Software Engineering for Self-Adaptive Systems & Self-Aware Computing

Dagstuhl Seminar 15041 on Model-driven algorithms and architectures for self-aware computing systems. January 18 – 23, 2015.

Holger Giese
Head of the System Analysis & Modeling Group
Hasso Plattner Institute for Software Systems Engineering
University of Potsdam, Germany
holger.giese@hpi.uni-potsdam.de

Outline

1. **MechatronicUML**
2. **ExecUtable RuntimE MegA models (EUREMA)**
3. **Challenges Ahead**
4. **Outlook**

1. MechatronicUML

At the level of code it seems impossible to build trustworthy advanced system of systems:

Modeling separately

- the integration of intelligent behavior,
- the integration with control theory,
- the real-time coordination, and
- the reconfiguration at the level of agents.

} Micro Architecture (first two items)

} Macro Architecture (last two items)

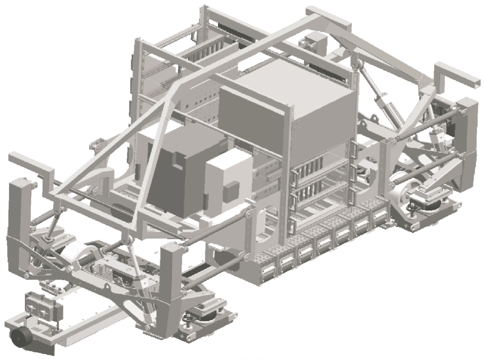
- Analyze the models in a compositional manner
- Synthesize the code

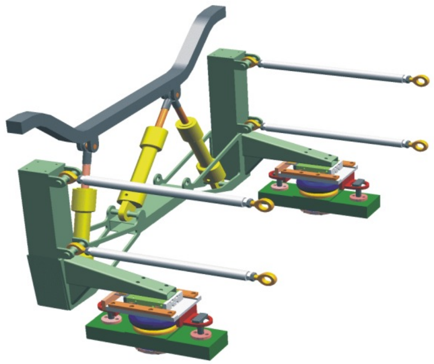
Application Example: Railcab System (1/2)

A system of **autonomous shuttles** that operate on demand and in a decentralized manner using a **wireless network**.

System of systems
- Hard real-time
- Safety-critical
- Self-Optimization

Application Example: Railcab System (2/2)

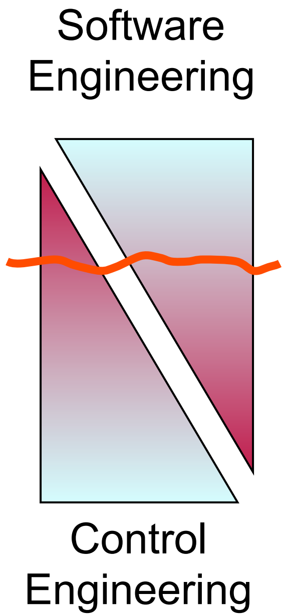




Domains:

- Logistic
- Real-time coordination
- Local control
- Electronics
- Mechanics

} Classical Engineering (Mechatronics)



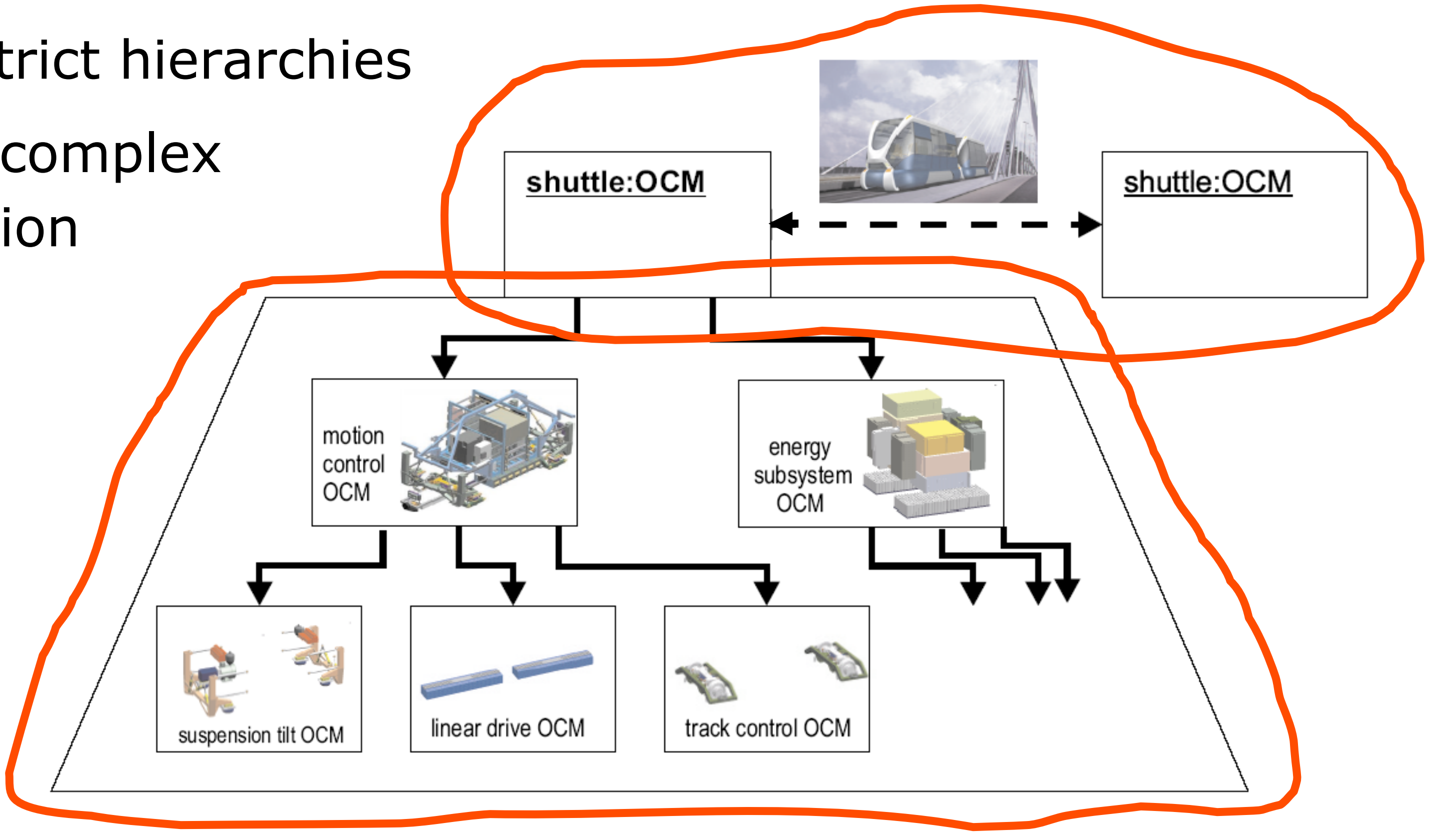
⇨ Integration of the different worlds

⇨ Self-optimization at multiple levels

⇨ Self-adaptation/self-coordination via software

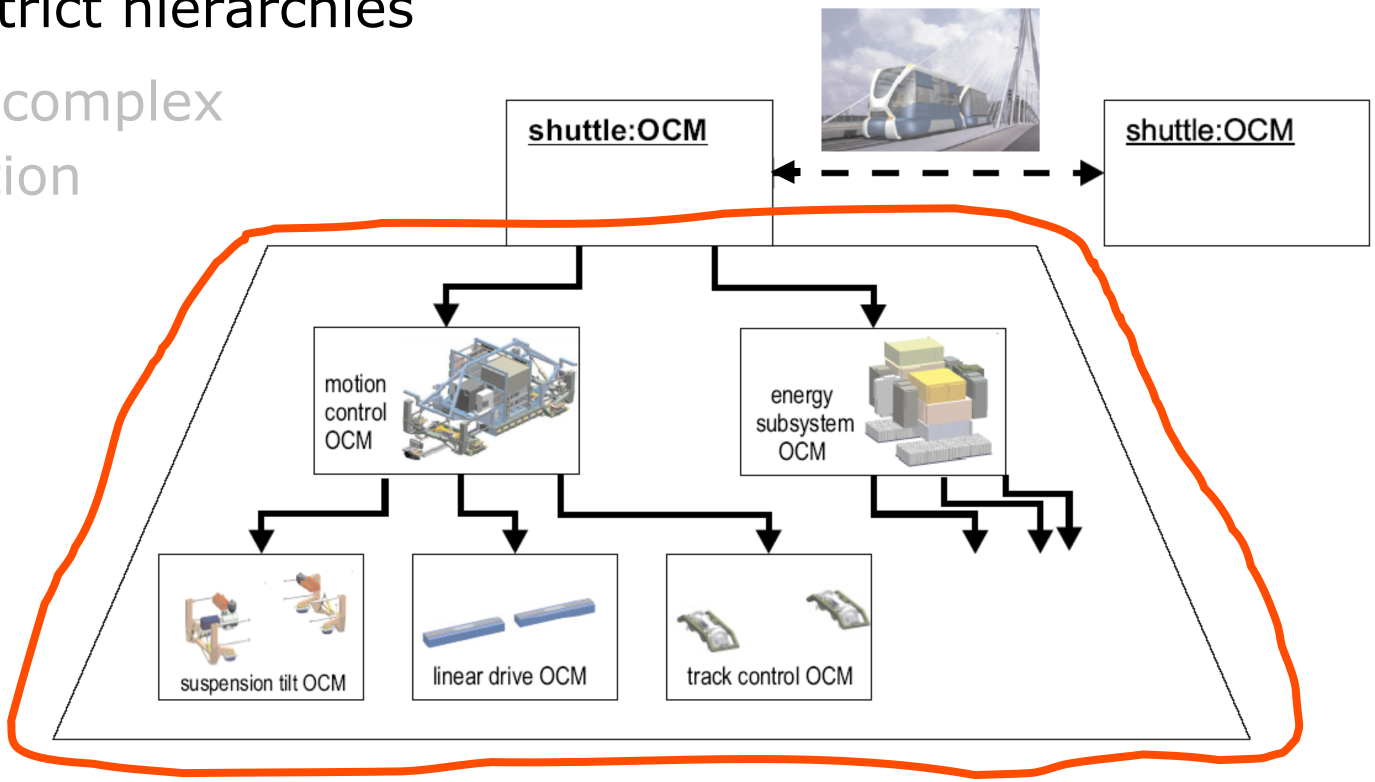
Micro and Macro Architecture

- **Autonomous subsystems (shuttles)**
- Within: strict hierarchies
- Outside: complex coordination



Micro Architecture

- **Autonomous subsystems (shuttles)**
- Within: strict hierarchies
- Outside: complex coordination



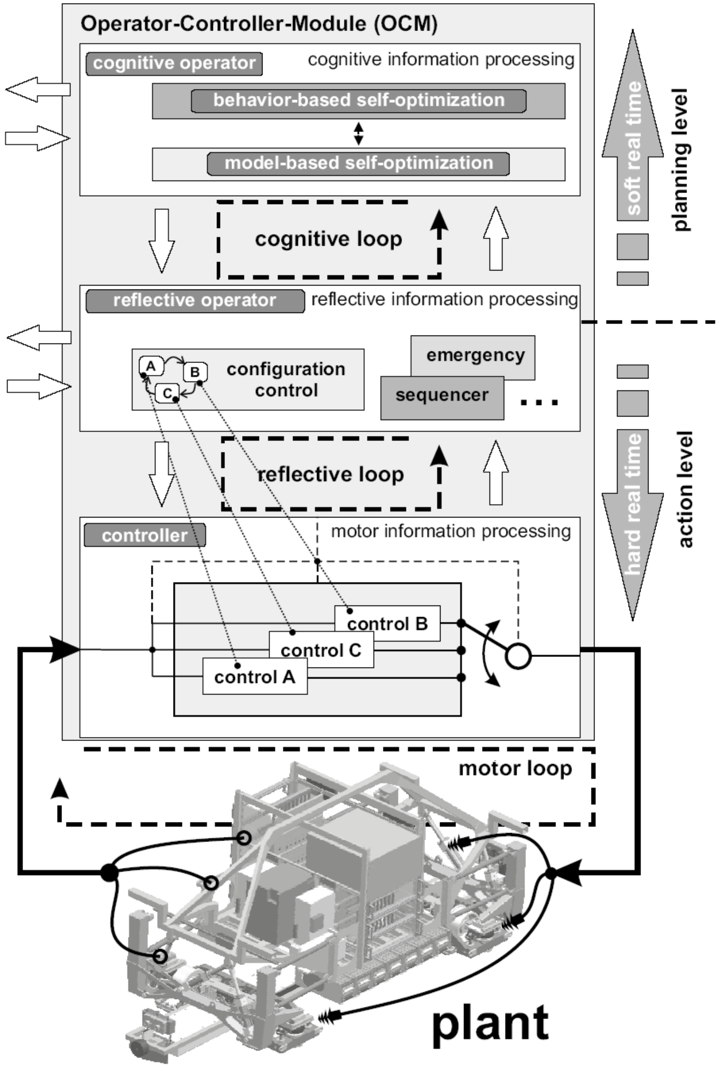
Micro Architecture

- **Operator-Controller Module [ICINCO04]**
- **Cognitive operator ("intelligence")**

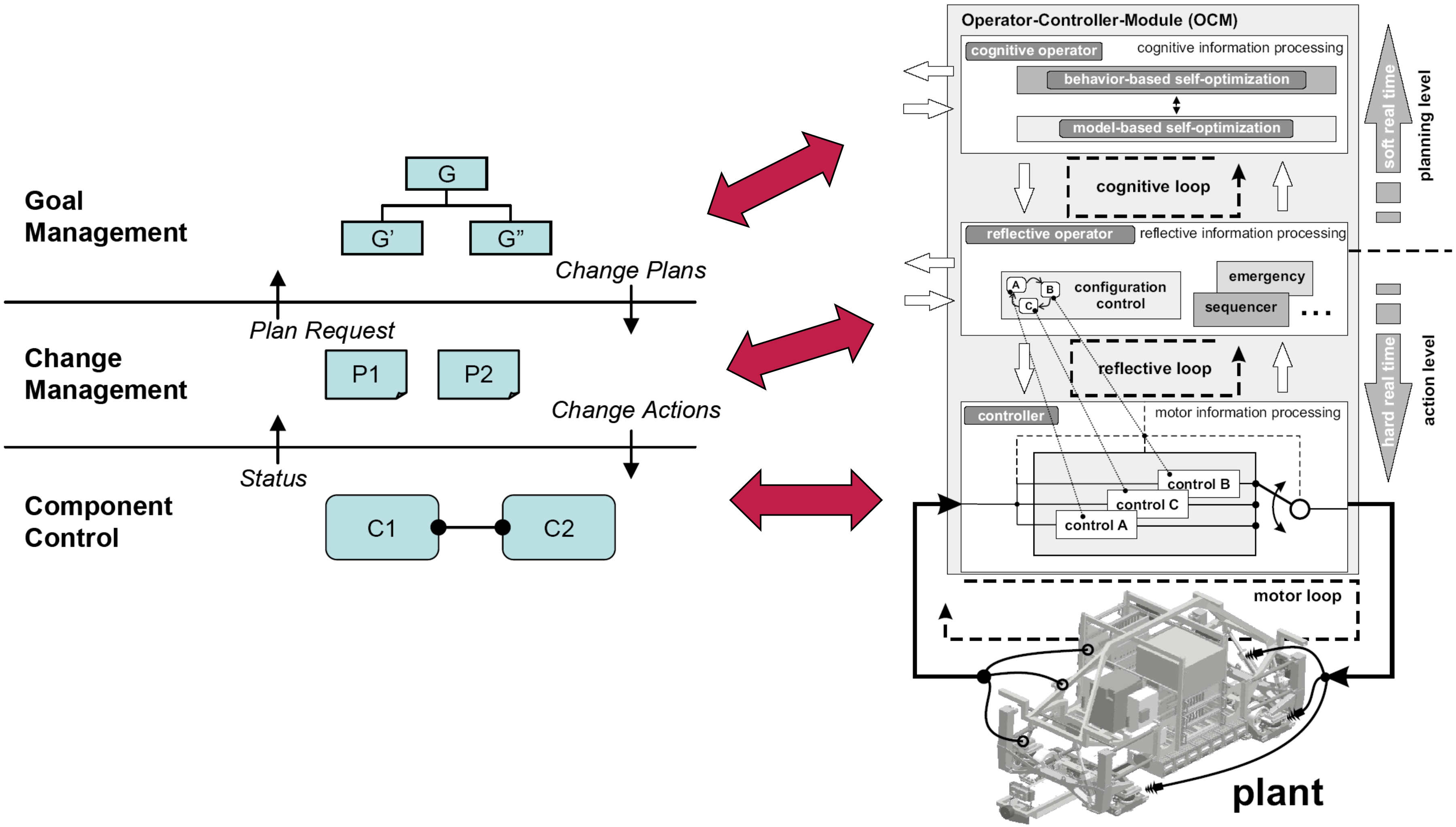
  decoupled from the hard real-time processing
- **Reflective operator**

  Real-time coordination and reconfiguration
- **Controller**

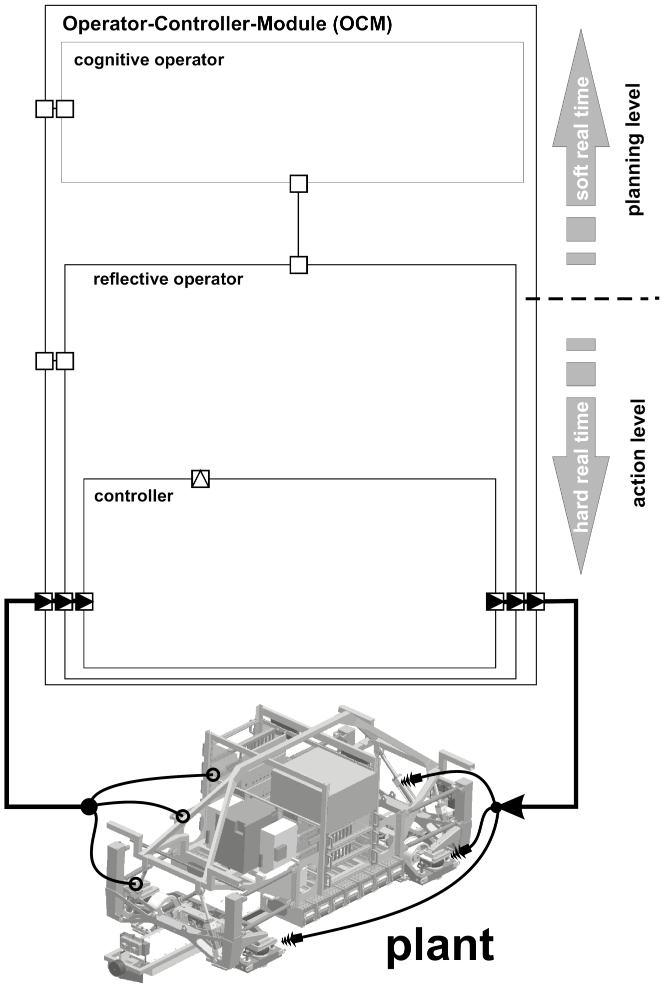
  Control via sensors and actuators in hard real-time



