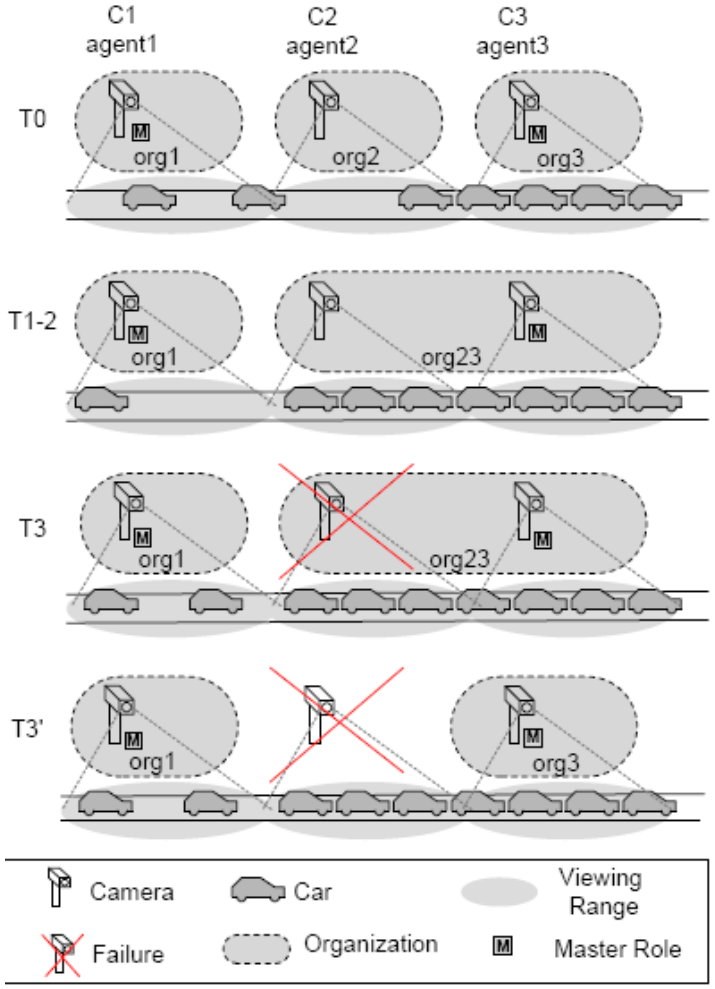


FORMS MAPE Perspective

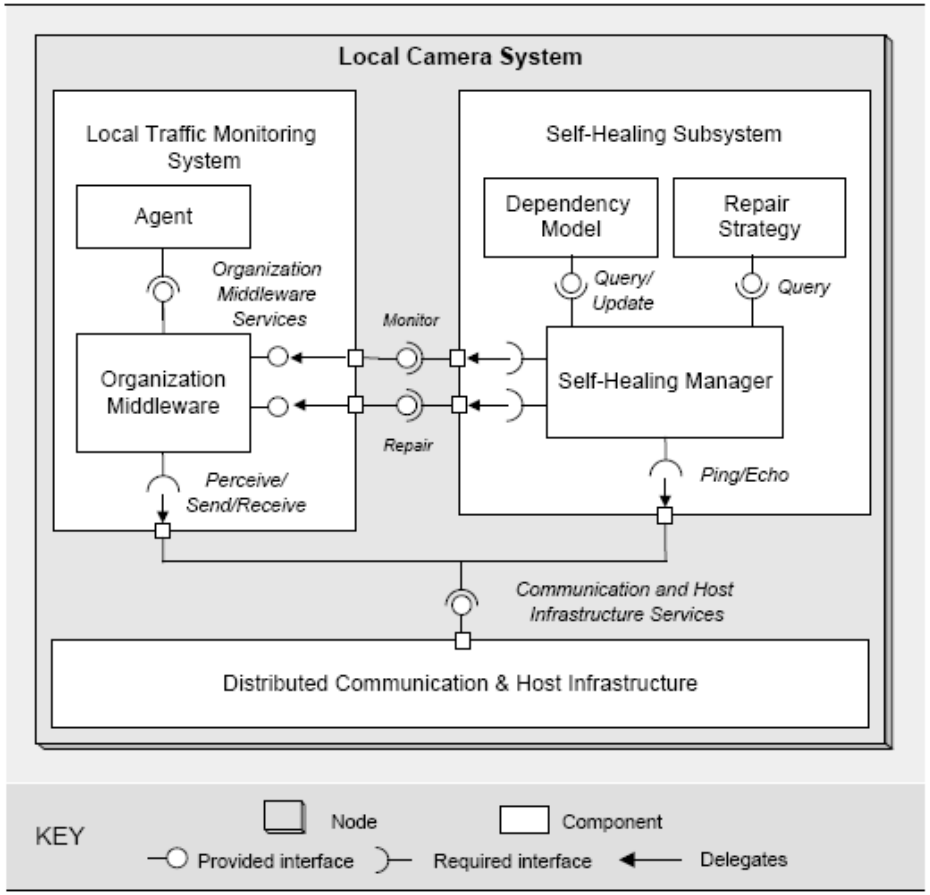
Reflection Models



(A glimpse of) FORMS in action

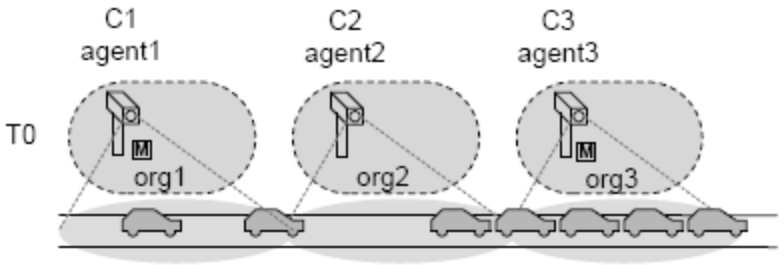


(a) Scenario with a failing camera



(b) Deployment view of one camera

FORMS in action



Environment

$attributes : \mathbb{P}\ Attribute$
$processes : \mathbb{P}\ Process$

$attributes \neq \varnothing$

TrafficEnvironment$_{T0}$

TrafficEnvironment

$attributes = \{camera_1\ , camera_2\ , camera_3\ , freeflow\_zone_1\ ,$
$\quad freeflow\_zone_2\ , congested\_zone_3\}$
$processes = traffic\_domain\_processes \cup$
$\quad communication\_infrastructure\_processes$

$traffic\_domain\_processes ==$
$\quad \{monitor\_camera_1\ , monitor\_camera_2\ , monitor\_camera_3\}$

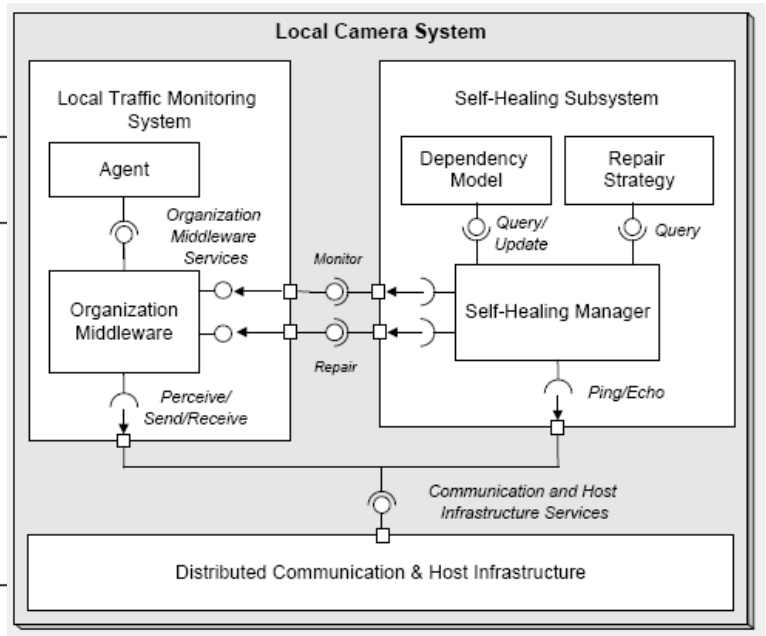
$communication\_infrastructure\_processes == \{transmit\}$