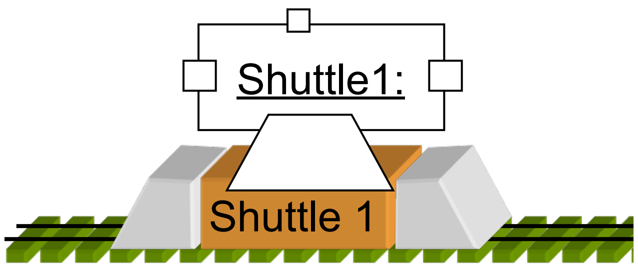
OCM & Reference Architecture



MECHATRONIC UML: Components

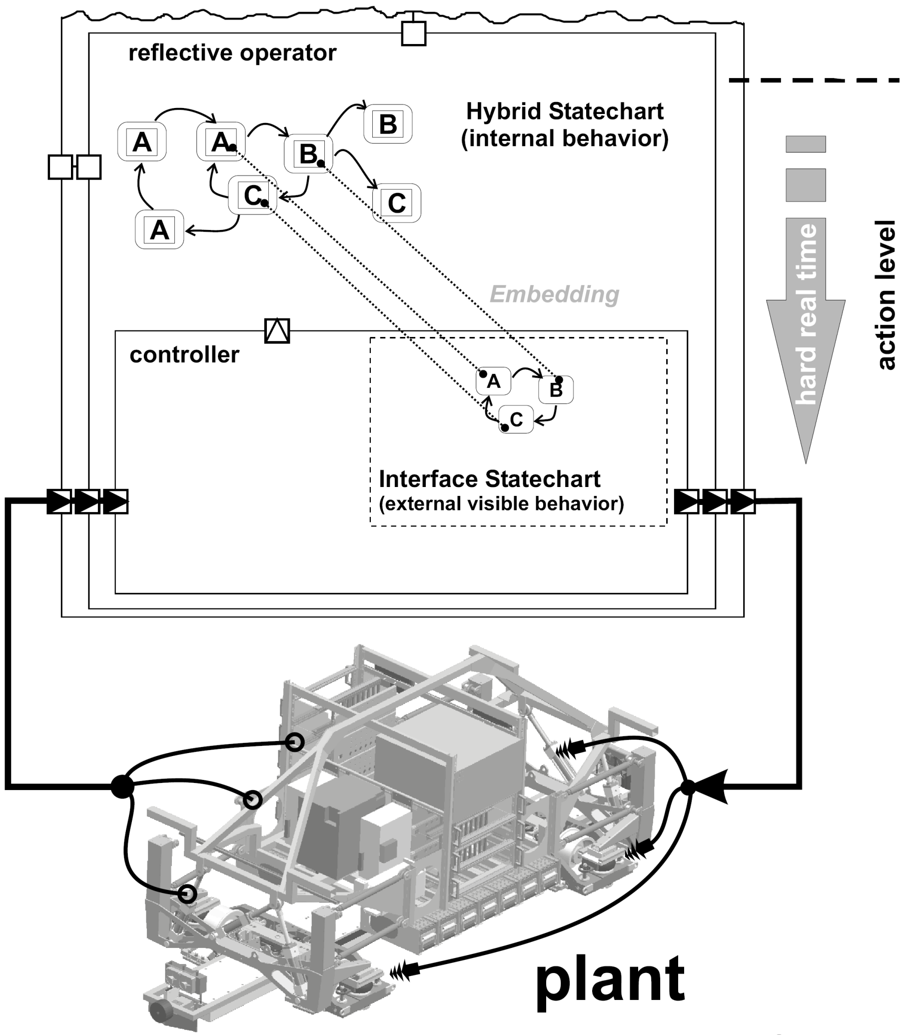


- Model the structure of the Software with hybrid UML components with
- Hybrid behavior
  - Regular ports (discrete)
  - Continuous ports
  - Hybrid ports
- Reconfiguration
  - Permanent ports
  - potential ports



Integration Reflective Operator & Controller

- Hybrid components
  - UML components (Fujaba)
  - Block diagrams (CAMeL)
- Hybride Statecharts can embed subordinated hybrid components
  - Controller or
  - The reflective operator of subordinated OCMs
- Interface statecharts enable **modular reconfiguration** across the boundaries of hybrid components
- Automatic **check** for correct embedding

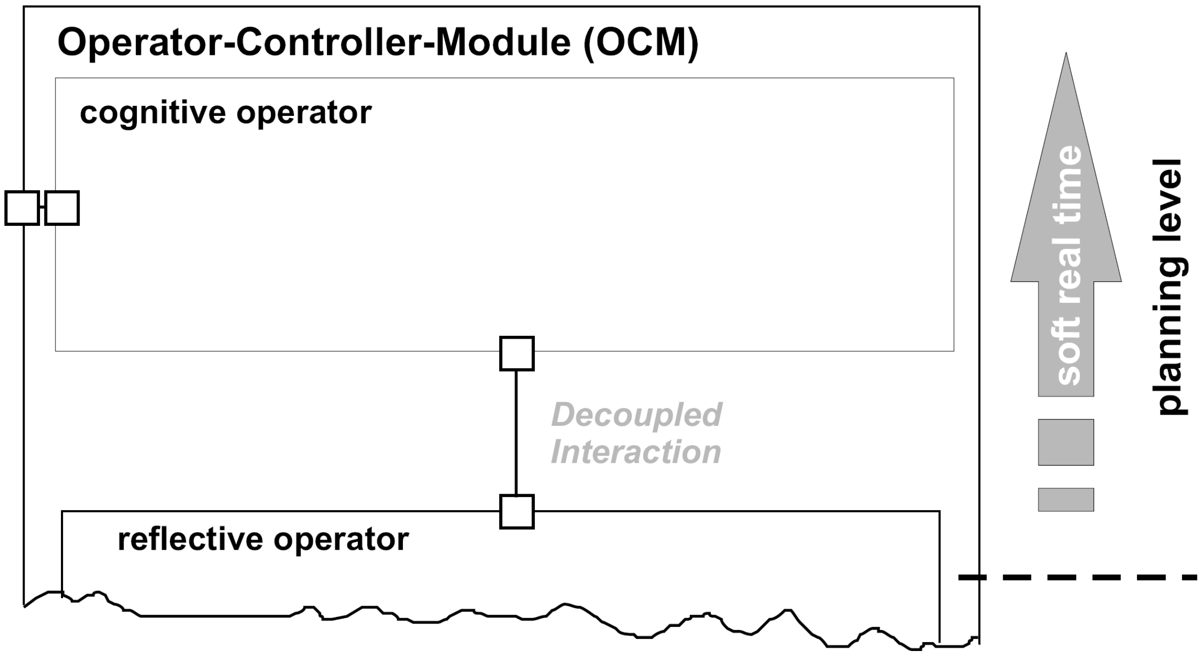


[FSE04]

Integration Cognitive & Reflective Operator

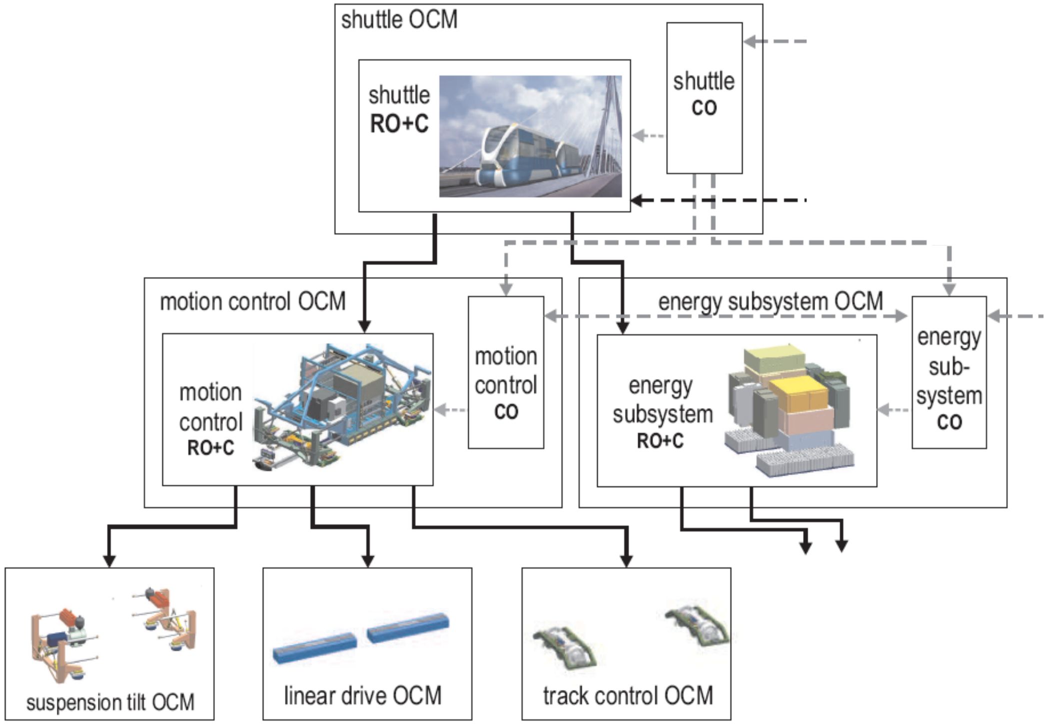
The cognitive operator is decoupled from the rest:

- We **check** that the reflective operator realizes a "***Filter***" which excludes unsafe reactions.
- The cognitive operator can "**guide**" the reflective operator as long as the commands given are considered to be safe and occur in time.

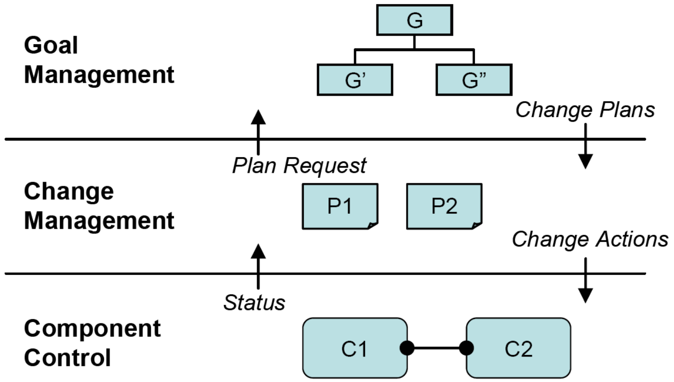


Strict Hierarchies

- **Concepts [FSE04]:**
- Hybrid components: UML components or block diagrams
- Hybride Statecharts embed hybrid components (controller or the **reflective operator** of subordinated OCMs)
- Interface statecharts enable **modular reconfiguration** across the boundaries of hybrid components



Strict Hierarchies & Reference Architecture

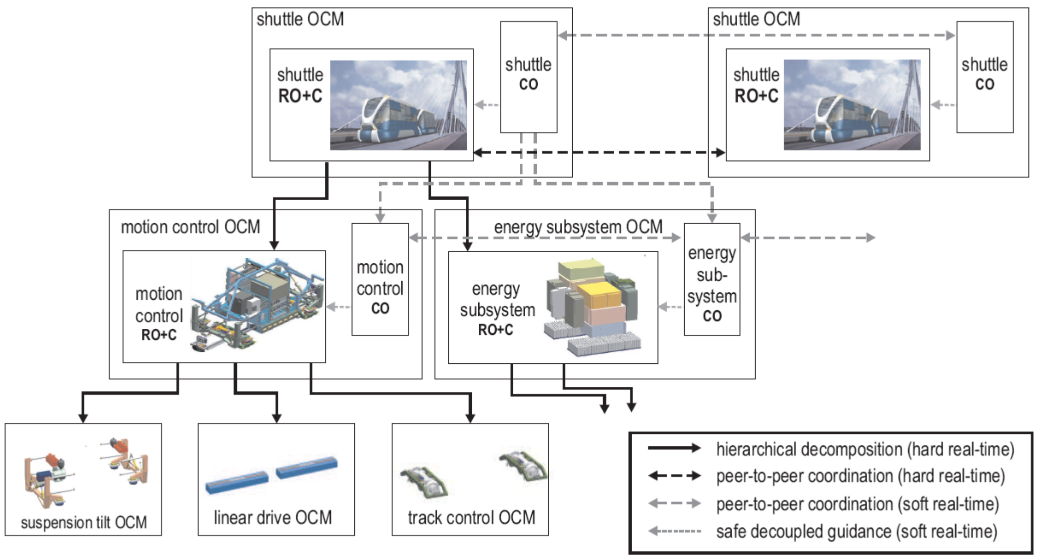


distributed over the cognitive operators (may build a hierarchy)

distributed over the reflective operators (strict hierarchical coordination)

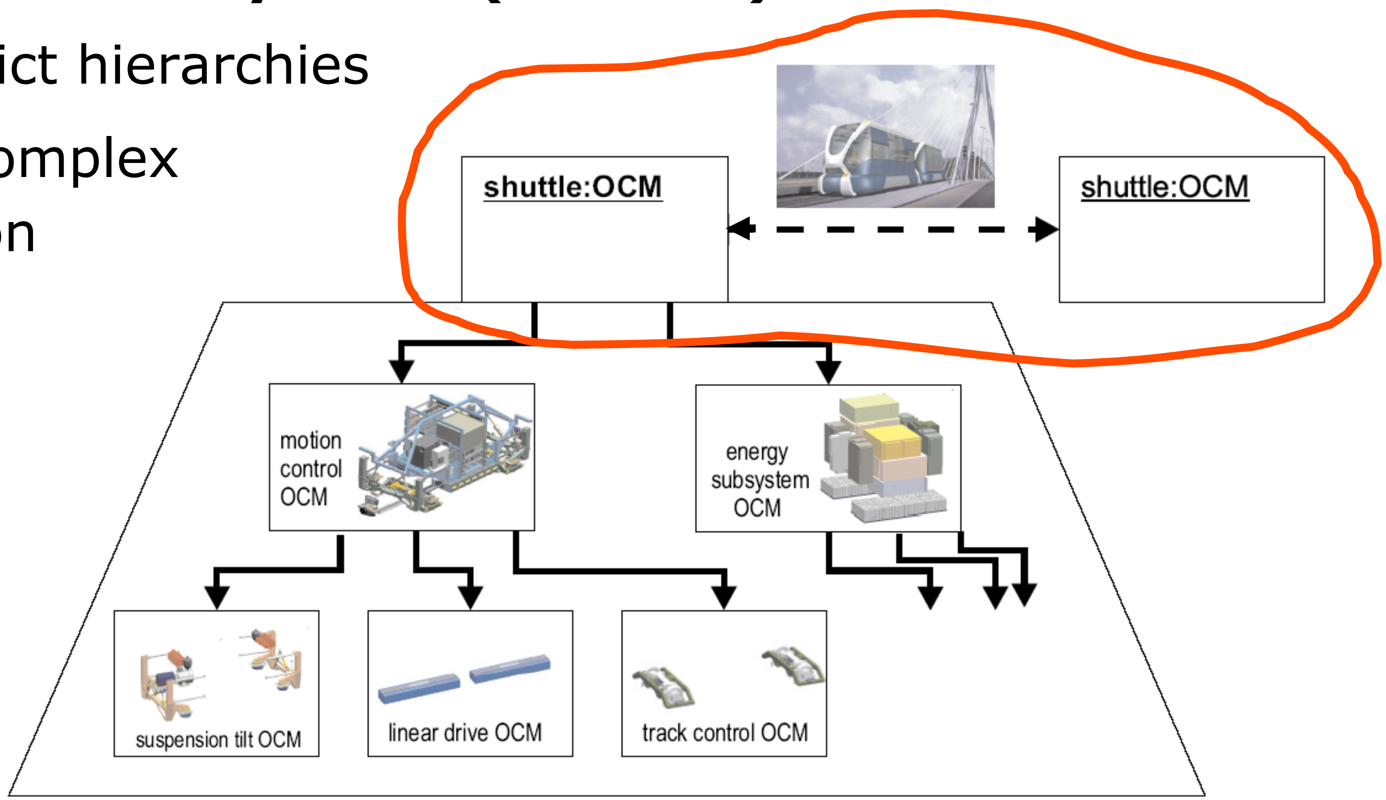
distributed over the controllers and reflective operators (may build a hierarchy)

- **Difference:**
- Hierarchy of parts which include change management functionality ⇨ self-adaptation at multiple levels
- Reflective operator includes functionality as well as change management ⇨ separation is less strict!