FORMS in action

$$
\begin{array}{|l}
\hline
\textit{LocalCameraSystem} \\
\textit{localTrafficMonitoringSystem} : \textit{LocalTrafficMonitoringSystem} \\
\textit{selfHealingSubsystem} : \textit{SelfHealingSubsystem} \\
\textit{myName} : \textit{Name} \\
\hline
\ldots \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{SituatedLocalCameraSystem} \\
\textit{TrafficEnvironment} \\
\textit{LocalCameraSystem} \\
\textit{context} : \textit{Context} \\
\hline
\textit{context} \subseteq \textit{attributes} \land \\
\hline
\ldots \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{TrafficJamMonitoringSystem} \\
\textit{localCamaraSystems} : \mathbb{P}\ \textit{SituatedLocalCameraSystem} \\
\hline
\ldots \\
\hline
\end{array}
$$



FORMS in action

$TrafficEnvironment_{T3}$

$\Delta TrafficEnvironment_{T2}$
$e? : events$
$s? : shutdowns$

$e? = \{camera_2\} \mapsto \{\}$
$s? = monitor\_camera_2$
$attributes' = attributes \setminus first(e?) \cup second(e?)$
$processes' = processes \setminus \{s?\}$

$Clock$

$time : Time$

$Tick$

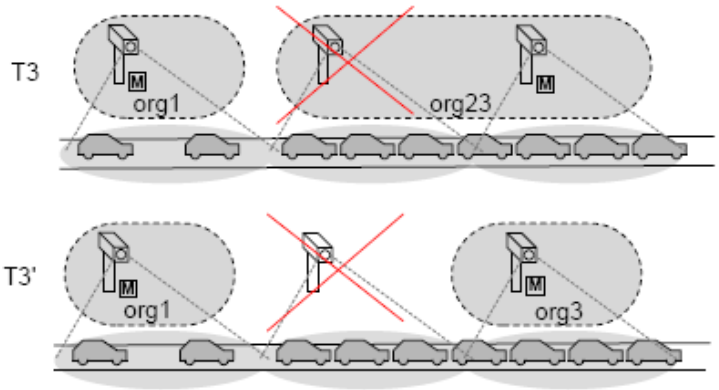
$\Delta Clock$

$time' = time + 1$

$Timeout$

$\Xi SelfHealingManager$
$Tick$
$n! : Name$

$\exists\, n! : Name;\ t : Time \bullet$
$(n!, t) \in coordinationMechanism.ping\_time \land$
$t + coordinationMechanism.wait\_time > time'$



FORMS in action

$TrafficEnvironment_{T3}$

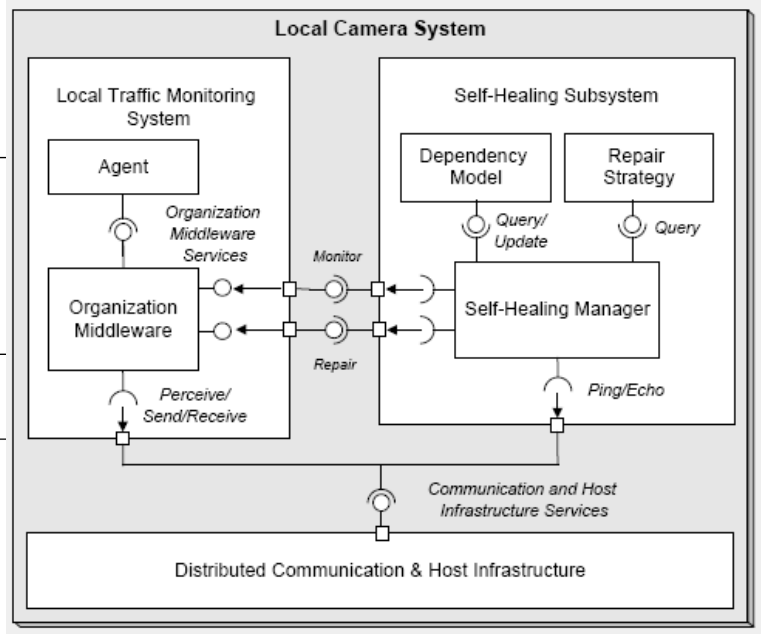
$\Delta TrafficEnvironment_{T2}$
$e? : events$
$s? : shutdowns$

$e? = \{camera_2\} \mapsto \{\}$
$s? = monitor\_camera_2$
$attributes' = attributes \setminus first(e?) \cup second(e?)$
$processes' = processes \setminus \{s?\}$

$Timeout_1$

$Timeout$
$\Xi SelfHealingManagerOne_{T2}$

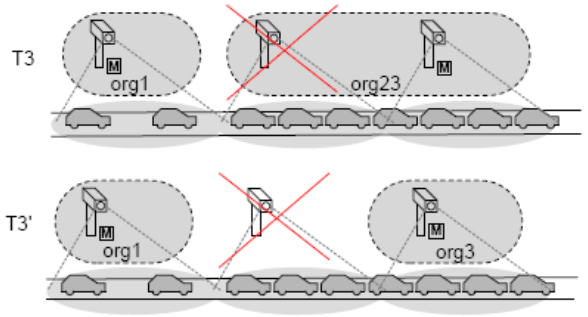
$time = 4470$
$n! = 2$



$CameraOneRecoversFromFailureCameraTwo$

$\Delta TrafficJamMonitoringSystem_{T3}$
$TrafficEnvironment_{T3}$
$Timeout_1$
$lcs1?, lcs1! : SituatedLocalCameraSystem$
$camera : Attribute$
$cam : EnvironmentRepresentation$
$n : Name$

$\{camera\} = first(e?) \land$
$traffic\_communication\_channel = traffic\_communication\_channel \setminus \{n \mapsto cam\} \land$
$\ldots$
$lcs1?.myName = 1 \land$
$lcs1!.context = lcs1?.context \setminus \{camera\} \land$
$lcs1!.selfHealingSubsystem = updateSelfHealingSubsystem(lcs1?, camera, cam, n) \land$
$lcs1!.localTrafficMonitoringSystem =$
$\quad adaptLocalTrafficMonitoringSystem(lcs1?, camera, cam, n) \land$
$localCamaraSystems' = localCamaraSystems \setminus \{lcs1?\} \cup \{lcs1!\}$



Reference approaches for self-adaptation
FORMS 2012

- Integrated, extensible model
- Formal underpinning
- Focus on modeling and reasoning about structural aspects of self-adaptive systems
- Reference model can be mapped to different architectures
- Vocabulary for domain of self-adaptive systems

Overview

- Architecture-based self-adaptation vs. control-based self-adaptation
- Reference approaches for architecture-based self-adaptation
- Formal methods for self-adaptive systems
- Active formal methods for self-adaptation
- Wrap up

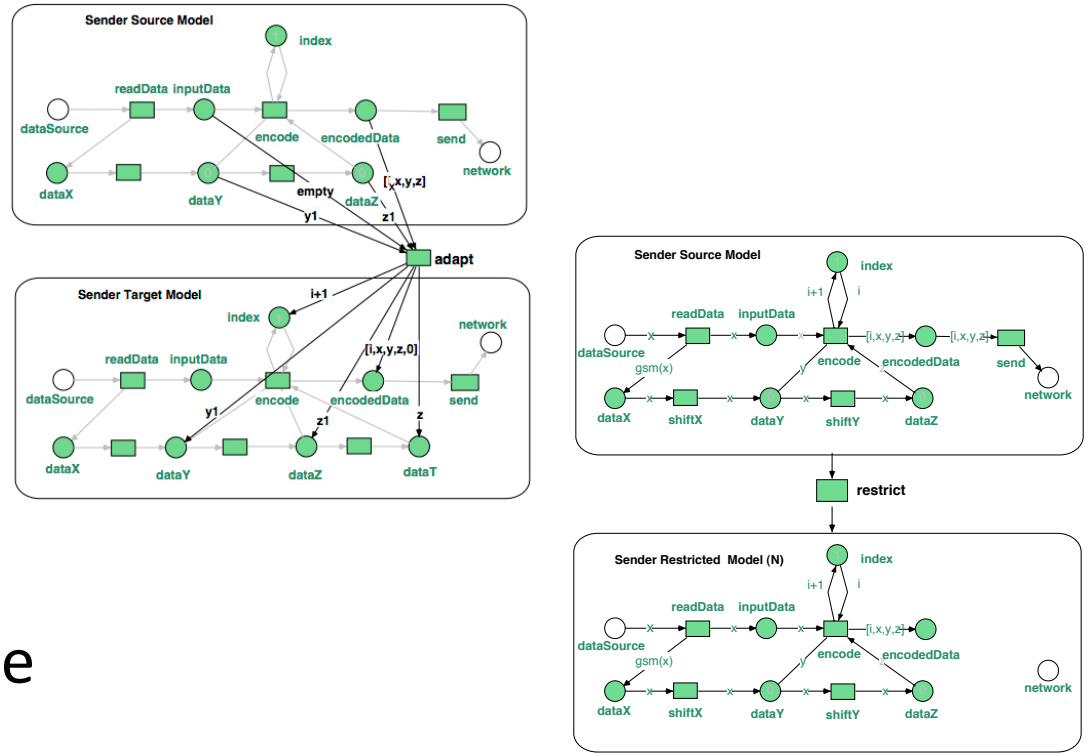
Formal methods for self-adaptation
A selection