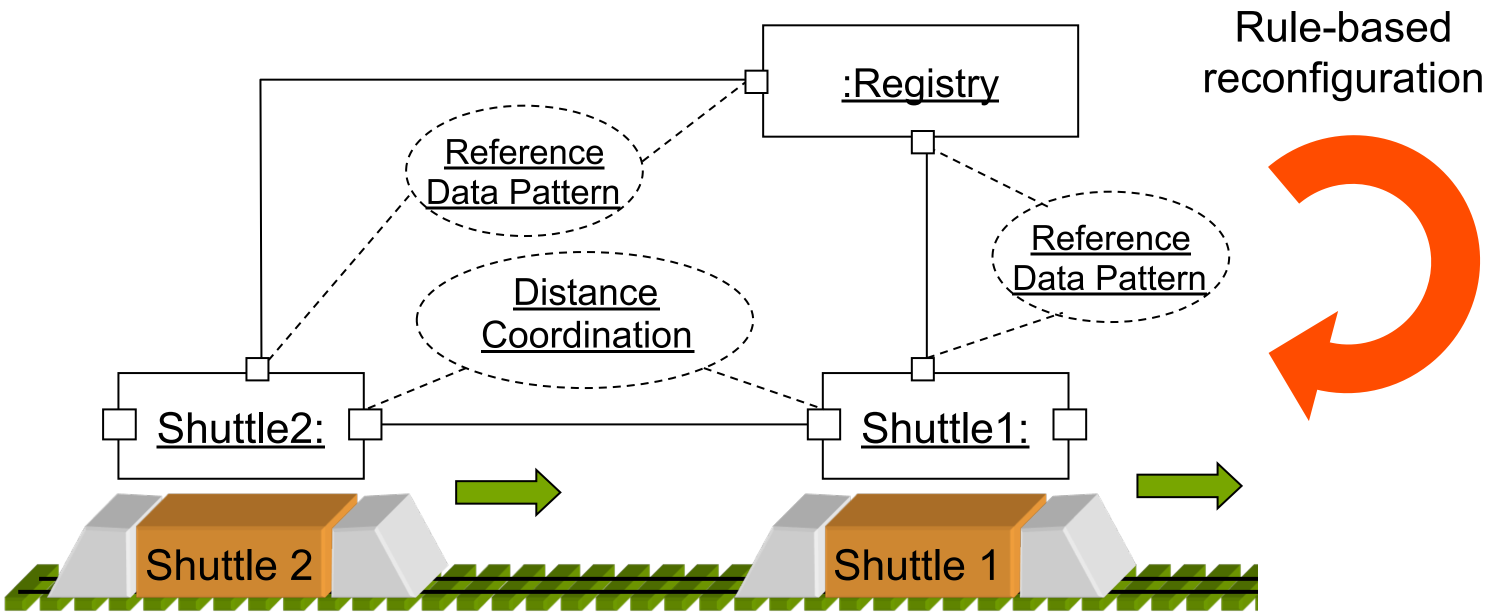
Macro Architecture

- **Autonomous subsystems (shuttles)**
- Within: strict hierarchies
- Outside: complex coordination

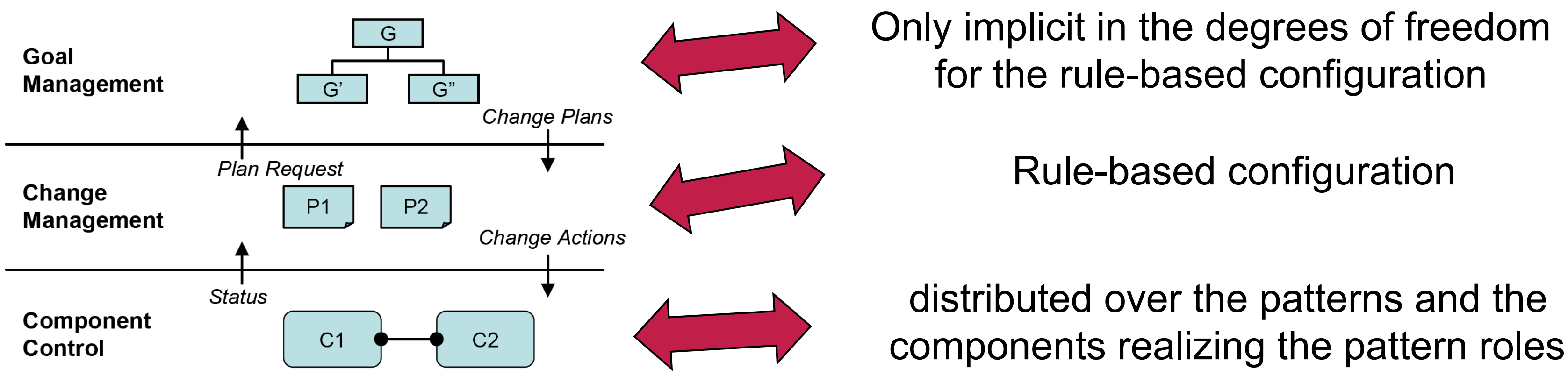


Macro Architecture: Coordination

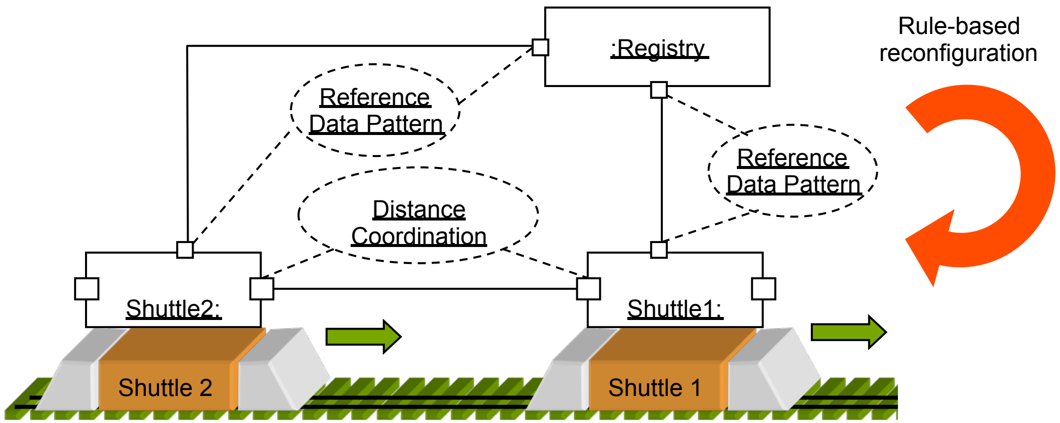
- **Real-time coordination via pattern [ESEC/FSE03]**
  - Real-time protocol state machines for each role
  - Real-time state machines for each connector
- **Rule-based reconfiguration (self-coordination) [ICSE06]**
  - Rules for instantiation and deletion of patterns



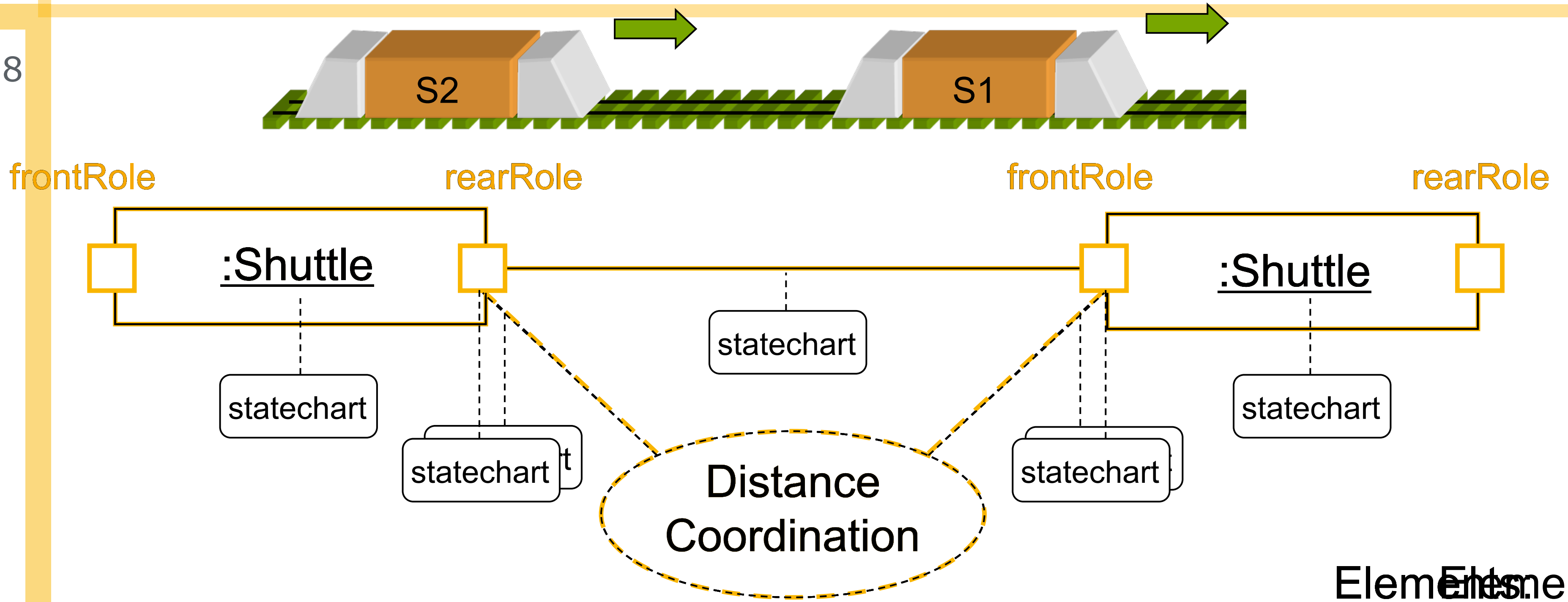
Complex Coordination & Reference Architecture



- **Difference:**
- Pattern capture component interaction as well as its instantiation ⇨ self-coordination
- No new change plans but only choices which can be made by the local cognitive operators



Real-Time Coordination via Patterns



Pattern (Distance Coordination):

- Model: Statecharts for roles and connector
- Specification: required OCL RT properties

Components (Shuttles):

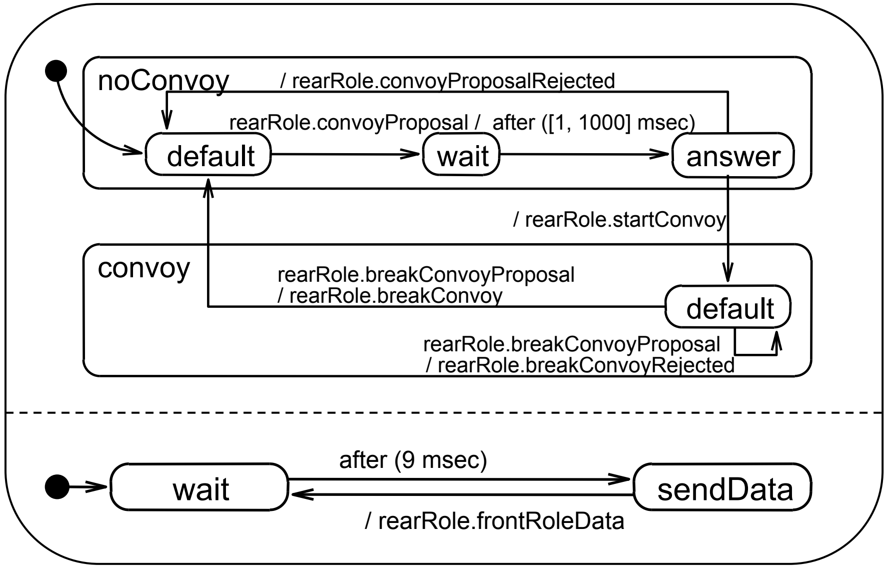
- Model: Statecharts for ports (refined roles) and synchronization
- Specification: local OCL constraints

Elements:

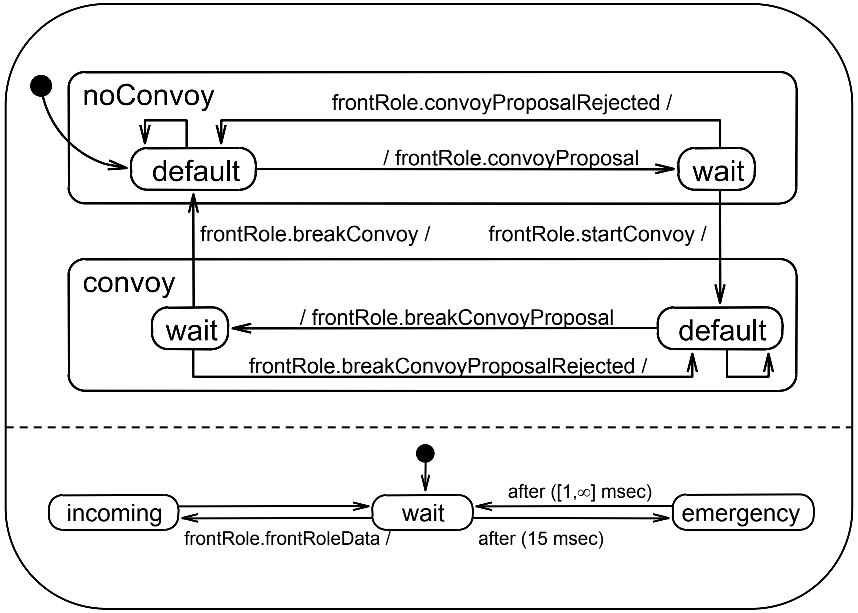
- Components
  - Ports
- Connectors
  - Patterns
  - Roles

Complex Coordination: Role Protocols

Role: FrontRole



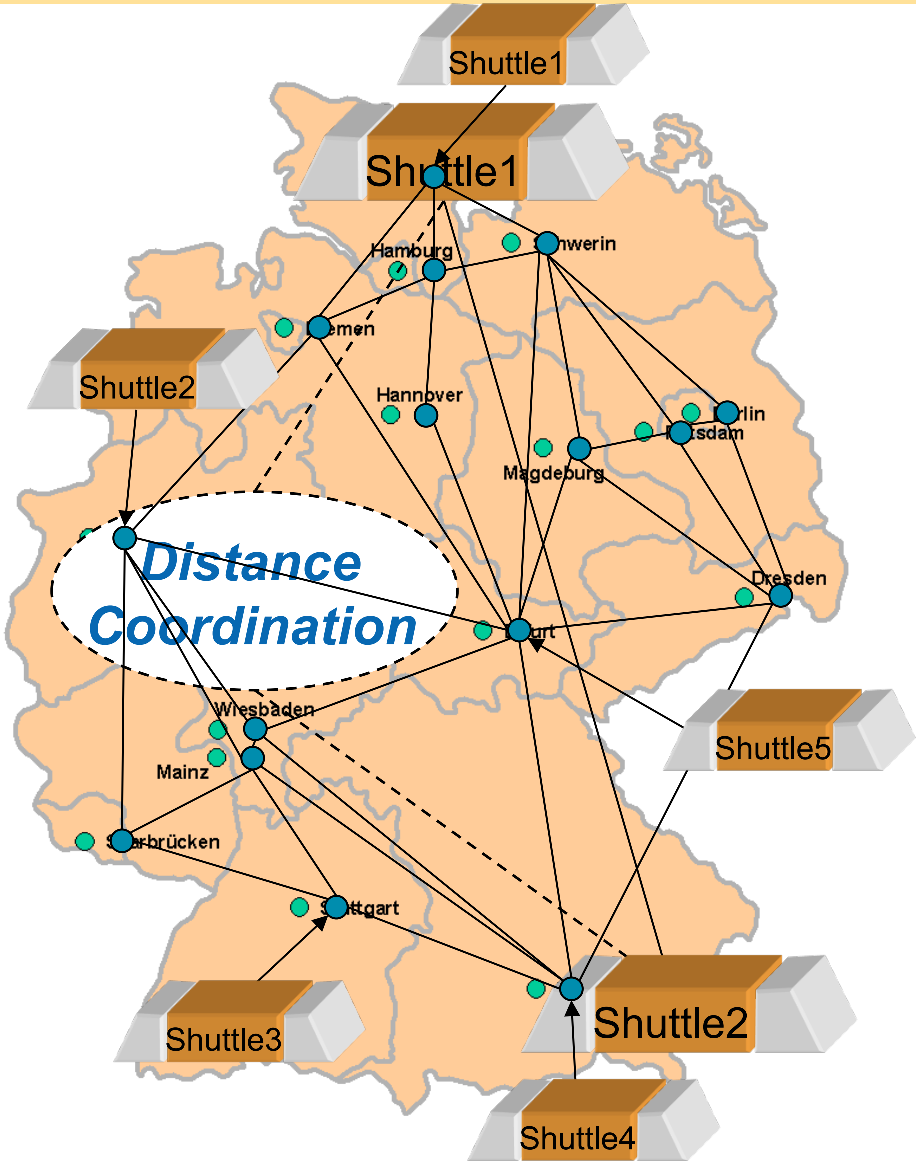
Role: RearRole



Connector:

- buffer with maximal delay of 5 msec
- modeled faults: only full communication break down

Rule-Based Reconfiguration (1/2)



Problem:

- Shuttles move and create resp. delete Distance Coordination patterns
- Arbitrary large topologies with moving shuttles

Solution:

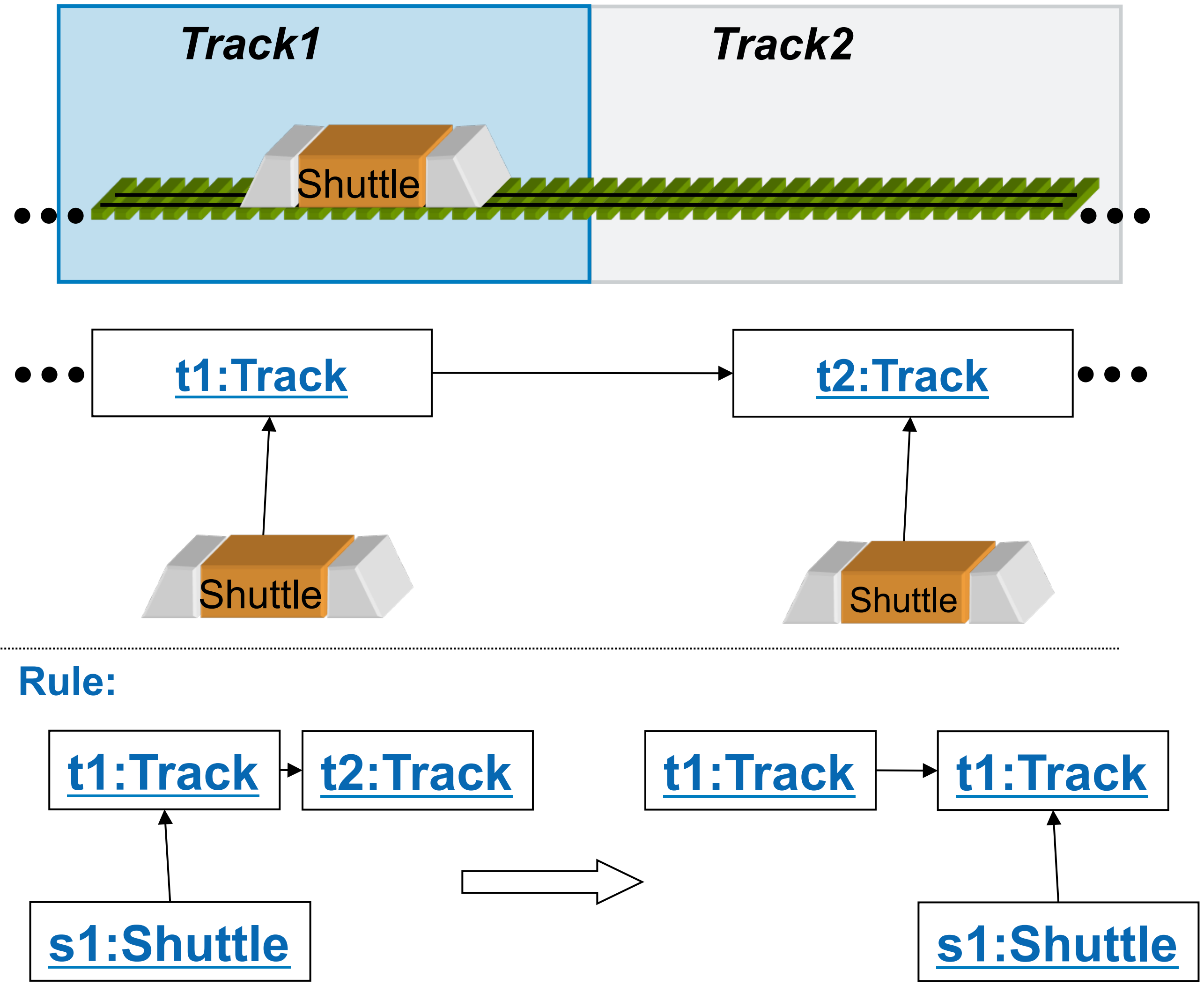
- State = Graph
- Reconfiguration rules = graph transformation rules
- Safety properties = forbidden graphs

⇨ Formal Verification possible

Rule-Based Reconfiguration (2/2)

Apply Graph Transformation Systems

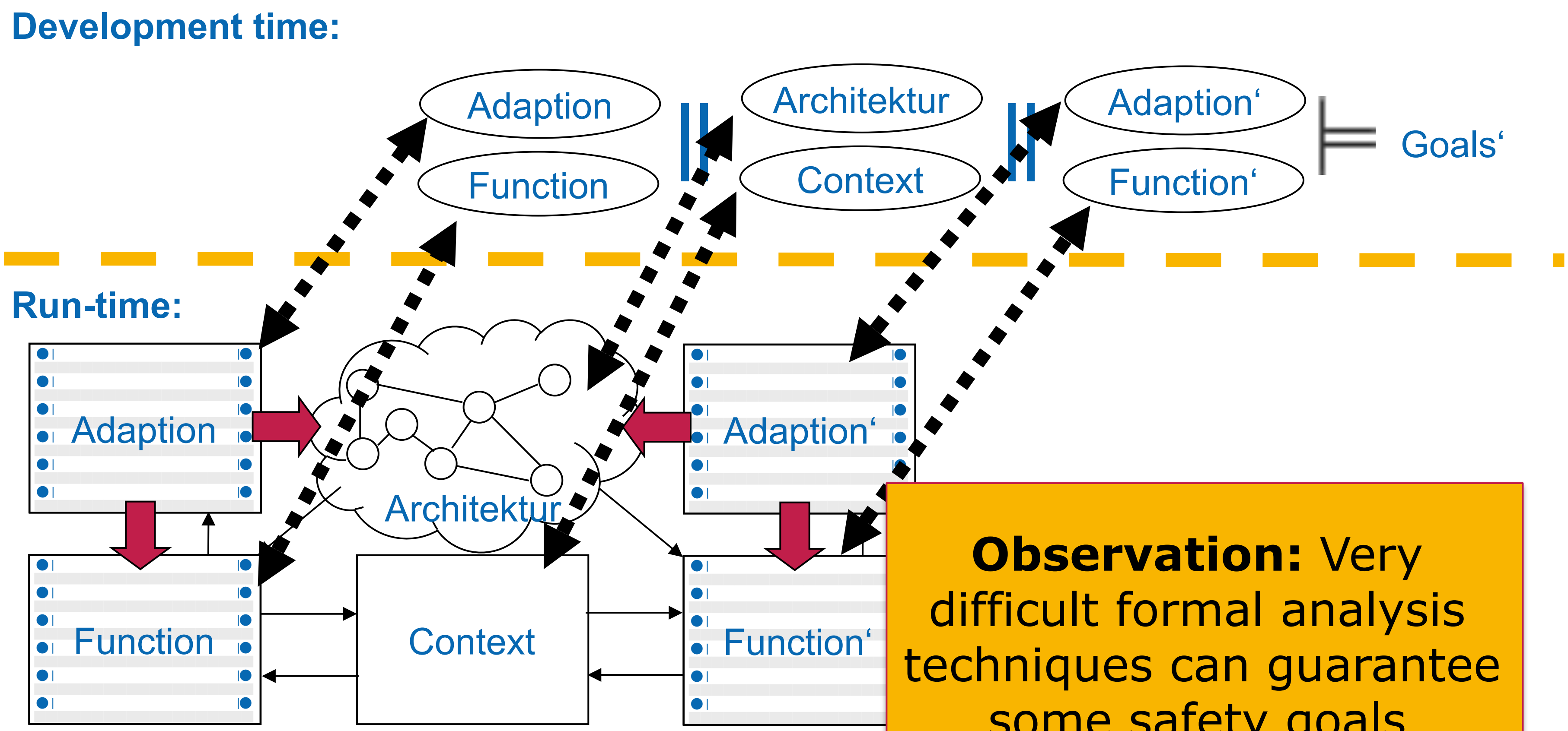
- □Map the tracks
- □Map the shuttles
- □Map the movement of shuttles to rules
- □Map the re-configuration to rules



Application Example: Self-Coordination

- **Cognitive Operators:** do self-optimization
  - Maneuver planning
  - Convoy planning
  - Shuttle planning
- **Reflective Operator:** switch to guarantee safety
  - Realize maneuvers planned by the cognitive operator(s)
  - Recognize timeouts and enforced related safety maneuvers
  - Detect problems of controllers and enforced related safety maneuvers

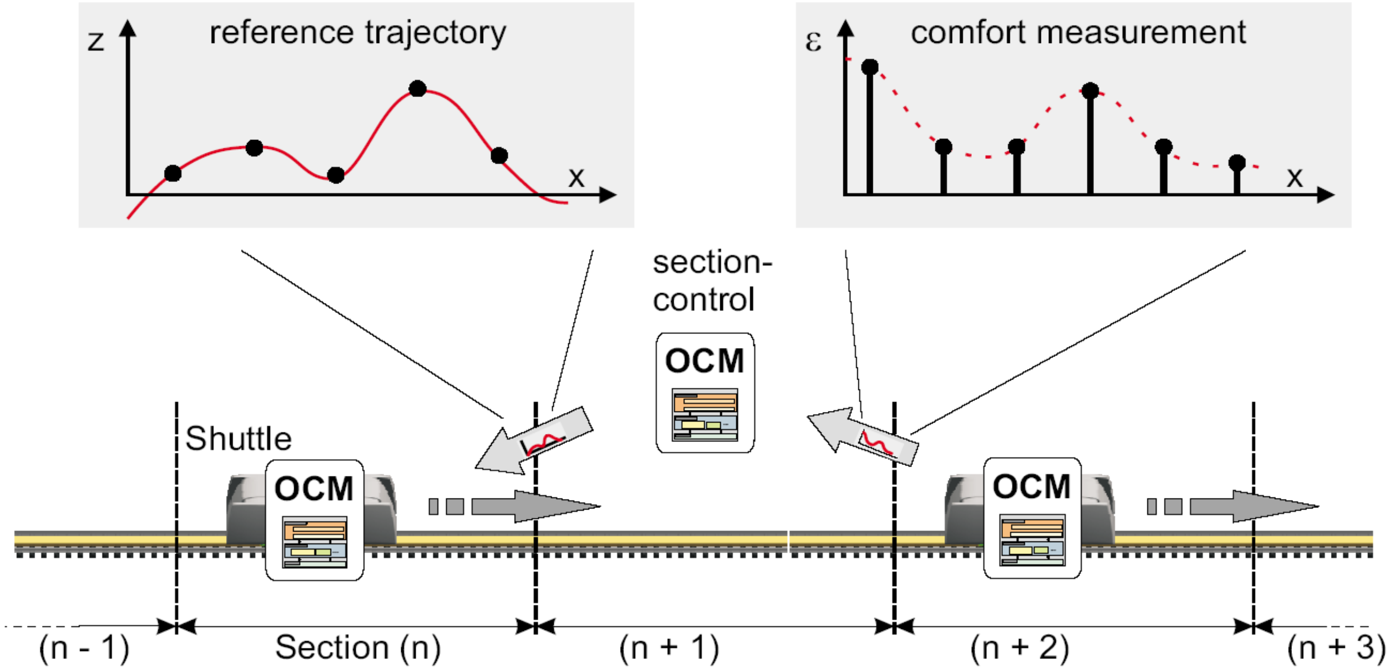
Models at Development Time (2/2)



Observation: Very difficult formal analysis techniques can guarantee some safety goals (validity of the models is guaranteed to some extent by synthesis)

Application Example: Self-Optimization

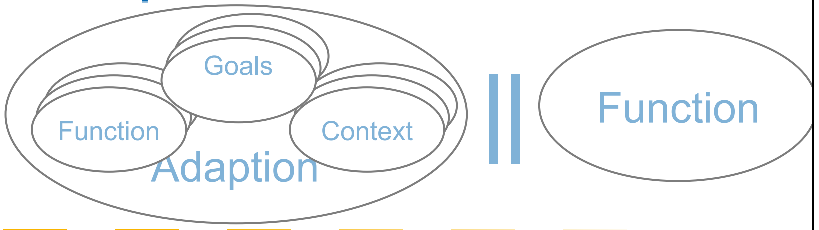
[STTT2008]

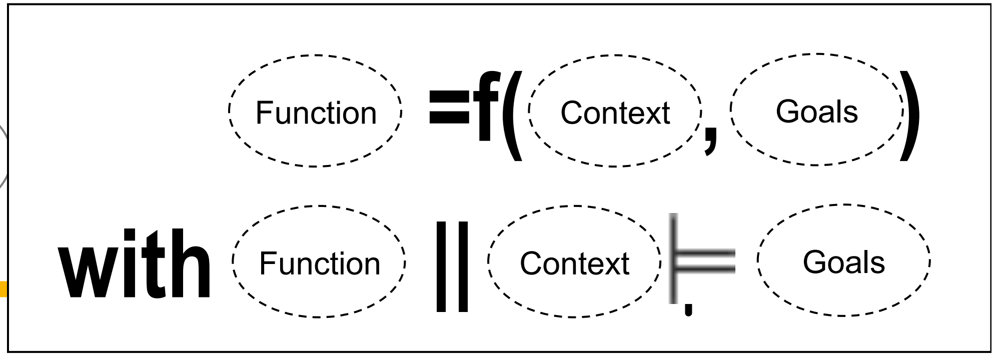


- **Cognitive Operators:** do distributed self-optimization
  - Distributed learning of a model of the track (environment)
  - Local learning of a model of the shuttle (system hardware)
  - Planning an adaptation in form of an optimal trajectory
- **Reflective Operator:** switch to robust local control if necessary

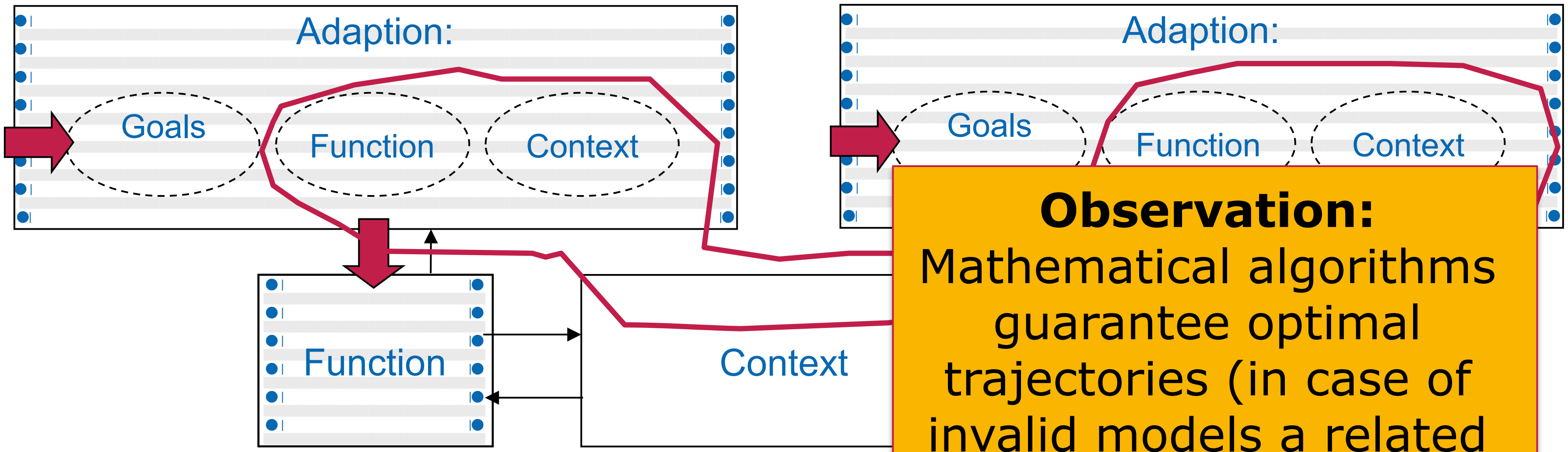
Models at Run-Time (2/2)

Development time:





Run-time:

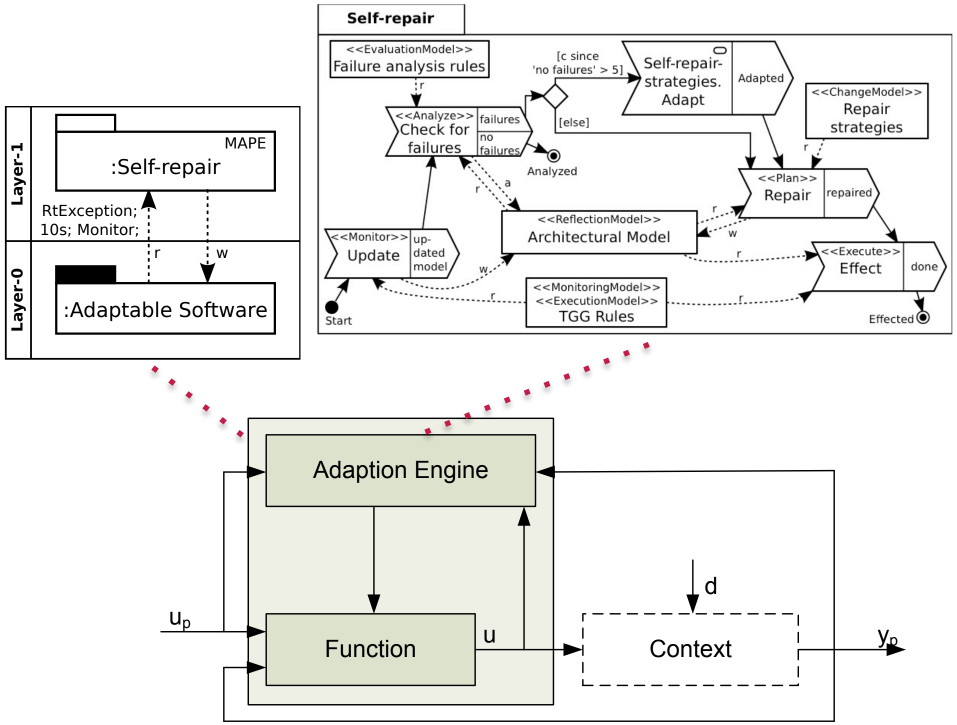


Observation: Mathematical algorithms guarantee optimal trajectories (in case of invalid models a related diagnosis and fallback to a robust control is used as backup)

Possible benefits: Up-to-date context models are
when multiple subsystems exchange data about

2. ExecUtable RuntimE MegA models (EUREMA)

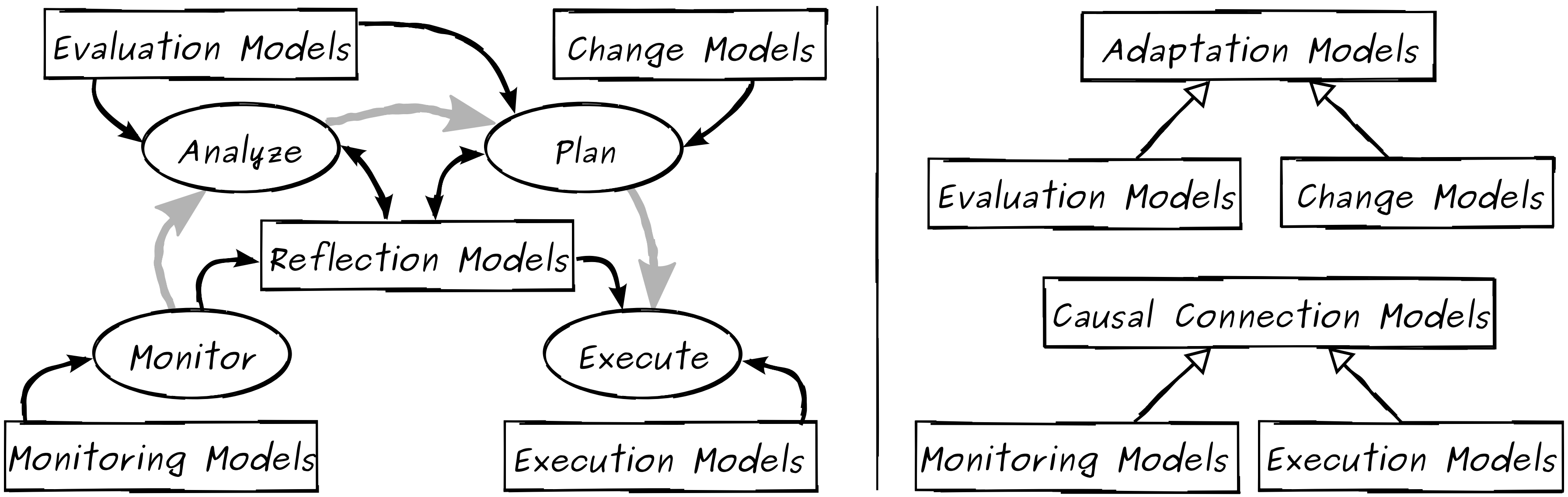
- Executable EMF megamodels kept alive at runtime with
  - Multiple runtime models
  - Activities are model operations (e.g., monitor + execute for EJBs with TGG)
  - Multiple loops
  - Multiple layers
  - Runtime interpreter for adaptation engines permits high degree of flexibility
- Leverages the co-existence of self-adaptation and evolution
- Modules and runtime models can to some extent be reused



Thomas Vogel and Holger Giese. A Language for Feedback Loops in Self-Adaptive Systems: Executable Runtime Megamodels. In Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012), pages 129-138, 6 2012. IEEE Computer Society.

homas Vogel and Holger Giese. 2014. Model-Driven Engineering of Self-Adaptive Software with EUREMA. *ACM Trans. Auton. Adapt. Syst.* 8, 4, Article 18 (January 2014), 33 pages. DOI=10.1145/2555612 http://doi.acm.org/10.1145/2555612