- 2006: Zhang & Cheng (design time verification and model transformation)
- 2009: Epifani et al. (K models at runtime)
- 2011: Calinescu et al. (MAPE functions at runtime)
- 2013: Ghezzy et al. (model interpretation)

D. Weyns, U. Iftikhar, D. Gil de la Iglesia, and T. Ahmad, A Survey on Formal Methods in Self-Adaptive Systems, Fifth International C* Conference on Computer Science and Software Engineering 2012

Formal methods for self-adaptive systems
Zhang and Cheng 2006

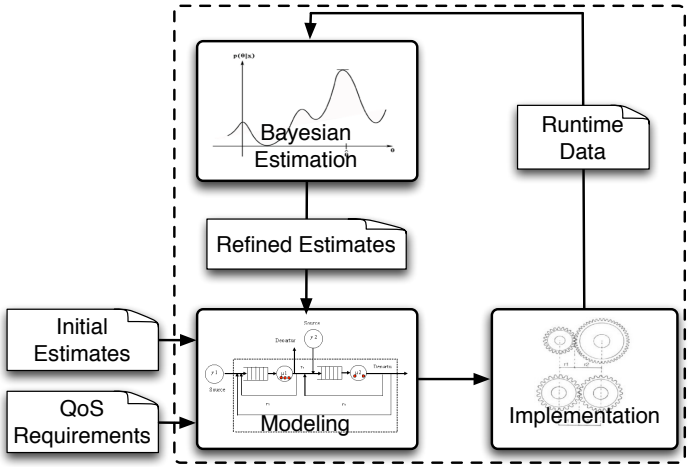
- Different classes of adaptations
  - one-point, guided adaptation, overlap adaptation
- Process to create and verify formal models (Petri nets and LTL)
- Automatically generate programs from them

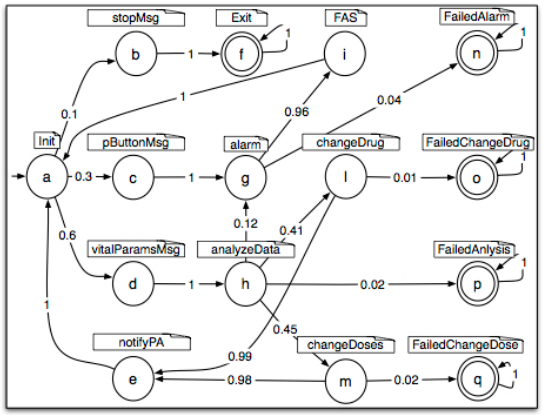


J. Zhang and B. Cheng, Model-based development of dynamically adaptive software, International Conference on Software Engineering, ICSE 2006

Formal methods for self-adaptive systems
Epifani et al. 2006

- Probabilistic model represents reliability of execution flows of system
- Probabilities are dynamically updated based on observations
- Formal model of system behavior at runtime: focus on K of MAPE-K

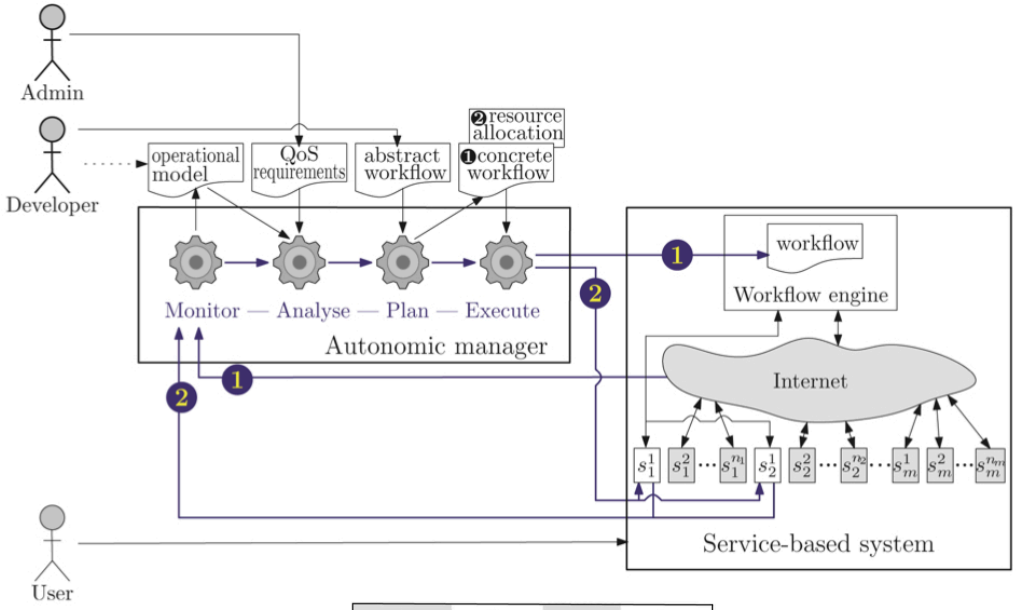


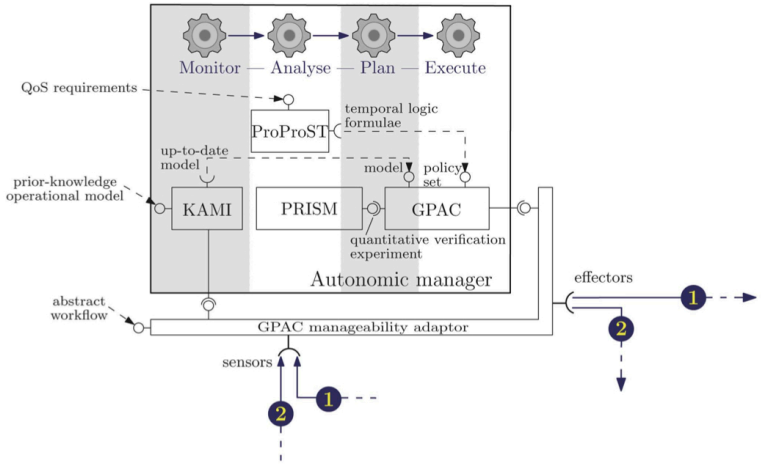


I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. 2009. Model evolution by run-time parameter adaptation, International Conference on Software Engineering, ICSE 2009

Formal methods for self-adaptive systems
Calinescu et al. 2011

- Probabilistic model of reliability and performance properties of service-based system
- Requirements specified in probabilistic computation tree logic
- Online verification of properties using Prism
- Adaptation of workflow engine (service selection + resources)
- Adaptation logic consists of set of tools that are glued together

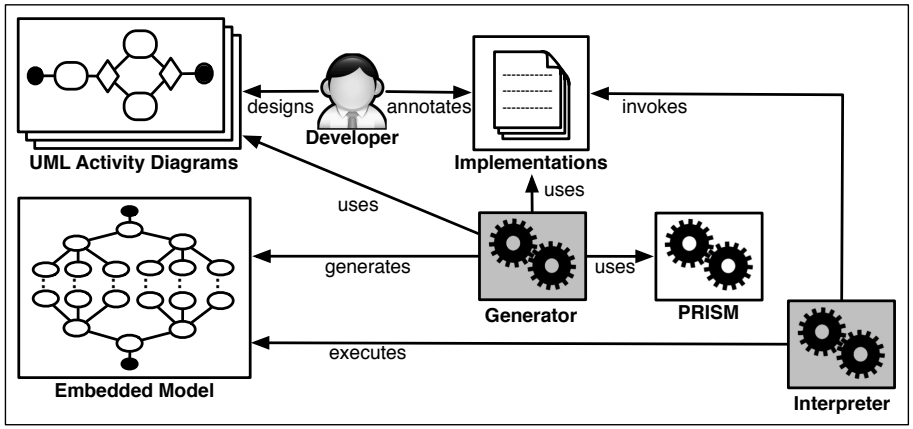


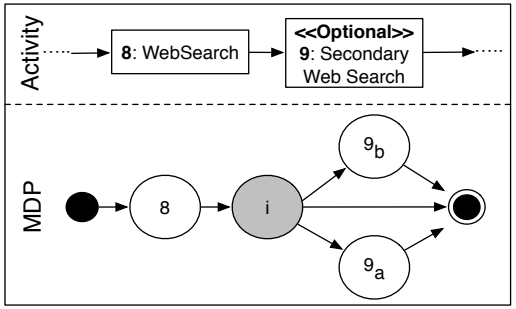


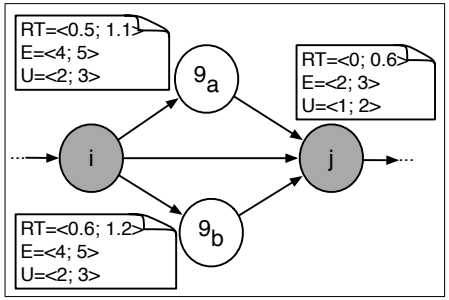
R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS Management and Optimization in Service-Based Systems, IEEE Transactions on Software Engineering, TSE 2011

Formal methods for self-adaptive systems
Ghezzy et al. 2013

- Annotated UML diagram models response time, energy consumption and usability of different execution paths of the system
- Diagram is automatically translated to Markov decision process using Prism
- Interpreter guides the execution of the system using the model
- Cumulative reward is used to select path with highest utility
- Adaptation logic is encoded in ad-hoc interpreter







C. Ghezzi, L.S. Pinto, P. Spoletini, G. Tamburrelli: Managing non-functional uncertainty via model-driven adaptivity, International Conference on Software Engineering, ICSE 2013

Summary SOTA

- Increasing attention for formal models at runtime to provide guarantees of adaptation
- Probabilistic approaches dominate
- Focus on formal models of system, environment and goals (K of MAPE-K)
- No systematic formalization and verification of of adaptation functions (MAPE of MAPE-K)
- Limited support for unpredicted changes

Overview

- Architecture-based self-adaptation vs. control-based self-adaptation
- Reference approaches for architecture-based self-adaptation
- Formal methods for self-adaptive systems
- Active formal methods for self-adaptation
- Wrap up

Starting points

- Formalize adaptation functions to provide guarantees about adaptation capabilities
  - E.g., does analysis detect errors correctly?
  - Are adaptations performed in order of selected plan?
- Support unanticipated changes
  - Requires support for adaptations of adaptation functions

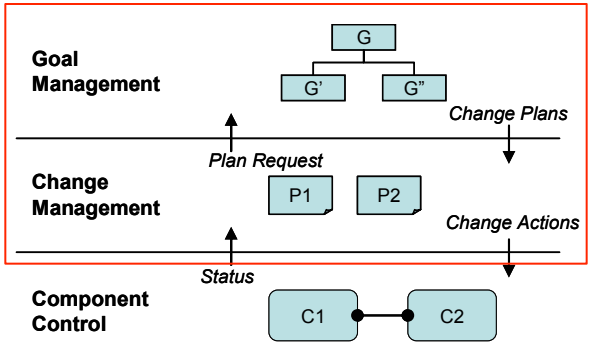
ActivFORMS

Active formal models for self-adaptation

- Formal model of complete MAPE-K loop
- Model is directly executed to adapt the managed system
- Model directly supports online verification of goal satisfaction/violation
- Model can be adapted at runtime to support unanticipated changes

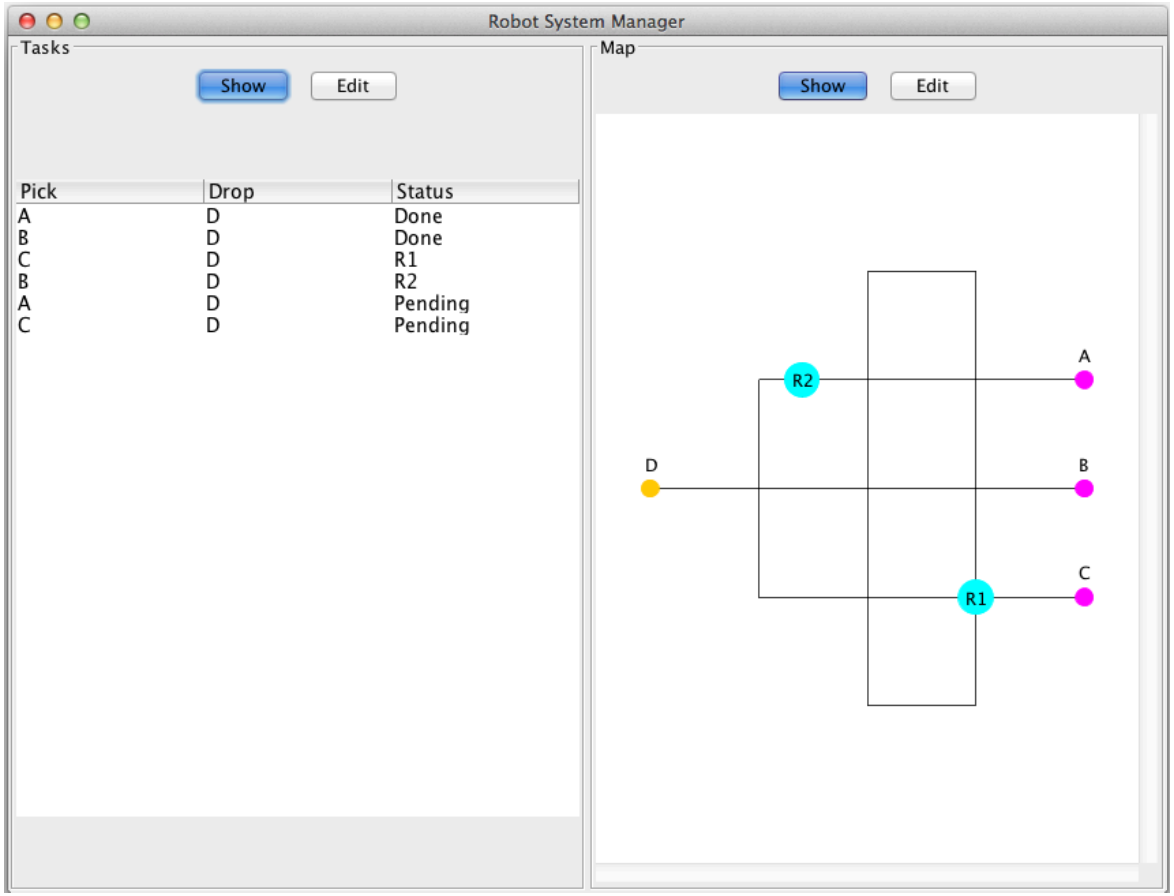
http://homepage.lnu.se/staff/daweaa/ActivFORMS.htm (from October 15, 2013)