EUREMA: Knowledge & Runtime Models

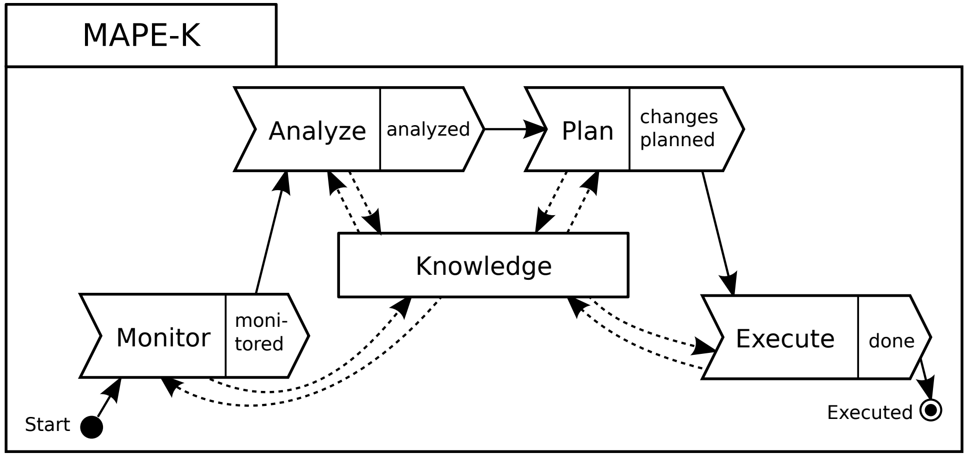


- Mega Model = „Model of Models and Operations on Models"

Idea:

- **Runtime models** are maintained at runtime
- **Runtime mega models** describe adaptation activities (MAPE)
- **Runtime interpreter** for runtime mega models

EUREMA: Use MDE for Model Operations

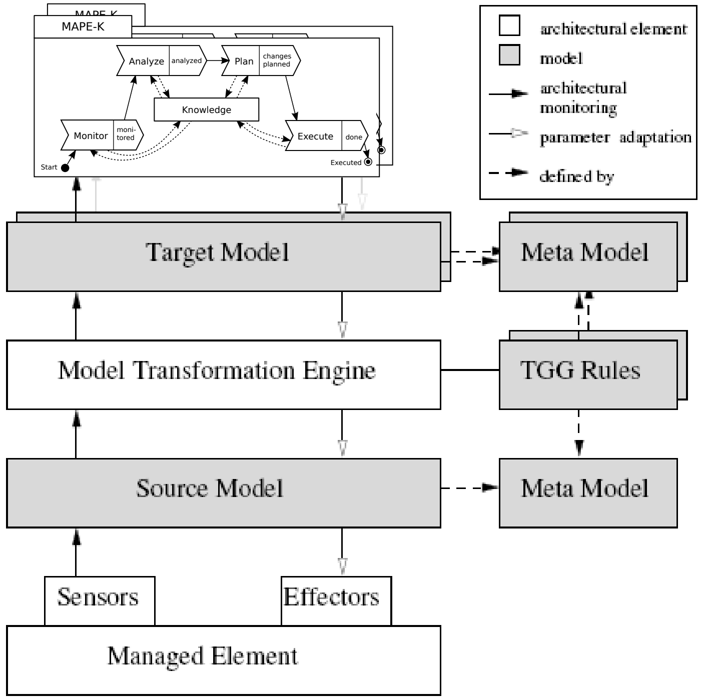


- **Options for using MDE for model operations:**
  - **Monitor/Execute:** techniques for model synchronization can be employed (e.g., Triple Graph Grammar (TGG))
  - **Analysis:** techniques that can operate on models with meta models such as OCL, model transformations, etc. can be employed.
  - **Plan:** techniques that can operate on models with meta models such as OCL, model transformations, etc. can be employed.

EUREMA: Use MDE for Model Operations (1/2)

MDE for Monitor/Execute:

- Employ **Triple Graph Grammar (TGG)** for the model operations monitor and execute (at once)
- Synchronize runtime models **incrementally** between the modules and the managed element (faster as manual implementations)
  - Extract abstract runtime models for different modules as required from **unchanged** EJB applications
  - Adapt managed subsystem incrementally via model (parameter and structural adaptation)



Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems. In: Proc. of the 6th International Conference on Autonomic Computing and Communications (ICAC'09), Barcelona, Spain, ACM (15-19 June 2009).

Thomas Vogel and Stefan Neumann and Stephan Hildebrandt and Holger Giese and Basil Becker. Incremental Model Synchronization for Efficient Run-Time Monitoring. In Sudipto Ghosh, ed., Models in Software Engineering, Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009, Reports and Revised Selected Papers, vol. 6002 of Lecture Notes in Computer Science (LNCS), pages 124-139. Springer-Verlag, 4 2010.

EUREMA: Use MDE for Model Operations (2/2)

Hasso Plattner Institut

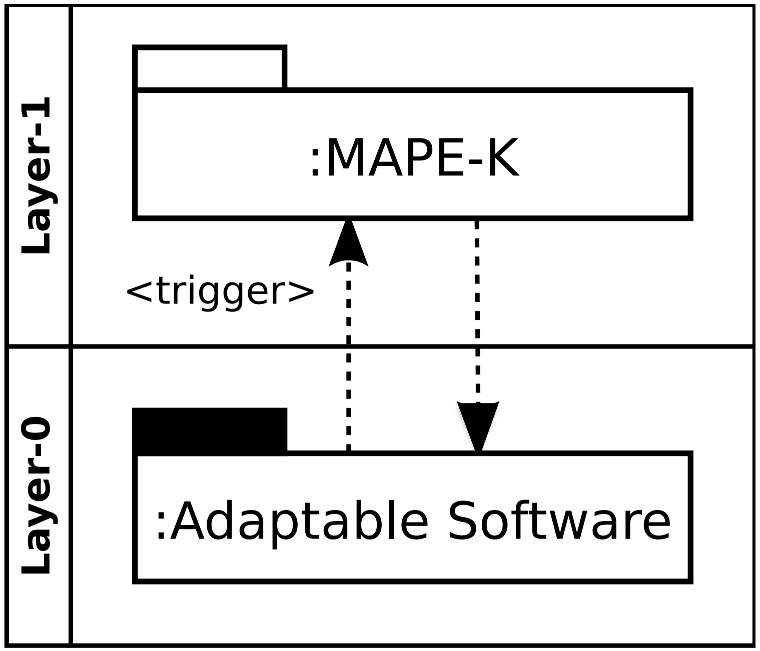
- **Benefits:**
  - The supported incremental processing provides low overhead monitoring and executing solution
  - Permits sensors and effectors at a higher level of abstraction
  - Model transformation technique can be used to map this high level information to analysis models used by the EUREMA modules

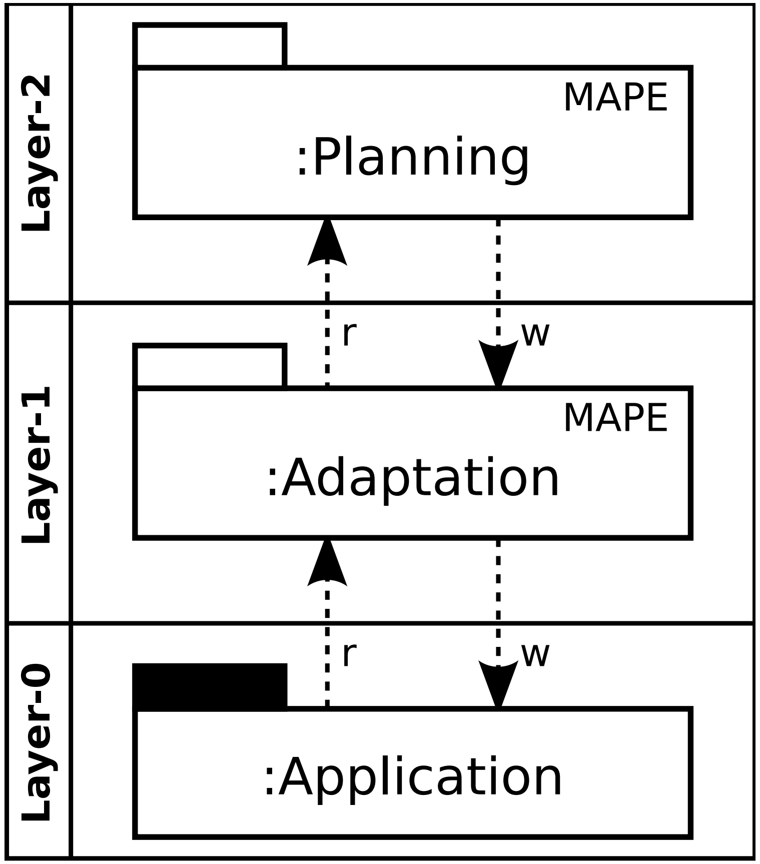
Target Model	Model-Driven Approach			NIA
	Rules	Nodes/Rules	LOC	LOC
Simpl. Architectural Model	9	7,44	15259	357
Performance Model	4	6,25	5979	253
Failure Model	7	7,14	12133	292
Sum	20		33371	902

Size	NIA		Model-Driven Approach						
	S	B	n=0	n=1	n=2	n=3	n=4	n=5	B
5	8037	20967	0	163	361	523	749	891	10733
10	9663	43054	0	152	272	457	585	790	23270
15	10811	72984	0	157	308	472	643	848	36488
20	12257	105671	0	170	325	481	623	820	55491
25	15311	142778	0	178	339	523	708	850	72531

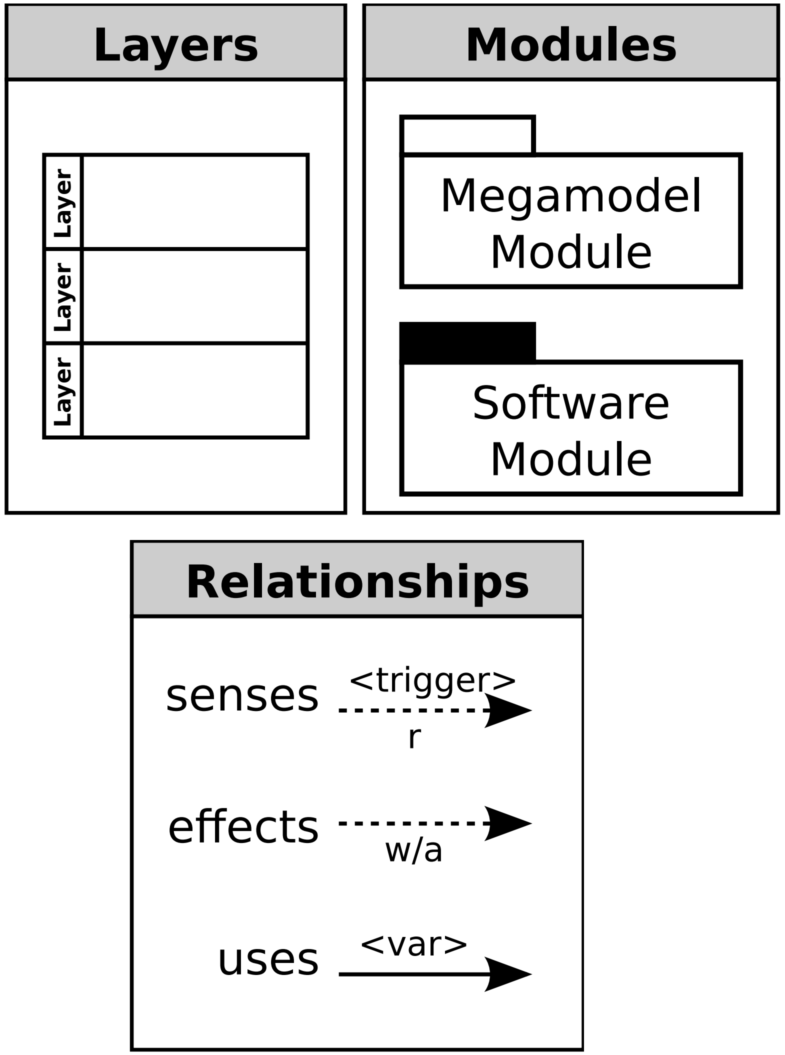
- **Limitations:**
  - One generic adapter to the model world is initially required that requires usually more effort than an ad hoc monitoring effort

Layer Diagrams: Example



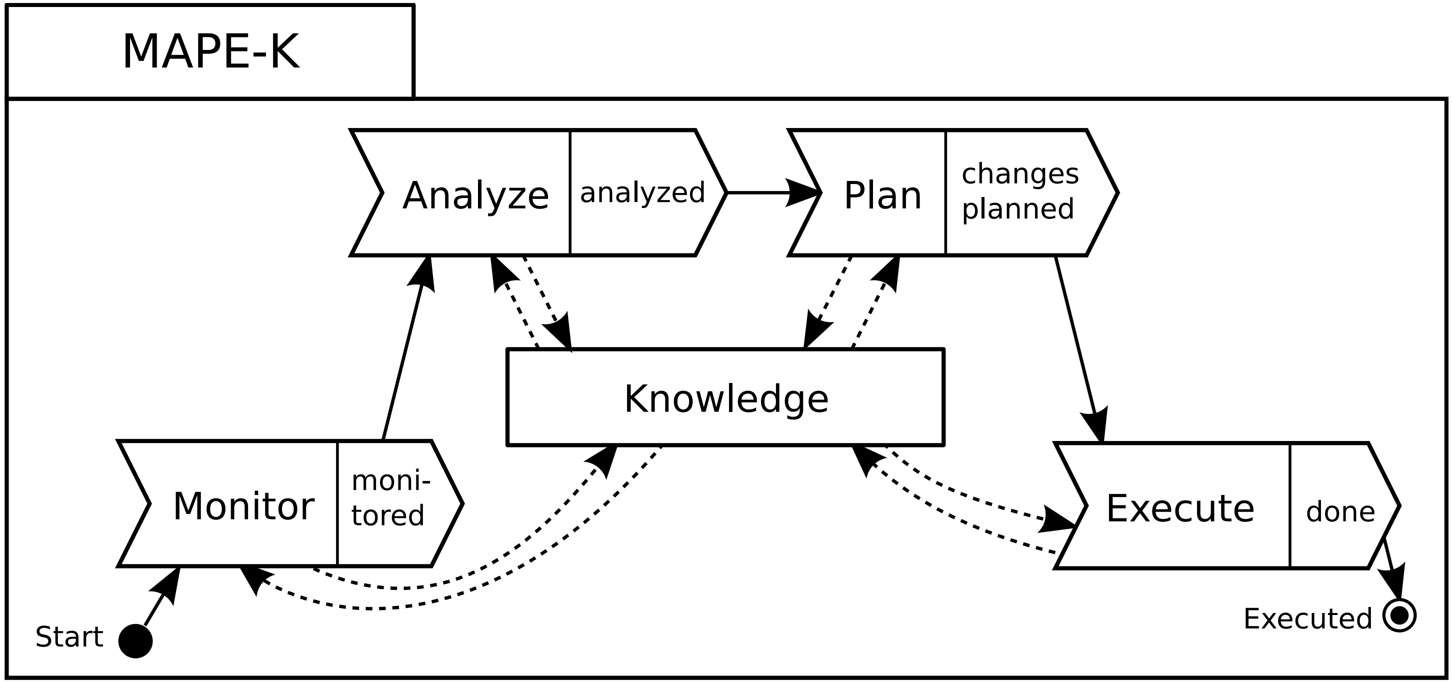


Layer Diagrams: Notation

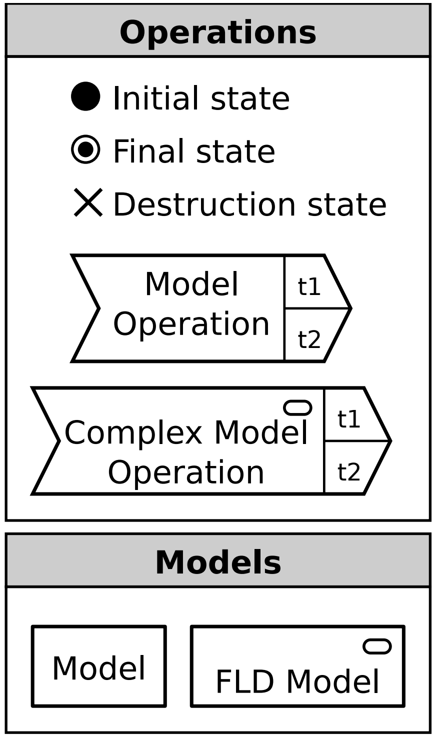


- **Main concepts:**
- **Layers:**
  - Layer 0: core software
  - Layer 1: adaptation engine
  - Layer 2: higher-order adaptation behavior (e.g., planning)
- **Modules:**
  - megamodel modules: FDLs
  - software modules: legacy software
- **Relationships:**
  - Sense: trigger modules
  - Effects: effects of the modules
  - Use: use of megamodel elements of a module

Feedback Loop Diagrams (FLDs): Example

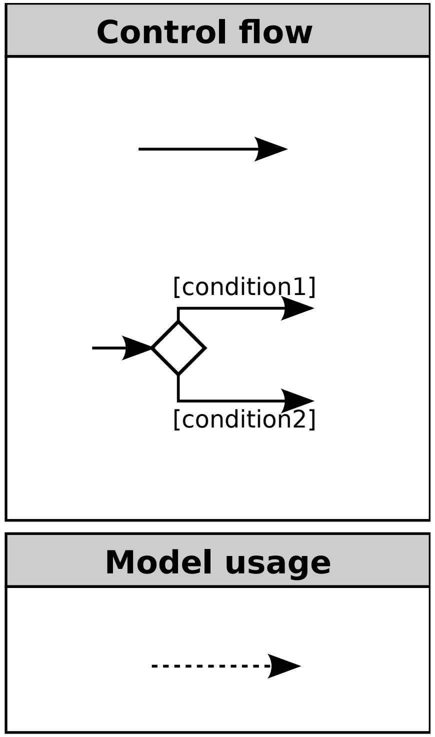


Feedback Loop Diagrams (FLDs)