Focus

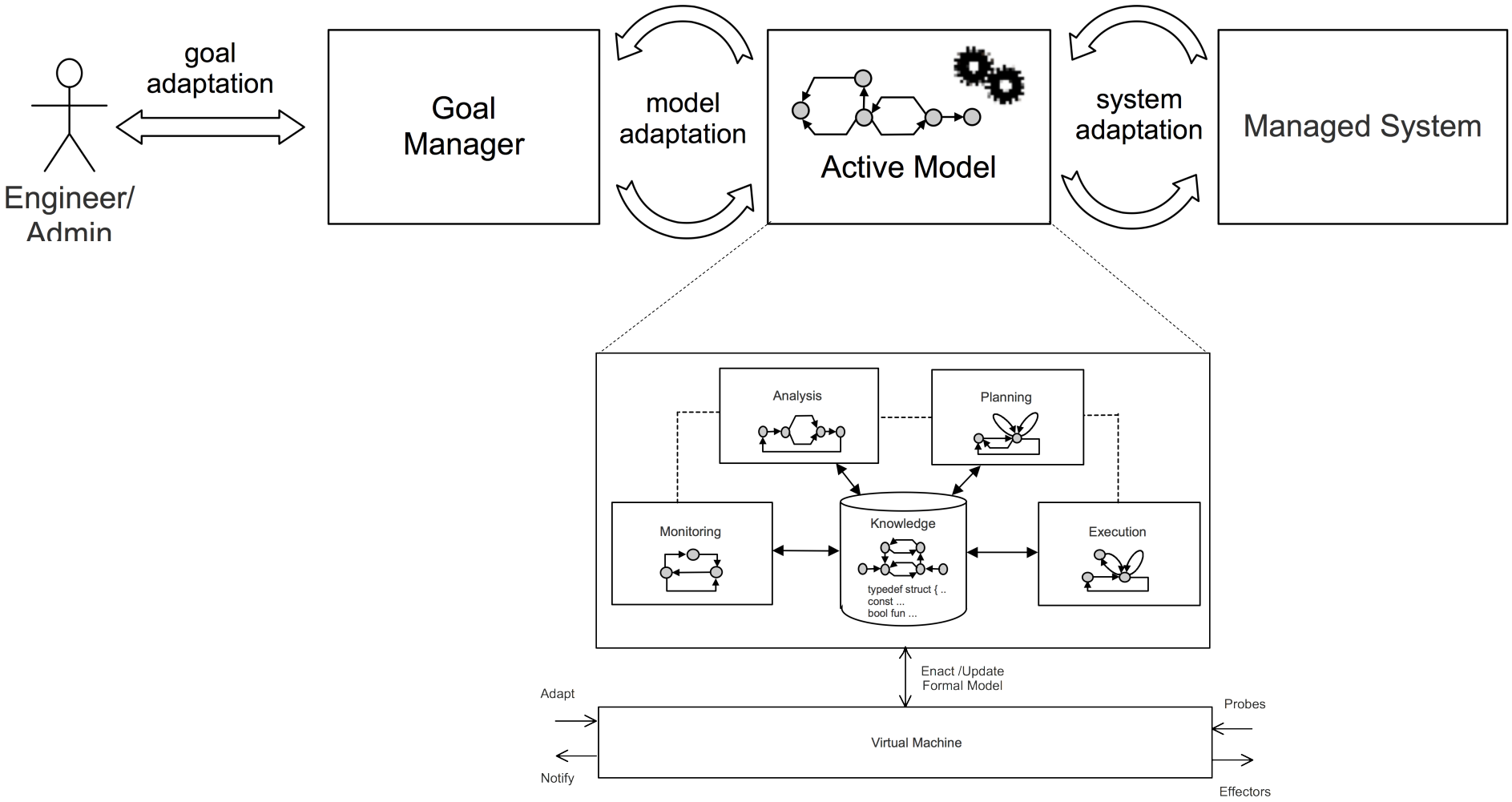


- 3 layered model of Kramer & Magee
  - Component control (layer 1), change management (2), goal management (3)
- Focus on layer 2 and 3
  - Assumption: managed system is equipped with required sensors and effectors
  - Instrumentation of managed system is research subject in its own right
- Case study: logistic multi-robot system

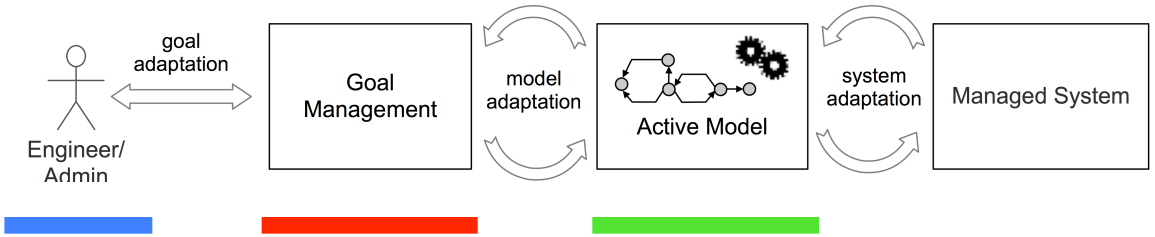
Case study


Robot System Manager
Tasks
Show
Edit
Pick
Drop
Status
A
D
Done
B
D
Done
C
D
R1
B
D
R2
A
D
Pending
C
D
Pending
Map
Show
Edit
R2
A
D
B
R1
C

Approach

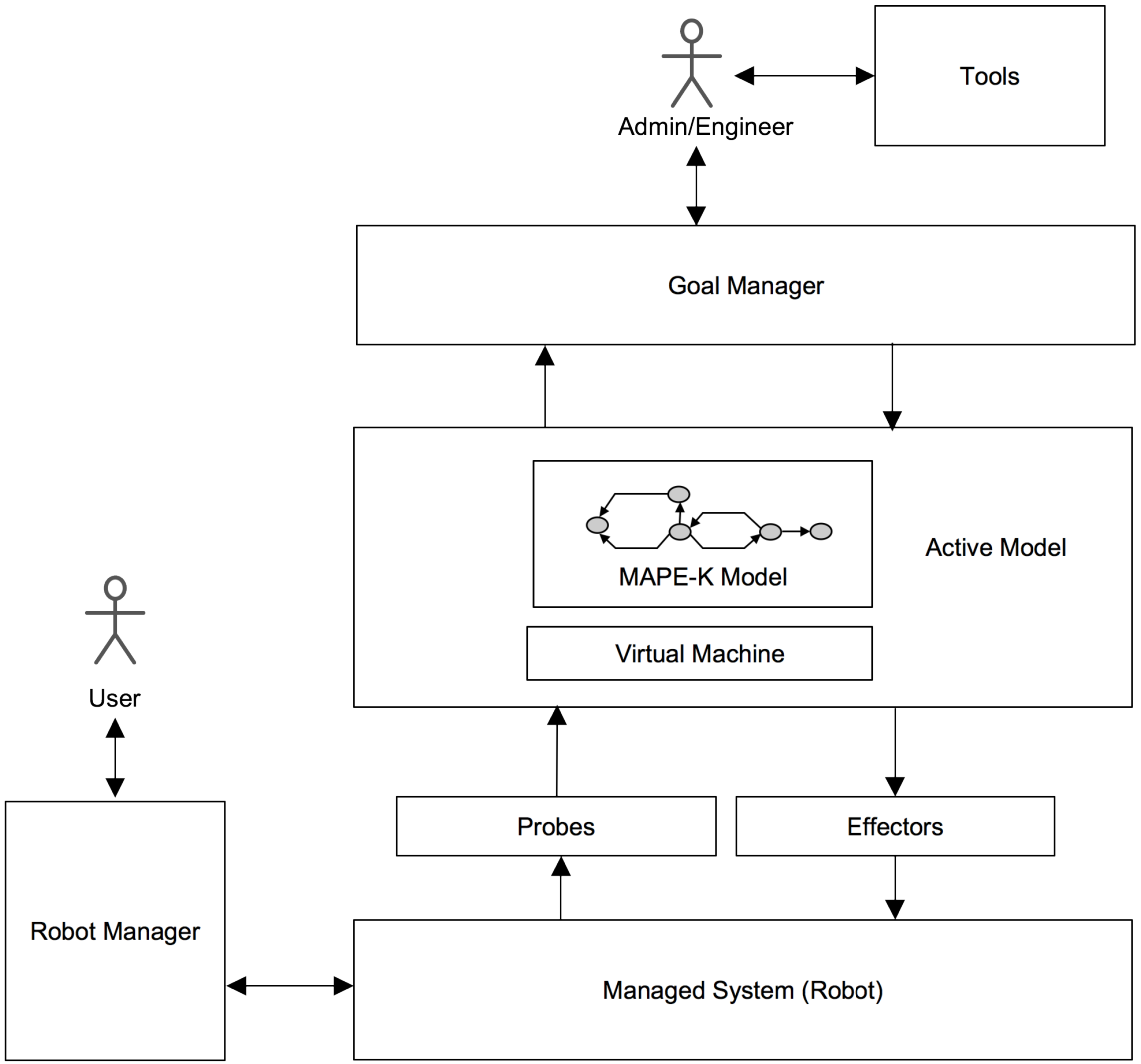

goal
adaptation
Engineer/
Admin
Goal
Manager
model
adaptation
Active Model
system
adaptation
Managed System
Analysis
Planning
Monitoring
Knowledge
typedef struct { ..
const ...
bool fun ...
Execution
Enact /Update
Formal Model
Adapt
Notify
Virtual Machine
Probes
Effectors