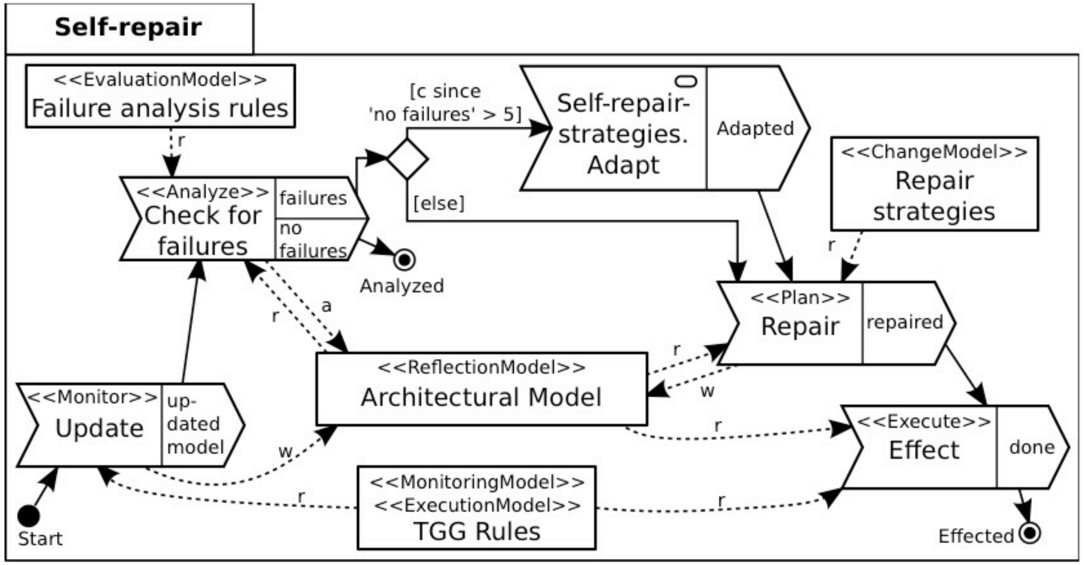
- **Concepts:**
  - **Helper states:**
    - Initial state: start of the execution
    - Final state: end of the execution
    - Destruction state: end of the execution and termination of the module
  - **Model operations:**
    - Simple model operations: mapped to software or other modeling techniques (e.g., TGGs)
    - Complex model operations: mapped to modules
  - **Models:**
    - Runtime models
    - EUREMA models

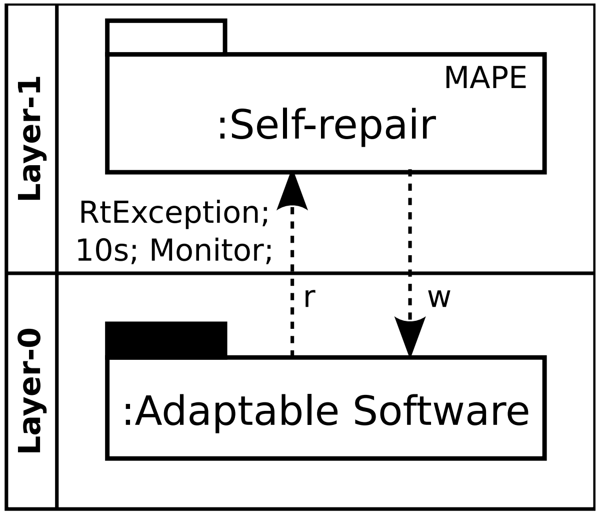
Feedback Loop Diagrams (FLDs)



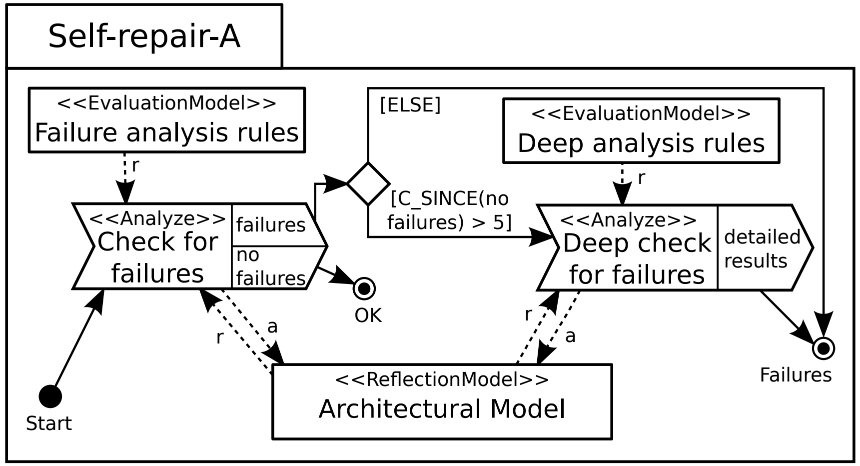
- **Concepts:**
  - **Control flow:**
    - Arrow: ordering
    - Rhombus: alternative flows of control
  - **Model usage:**
    - r: read
    - w: write
    - a: append

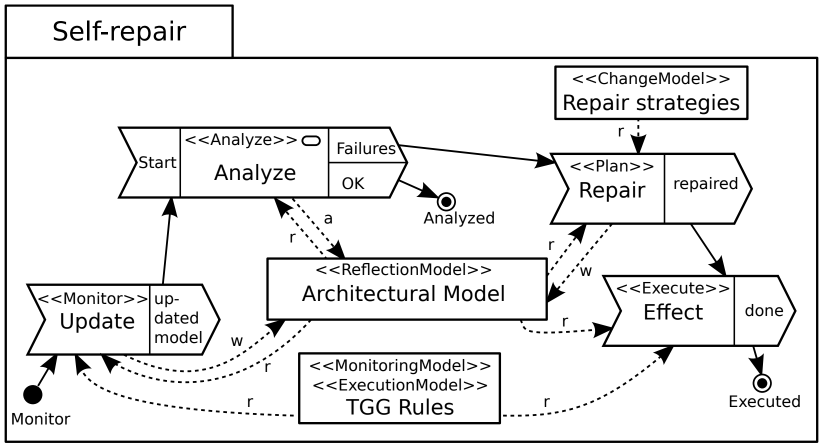
EUREMA: Self-Repair Example

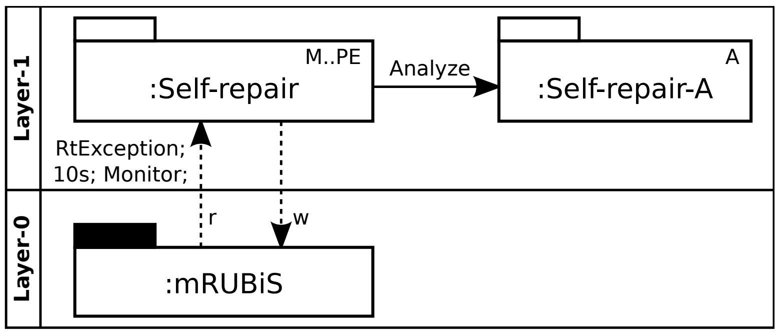




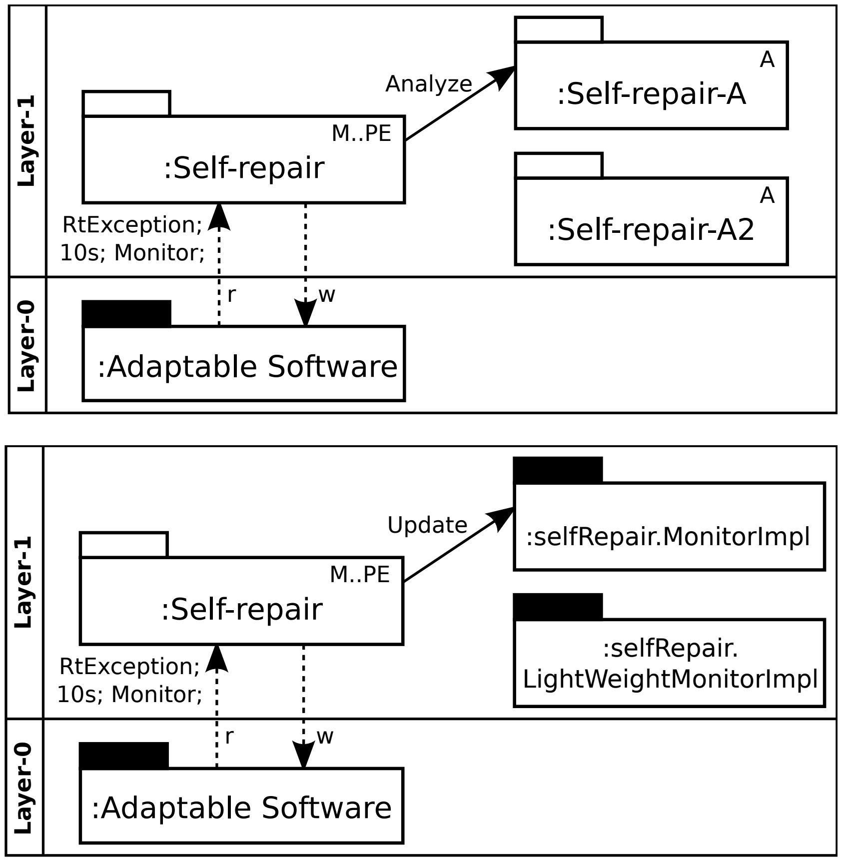
EUREMA: Modular Self-Repair Example





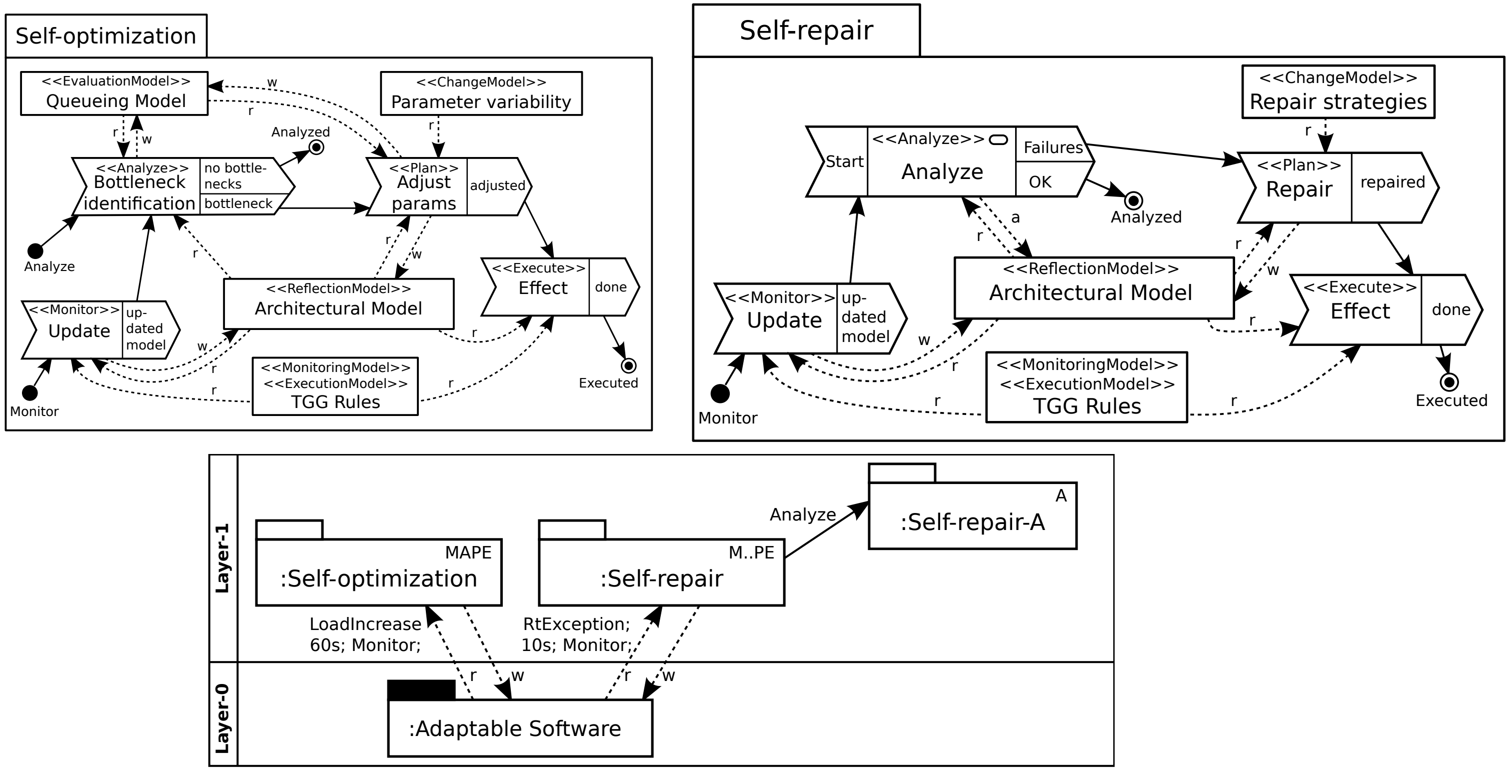


EUREMA: Alternatives & Variability Modeling



- EUREMA models can already include alternatives (variability) that can be activated by adjusting the EUREMA model at runtime.
  - Module-level
  - Software-level

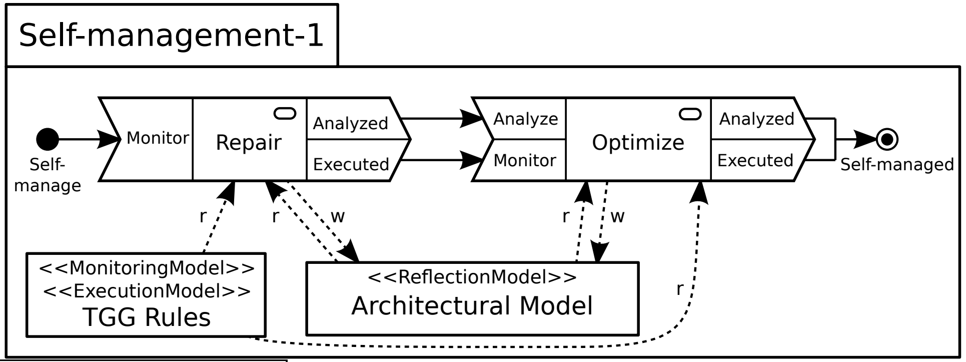
EUREMA: Independent MAPE Loops

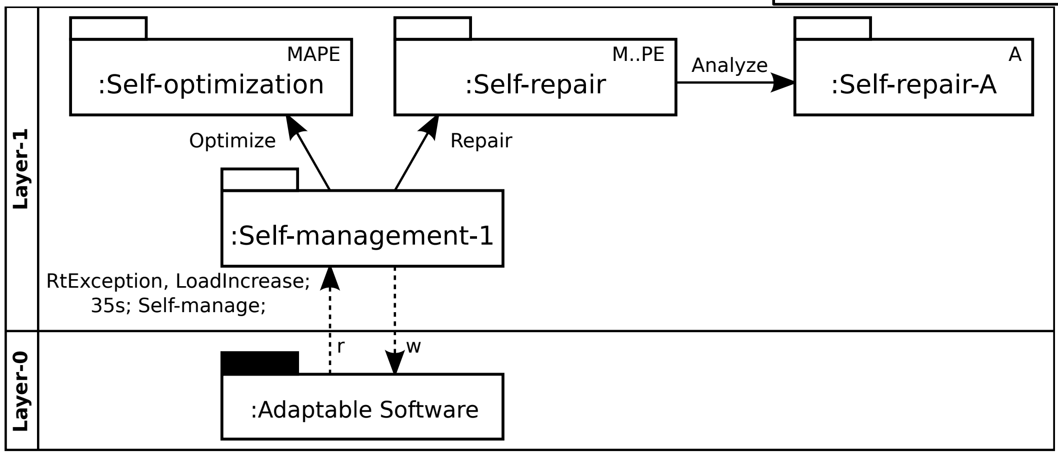


Solution:

- Use independent triggers for both loops
- Sequential execution will ensure that loops do not overlap

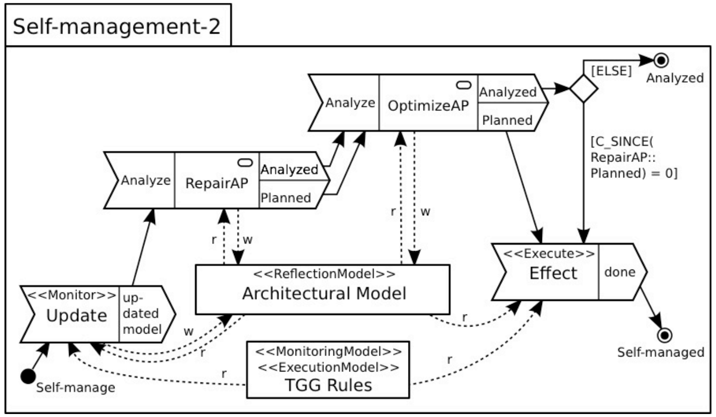
EUREMA: Sequencing MAPE Loops Completely

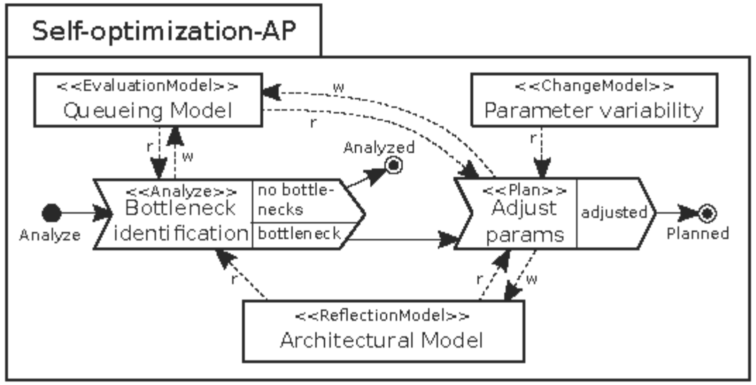


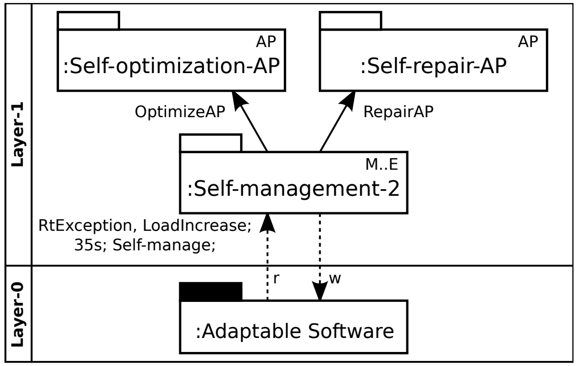


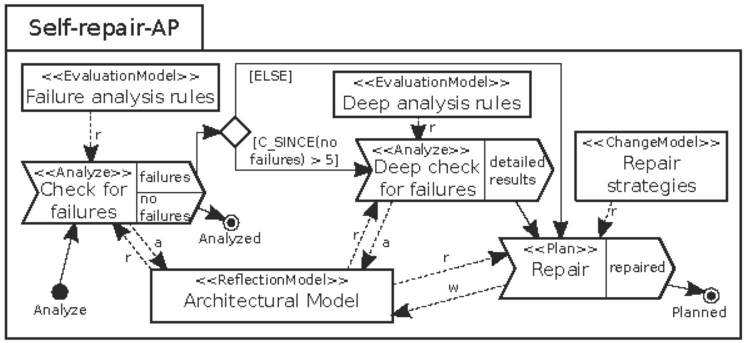
- **Solution:**
  - Extra module enforces the sequential execution such that the loops do not overlap