Approach

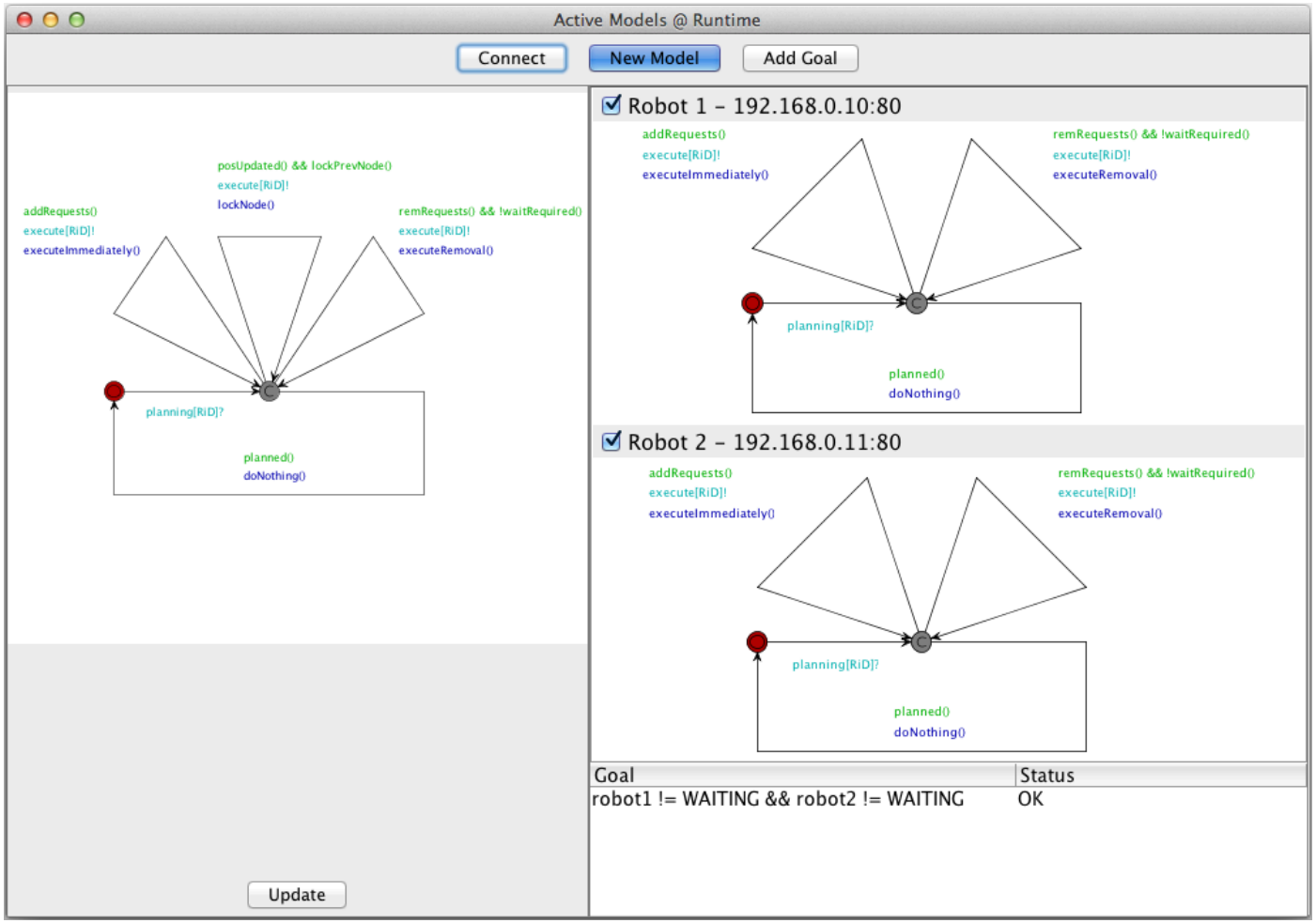


- Active model
  - Is a formally verified model
  - Realizes a MAPE-K loop
  - To adapt the managed system
- Goal management
  - Monitors the active model
  - Can adapt the active model (e.g., to improve it or deal with a particular adaptation problem)
- Engineer/Admin
  - Can monitor goal satisfaction/violation
  - Can change the active model, verify and deploy it, to manage (new) goals using goal management

Realization



Goal Management Interface



Virtual machine

- Transforms a formal model (network of timed automata) into a graph representation
- Executes that model
- Can adapt the current model at runtime
- Can detect and notify goal violations

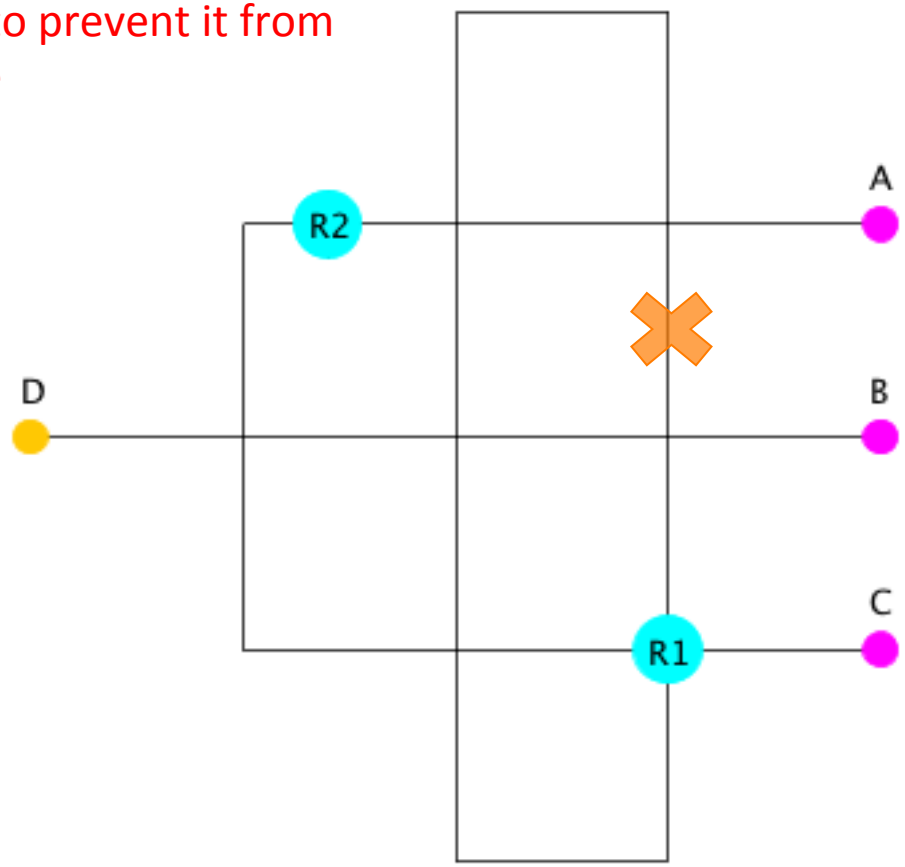
Levels of adaptation

- Level 1: active model adapts the managed system
  - Close temporally a lane in the warehouse for maintenance
- Level 2: adapt the active model (adapt MAPE)
  - Add a new drop location in the warehouse

Level 1 adaptations

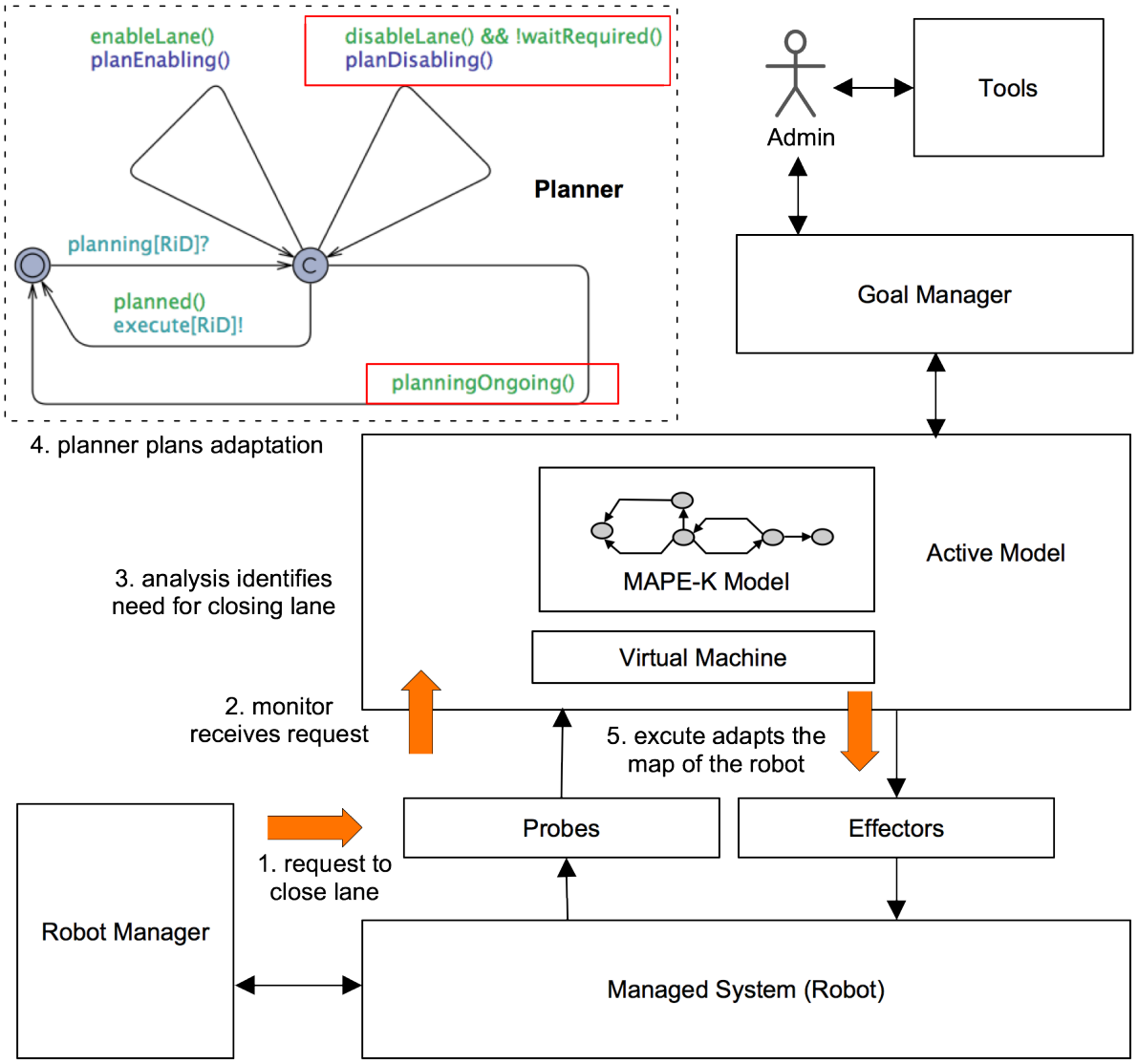
*Close temporally a lane in the warehouse for ma*intenance

- Adapt the robot to prevent it from using a closed lane



Level 1 adaptations

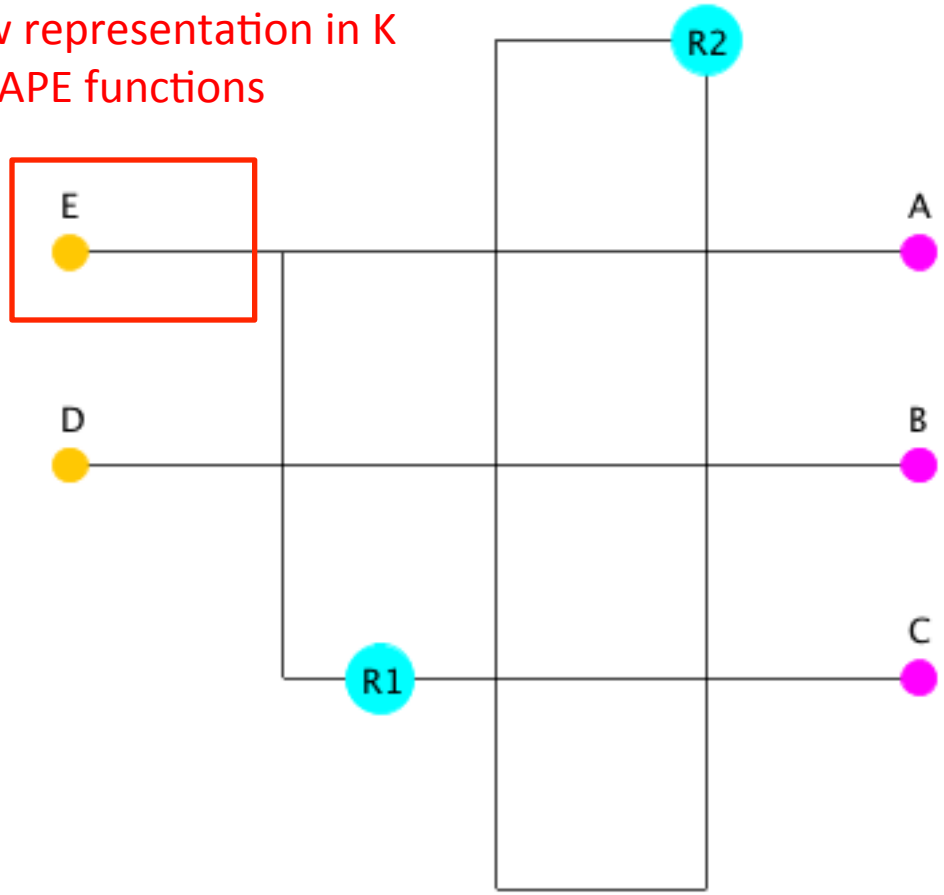
*Close temporally a lane in the warehouse for ma*intenance



Level 2 adaptations

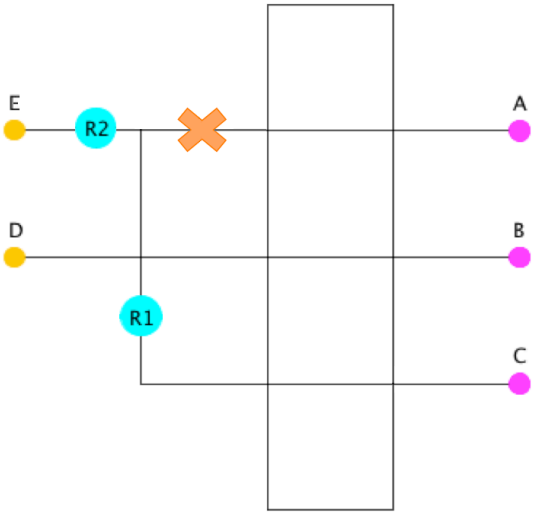
Add a new drop location in the warehouse

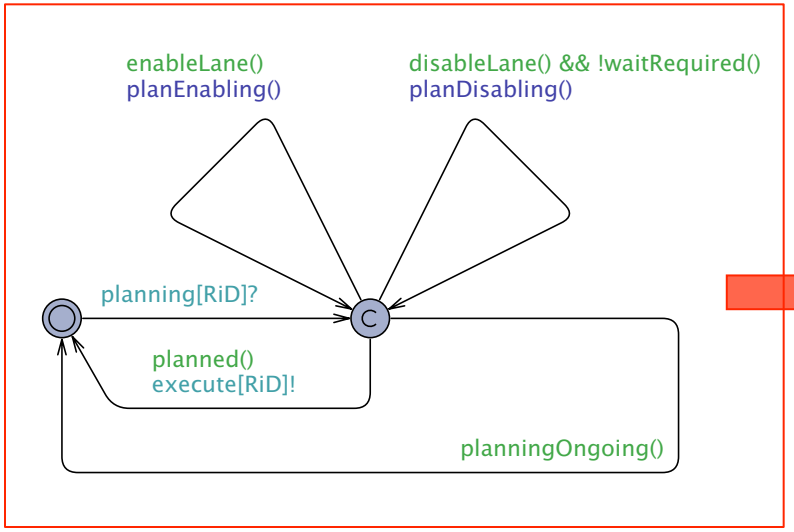
- Add new part of the map for the robot
- Creates new deadlock situations when certain lanes are disabled
- Requires adding new representation in K and adaptations of MAPE functions