EUREMA: Sequencing AP of MAPE



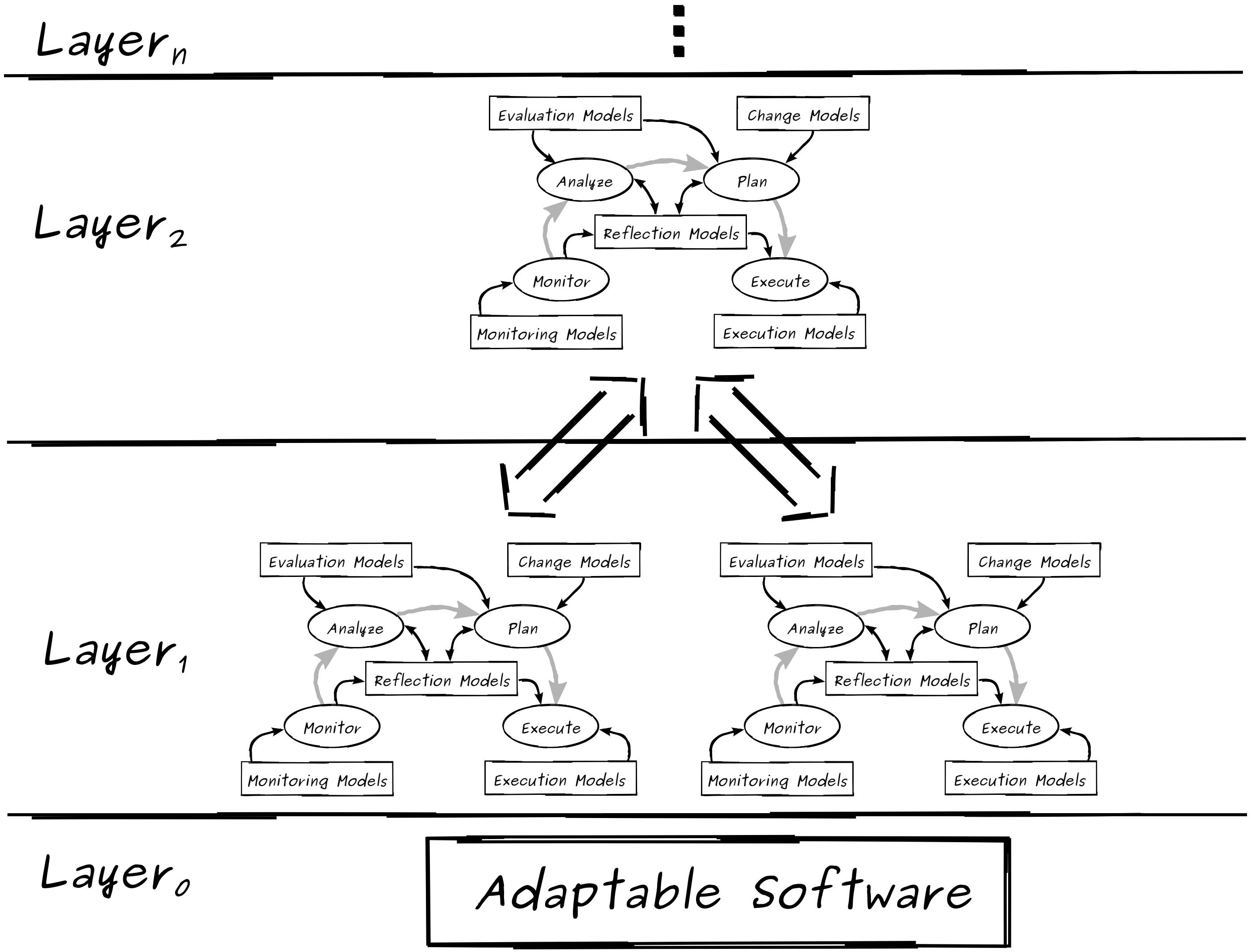




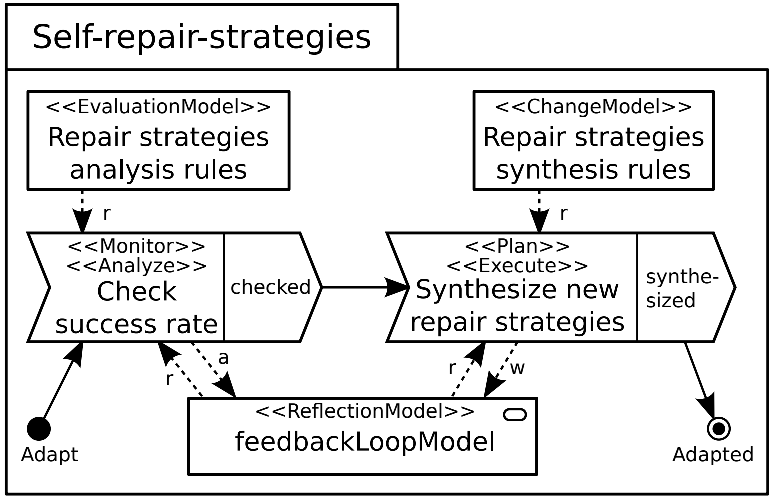


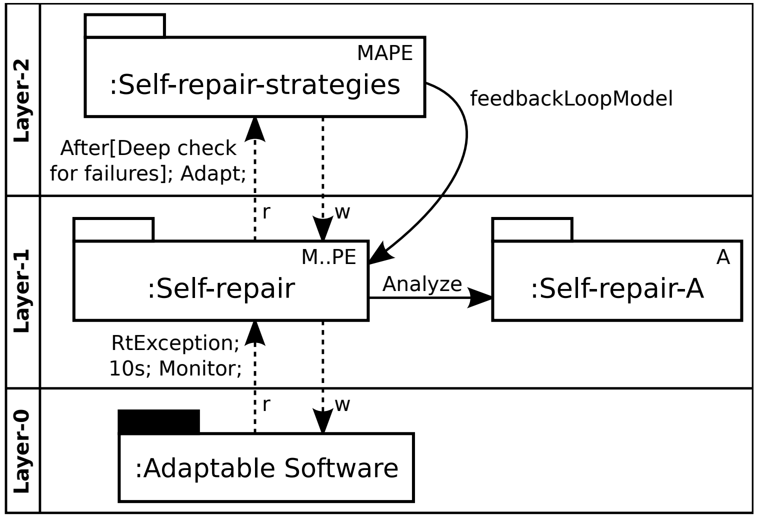
- **Solution:**
  - Join monitor and execute activities
  - Extra module enforces the sequential execution

EUREMA: Multiple Layers



EUREMA: Reflection via the Megamodels

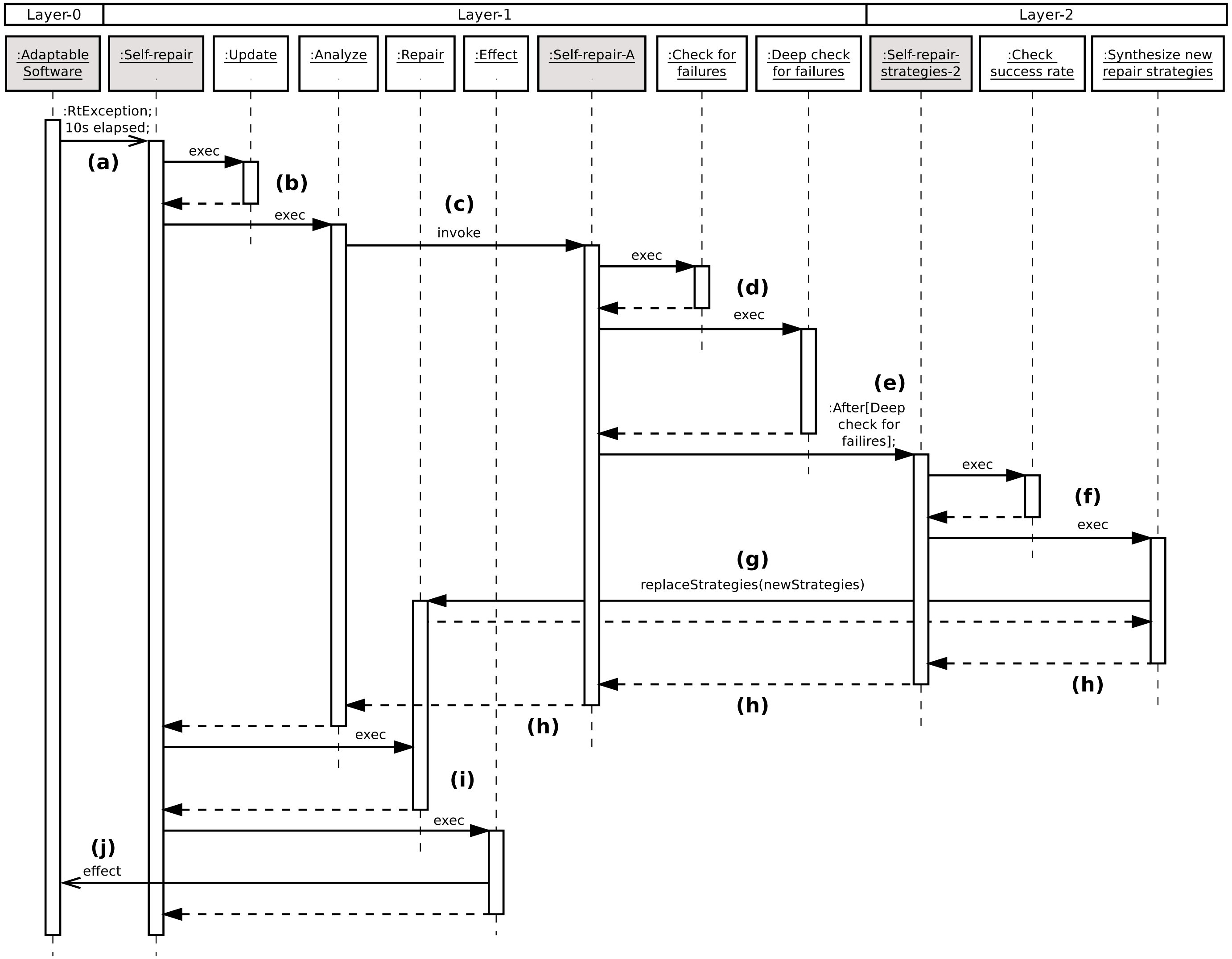




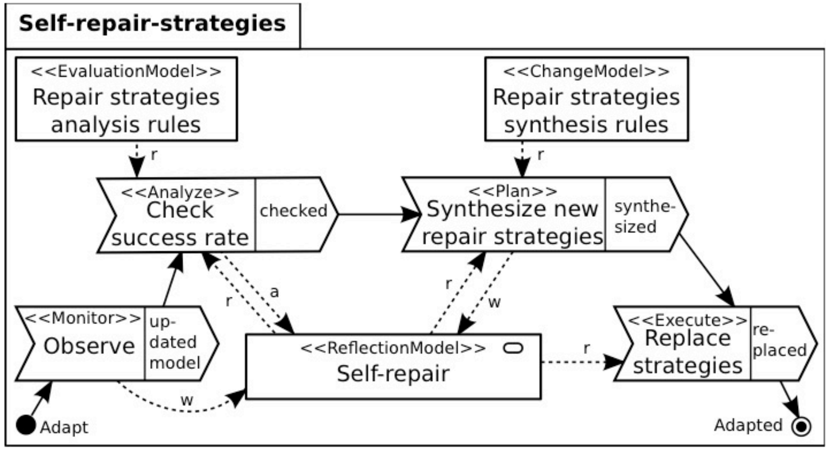
- **Benefits:**
  - No extra model has to be developed
  - Causal connection is guaranteed by construction
- **Disadvantages:**
  - No abstraction of the underling layer
  - No temporal decoupling as no copy is maintained

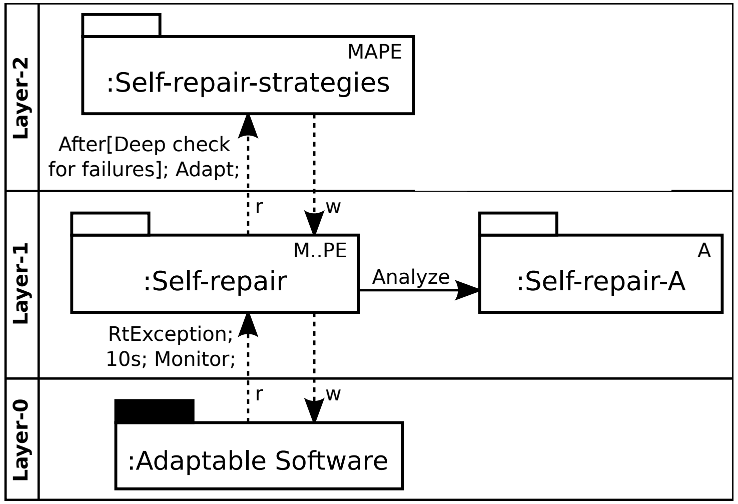
Complex Behavior of Self-Adaptation Activities

Hasso Plattner Institut



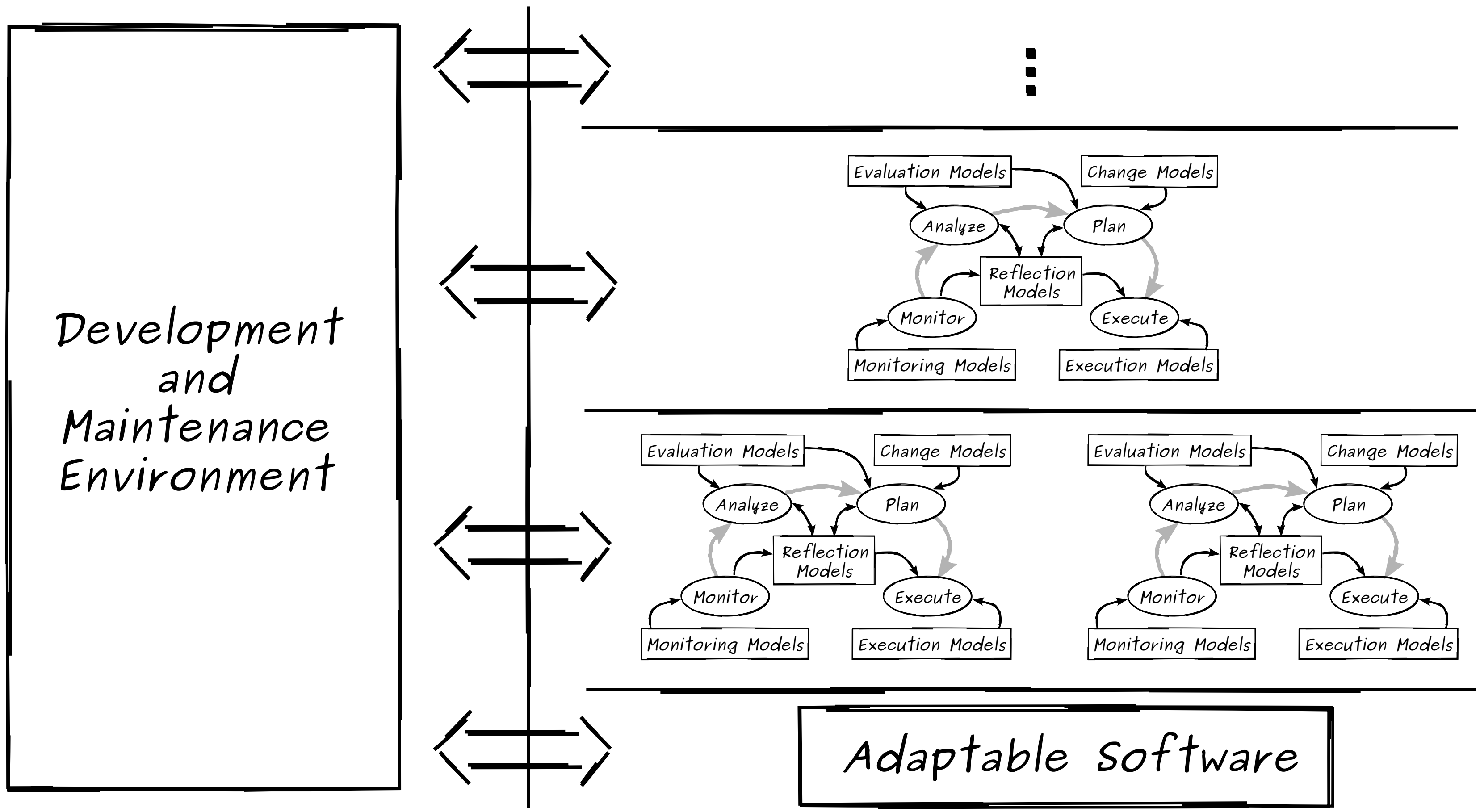
EUREMA: User-Defined Reflection





- **Benefits:**
  - Abstraction of the underling layer
  - Temporal decoupling
- **Disadvantages:**
  - Extra model has to be developed
  - Causal connection has to be maintained explicitly

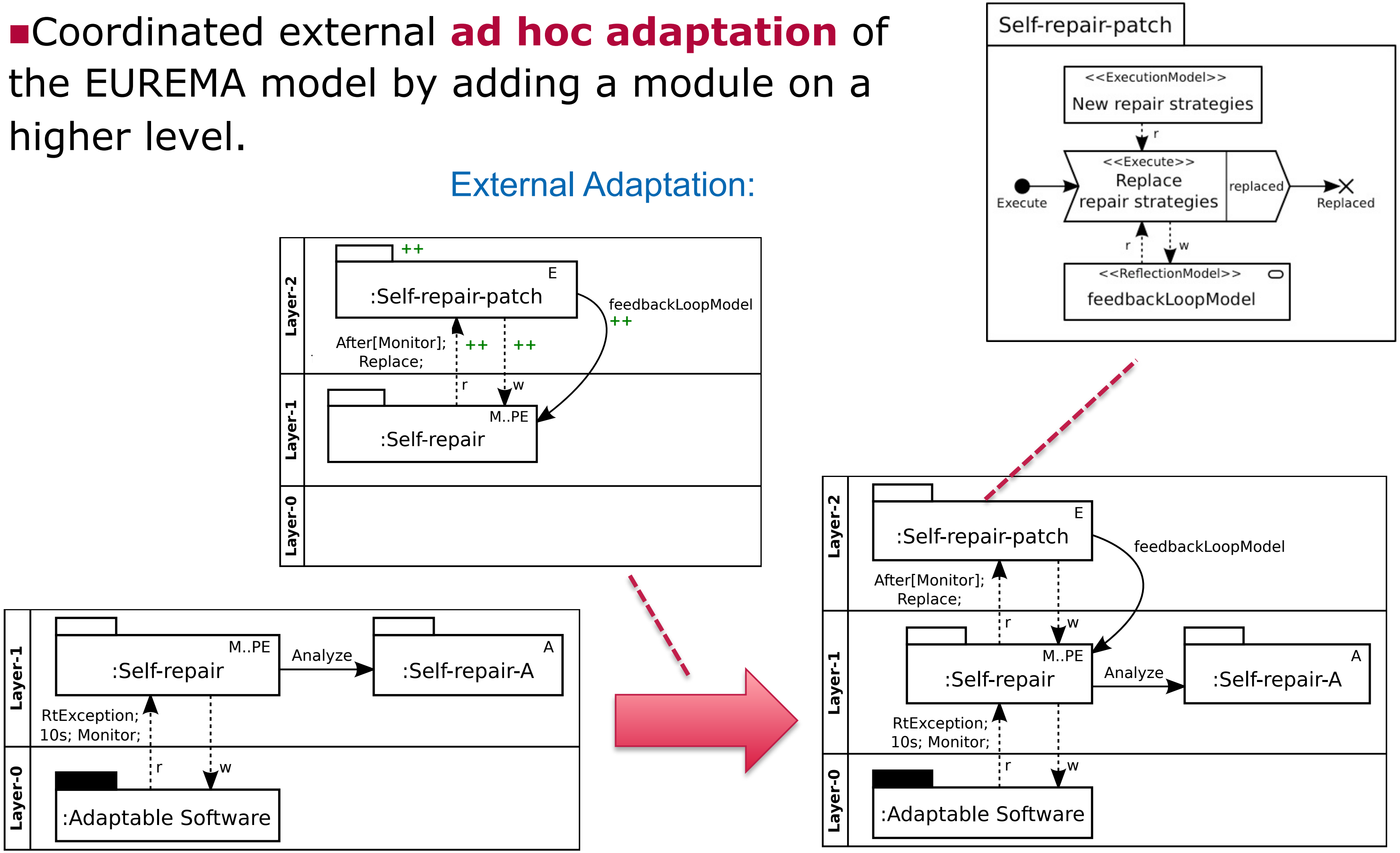
Co-Existence of Self-Adaptation & Maintenance



EUREMA: Evolution via Off-line Adaptation (1/2)

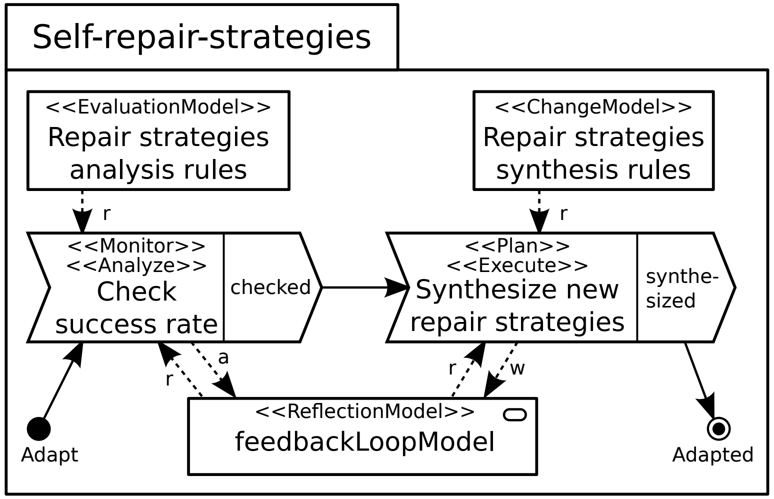
- Coordinated external **ad hoc adaptation** of the EUREMA model by adding a module on a higher level.

External Adaptation:

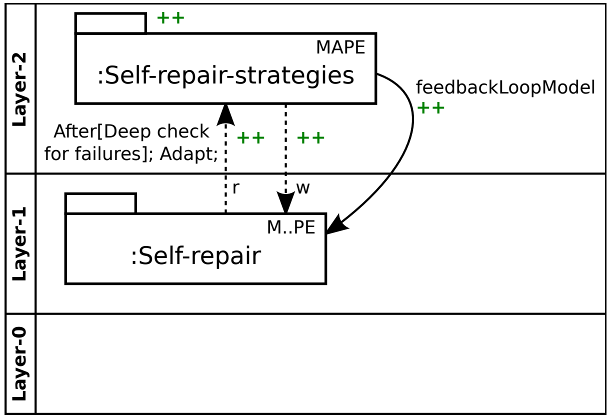


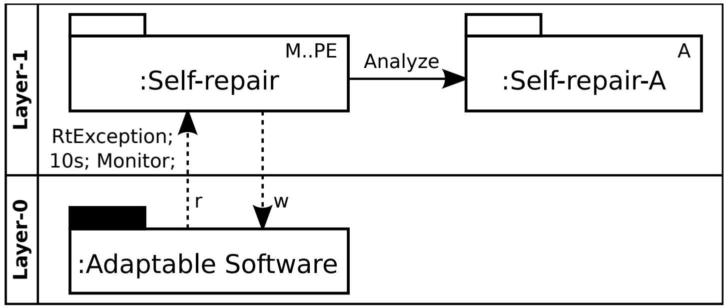
EUREMA: Evolution via Off-line Adaptation (2/2)

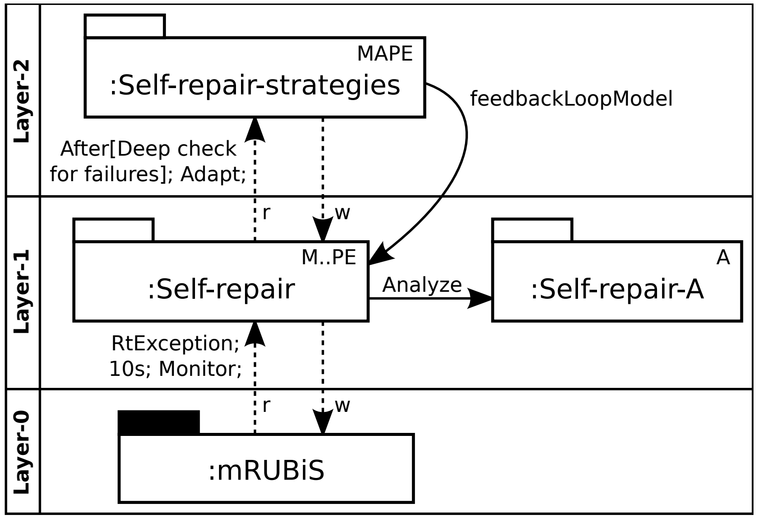
- Add **self-adaptation layer** in an EUREMA model on the fly.



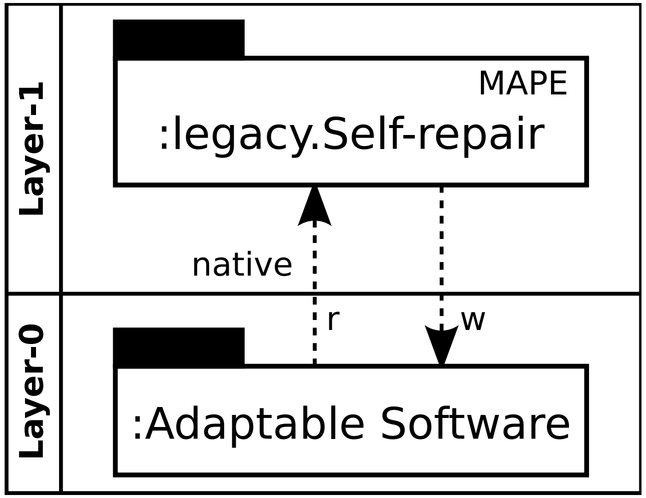
External Adaptation:

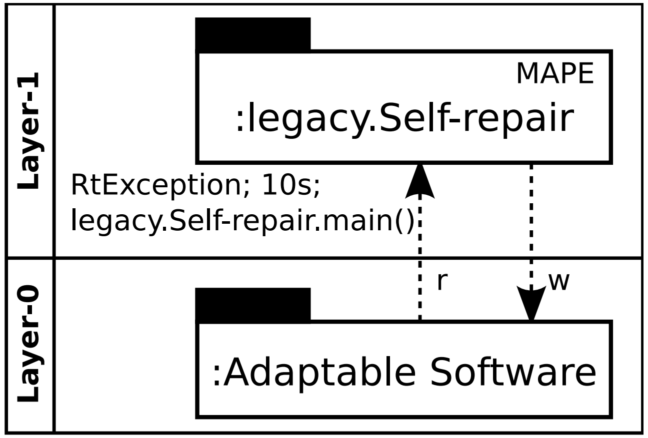






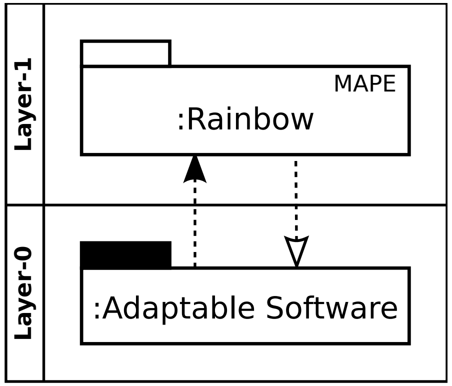
EUREMA: Legacy & Triggering

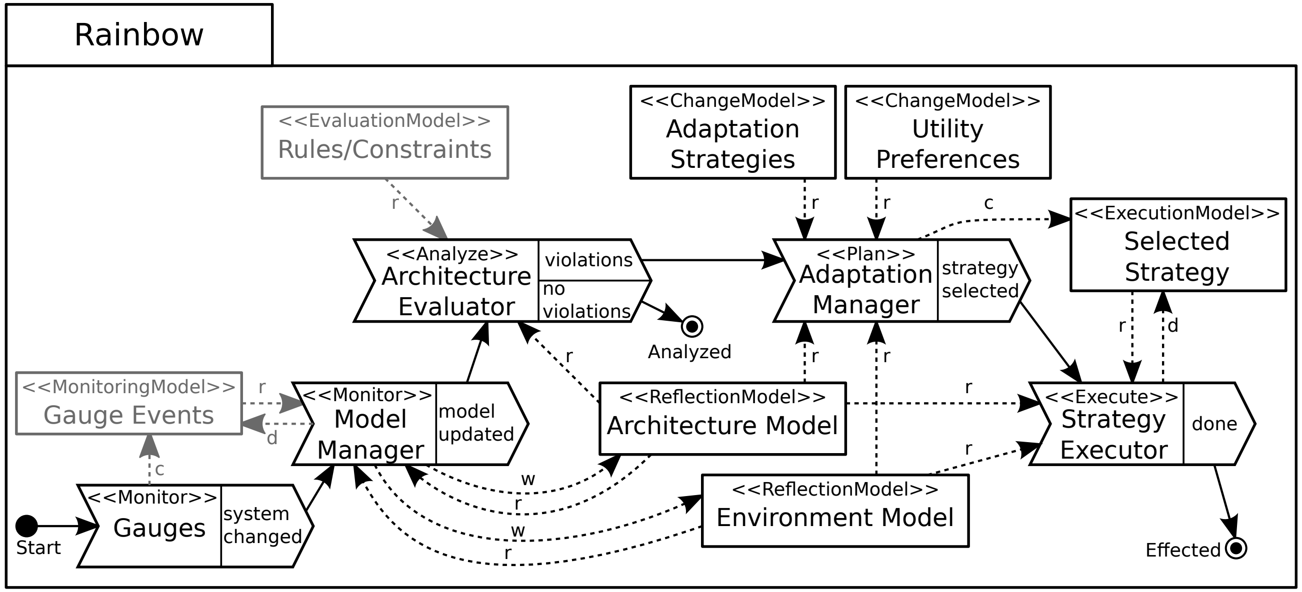




- **Options:**
  - Only model native triggering with EUREMA (no evolution is possible later)
  - Model and realize triggering with EUREMA (evolution is possible later !)

EUREMA: Modeling Rainbow





- The general separation of the adaptation behavior into runtime models and activities can be captured. Emulation would in addition permit evolution due to the co-existence with offline maintenance.

EUREMA: Discussion

Hasso Plattner Institut

- Model-driven engineering approach for adaptation engines
- Domain-specific modeling language for layers, modules, and control flow
- Leverages advanced solutions, like layered feedback loops
- Executable megamodels are kept alive at runtime
- Runtime models are employed at runtime
- Runtime interpreter for adaptation engines permits high degree of flexibility
- Leverages the co-existence of self-adaptation and off-line adaptation for evolution
- Modules and runtime models can to some extent be reused

- **Limitations:**
  - Concurrency and a distributed setting are not supported yet
  - …

3. Challenges Ahead
- enable self-aware computing

- Capture goals explicitly
- Guide tradeoffs
- …

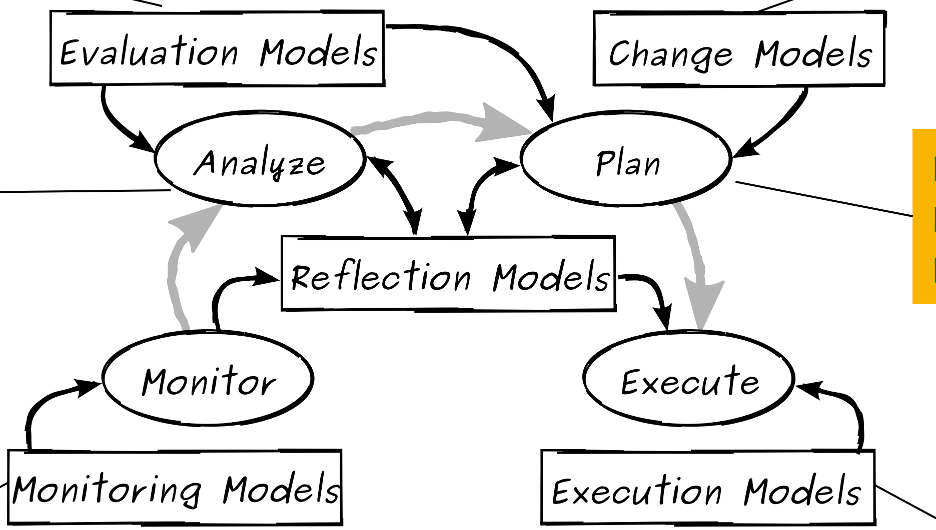
- Capture possible changes
- …

param

param

- Predictions
- …

- Selection
- Synthesis
- …



- Learning model from observations
- Abstract from details
- …

- Enact runtime model for the adaptable software
- Add details
- …

Loop:
- **Speed vs. accuracy**
- **exploration vs. exploitation**
- **limits/guarantees for model learning**
- **proper treatment of uncertainty**

REUSE!

Challenges Ahead
- long term (meta self-aware?)

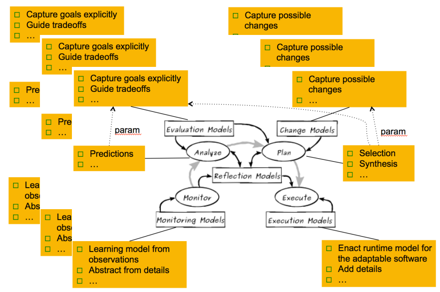
- Executable mega models kept alive at runtime with
  - Multiple runtime models
  - Activities are model operations (e.g., TGG)
  - Multiple loops
  - Multiple layers
  - Runtime interpreter for adaptation engines permits high degree of flexibility
- Leverages the co-existence of self-adaptation and evolution
- Modules and runtime models can to some extent be reused

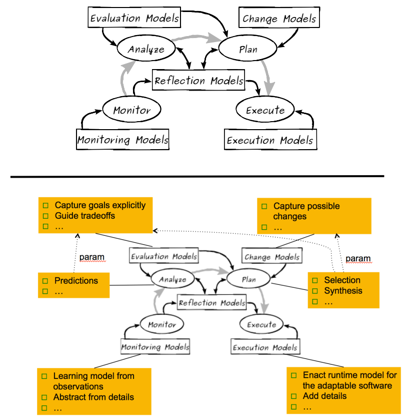
Coordinate multiple loops for:

- Multiple goals
- Multiple runtime models
- Multiple activities (runtime model operations)
  - Learning strategies
  - Prediction strategies
  - Synthesis strategies
- …

Higher layer must steer:

- diverse runtime models: number + selection
- diverse activities (runtime model operations): number + selection
- speed / accuracy
- exploration / exploitation
- …





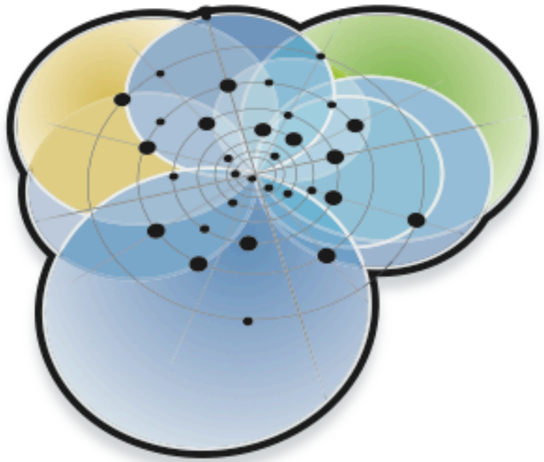
4. Outlook: Beyond centralized MAPE-K …

HPI Hasso Plattner Institut

(Networked)
Cyber-Pyhsical Systems

Internet of Things

Smart City



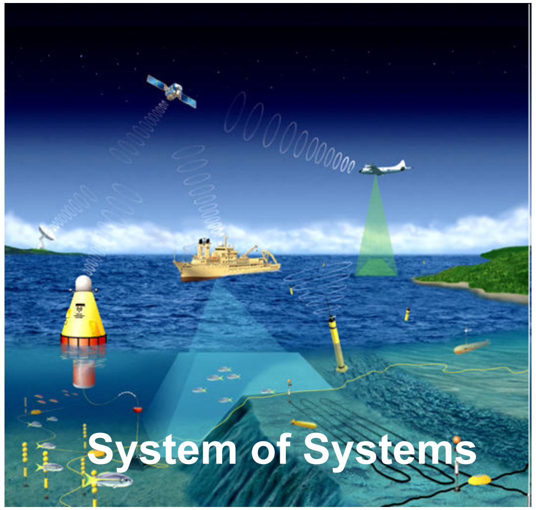
Ultra-Large-Scale Systems

Smart Home

E-Health

Smart Factory -
E.g. Industry 4.0

Smart Logistic



http://oceanservice.noaa.gov/news/weeklynews/nov13/ioos-awards.html

Ambient
Assisted Living

Micro Grids

Collabrarive self-aware computing
- Exchange runtime models