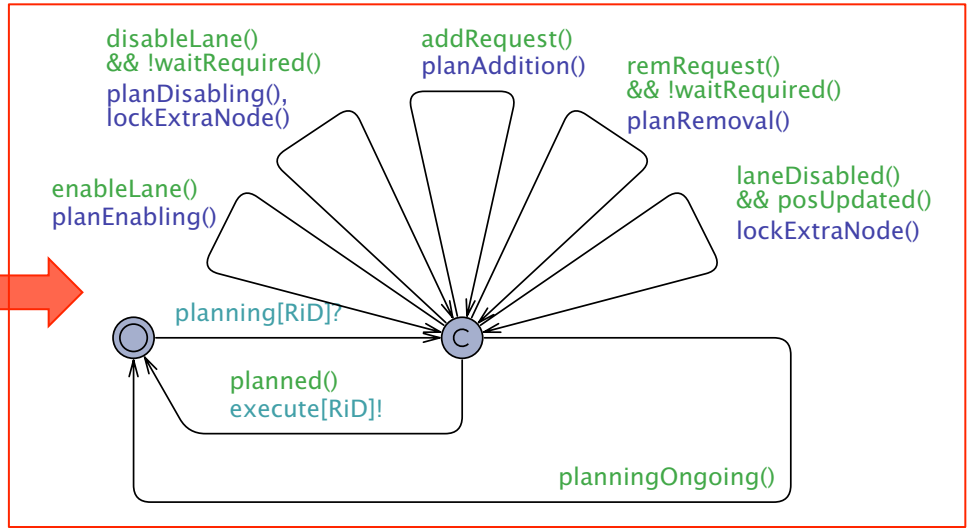


Level 2 adaptations

Deal with new deadlock threat (close additional lane): e.g., update planner

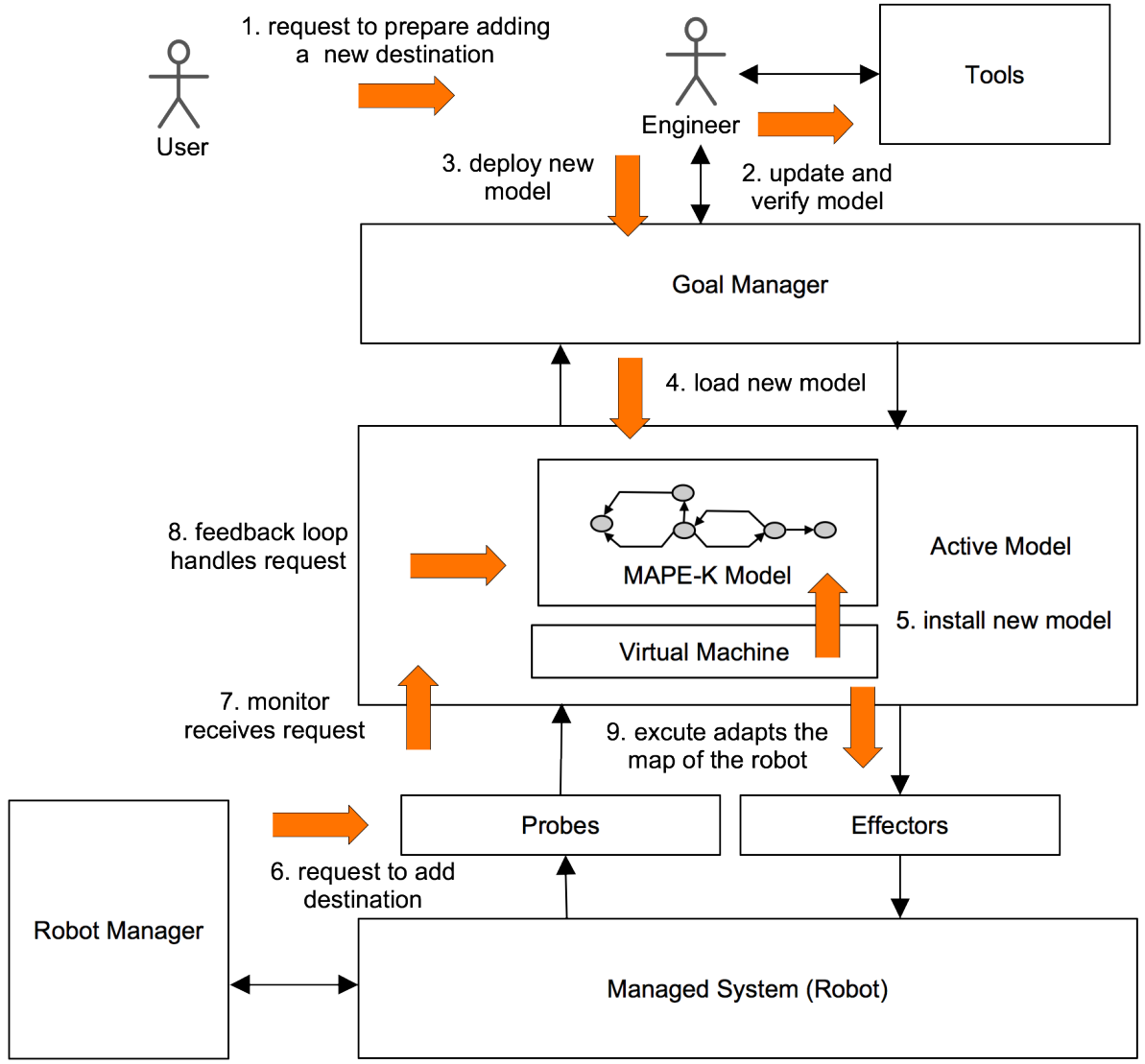






Level 2 adaptations

Add a new drop location in the warehouse



ActivFORMS summary

- Formal active model guarantees verified properties of the adaption process
- Active model directly executes the adaptation: no coding, no model transformations
- Adaptation of adaptation functions: lightweight process to add new goals
- Online detection of goal violations

Tradeoffs

- Expert knowledge to design and change the formal models
- Modeling is limited by the expressive power of the modeling language
- Language might not be appropriate to model adaption logic for particular types of systems
- Possible performance overhead

Paves the way for future research

- Domain specific design primitives to support the designer
- Different modeling languages (e.g. probabilistic automata to model domain)
- Coordination between Active Models in decentralized setting
- Automation goal management by learning
- Scalable runtime verification

Overview

- Architecture-based self-adaptation vs. control-based self-adaptation
- Reference approaches for architecture-based self-adaptation
- Formal methods for self-adaptive systems
- Active formal methods for self-adaptation
- Wrap up

Wrap up: Goals of this tutorial

- Understand the notion of self-adaptation
- Get familiar with references approaches for architecture-based self-adaptation
- Get familiar with state of the art in formal methods for self-adaptive systems
- Understand the challenges in formal methods at runtime for self-adaptive systems

Wrap up
Understand the notion of self-adaptation

- Self-adaptation is motivated by the need to deal with design time uncertainties
- Two key families are
  - Control-based self-adaptation: controller design and analysis based on control theoretic foundation
  - Architecture-based self-adaptation: feedback loop reasons about self-model and adapts system when needed
- Separation between managed and managing system
  - Concerns of managed system are about the domain at hand
  - Concerns of managing system are about system

Wrap up

Get familiar with reference approaches for architecture-based self-adaptation

- MAPE-K reference model
  - MAPE: primary functions to realize self-adaptation
  - K: domain models
- Rainbow framework maps reference model to concrete architecture and implementation
- 3 layer model of Kramer and Magee
  - Component control – adaptation management – goal management
- FORMS: rigorous specified model that integrates different perspectives on self-adaptation

Wrap up

Get familiar with state of the art formal methods in self-adaptive systems

- Verification at construction time to provide guarantees about system goals
- Model driven approaches to guarantee conformance between models and implementation
- Recent years a clear trend towards the application of formal methods at runtime
- Dominating focus on probabilistic models of the domain
- Main focus on “parametric uncertainty” (e.g., reliability of services change over time)

Wrap up

Understand the challenges on formal methods at runtime for self-adaptive systems

- Guaranteeing domain goals under uncertainty is one part of assurances of self-adaptation
- Guaranteeing correct adaptation behavior is the other part (lack of attention so far)
- Need for solutions that deal with “structural uncertainty”
  - i.e., unanticipated change; e.g., change goals
- Scalable runtime verification

Bibliography

- B. Cheng et al., Software Engineering for Self-Adaptive Systems: A Research Roadmap, Lecture Notes in Computer Science, vol. 5525, 2009
- P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, An Architecture-Based Approach to Self-Adaptive Software, IEEE Intelligent Systems, May/June 1999
- Kephart and Chess, The vision of autonomic Computing, IEEE Computer, January 2003
- D. Garlan, S-W. Cheng, A.C. Huang, B. Schmerl, P. Steenkiste, Rainbow: Architecture- Based Self-Adaptation with Reusable Infrastructure, IEEE Computer, October 2004
- J. Kramer and J. Magee, Self-adaptation: an architecture challenge, Future of Software Engineering, FOSE 2007
- D. Weyns, S. Malek, J. Andersson, FORMS: Formal reference model for self-adaptation, ACM Transactions on Autonomous and Adaptive Systems, TAAS 7(1), 2012
- D. Weyns, R. Haesevoets, A. Helleboogh, T. Holvoet, W. Joosen, The MACODO Middleware for Context-Driven Dynamic Agent Organzations, ACM Transaction on Autonomous and Adaptive Systems, 5(1), 2010.
- D. Weyns, U. Iftikhar, D. Gil de la Iglesia, and T. Ahmad, A Survey on Formal Methods in Self-Adaptive Systems, Fifth International C* Conference on Computer Science and Software Engineering 2012
- J. Zhang and B. Cheng, Model-based development of dynamically adaptive software, International Conference on Software Engineering, ICSE 2006
- I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. 2009. Model evolution by run-time parameter adaptation, International Conference on Software Engineering, ICSE 2009
- R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS Management and Optimization in Service-Based Systems, IEEE Transactions on Software Engineering, TSE 2011
- C. Ghezzi, L.S. Pinto, P. Spoletini, G. Tamburrelli: Managing non-functional uncertainty via model-driven adaptivity, International Conference on Software Engineering, ICSE 2013
- http://homepage.lnu.se/staff/daweaa/ActivFORMS.htm (available from October 15, 2013)